Computer-Graphics-Module-3-Important-Topics

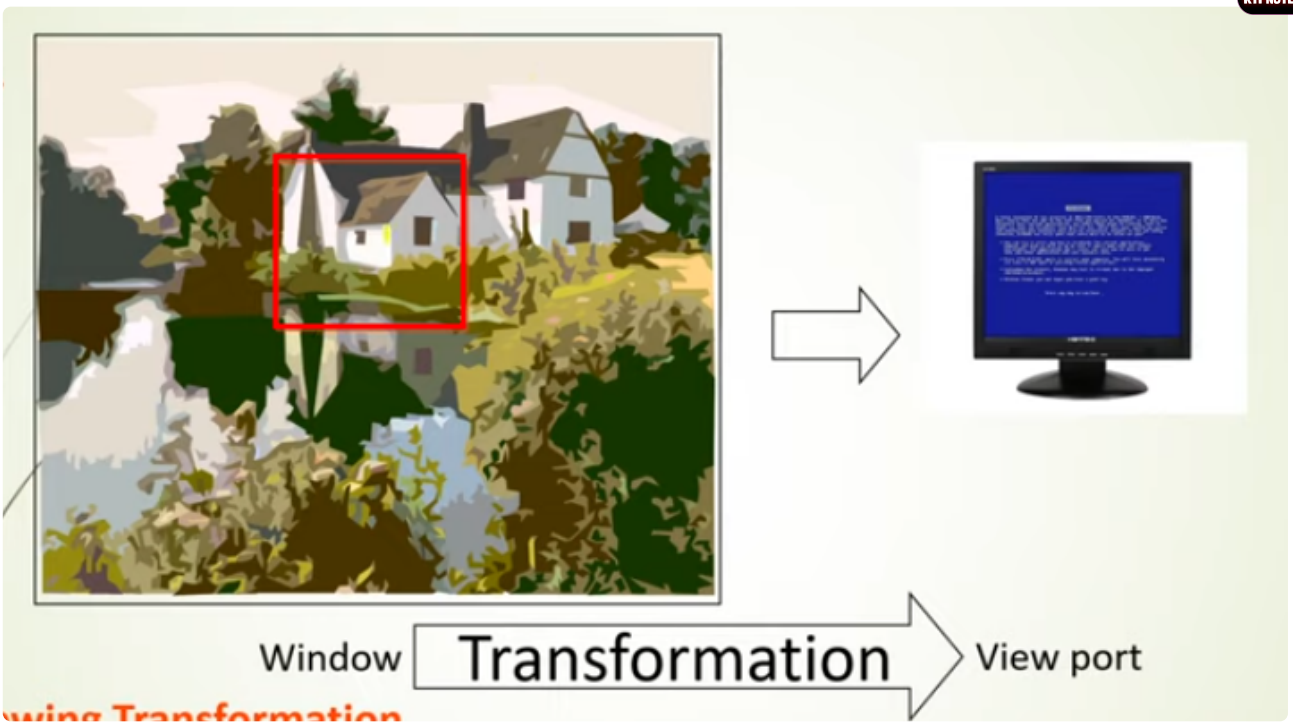
1. Window to viewport

What is window?

- Window is the area where image is viewed in real world
- Defines what is to be displayed

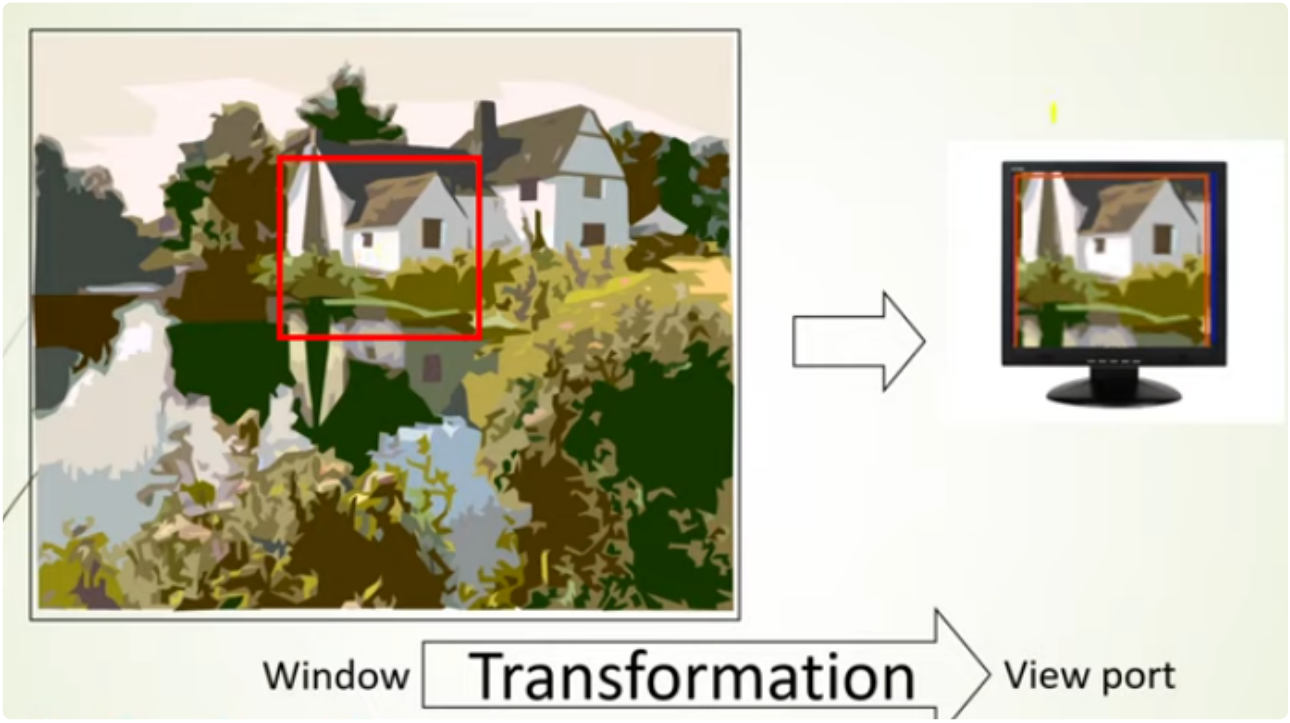
What is a view port?

- Its the area where image is viewed in o/p device
- defines where it is to be displayed

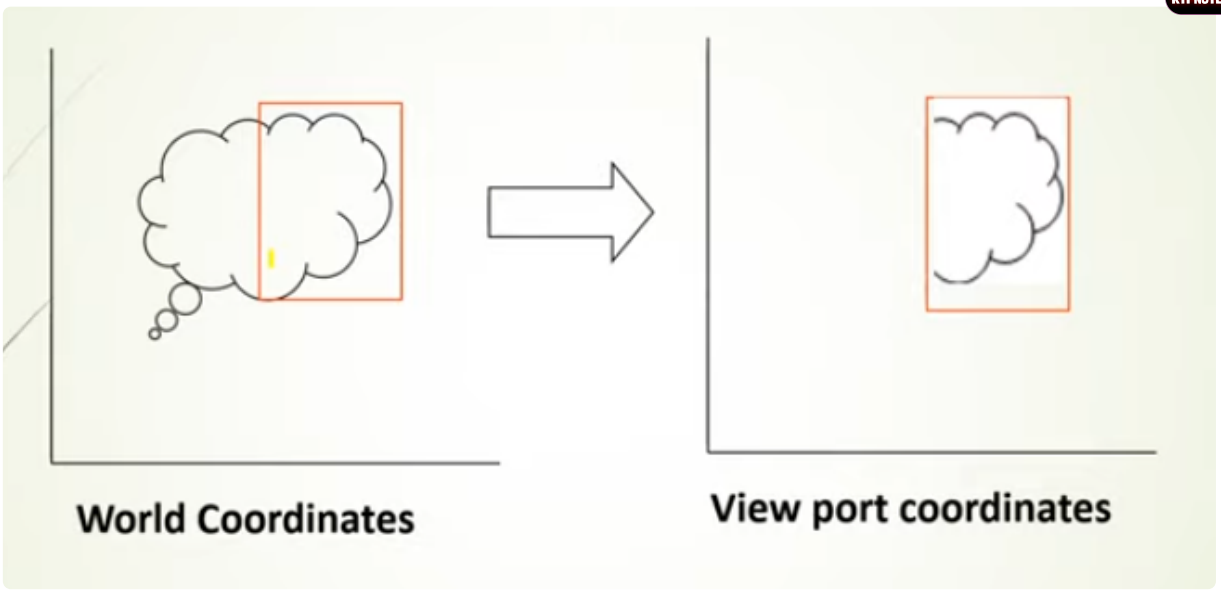
- 

- Here the Red box is the window, and defines what is to be displayed
- The monitor is the viewport

Window to viewport transformation

- The contents of window is transferred to the viewport
- 

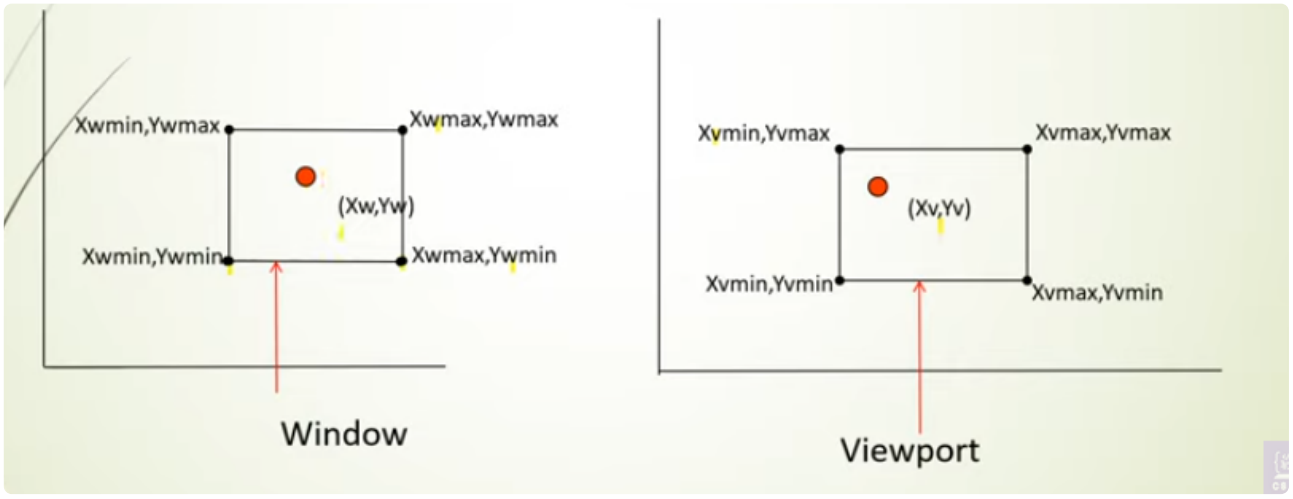
- Window to viewpoer transformation can be defined as the mapping of world coordinate scene to device coordinates
- Another example

- 

Clipping

- Define the window area
- Discarding portions outside window is called clipping

Window to viewport transformation - Relation

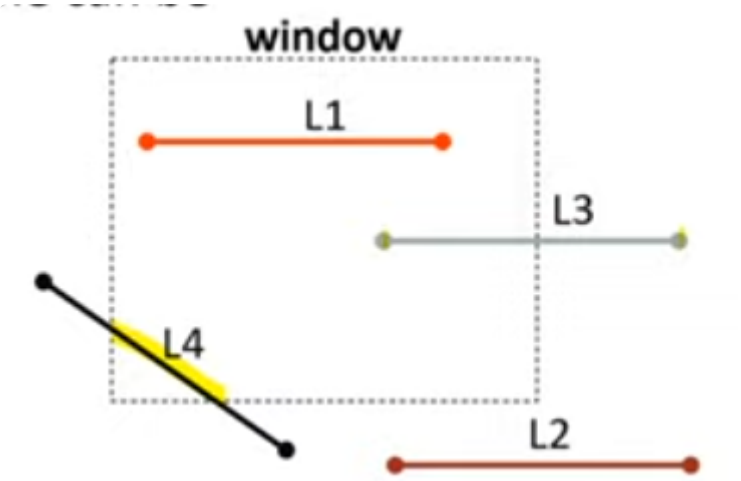
- For every point (Xw,Yw) in the window there is a corresponding or equivalent of relative point(Xv,Yv) in viewport
- 

---

2. Line clipping problem(Cohen Sutherland, Sutherland hodgeman)

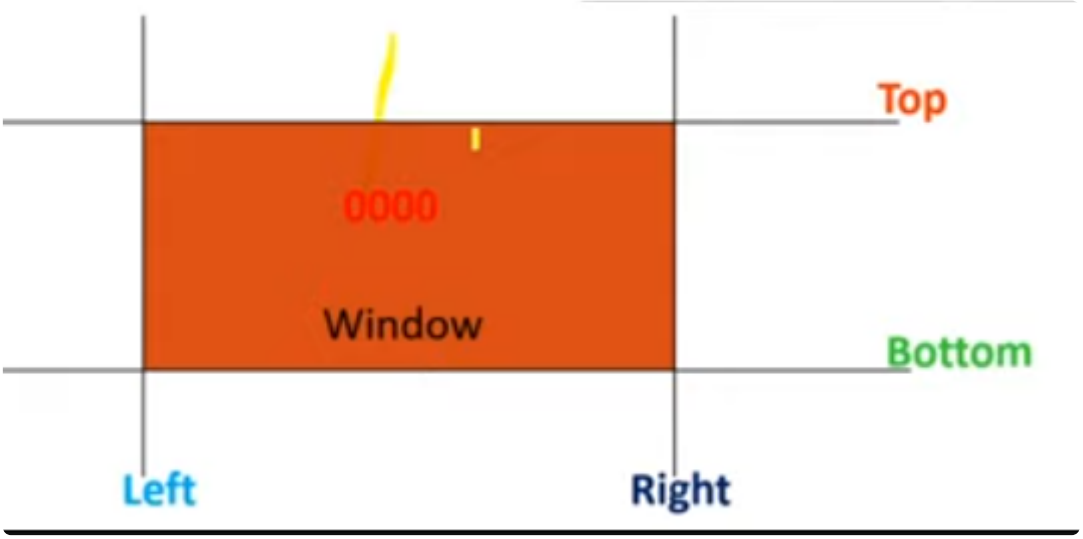
Cohen Sutherland problem

- With respect to a window, the line can be
  - Perfectly inside (L1)
  - Perfectly outside (L2)
  - Partially inside (L3)
  - Endpoints outside, Line segment inside (L4)



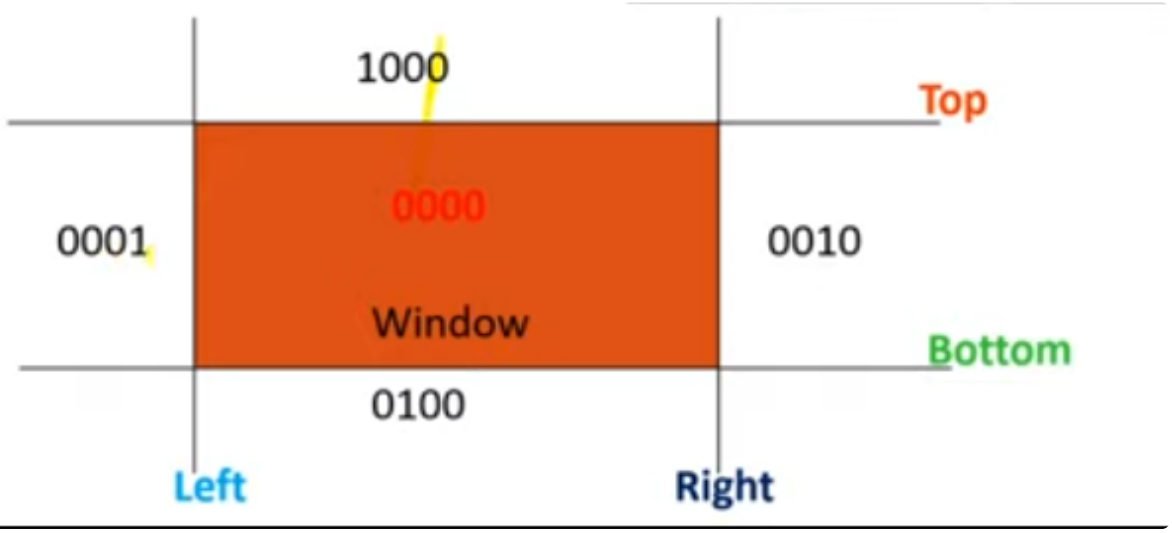
- If the line is
  - Perfectly inside (L1) -- Accept the line
  - Perfectly outside (L2) -- Reject the line
  - Crosses window boundary -- Clip the line (L3 and L4)
- For Clipping, we use Cohen Sutherland Line Clipping algorithm
- Uses a coding scheme where each endpoint is assigned a region code
  - Region code = 0, if inside boundary
  - Region code = 1 if outside boundary
- This is the bit pattern that is used
  - 

- Example 1, A point in the middle



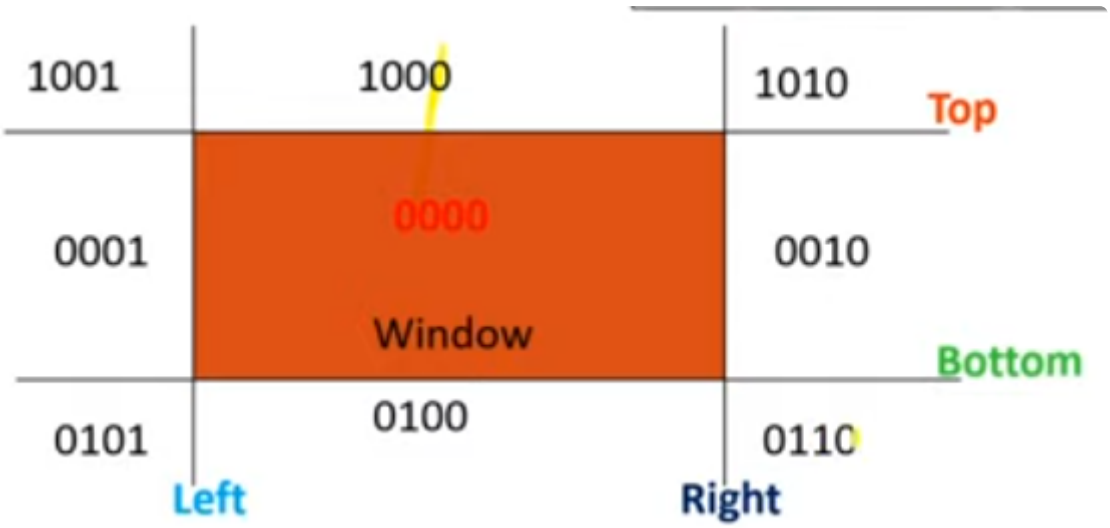
- Since the point is in the middle
  - Top? No, Assign value 0
  - Bottom? No, Assign value 0
  - Right? No, Assign value 0
  - Left? No, Assign value 0
- We get Top Bottom Right Left = **0 0 0 0** as the bit pattern

- Example 2, Points at top, bottom, right and left



- Top point
  - Top? Yes, 1
  - Bottom? No. 0
  - Right? No. 0
  - Left? No. 0
  - Bit pattern = 1000
- Bottom point

        - Top? No, 0
        - Bottom? Yes. 1
        - Right? No. 0
        - Left? No. 0
        - Bit pattern = 0100
    - Left = 0001
    - Right = 0010
- Example 3, Corners

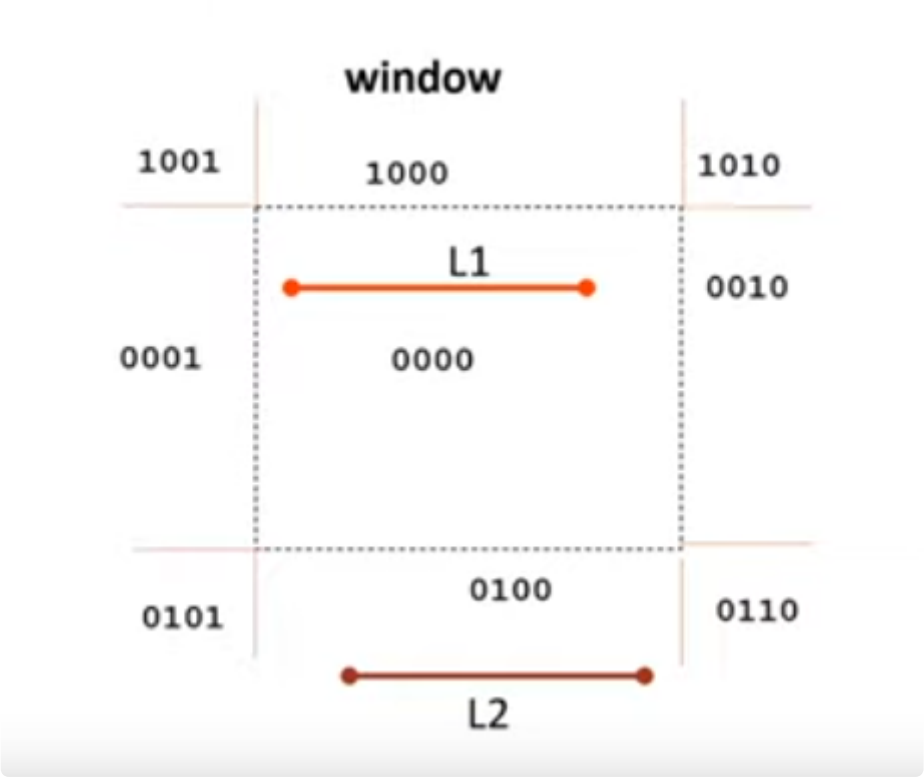


    - Top left corner
        - Top? Yes, 1
        - Bottom? No. 0
        - Right? No. 0
        - Left? Yes. 1
        - Bit pattern = 1001
    - Top right corner
        - Top? Yes, 1
        - Bottom? No. 0
        - Right? Yes. 1
        - Left? No. 0
        - Bit pattern = 1010
    - Bottom Left = 0101
    - Bottom Right = 0110

Algorithm

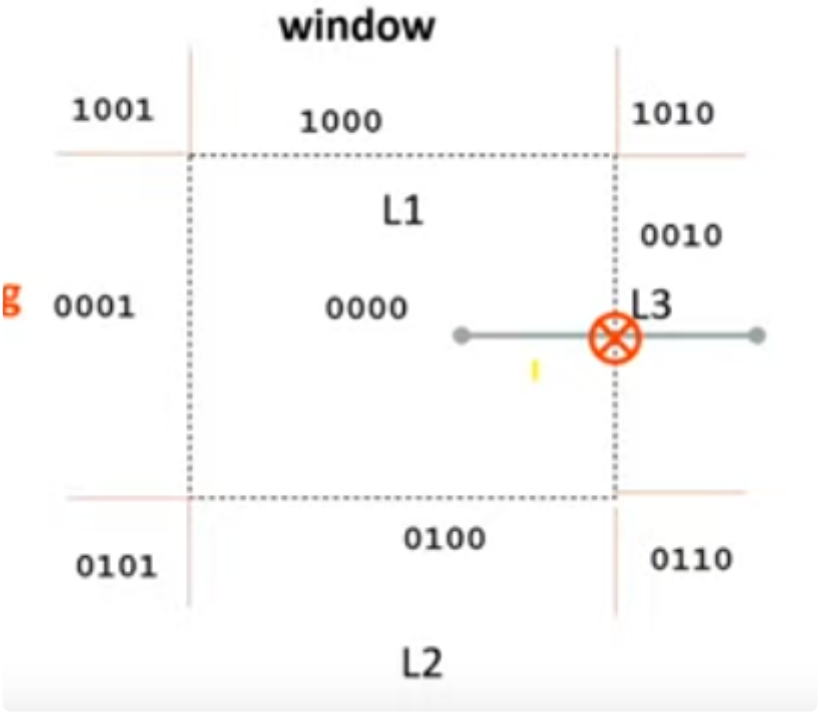
- Encode the endpoints of the line
  - If 2 endpoints have the code 0000 and their AND is 000 then line is completely inside. So Accept the line
  - If the 2 endpoints are non zero and their AND is non zero, the line is completely outside. So Reject the line
  - if 2 Endpoints are non zero and their AND is zero, the line is partially inside. **So clip the line**
- Clipping needs intersection points(s)
- If a point is outside any window boundary, find the intersection point on window boundary

Example

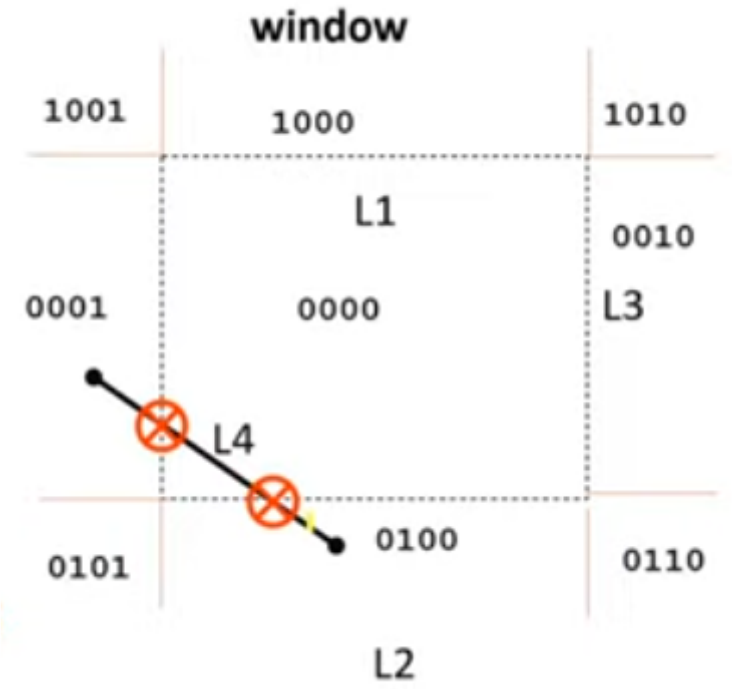


- Consider Line L1
  - The endpoints are
    - 0000
    - 0000
  - Doing AND Operation gives 0000
  - Completely inside
- Line 2
  - The endpoints are

    - 0100
    - 0100
  - Doing AND Operation gives 0100
  - Completely outside

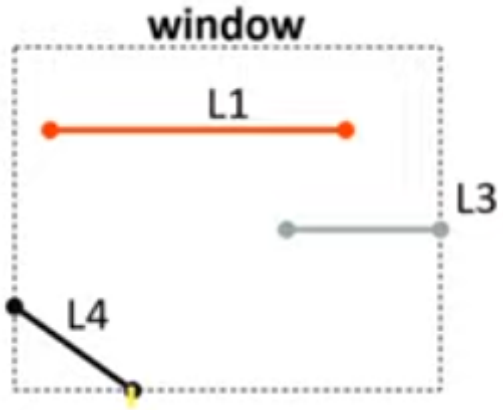


- Line 3
  - The endpoints are
    - 0000
    - 0010
  - Doing AND Operation gives
    - 0000
    - 0010 +
    - 0000
  - Partially inside, needs clipping
  - Find intersection point
  - New endpoints are
    - 0000
    - 0000 (Intersection point at window boundary)
  - 0000 + 0000 = 0000 (Completely inside)



- Line 4
  - Endpoints
    - 0001
    - 0100
  - 0001 + 0100 = 0000, Partially inside, needs clipping
  - Find intersection point
  - New endpoints
    - 0000
    - 0000
    - 0000 + 0000 = 0000 (Completely inside)

Final clipped lines

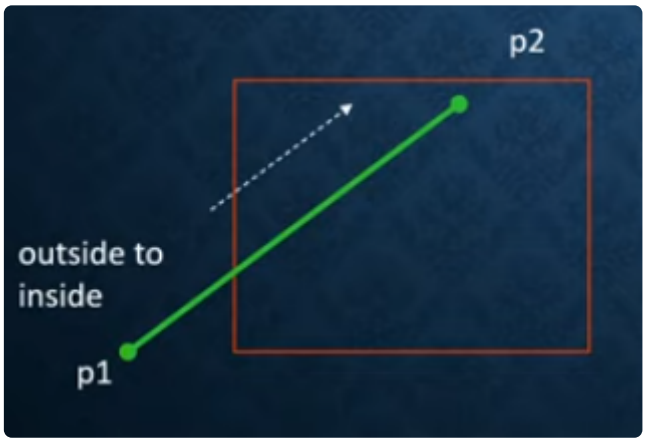


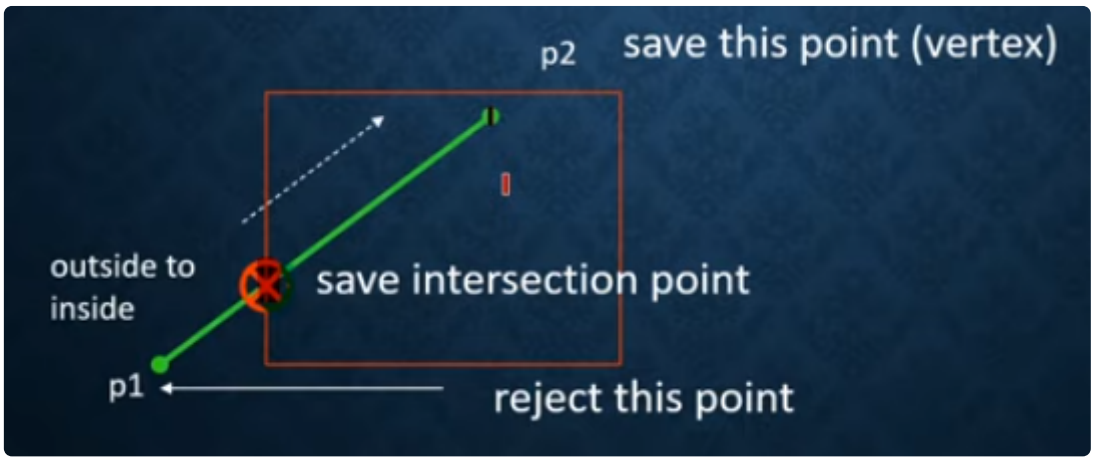
Sutherland Hodgeman Polygon clipping

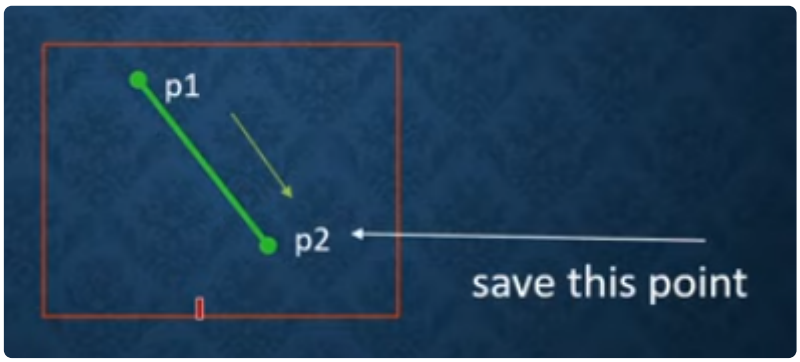
What is a polygon?

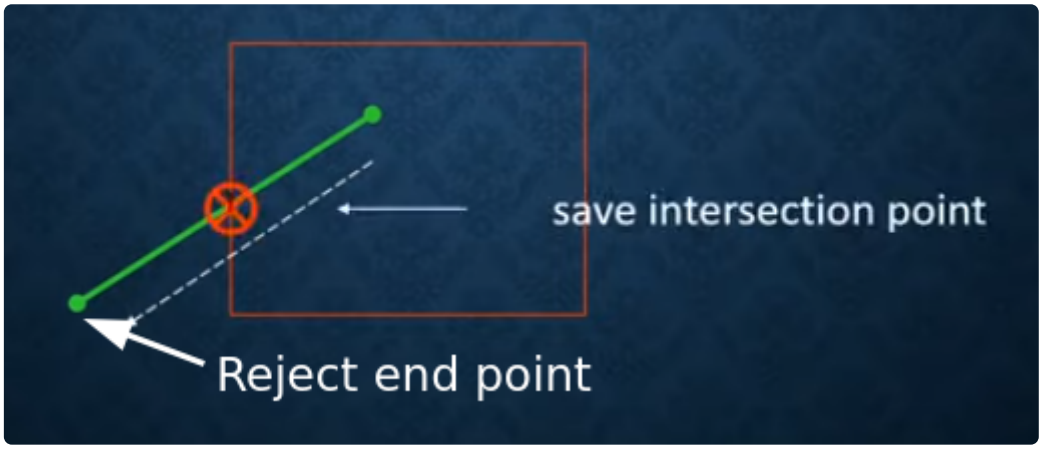
- Polygon is a set of lines joined together
- Polygon clipping has 4 Stages
  - Left clip
  - Right clip
  - Top clip
  - Bottom clip
- For each stage there are 4 cases to be checked for

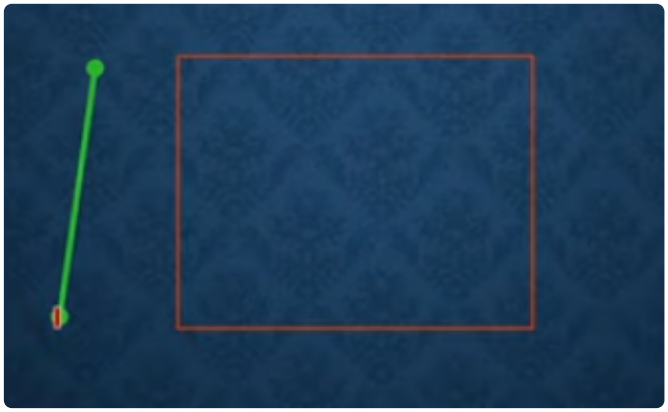
What are the 4 cases?

1. if an edge is moving from outside to inside
1. 

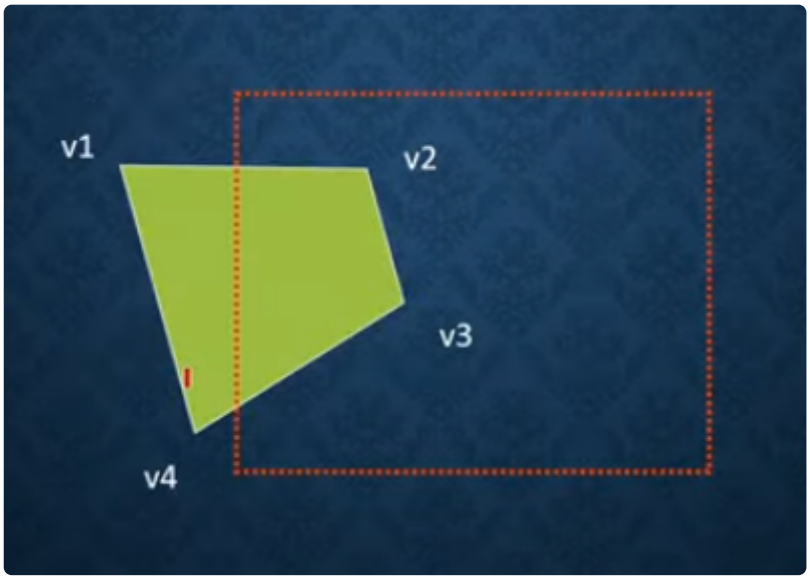
2. Reject the start point
    1. Here the start point is p1
3. Save the intersection point on window boundary and the other vertex
    1. Saving p2
4. 

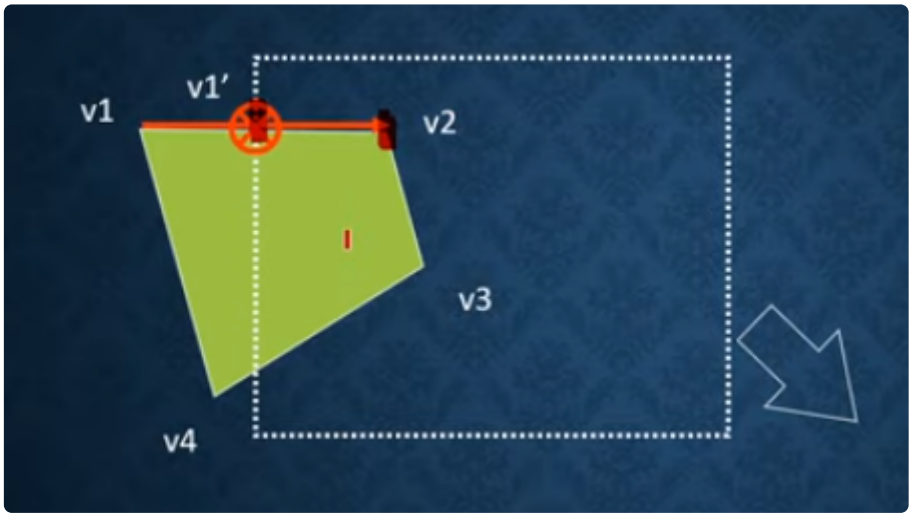
2. If an edge is moving from inside to inside
    1. Save the second vertex
    2. 

3. If moving from inside to outside
    1. Save intersection point
    2. Reject end point
    3. 

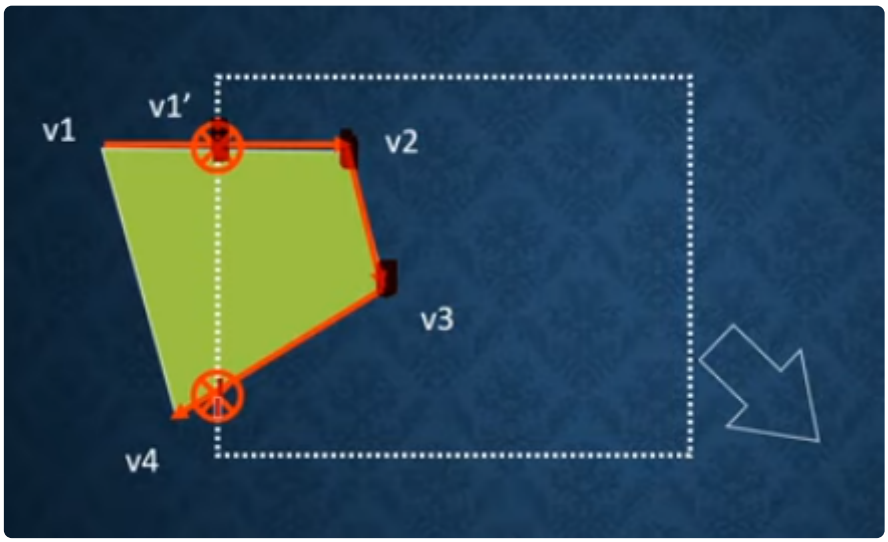
4. If moving from outside to outside
    1. Save none
    2. 

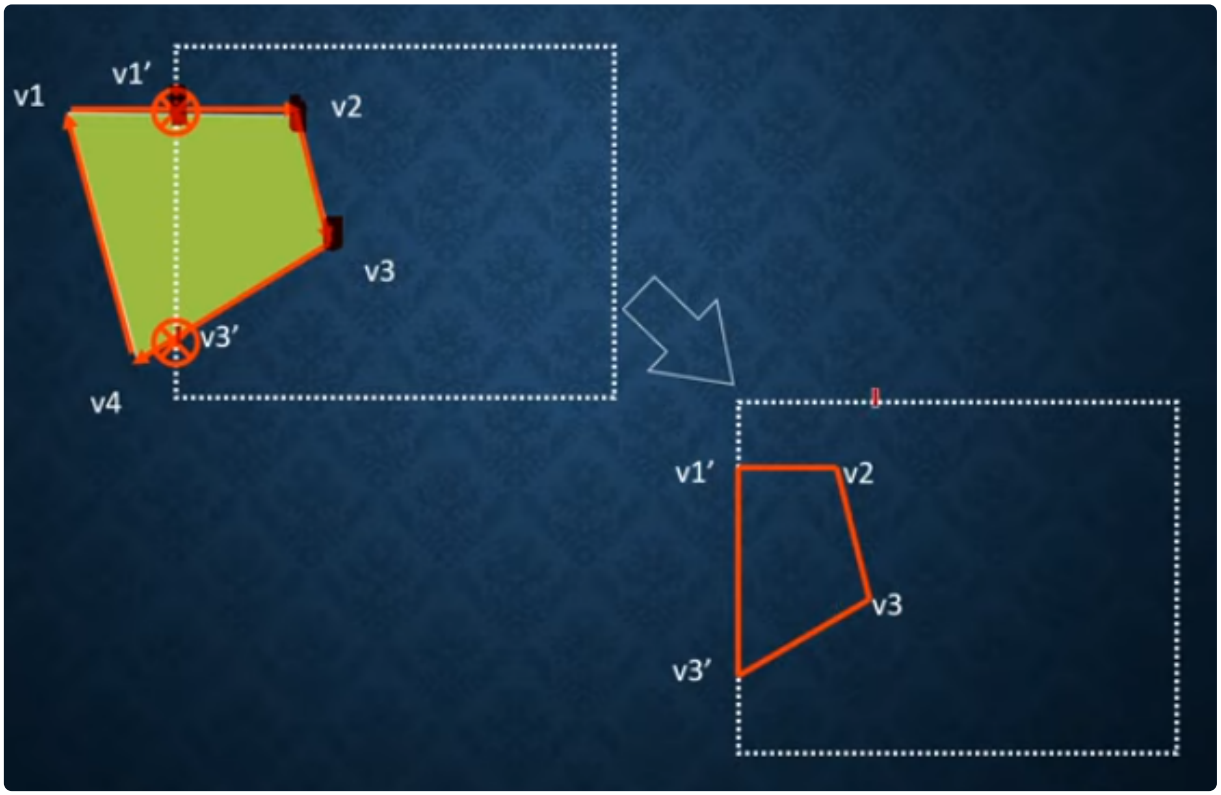
Example of polygon clipping



- Only the left side is outside, so we need to do left clipping
- v1 - v2
  - outside to inside
  - Reject start point (v1) and keep intersection
  - 

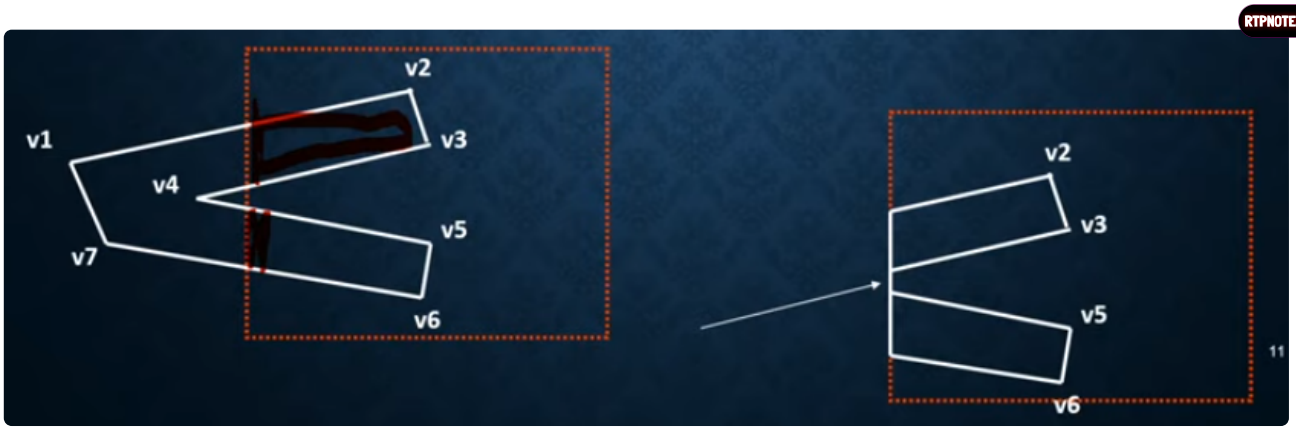
- v2 - v3
  - inside to inside
  - Keep second point (v3)
- v3 - v4
  - inside to outside
  - Save intersection
  - Reject end point

- 

- v4 - v1
  - v4 - v1
  - Outside to outside
  - Reject both points
- After clipping
  - 

Disadvantage

- This algorithm will not clip concave polygon properly
- A line is created through the window boundary

- 

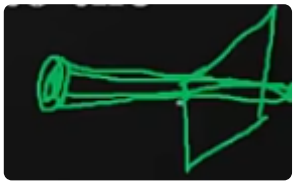
3. Parallel and Perspective Projections

- There are 2 basic Projection methods based on Property of projection line
  - Parallel Projection
  - Perspective Projection

Parallel Projection

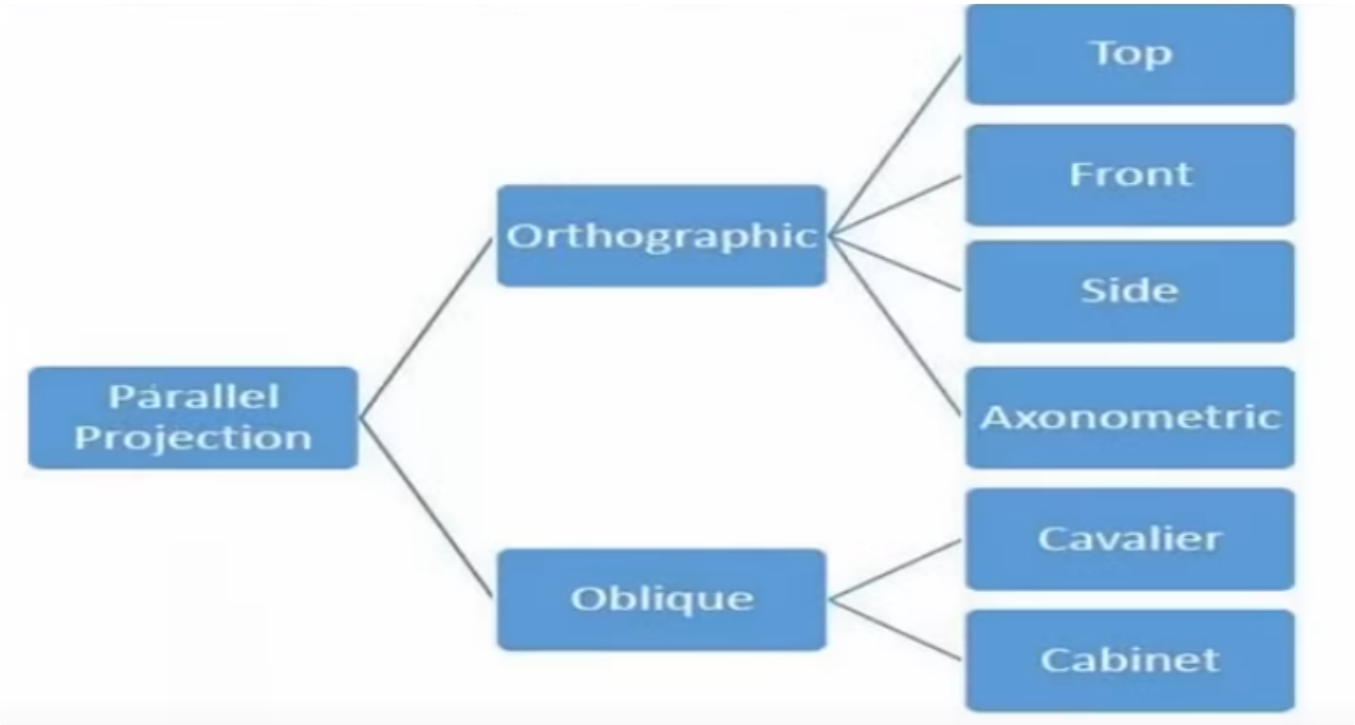
- There will be Projecting lines from Object to Frame for mapping different object points. If they are parallel to each other its paralllel projection
- 
- In the above image, object is at the left and plane is at right, and there are parallel lines
- They Preserves relative proportions of objects
- Accurate views of various sides of an object is obtained with parallel projection
- But does not give a realistic representation of the 3D object

Perspective Projection

- In this case, the lines are not parallel, somewhere in infinity they are considered to be merged
- 

- Produces realistic view but does not relative proportion

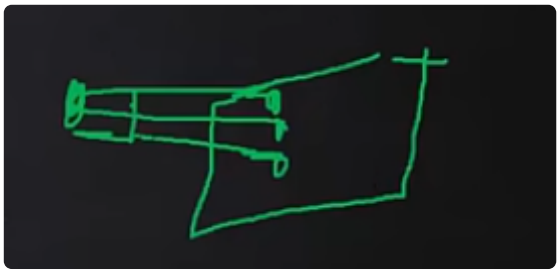
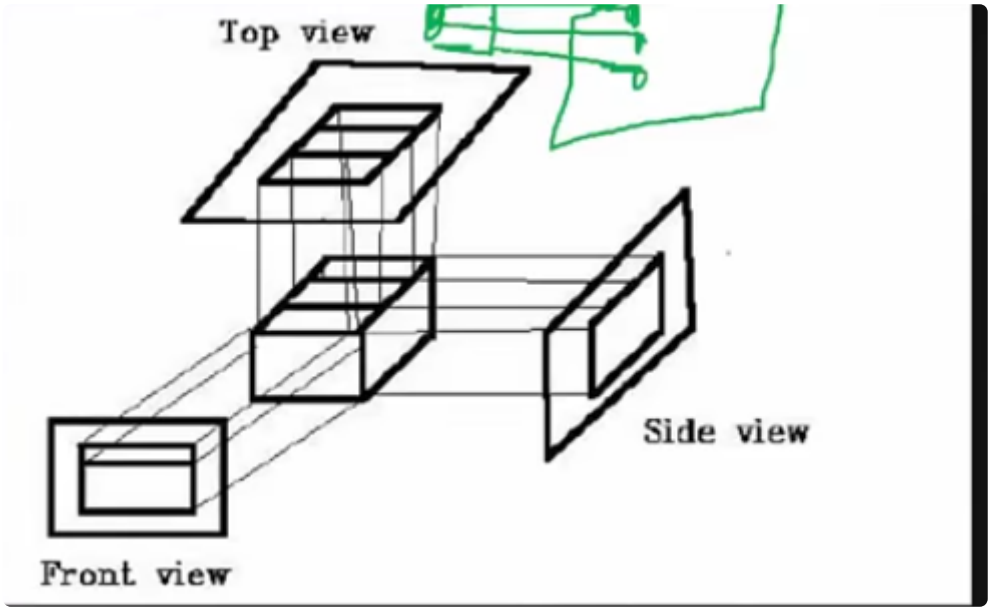
Types of Parallel Projection



- Its divided into 2 Types
  - Orthographic
  - Oblique
- Under Orthographics we have
  - Top
  - Front
  - Side
  - Axonometric
- Under Oblique we have
  - Cavalier
  - Cabinet

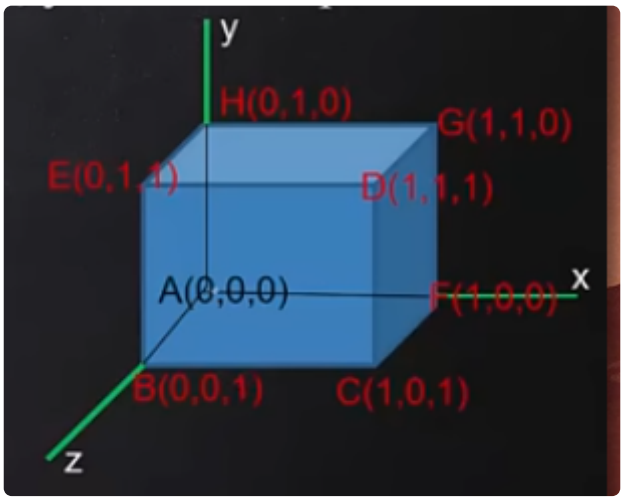
4. Orthographic Projections

Orthographic Projection

- Object projection lines are parallel to each other
- These parallel Projection lines and projection plane makes an angle of 90 degrees
  - 
- There are 3 types of orthographic projections
  - Front Projection
  - Top Projection
  - Side Projection
    - 

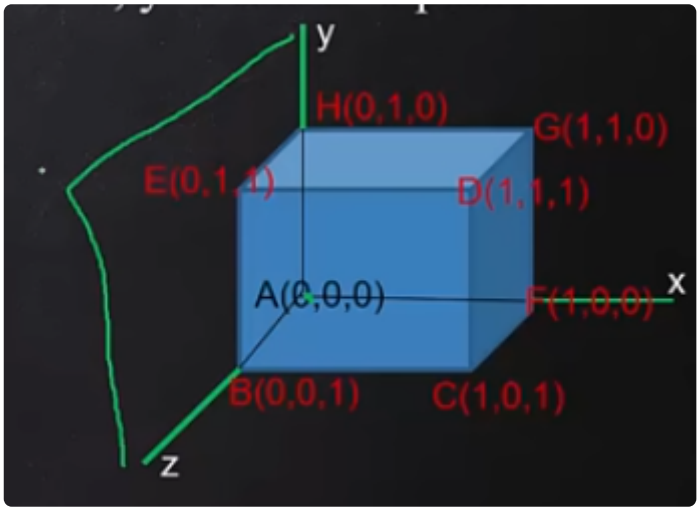
Example Problem

Find the orthographic projection of a unit cube onto x = 0, y = 0 and z = 0 plane



X = 0 Plane projection

- X = 0 means it is YZ plane Projection
- Projection where there is only Y and Z

- 

We need to take each point in the cube one by one

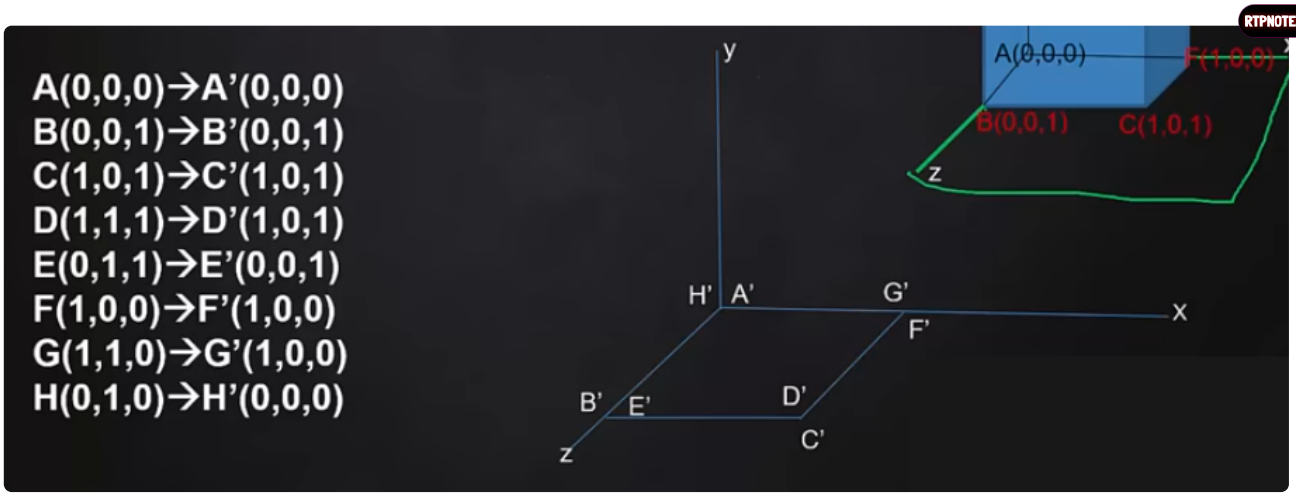
- A(0,0,0) -> A'(0,0,0)
  - A is already on the YZ plane, so its the same
- B(0,0,1) -> B'(0,0,1)
  - B is already on the YZ plane, so its the same
- C(1,0,1) -> C'(0,0,1)
  - Here Value of X = 1, But We need X = 0
  - Converted 1 to 0
  - (0,0,1) which is same as B
- D(1,1,1) -> D'(0,1,1)
  - X = 1, changing to X = 0
  - (0,1,1) which is same as E

- E(0,1,1) -> E'(0,1,1)
  - E is already on the YZ plane, so its the same
- F(1,0,0) -> F'(0,0,0)
  - X = 1, converting to 0
  - We get 0,0,0 which is same as A
- G(1,1,0) -> G'(0,1,0)
  - X = 1, changing to X = 0
  - We get (0,1,0), which is same as H
- H(0,1,0) -> H'(0,1,0)
  - H is already on the YZ plane, so its the same
- Putting the new coordinates together we get
  - 

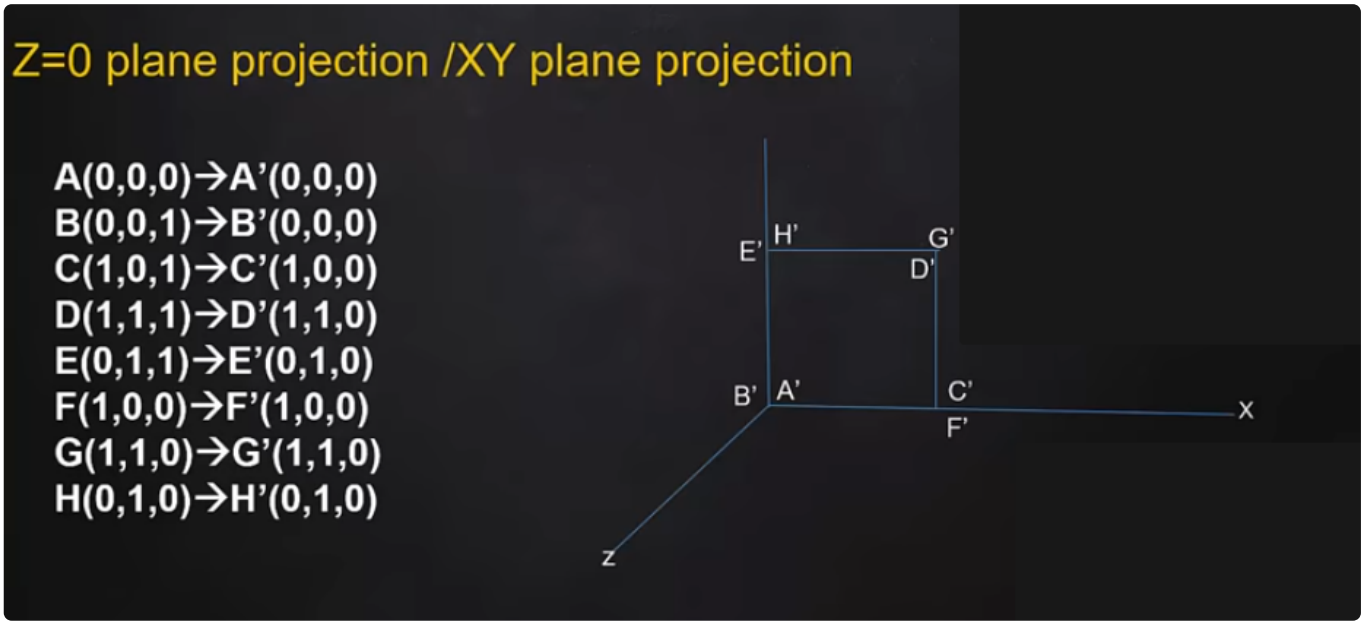
Y = 0 Projection

- Y = 0 means, the plane is in XZ

  -

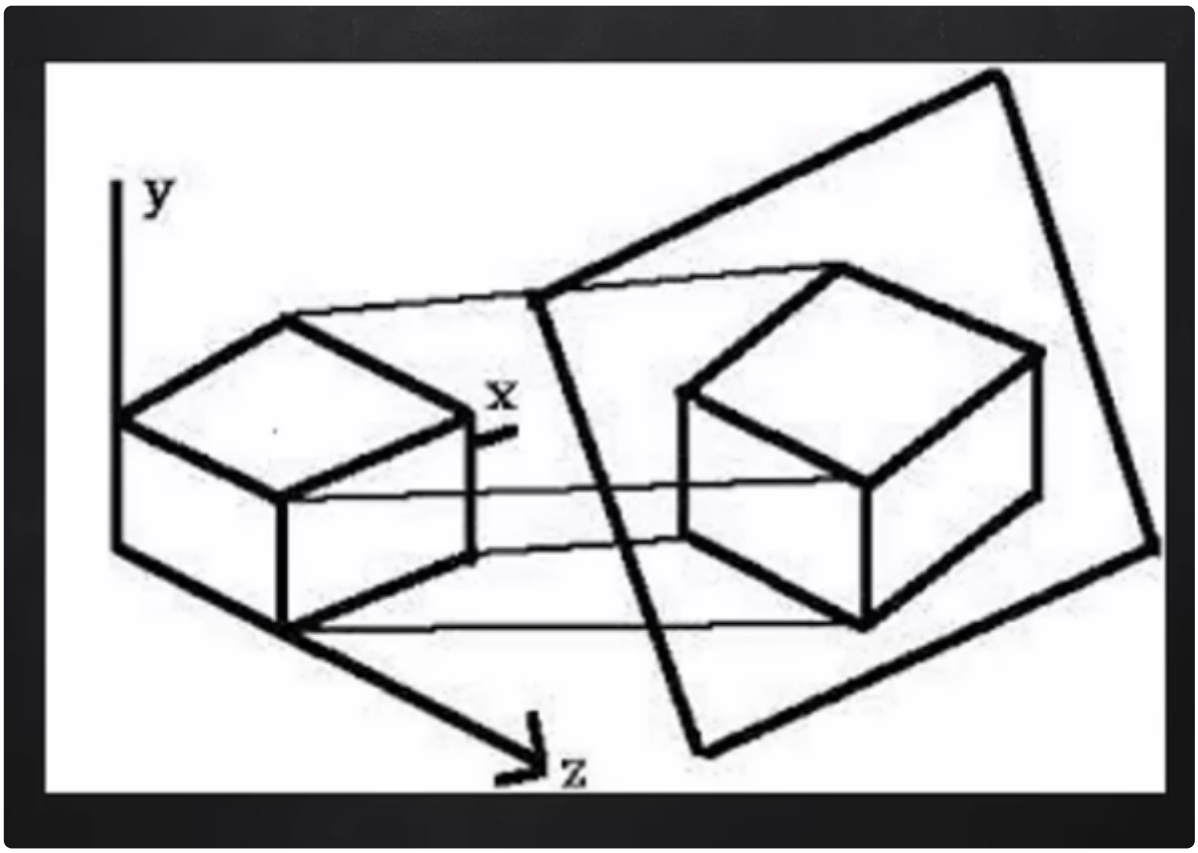


Z = 0 Projection



Isometric Projections

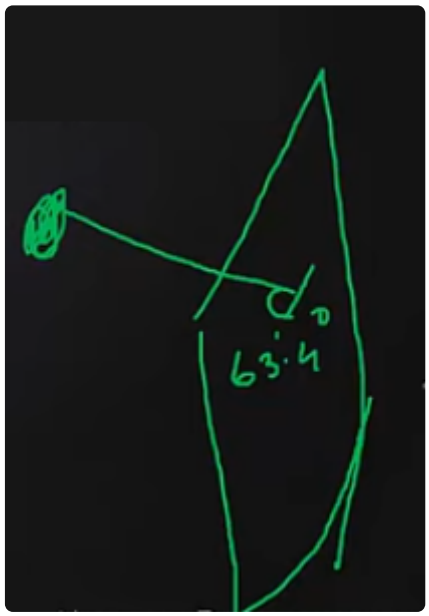
- Orthographic projections that show more than one side of an object are called **axonometric** orthographic projections
- The most common axonometic projection is an isometric projection where the projection plane intersects each coordinate axis at an equal distance

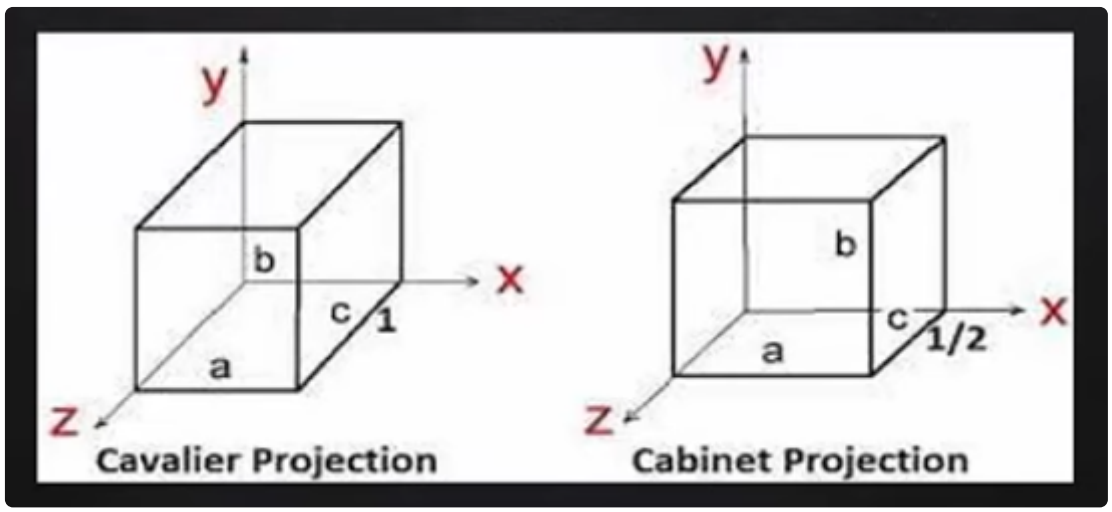
- 

5. Oblique Projections

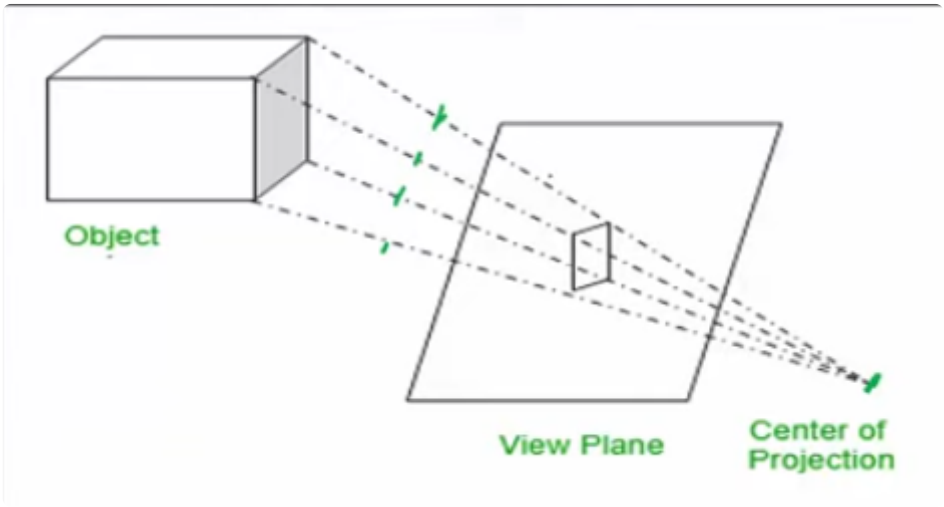
- Projecting the plane along parallel lines that are not perpendicular to projection plane
- There are two types of oblique projection
  - **Cavalier Projection**
    - Preserves length of line perpendicular to projection plan
    - Projection line an angle **45 degree** with projection plane

- 

- **Cabinet Projection**
  - Line perpendicular to viewing surface is projected at 1/2 original size
  - Projection line makes angle **63.4 degree** with projection plane
  - 

  - 

6. Perspective Projection, Types and Vanishing Points

- The distance and angles are not preserved and parallel lines do not remain parallel. Instead they all converge at a single point called **centre of projection or projection reference point**
- 

- The point at which the set of projected parallel lines appear to converge is called **vanishing point**
- Vanishing point for any set of lines that are parallel to one of the principal axes (X,Y,Z)of an object is referred to as the principal vanishing point

3 types of perspective projection

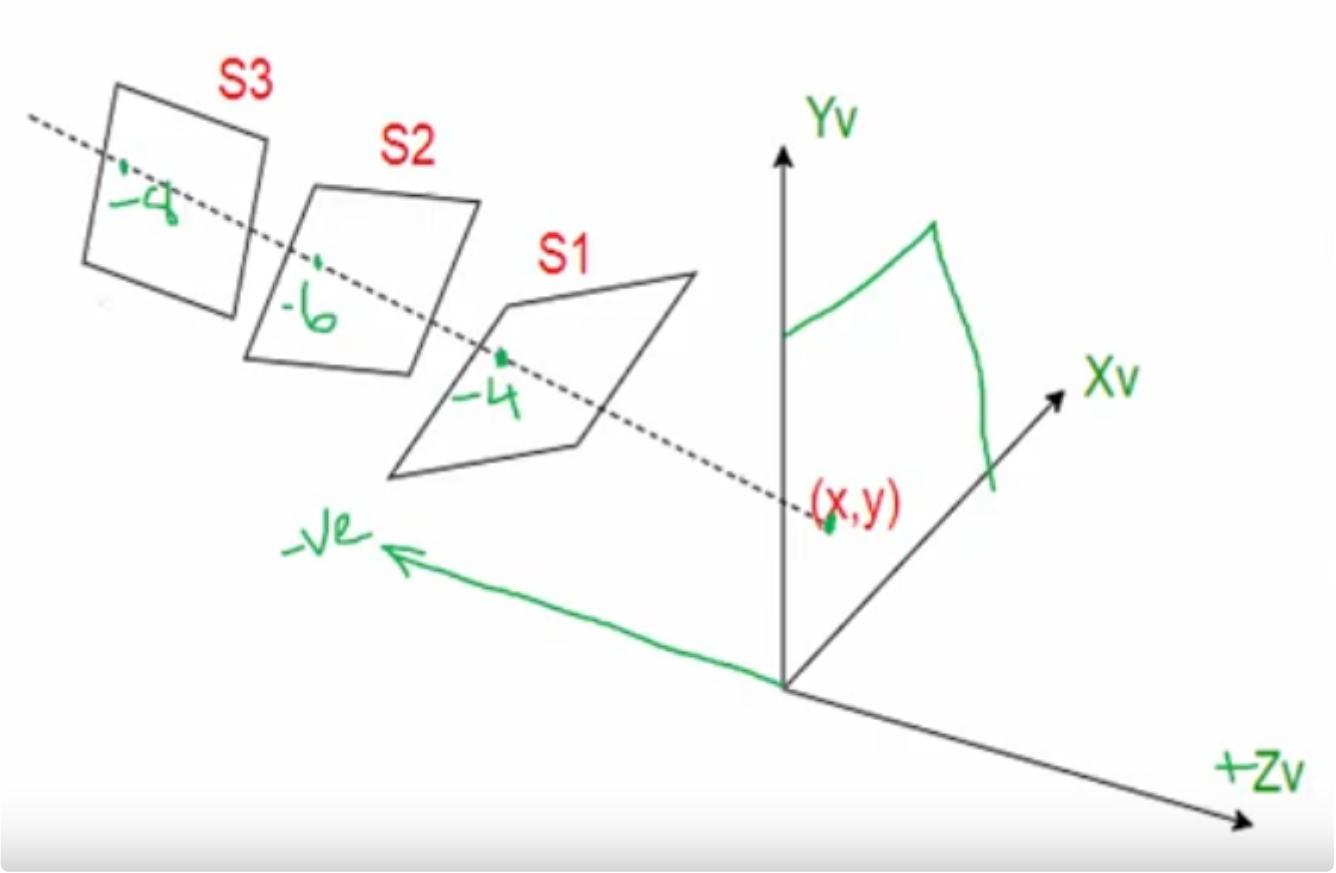
- One point perspective projection
  - Simple to draw
- Two point perspective projection
  - Better impression of depth
- Three point perspective projection
  - Most difficult to draw

7. Depth buffer algorithm

Depth Buffer Algorithm

- Its also called Z Buffer Algorithm

- Depth Buffer algorithm is the simplest image space algorithm
- For each pixel on the display screen, we keep record of the depth of an object within the pixel that lies closest to the observer
- In addition to the depth, we also record the intensity that should be displayed to show the object



- Consider the above diagram, here depth is measure in Z Axis
- S1 is the nearest to the observer

Algorithm

1. Initialize the depth buffer and refresh buffer so that for all buffer position (x,y)
    1. depth(x,y) = 0
    2. refresh (x,y) = Ibackground
2. For each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visiblity
    1. Calculate depth z for each (x,y) position on the polygon
    2. if z > depth(x,y) then set
        1. depth(x,y) = z, refresh(x,y) = Isurf(x,y)

- Ibackground = Value of background intensity
- Isurf(x,y) is the intensity value for the surface at pixel position (x,y)
- After all surfaces has been processed, the depth buffer contains depth values for the visible surfaces and refresh buffer contains intensities to display

8. Scanline algorithm

What is Scanline algorithm?

- It's a technique used in computer graphics to determine which surfaces of objects are visible in a scene.
- It builds on a method used to fill polygons (shapes with many sides) with color.

Features of scan algorithm

- Unlike the depth buffer method, which can only handle one surface (or layer) at a time, the scanline algorithm can handle multiple surfaces at once.
- It works by looking at one horizontal line (scan line) of the image at a time.'
- To figure out which surface is visible at each point on a scan line, the algorithm processes all the surfaces that intersect that scan line together before moving to the next one.
- It uses two main data structures: the edge table and the polygon table.

Edge Table

- This table lists all the edges (lines) in the scene.
- For each edge, it records the endpoints (start and end points), the slope (how steep the line is), and links to the corresponding surfaces in the polygon table.

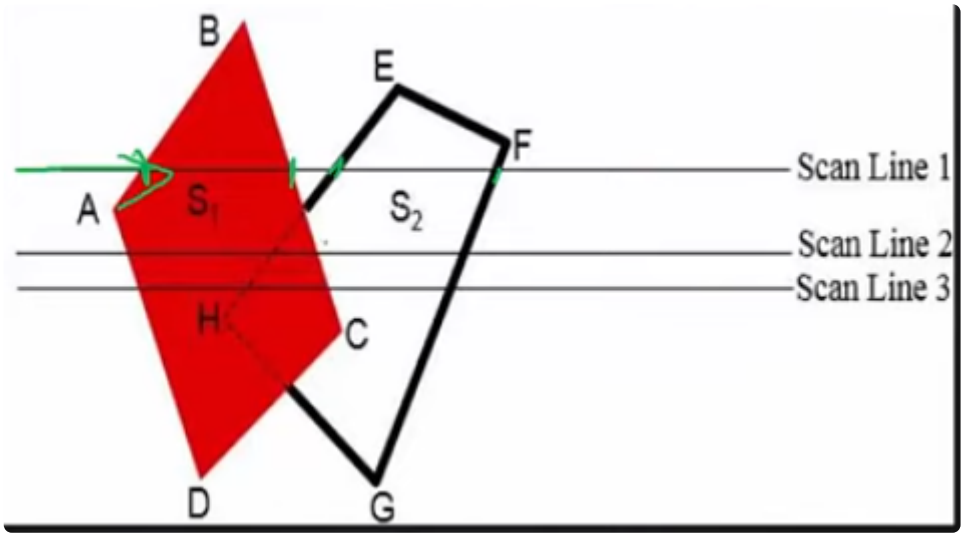
Polygon Table

- This table contains details about each surface (polygon), such as
  - The color of the polygon.
  - A flag that starts off as false (used during processing).

Scanline algorithm

1. Process Left to Right:

- For each horizontal line (scan line) across the image, look at all the surfaces (polygons) that intersect that line.
- Process these surfaces from the left side of the image to the right.
  2. Handle Overlapping Faces:
- When surfaces overlap, figure out which one is closest to the viewer.
- Only the closest surface is shown, and its color (intensity) is saved to be displayed on the screen.
  3. Maintain Active Edge List:
- For each scan line, keep a list of edges (lines) from the surfaces that intersect that line.
- Sort these edges by their x-coordinates (horizontal positions) from left to right.



- In the figure there are two polygon faces **ABCD and EFGH**
- In the figure you can also see 3 scan lines
- Basically When a scan line enters a polygon, we Turn the Flag on, and when it exits the polygon we Turn the flag off
- When inside the polygon we need to color it (If there is no overlapping)
  - We check this from left to right
  - Check the below example on how it works

Scanline 1

- Scan line 1 passes through the following edges
  - AB, BC,EH,FG
- **Active Edge List**: {AB, BC, EH, FG}

- **Process Left to Right**
  - **At the leftmost position**: Turn on the flag for surface S1 (ABCD).
  - **Between edges AB and BC**: The flag for S1 is ON, so we fill this area with the intensity (color) of surface S1 in the refresh buffer.
  - **At edge BC**: Turn off the flag for S1.
  - **Between BC and EH**: No flags are ON, so no intensity is stored.
  - **At edge EH**: Turn on the flag for surface S2 (EFGH).
  - **Between EH and FG**: The flag for S2 is ON, so we fill this area with the intensity of surface S2 in the refresh buffer.
  - **At edge FG**: Turn off the flag for S2.
  - **Depth Calculation**
    - Not needed for Scanline 1 as there are no overlapping regions between surfaces S1 and S2.

Scanline 2

- Scanline passes through the following edges
  - AD,HE,BC,FG
- **Process Left to Right**
  - **Between edges AD and BC**: The flag for S1 is ON, so we fill this area with the intensity of surface S1 in the refresh buffer.
  - **Between EH and BC**: Both flags for S1 and S2 are ON, indicating overlap. Calculate the depth from the viewpoint to each surface.
- **Depth Calculation**: Compare depths of S1 and S2 and store the intensity of the surface closer to the viewpoint in the refresh buffer.

Scanline 3

- **Active Edge List**: Same as Scanline 2 {AD, EH, BC, FG}
- Since the active edge list and conditions are the same as in Scanline 2, we use the previously calculated depth information.
- **Reuse Depth Information**: No need to recalculate depth; use the same intensity details as in Scanline 2

Advantages of scan line algorithm

- Any number of overlapping surfaces can be processed at a time

- Take advantage of coherence properties

Disadvantages

- It does not work for surfaces that cut through or cyclically overlap each other