Computer Graphics and Animation

Lecturer: Bijay Mishra (बिजय मिश्र)

 **Contact No: 9841695609**

 **Email ID: biizay@gmail.com**

 **@jijibisha**

Putting your Mobile away and paying attention to those talking to you? There's **App** for that, it's called **RESPECT**!

Computer Graphics and Animation

Unit 3 : CLIPPING

BIJAY MISHRA

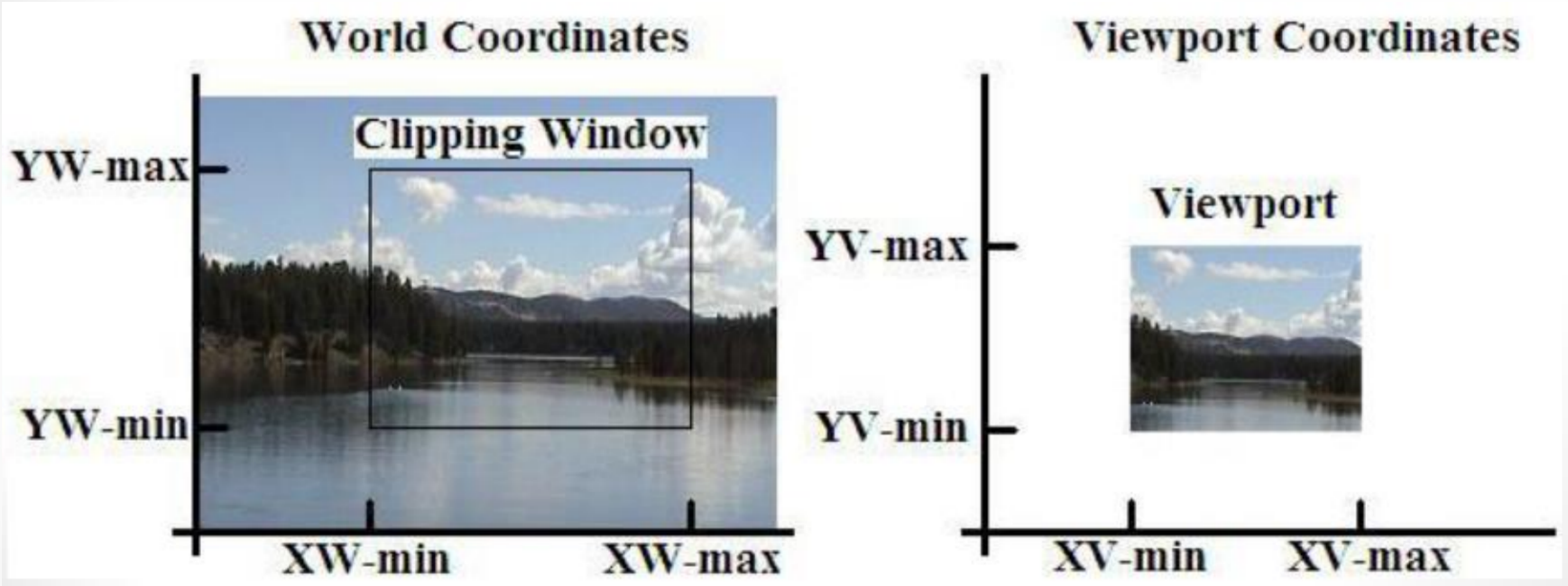
2D Viewing

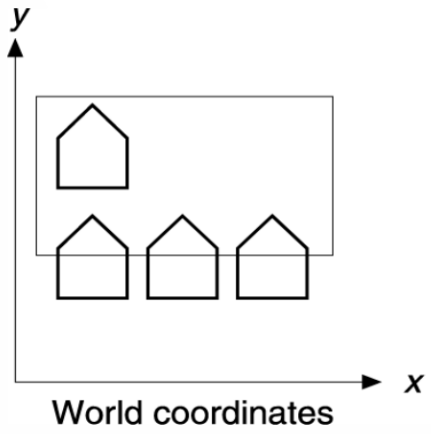
2-D VIEWING

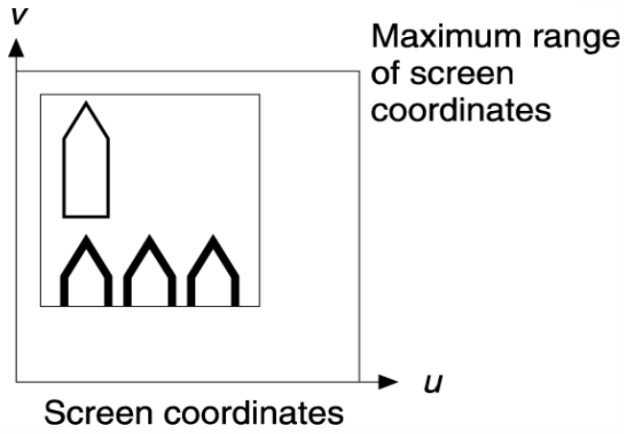
- **Window**
  - A world-coordinate area selected for display is called a window or clipping window.
  - Window is the section of the 2D scene that is selected for viewing.
  - The window defines *what is to be viewed*.

- **View-port**
  - An area on a display device to which a window is mapped is called a viewport.
  - The viewport defines *where it is to be displayed*.

2-D VIEWING


World Coordinates
Viewport Coordinates
Clipping Window
YW-max
YW-min
XW-min
XW-max
Viewport
YV-max
YV-min
XV-min
XV-max


y
x
World coordinates


v
Maximum range
of screen
coordinates
u
Screen coordinates

2-D VIEWING

World Co-ordinates:

- The space in which object are described is called *world co-ordinates*.
- A user defined application specific co-ordinate system having its own units of measure, axis, origin etc.
- Examples: cm, inches, km etc.

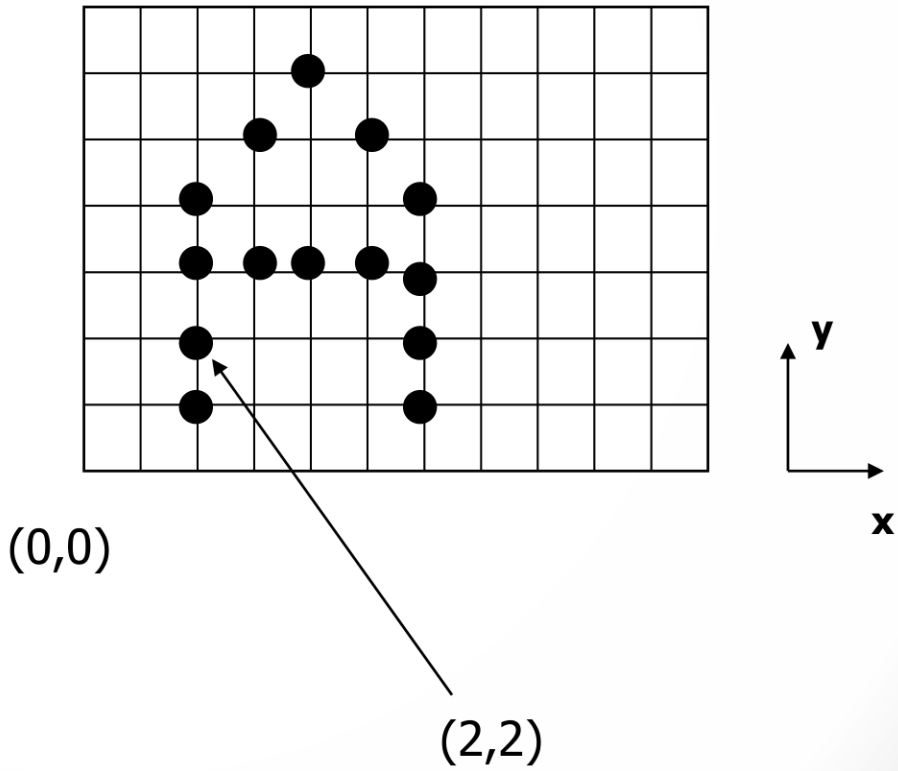
Screen Co-ordinates:

- We use the basic co-ordinate system of the screen window, co-ordinate (in pixels) extending from 0 to screen width-1 in x and from 0 to screen height -1 in y.
- The co-ordinate system used to address the screen.

2-D VIEWING

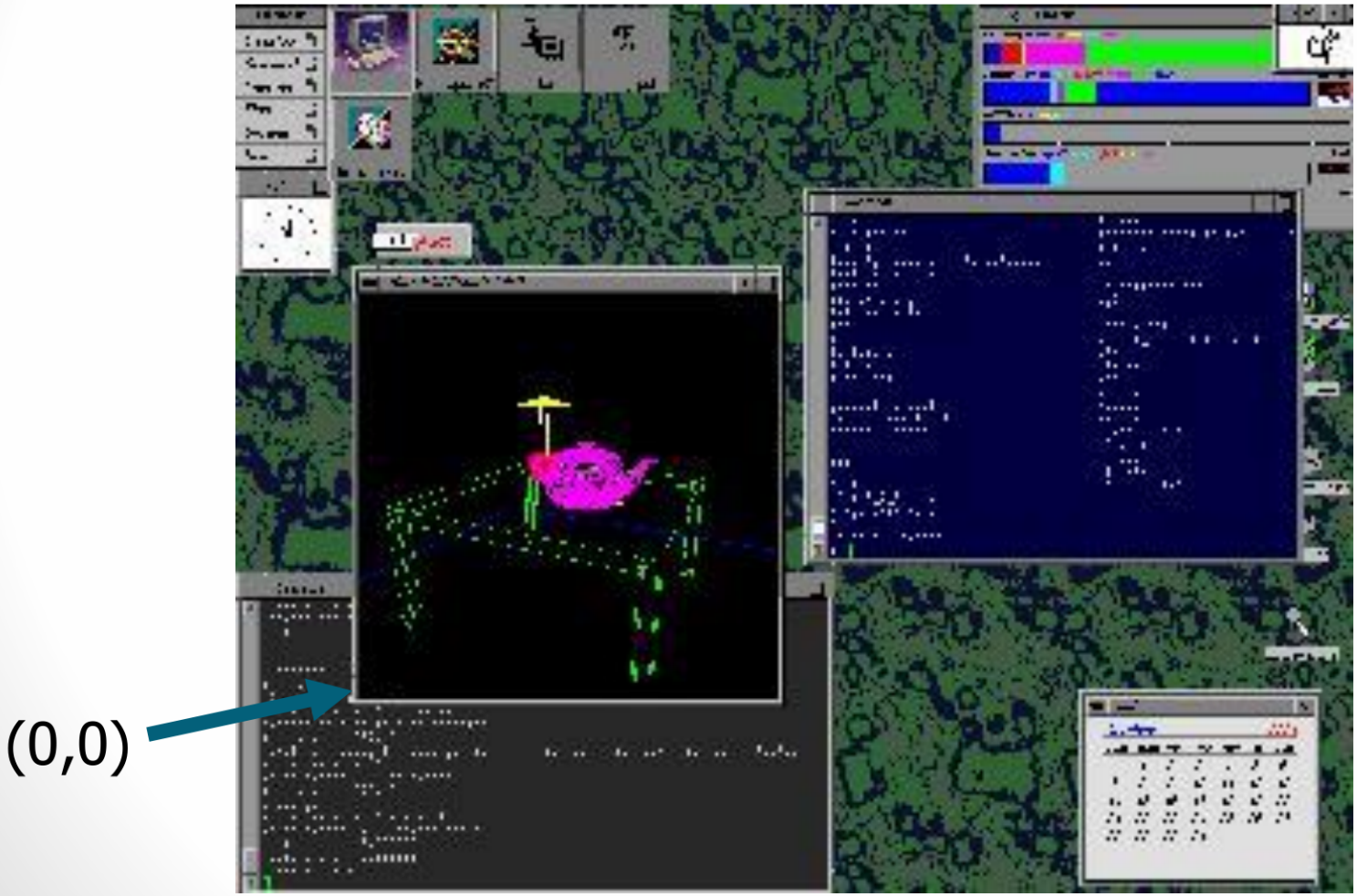
Screen Coordinate System

- Screen: 2D coordinate system (WxH)
- 2D Regular Cartesian Grid
- Origin (0,0) at lower left corner (OpenGL convention)
- Horizontal axis – x
- Vertical axis – y
- Pixels: grid intersections



2-D VIEWING

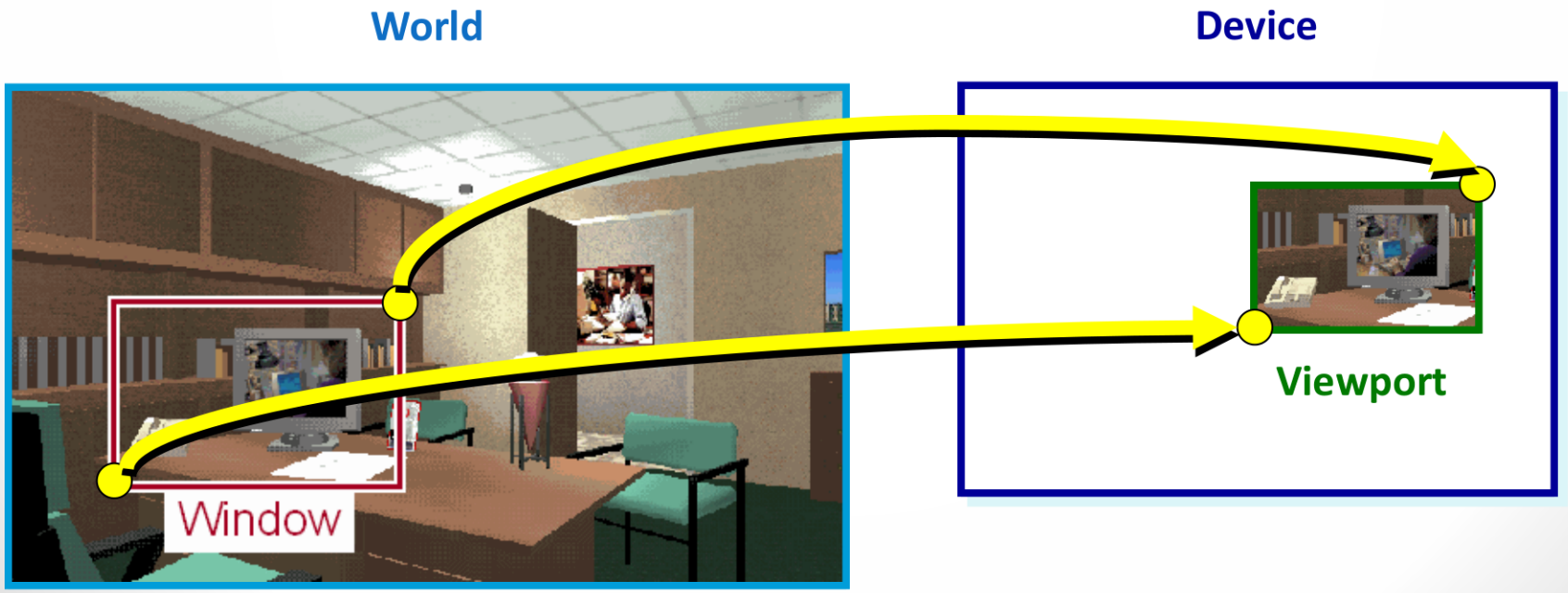
Screen Coordinate System



2-D VIEWING

Window and Viewport

- Transform 2D Geometric Primitives from World Coordinate System (Projection Coordinates) to Screen Coordinate System (Device Coordinates)



2-D VIEWING

Window and Viewport

- Window and viewport are often rectangular in standard positions, because it simplifies the transformation process and clipping process.
- Other shapes such as polygons, circles take longer time to process.
- By changing the position of the viewport, we can view objects at different positions on the display area of an output device.
- Also by varying the size of viewports, we can change size of displayed objects.
- Zooming effects can be obtained by successively mapping different-sized windows on a fixed-sized viewport.

2-D VIEWING

Viewing Transformation

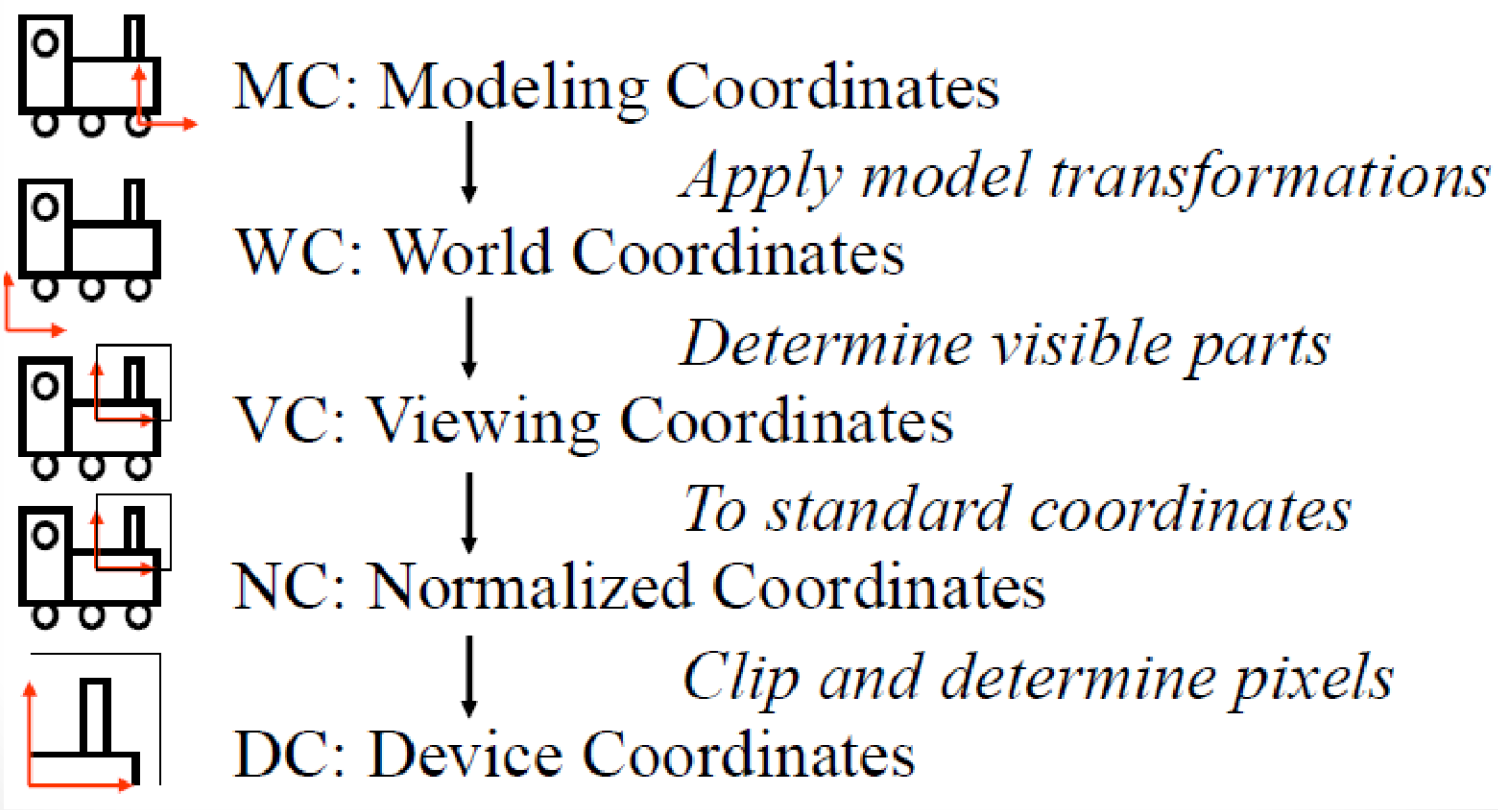
- The mapping of a part of a world-coordinate scene to device coordinates is referred to as a *viewing transformation*.
- 2D viewing transformation is simply referred to as the *window-to-viewport transformation* or the *windowing transformation*.

Transformation Pipeline

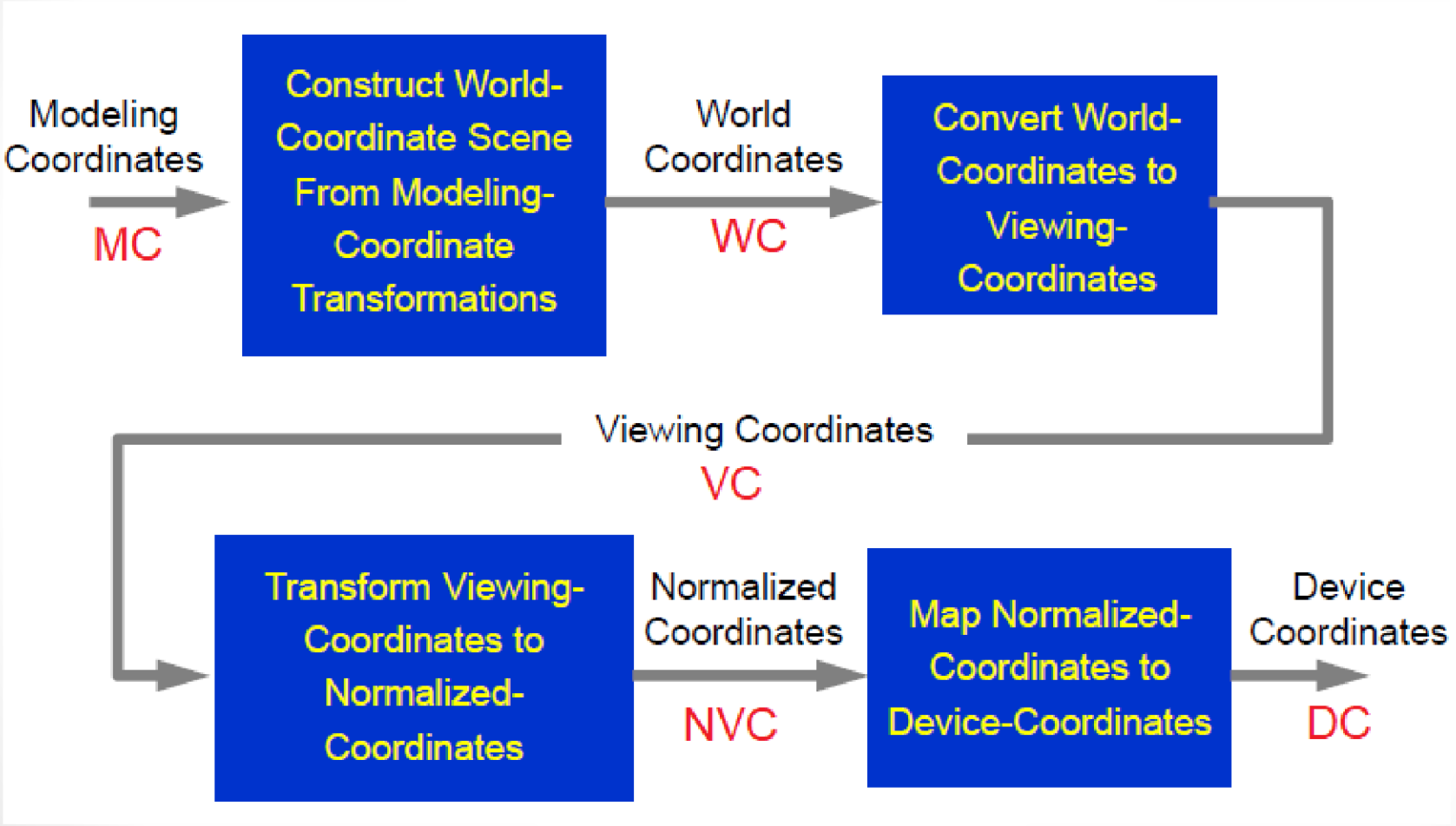
- Takes the object coordinates through several intermediate coordinates systems before finishing with device coordinates

2-D Viewing Transformation Pipeline

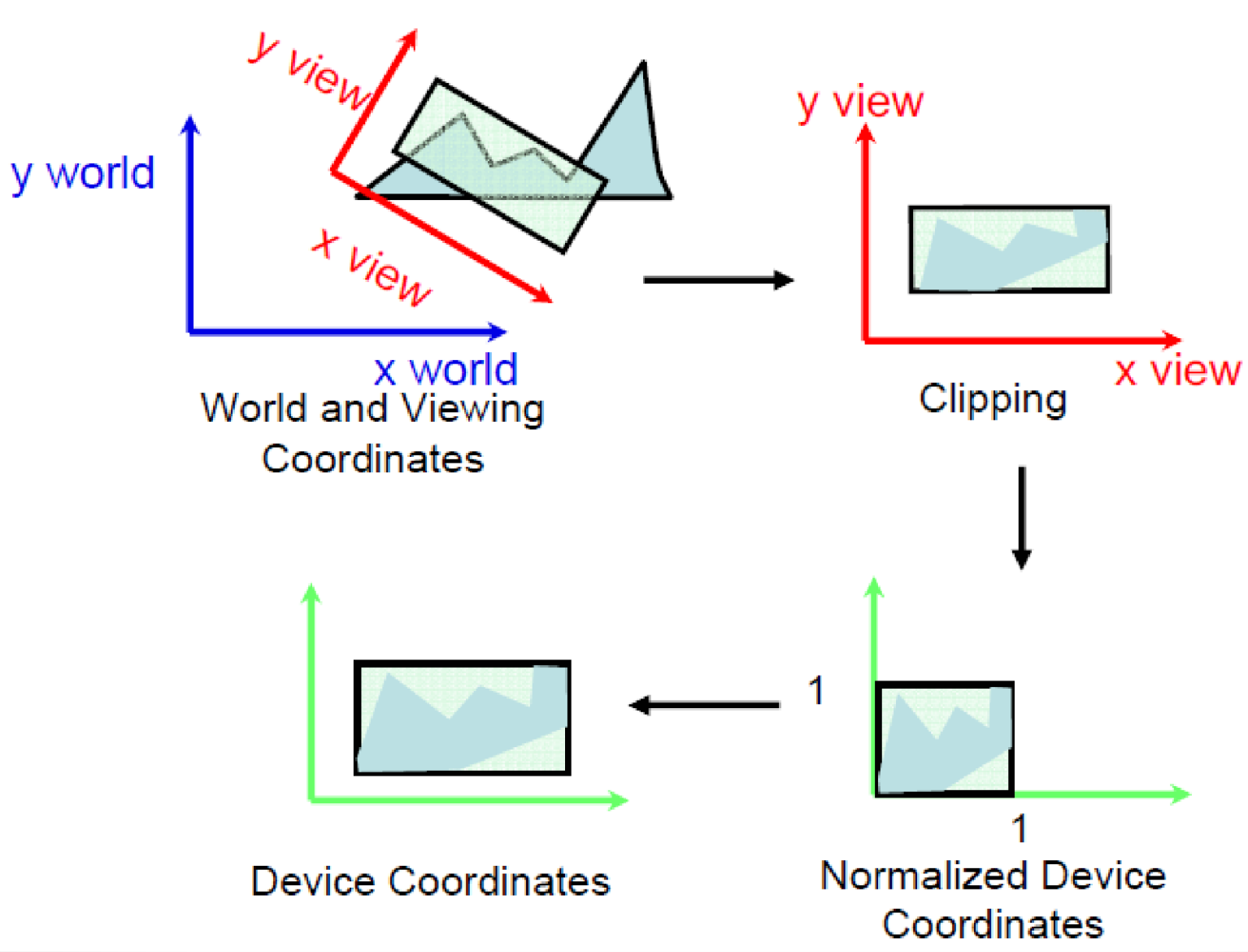
- The transformation that maps the window into the viewport is applied to all of the output primitives (lines, rectangles, circles) in world coordinates.
- This viewing transformation involves several steps.



2-D Viewing Transformation Pipeline



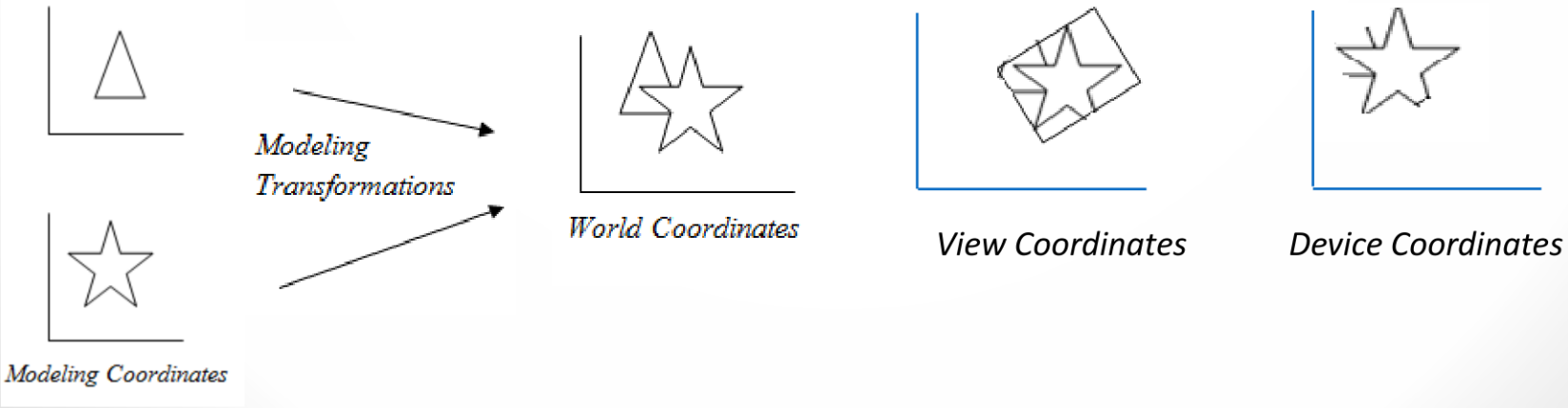
2-D Viewing Transformation Pipeline



2-D Viewing Transformation Pipeline

1. Modeling Coordinates

- Modeling coordinates are used to construct shape of individual parts (*objects or structures*) of a 2D scene. For example, generating a circle at the origin with a "radius" of 2 units.
- Here, origin (0, 0), and radius 2 units are modeling coordinates.
- Modeling coordinates define object shape.
- Can be floating-point, integers and can represent units like km, m, miles, feet etc.



2-D Viewing Transformation Pipeline

2. World Coordinates

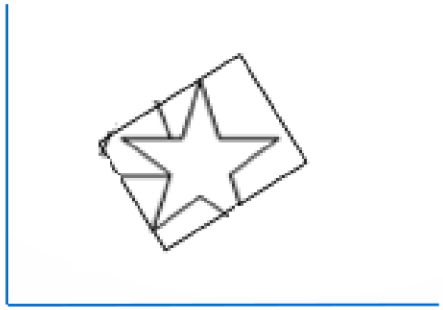
- World coordinates are used to organize the individual parts into a scene.
- World coordinates units define overall scene to be modeled.
- World coordinates represent relative positions of objects.
- Can be floating-point, integers and can represent units like km, m, miles etc.

World Coordinates

2-D Viewing Transformation Pipeline

3. Viewing Coordinates

- Viewing coordinates are used to define particular view of the user. Viewer‟s position and view angle i.e. rotated/translated.
- Viewing coordinates specify the portion of the output device that is to be used to present the view.
- *Normalized viewing coordinates* are viewing coordinates between 0 and 1 in each direction. They are used to make the viewing process independent of the output device (paper, mobile).



View Coordinates

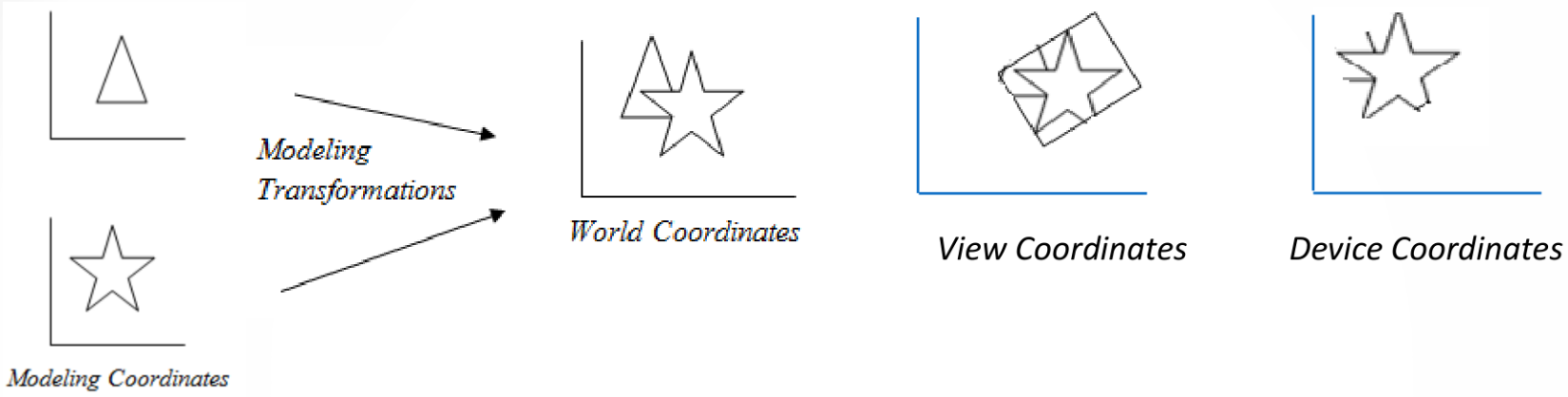
2-D Viewing Transformation Pipeline

4. Device Coordinates or Screen Coordinates

- The display coordinate system is called device coordinate system. Device coordinates are specific to output device.
- Device coordinates are integers within the range (0, 0) to (xmax, ymax) for a particular output device.

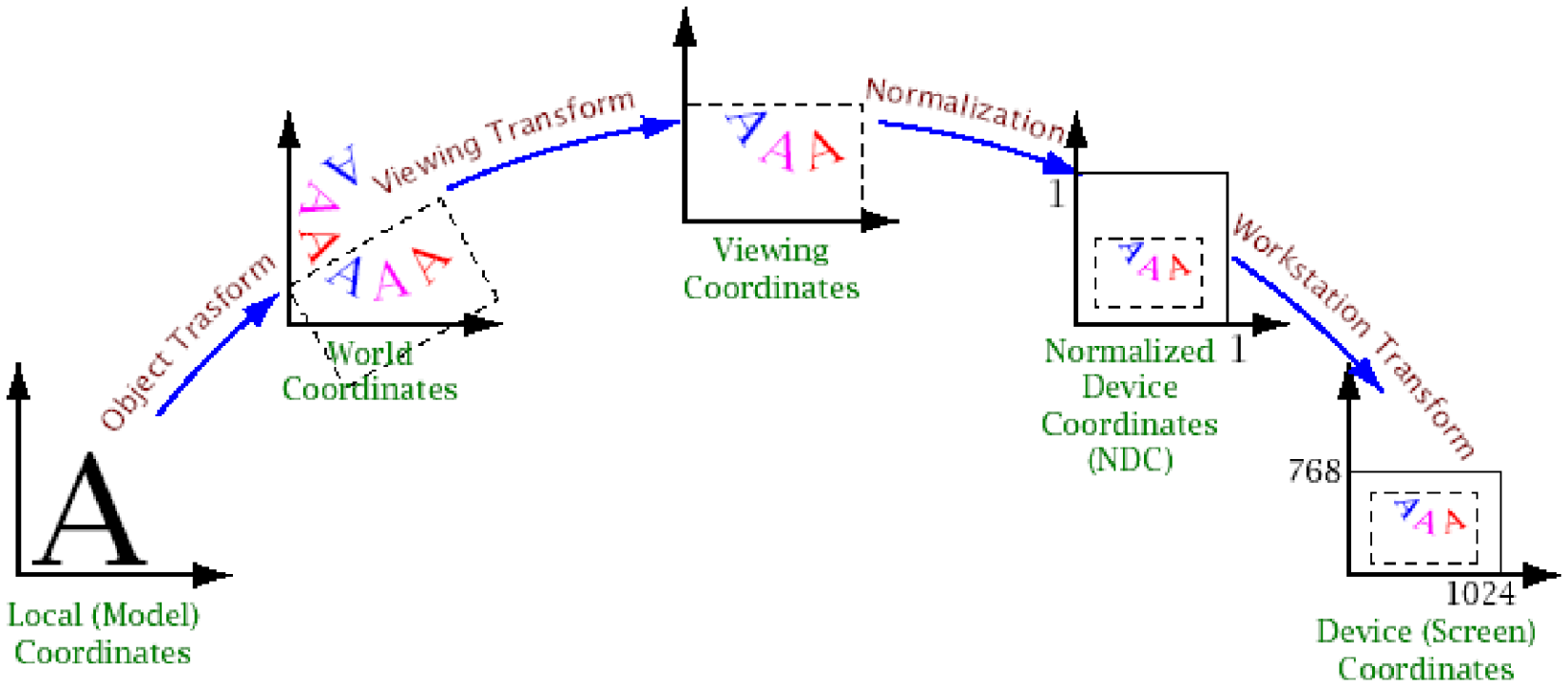
Device Coordinates

2-D Viewing Transformation Pipeline



2-D Viewing Transformation Pipeline

Local C. ⟶ World C. ⟶ Viewing C. ⟶ N.D.C. ⟶ Device C.

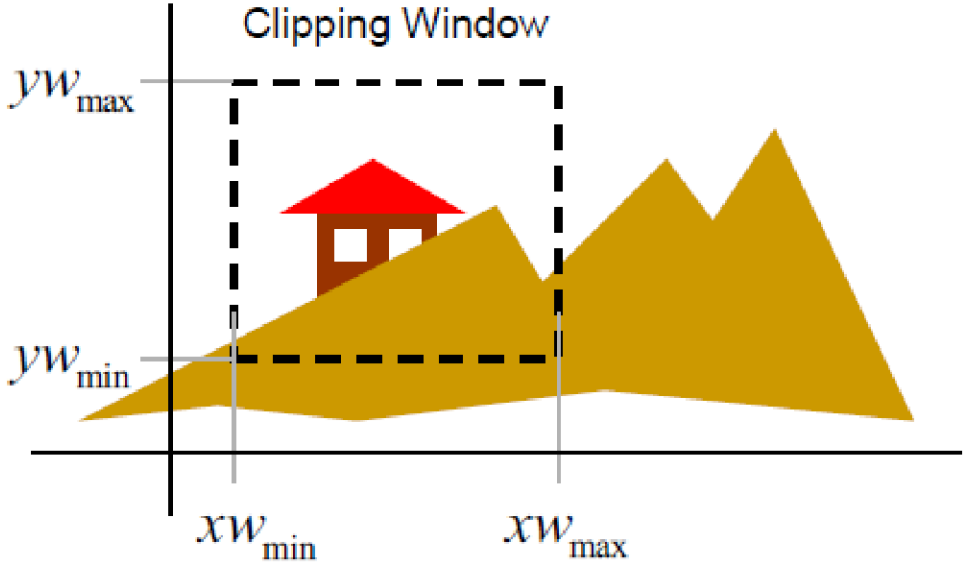


Window to Viewport Transformation

Window to Viewport Transformation

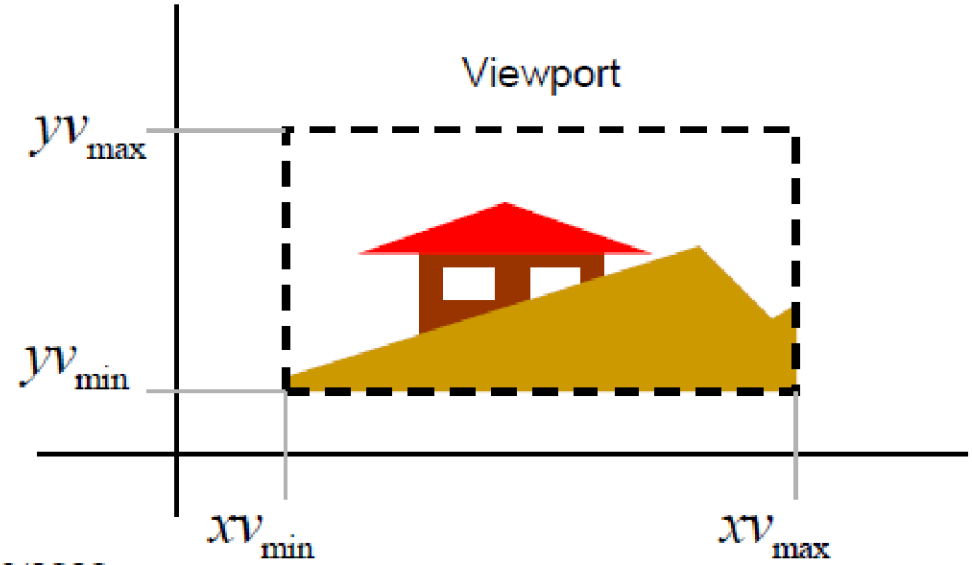
- The mapping of a part of a world co-ordinate scene to device co-ordinates is referred to as viewing transformation i.e. it transforms the pictures into screen co-ordinate.
- The world co-ordinate system is chosen that suit the application program and the screen co-ordinate system is inherent in the design of the display.
- After applying the transformation the transformed picture then clipped and visible portion is added to the display file.

Window to Viewport Transformation



World Coordinates

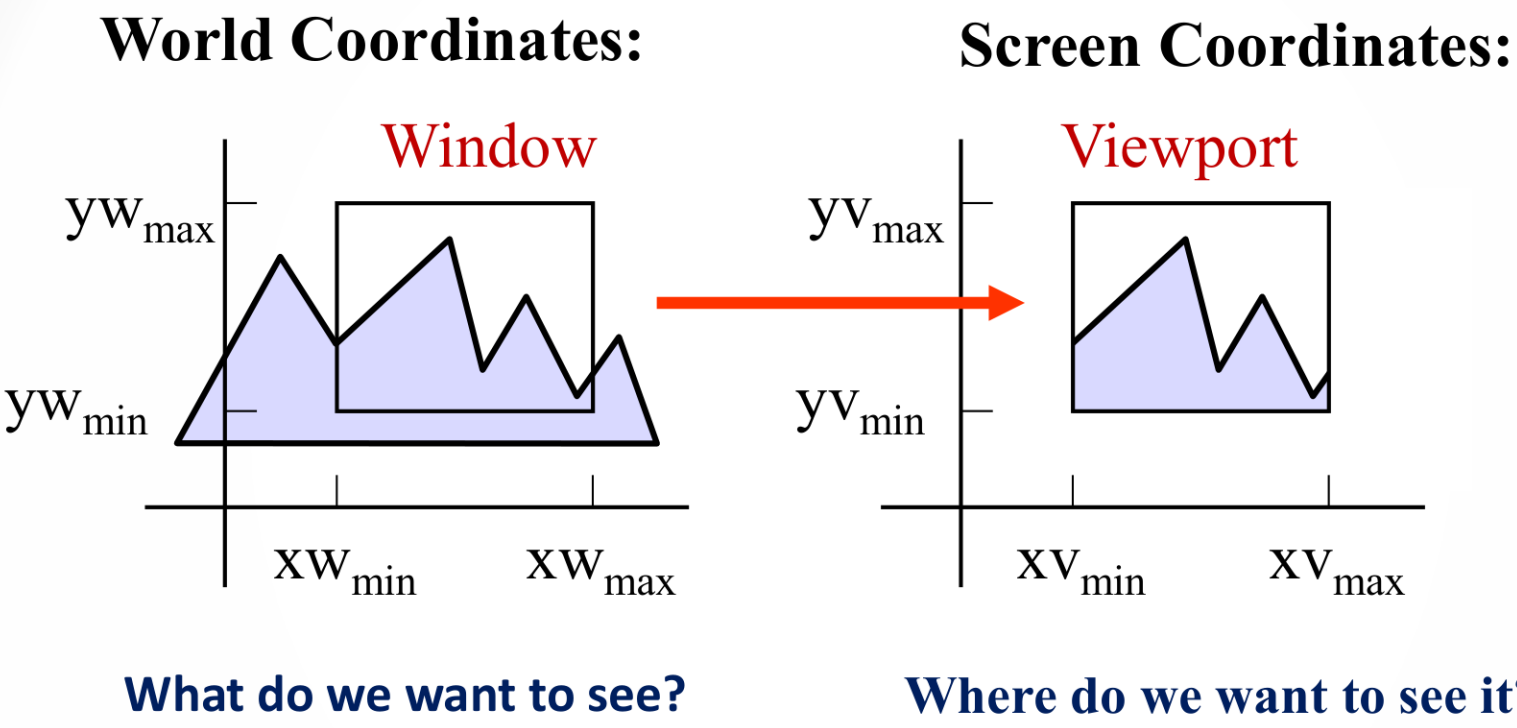
The clipping window is mapped into a viewport.



Viewport Coordinates

Viewing world has its own coordinates, which may be a non-uniform scaling of world coordinates.

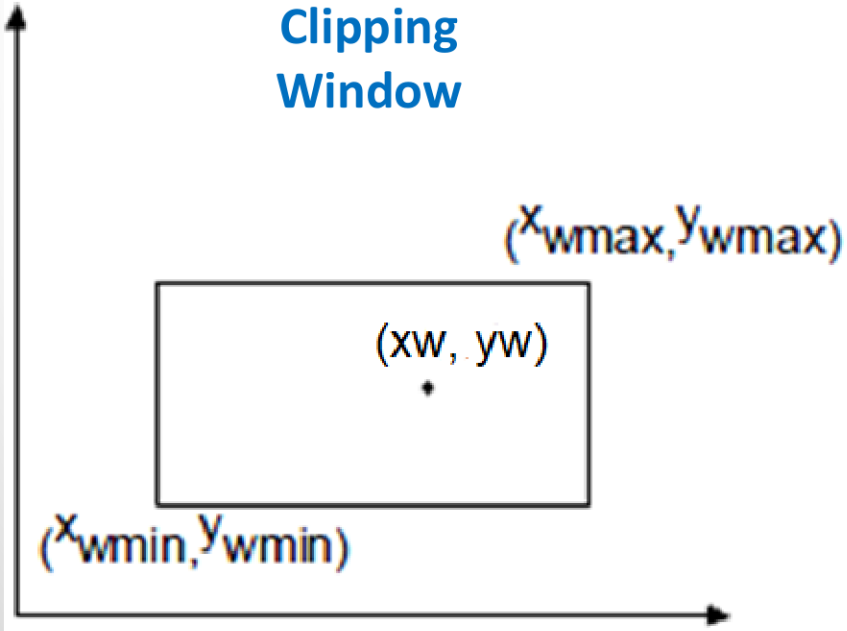
Window to Viewport Transformation

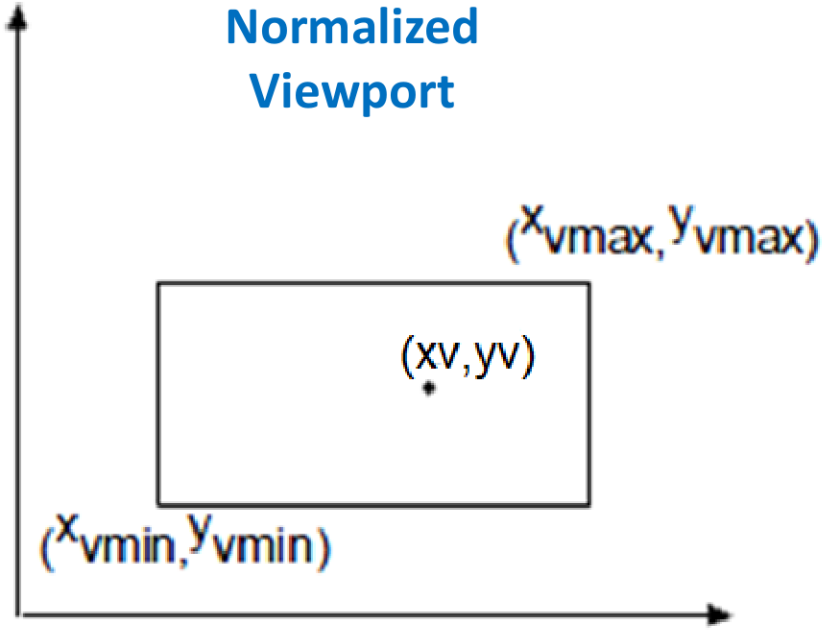


- We maintain the same relative placement in the viewport as in the window.
- If a coordinate position is at the center of the world window, for instance, it will be displayed at the center of the viewport.

Window to Viewport Transformation

A **window** is specified by four world coordinates: Xw_{min}, Xw_{max}, Yw_{min} and Yw_{max} . Similarly, a **viewport** is described by four normalized device coordinates: Xv_{min}, Xv_{max}, Yv_{min} and Yv_{max}.





Window to Viewport Transformation

- We maintain the same relative placement in the viewport as in the window.
- Let (xw, yw) be the world coordinate point that is mapped onto the viewport point (xv, yv), then we must have:

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

Window to Viewport Transformation

Solving these expressions for the viewport position (xv, yv), we have:

$$xv = xv_{min} + (xw - xw_{min}) \cdot s_x$$

$$yv = yv_{min} + (yw - yw_{min}) \cdot s_y$$

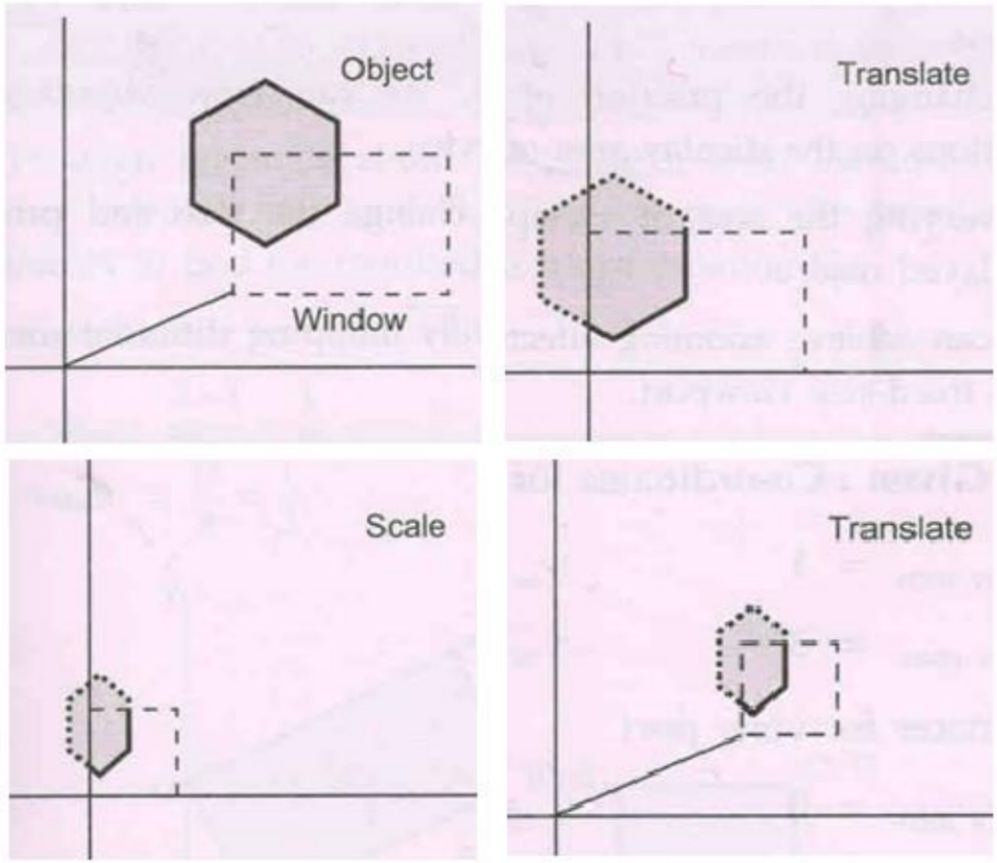
Where the scaling factors are:

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}, \qquad s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

Window to Viewport Transformation

Workstation Transformation

- Transformation of object description from normalized coordinates to device coordinates.
- It is accomplished by selecting a window area in normalized space and viewport area in coordinates of display device.



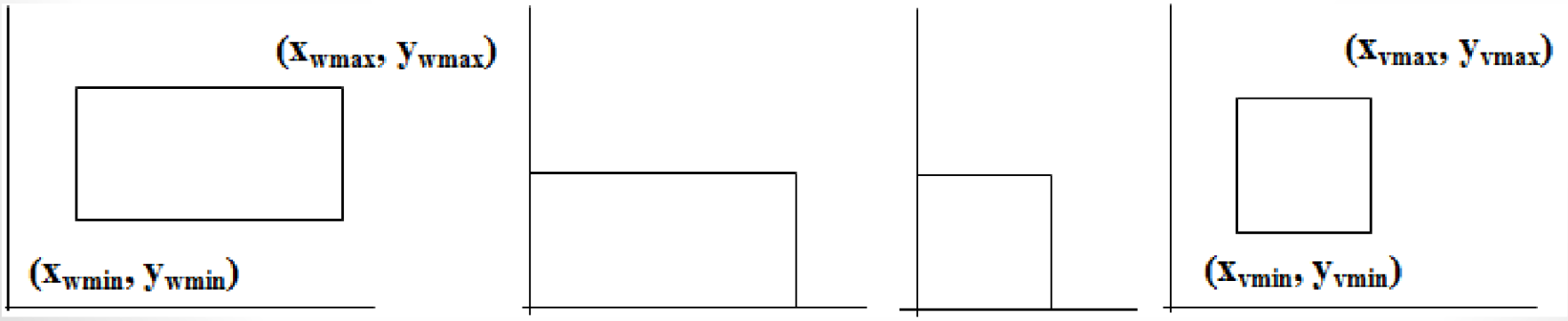
Window to Viewport Transformation

Workstation Transformation Steps:

1. **Translate** the window to the origin. i.e. apply $T(-Xw_{min}, -Yw_{min})$
2. **Scale** it to the size of the viewport. i.e. apply $S(S_x, S_y)$
3. **Translate** scaled window to the position of the viewport. i.e. apply $T(Xv_{min}, Yv_{min})$.

Therefore, Net transformation is given as:

$$T_{WV} = T(-Xw_{min}, -Yw_{min}) \cdot S(S_x, S_y) \cdot T(Xv_{min}, Yv_{min})$$



$$T_{WV} = T(-Xw_{min}, -Yw_{min}) \cdot S(S_x, S_y) \cdot T(Xv_{min}, Yv_{min})$$

Matrix Representation of the above three steps of Workstation Transformation is:

$$T_{wv} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -xw_{min} & -yw_{min} & 1 \end{bmatrix} \bullet \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ xv_{min} & yv_{min} & 1 \end{bmatrix}$$

or

$$T_{wv} = \begin{vmatrix} 1 & 0 & -Xw_{min} \\ 0 & 1 & -Yw_{min} \\ 0 & 0 & 1 \end{vmatrix} \bullet \begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix} \bullet \begin{vmatrix} 1 & 0 & Xv_{min} \\ 0 & 1 & Yv_{min} \\ 0 & 0 & 1 \end{vmatrix}$$

Numerical :

Solved in whiteboard, Check your Class Copy

Assignment

1. Define window and viewport. Explain 2D viewing transformation pipeline.
2. Find the normalization transformation window to viewport, with window, lower left corner at (1,1) and upper right corner at (3,5) onto a viewport with lower left corner at (0,0) and upper right corner at (0.5, 0.5).
3. Derive the formula for windows to viewport transformation.
4. Given a window bordered by (0,0) at the lower left and (4,4) at the upper right. Similarly, a viewport bordered by (0,0) at the lower left and (2,2) at the upper right. If a window at position (2,4) is mapped into the viewport. What will be the position of viewport to maintain same relative placement as in window?

Assignment

1. A normalized window has left and right boundaries of (-0.05 to +0.05) and lower and upper boundaries of (0.1 to 0.2). the viewport window left and right is (250,550) and lower to upper is (100,400), find the coordinate of any point (u,v) in the viewport window.
2. A normalized window has left (xmin=10) and right (xmax=50) boundaries and lower (ymin=5) and upper (ymax=30) boundaries .the viewport window left (umin=25) and right (umax=75) and lower (vmin=25) to upper (vmax=75) find the coordinate of any point (u,v) in the viewport window.

Assignment

1. Specify individually the translation and scaling matrices required to transform a 2D window of [Xmin=-234, Ymin=156] and [Xmax=66, Ymax=456] to a display viewport of [Umin=45, Vmin=35] and [Umax=245, Vmax=185].
2. A normalized window has left and right boundaries of (-0.05 to +0.05) and lower and upper boundaries of (0.1 to 0.2). The viewport window left and right is (250,550) and lower to upper is (100,400),find the transformation matrix.

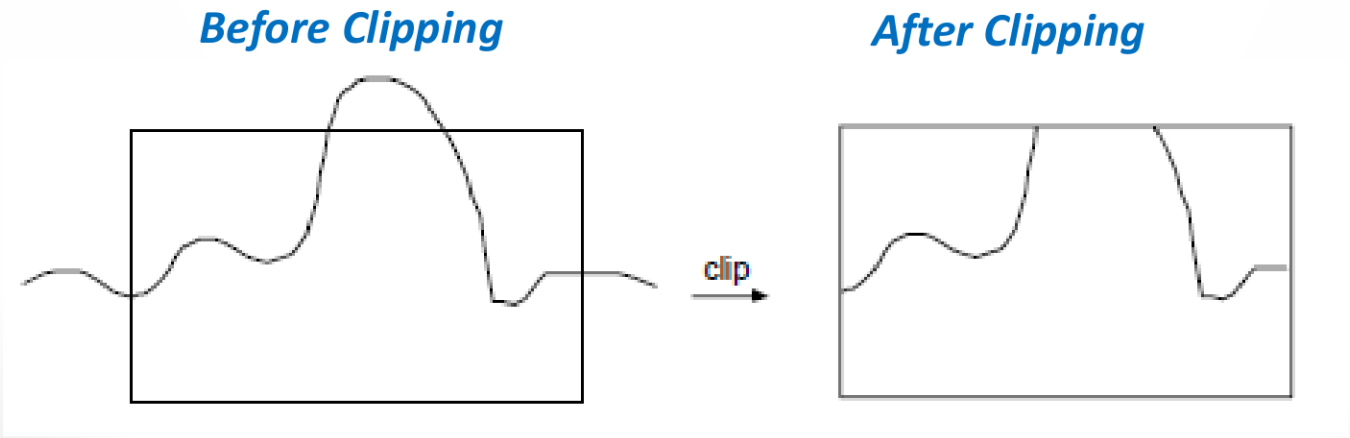
CLIPPING

Clipping

- The process of identifying those portions of a picture that are either inside or outside of the specified region of space is called *clipping*.

Two possible ways to apply clipping in the viewing transformation:

1. Apply clipping in the world coordinate system: *ignore objects that lie outside of the window.*
2. Apply clipping in the device coordinate system: *ignore objects that lie outside of the viewport*.



Clipping

***Why Clipping*?**

1. The primary use of clipping in computer graphics is to remove objects, lines, or line segments that are outside the viewing pane.
2. Excludes unwanted graphics from the screen.
3. Improves efficiency, as the computation dedicated to objects that appear off screen can be significantly reduced.

Clipping

Applications

Applications of clipping includes:

- Drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing, or duplicating.
- Extracting part of a defined scene for viewing;
- Identifying visible surfaces in three-dimensional views;
- Antialiasing line segments or object boundaries;
- Creating objects using solid-modeling procedures;
- Displaying a multi window environment;

Types of Clipping

1. Point Clipping
2. Line Clipping (Straight-line Segments)
3. Area Clipping (Polygons)
4. Curve Clipping
5. Text Clipping

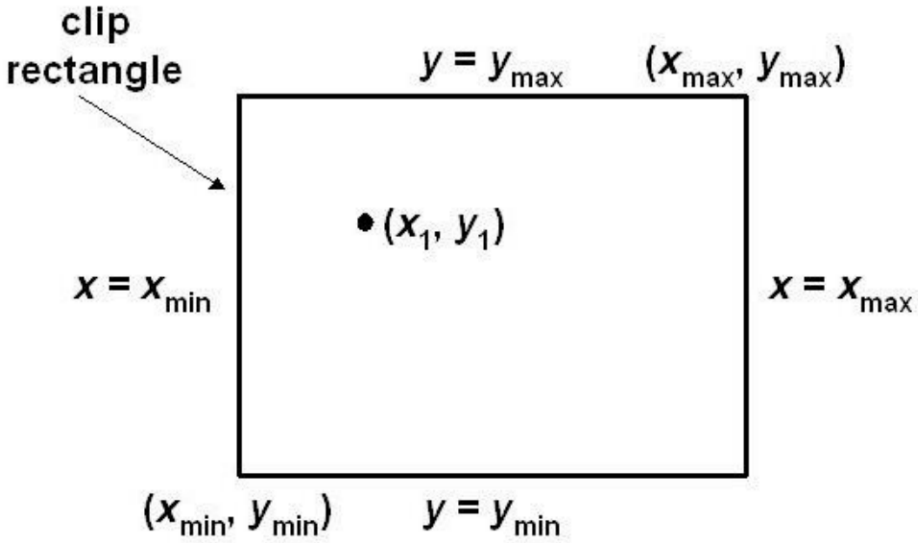
Point Clipping

- Let W denote a clip window with coordinates (xwmin, ywmin), (Xwmin, Ywmax), (Xwmax, Ywmin), (Xwmax, Ywmax), then a vertex (x, y) is displayed only if following "point clipping" inequalities are satisfied:

  $Xwmin \le x \le Xwmax,$

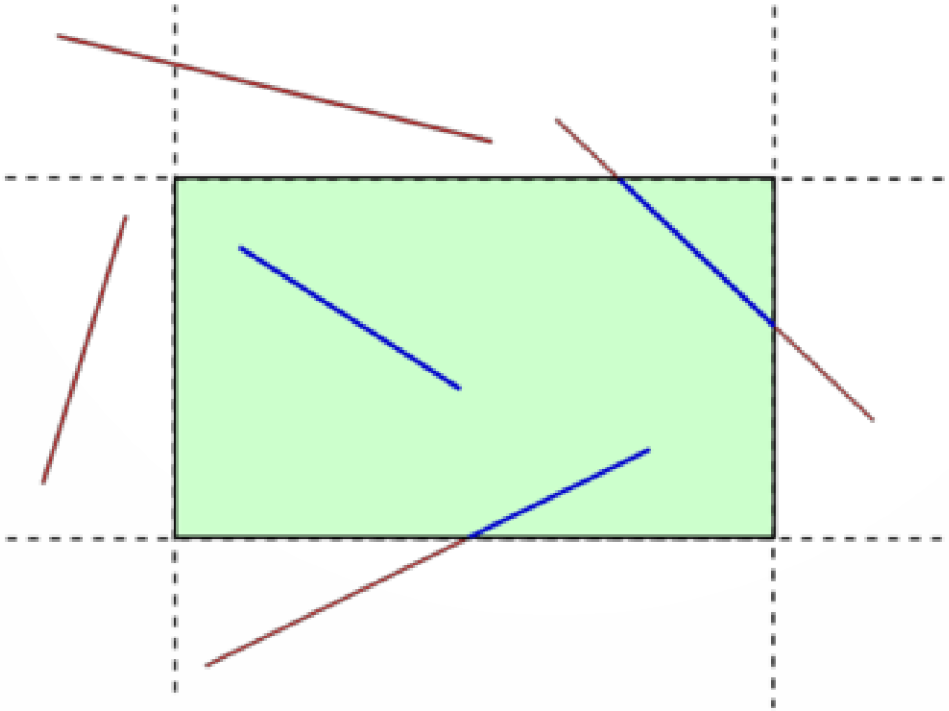
  $Ywmin \le y \le Ywmax.$

- Very simple and efficient.
- Only works for vertices.

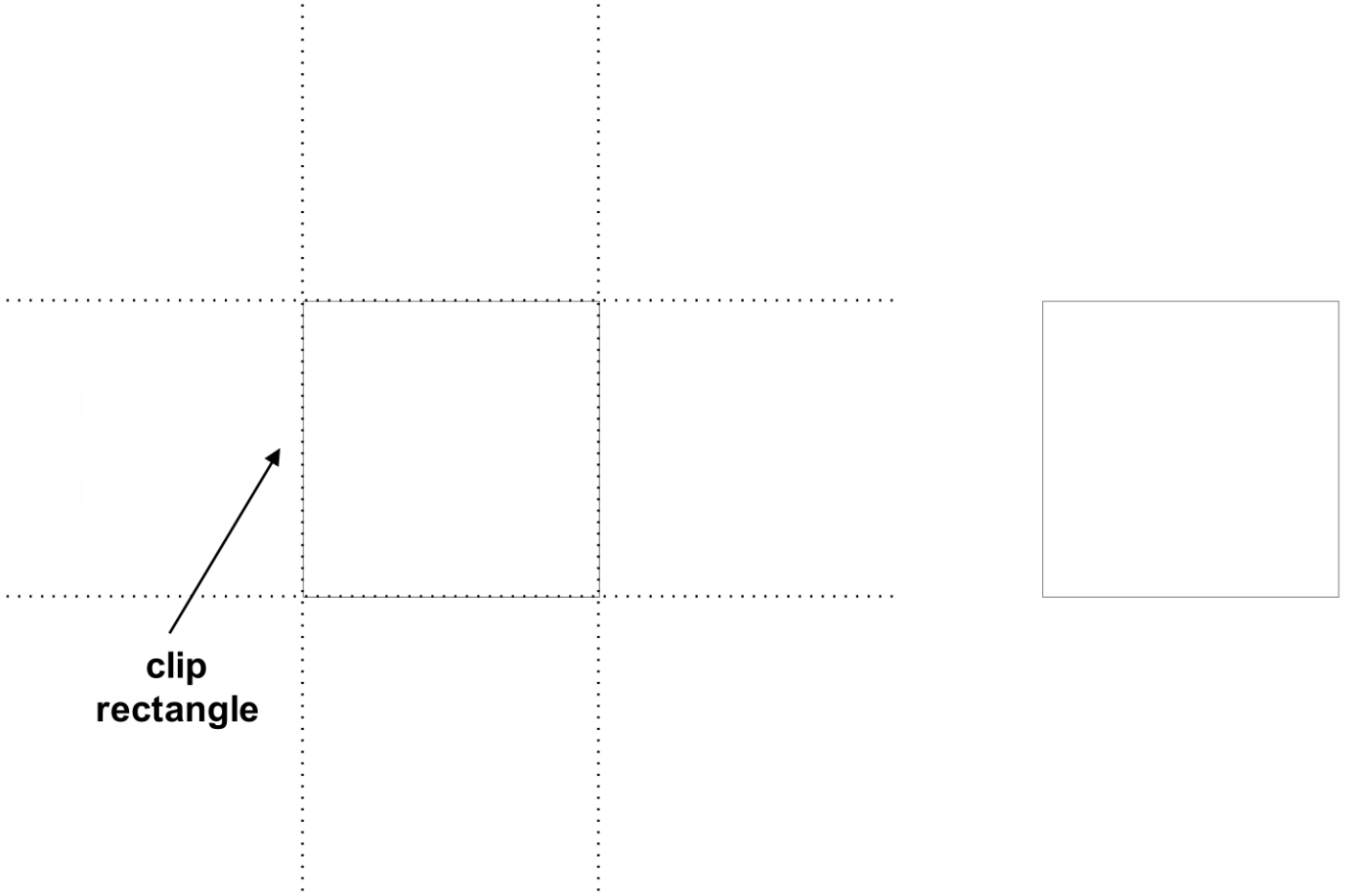


Line Clipping

- Lines that do not intersect the clipping window are either completely inside the window or completely outside the window.
- On the other hand, a line that intersects the clipping window is divided by the intersection point(s) into segments that are either inside or outside the window.

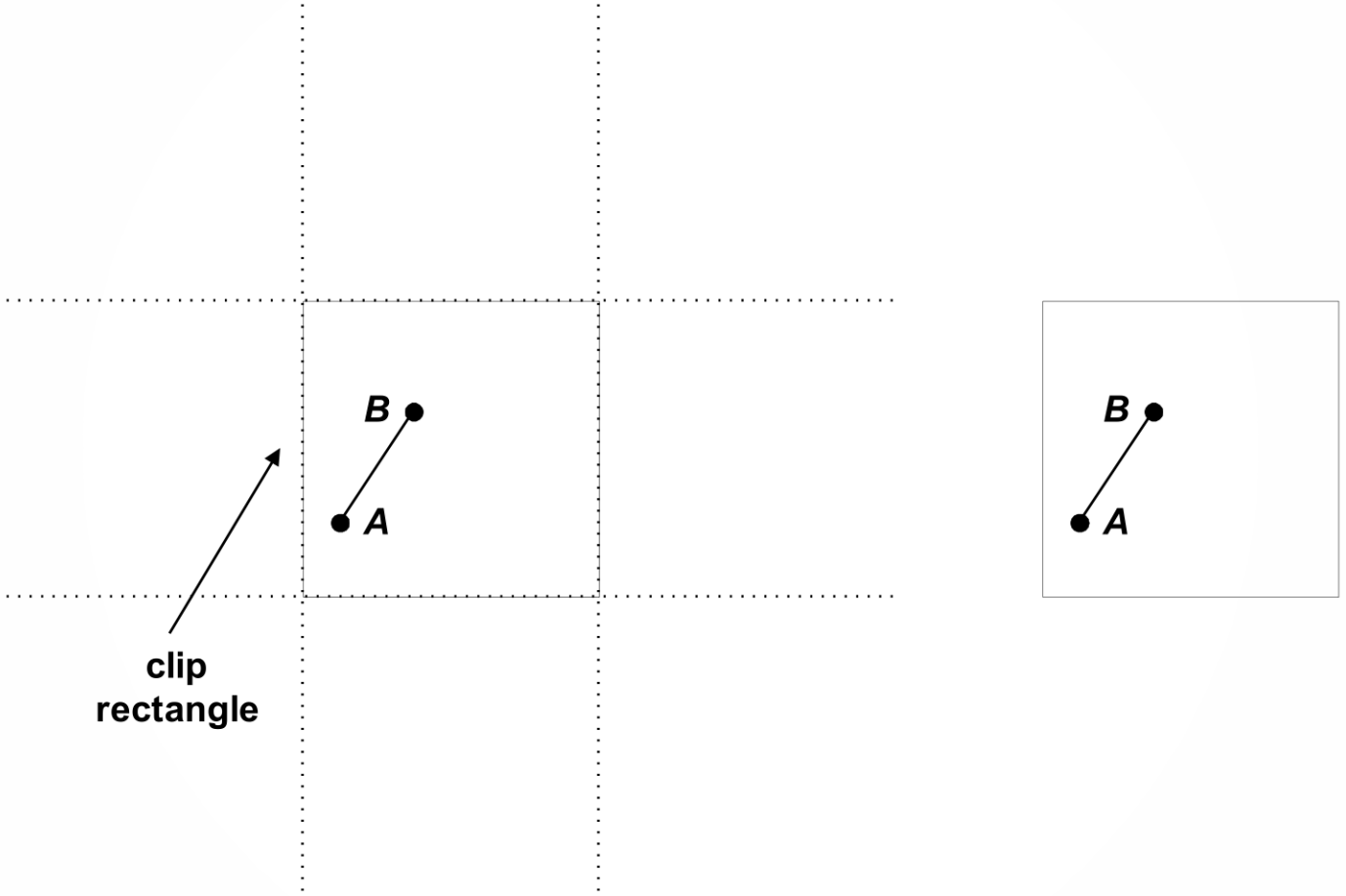


Line Clipping



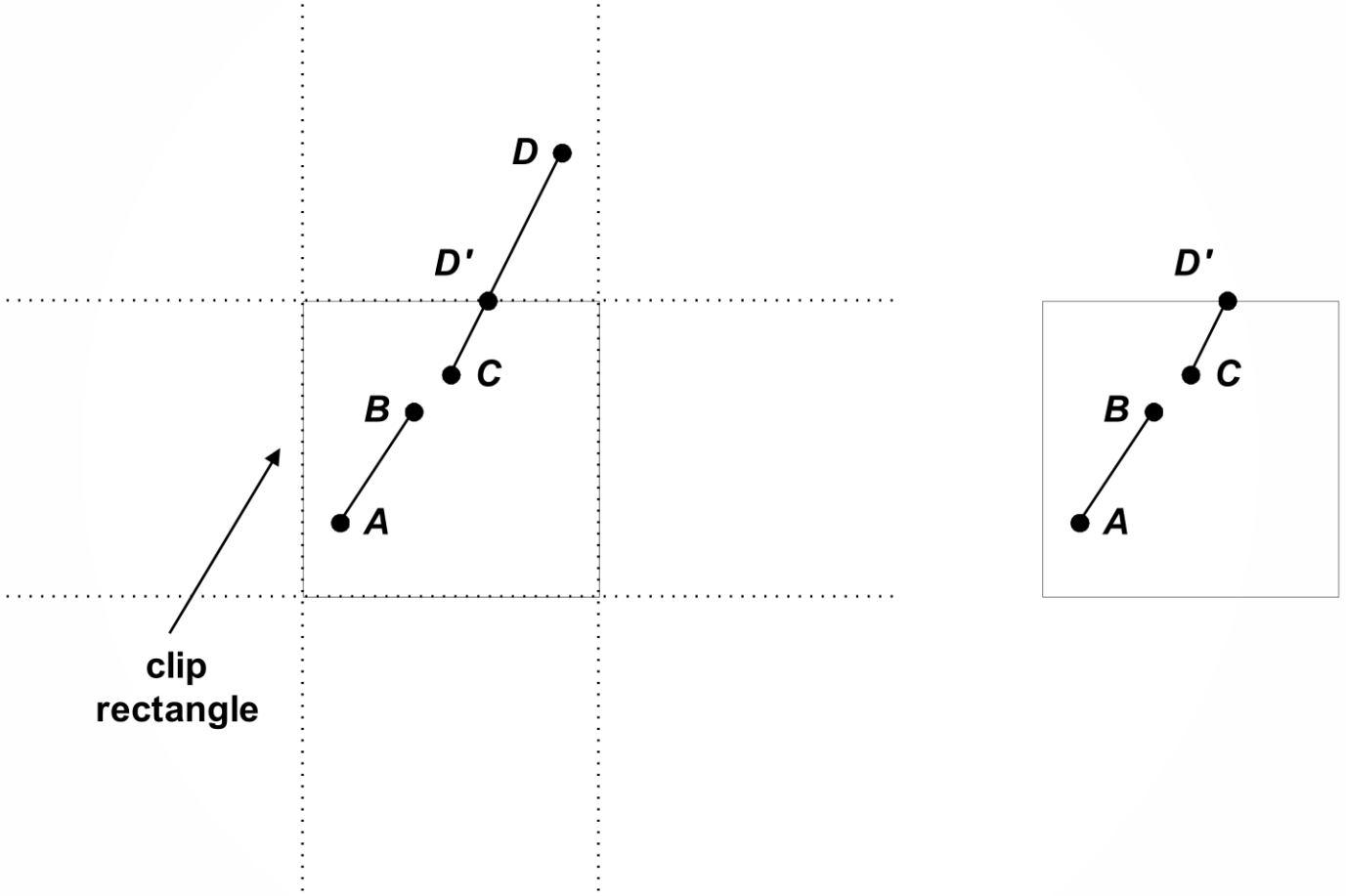
Cases for clipping lines

Line Clipping



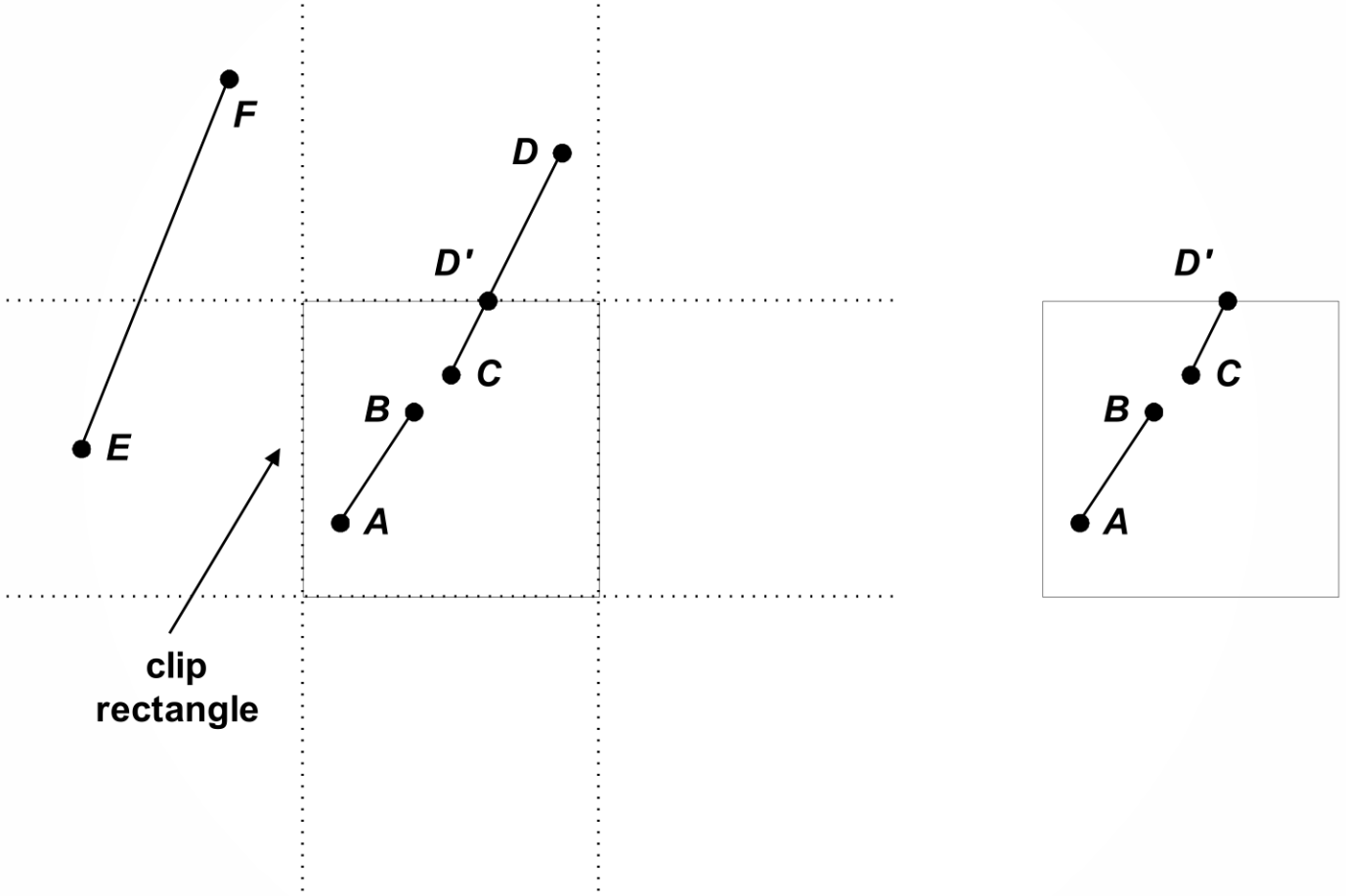
Cases for clipping lines

Line Clipping



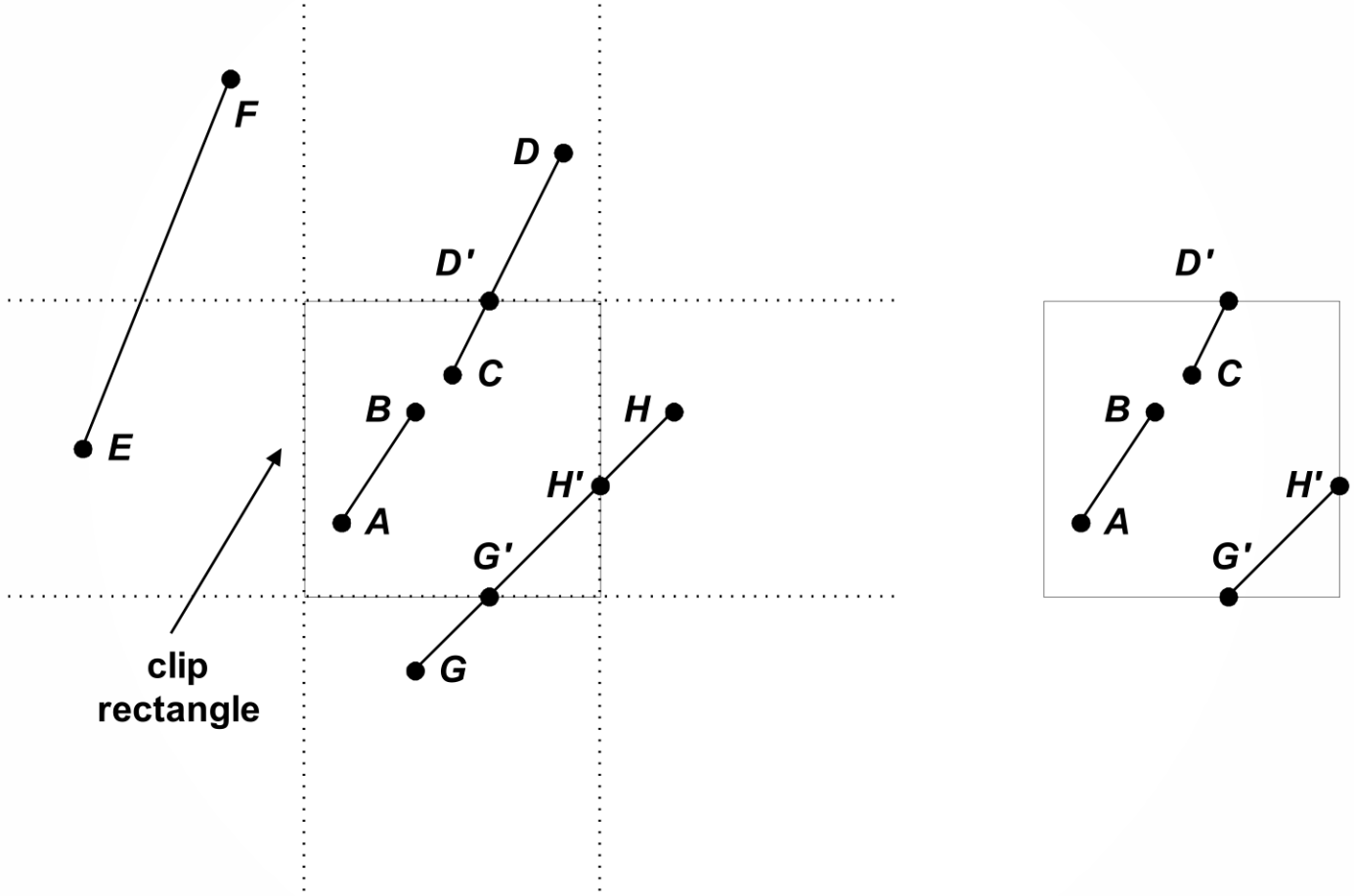
Cases for clipping lines

Line Clipping



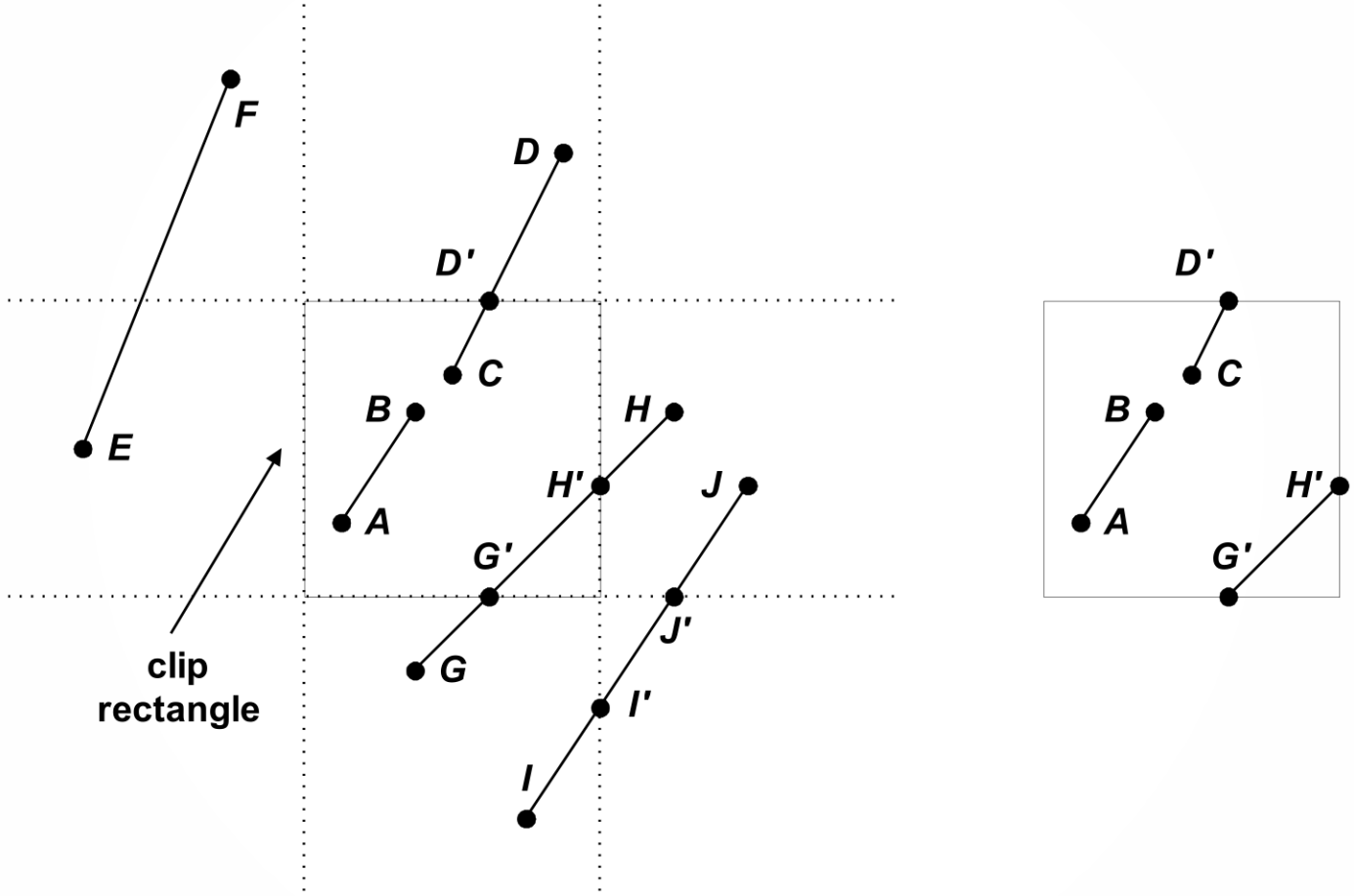
Cases for clipping lines

Line Clipping



Cases for clipping lines

Line Clipping



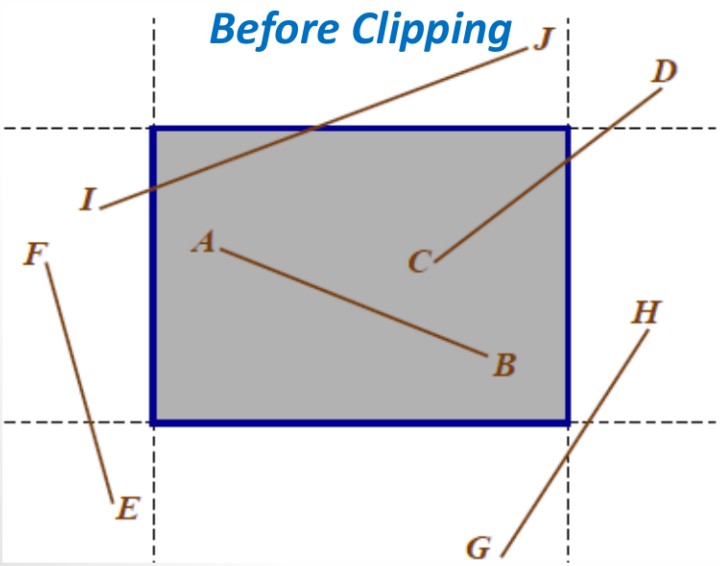
Cases for clipping lines

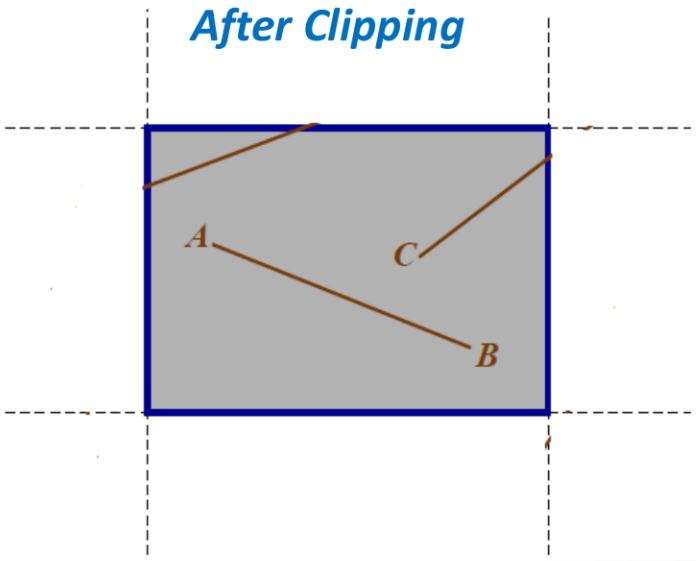
For any particular line segment:

a. **Both endpoints are inside the region (line *AB*).**
   - No clipping necessary.

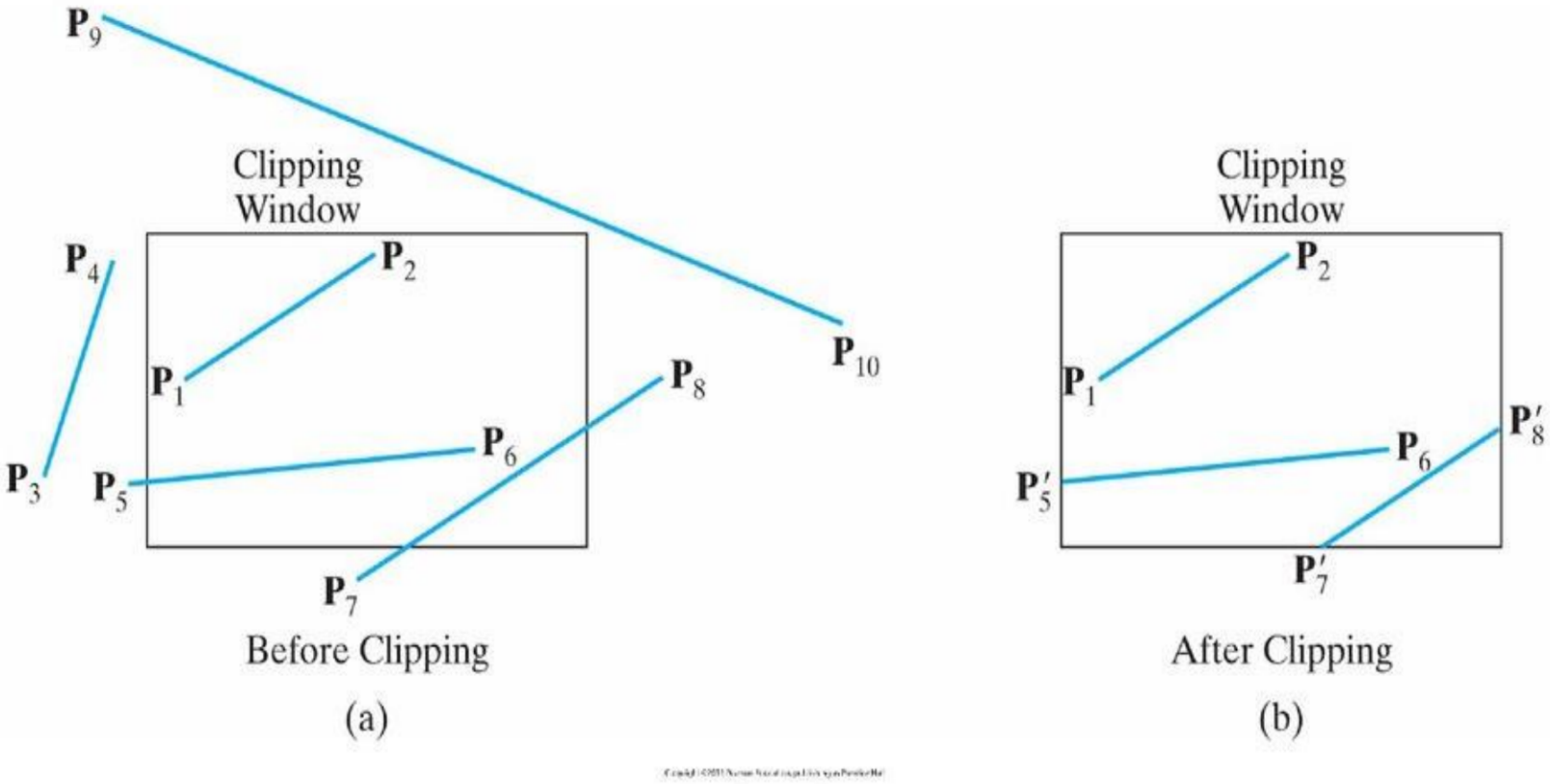
b. **One endpoint is inside and one is outside of the clipping window (line *CD*).**
   - Clip at intersection point.

c. **Both endpoints are outside the region:**
   - No intersection (lines ***EF***, ***GH***)
     - Discard the line segment.
   - Line intersects the region (line ***IJ***)
     - Clip line at both intersection points.



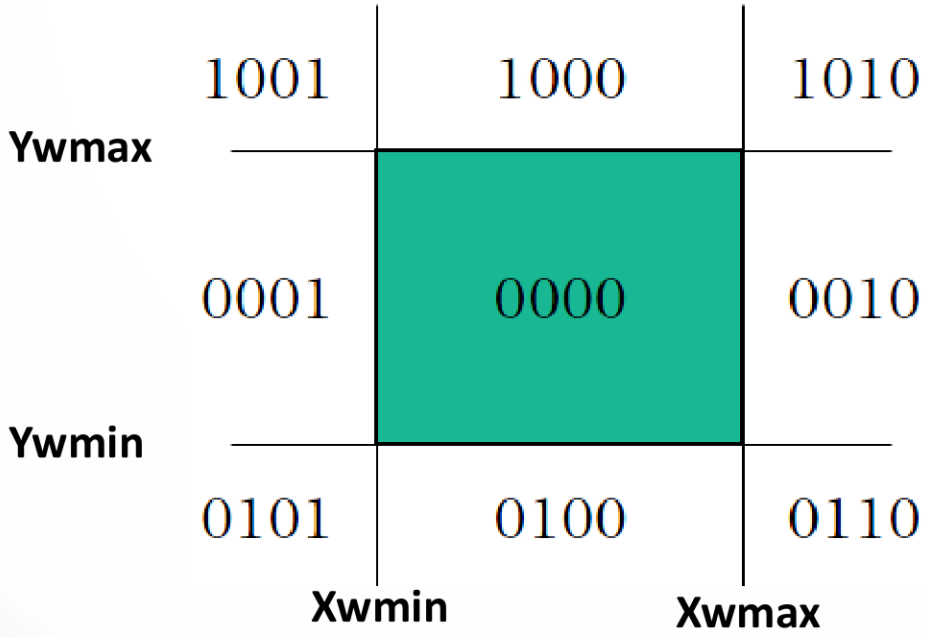


Line Clipping



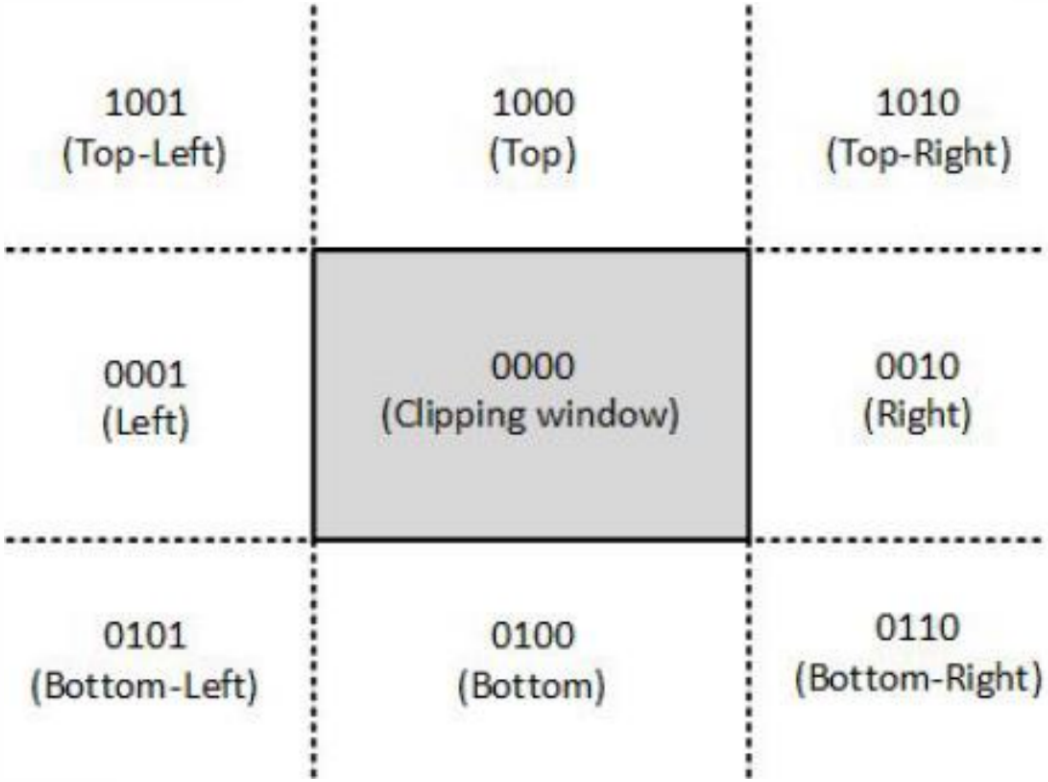
Cohen-Sutherland Line Clipping Algorithm

- One of the oldest and most popular line-clipping algorithms.
- In this method, coordinate system is divided into nine regions.
- All regions have their associated region codes.
- Every line endpoint is assigned a four digit binary code (*region code or out code*).
- Each bit in the code is set to either a 1(true) or a 0(false).
- Assign a bit pattern to each region as shown:



Cohen-Sutherland Line Clipping Algorithm

- Cohen–Sutherland codes for the nine regions code and endpoint region code



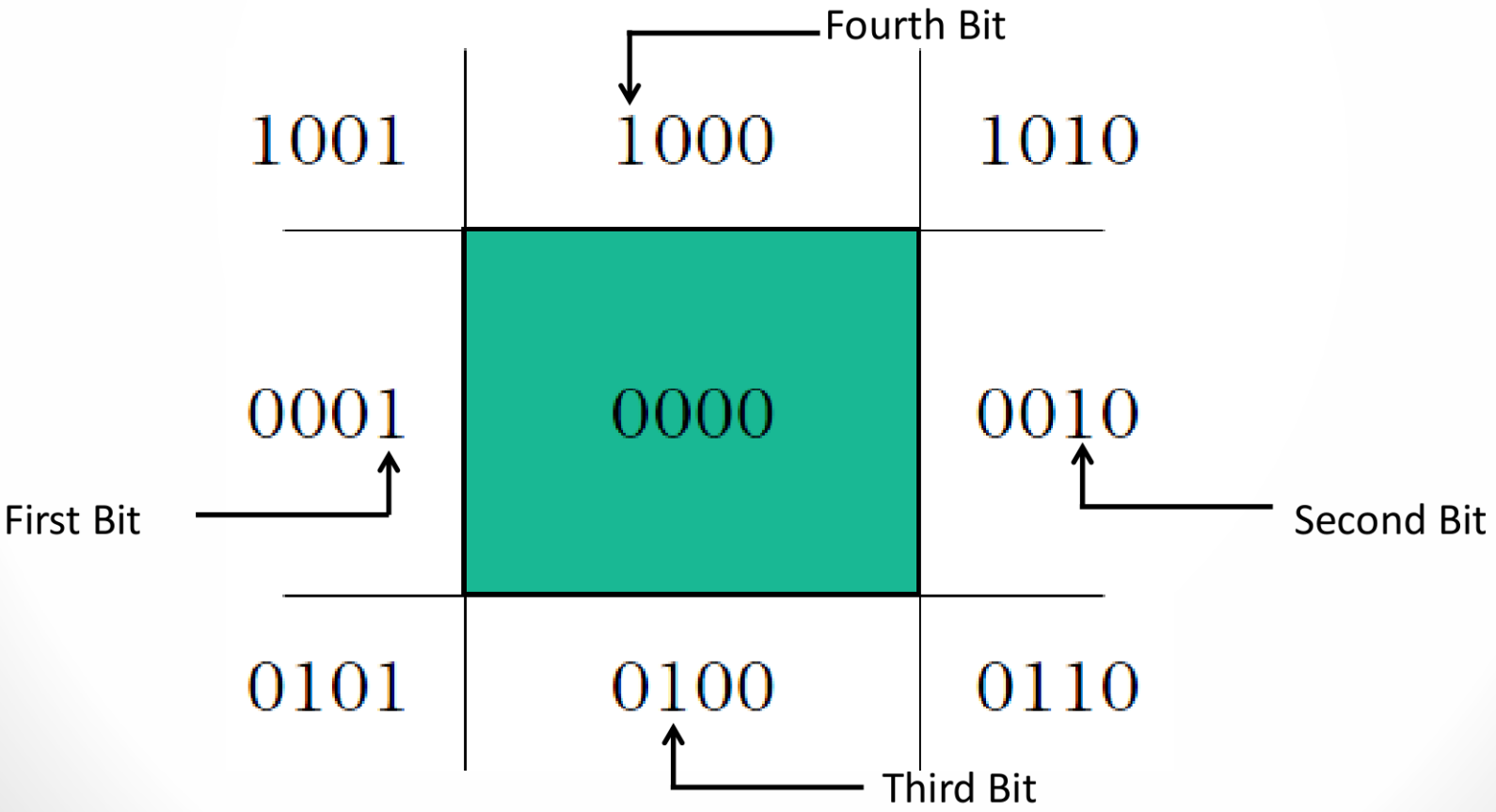
4-bit region codes (in TBRL format).

Cohen-Sutherland Line Clipping Algorithm

- Checking for *trivial acceptance* or *trivial rejection* using *region codes*
  1. Each endpoint of a line segment is assigned a region code;
  2. If both 4-bit codes are zero, the line can be *trivially accepted*;
  3. A logical "**AND**" is performed on both region code;
  4. If the result is nonzero, the line can be *trivially rejected*.
- If not trivially accepted or trivially rejected, divide the line segment into two at a clip edge;
- Iteratively clipped by test trivial-acceptance or trivial-rejection, and divided into two segments until completely inside or trivial-rejection.

Step 1: Establish Region code for all line end point

- First bit is 1, if x < Xwmin (Point lies to left of window) , else set it to 0
- Second bit is 1, if x > Xwmax (Point lies to right of window) , else set it to 0
- Third bit is 1, if y < Ywmin (Point lies to below window), else set it to 0
- Fourth bit is 1, if y > Ywmax (Point lies to above window) , else set it to 0



Step 2: Determine which lines are completely inside window and which are not

a) If both end points of line has region codes "0000" line is completely inside window.

b) If logical AND operation of region codes of two end points is NOT "0000". The line is completely outside (some bit position have 1's)

Astart 0000
Aend 0000

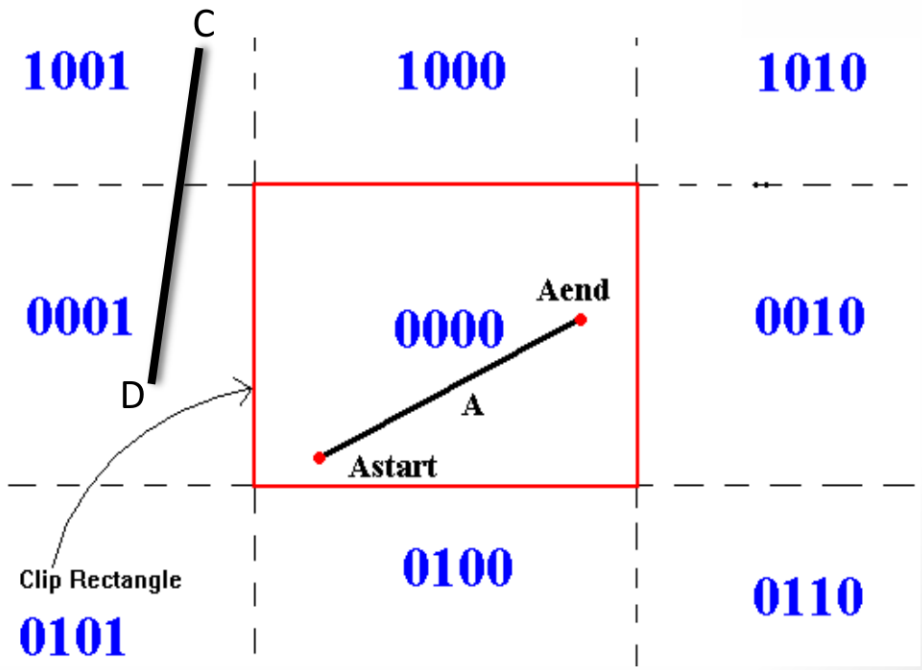
Completely inside

C 1001
D 0001

AND

0001

Completely Outside



Step 3: **If both tests fail then line is partially visible so we need to find the intersection with boundaries of window.**

a) If 1st bit is 1 then line intersect with **Left** boundary and

- $Y_i = Y_1 + m\,(X - X_1)$ *where* X= Xwmin

b) If 2nd bit is 1 then line intersect with **Right** boundary and

- $Y_i = Y_1 + m\,(X - X_1)$ *where* X= Xwmax

c) If 3rd bit is 1 then line intersect with **Bottom** boundary and

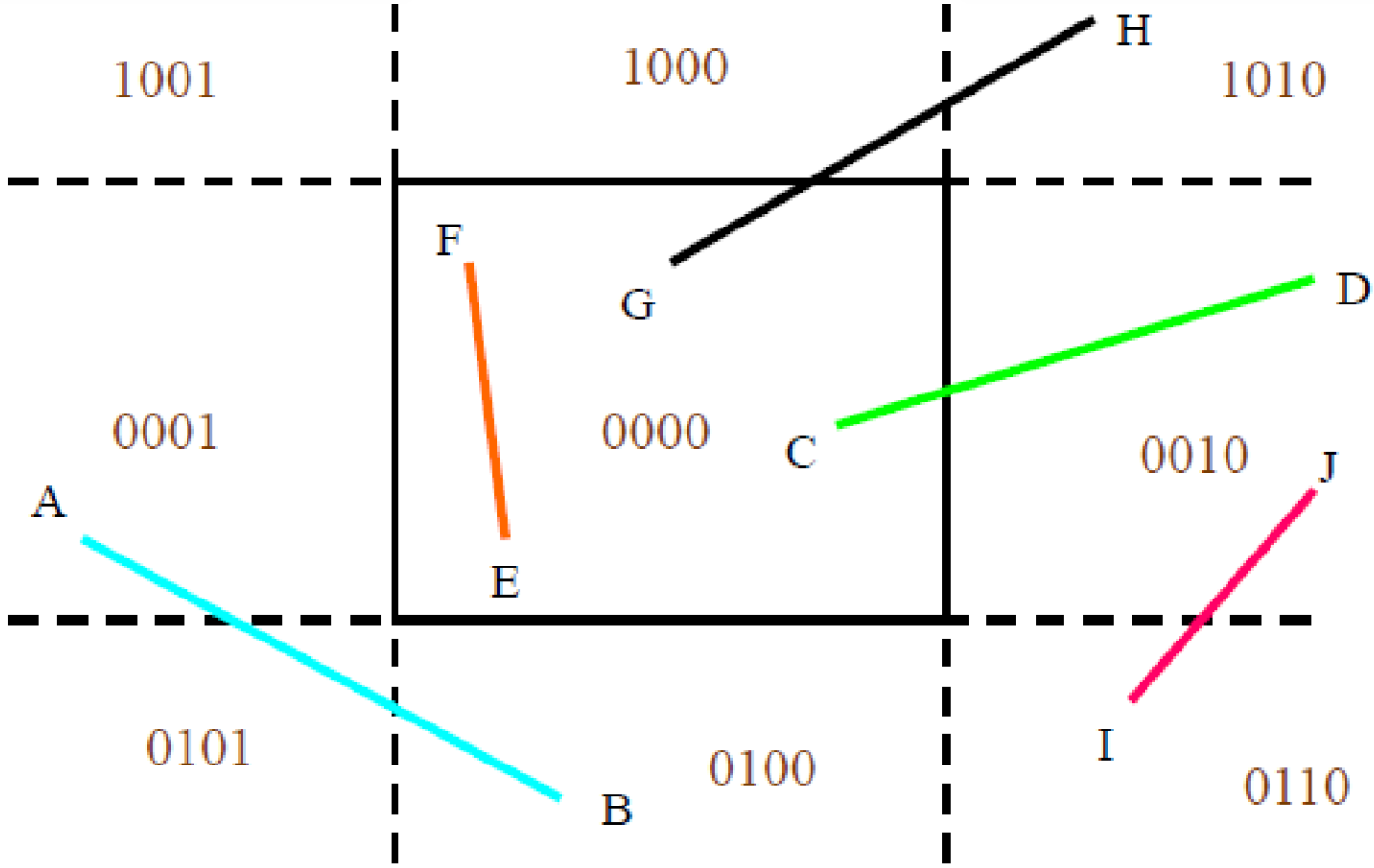
- $X_i = X_1 + (1/m)\,(Y - Y_1)$ *where* Y= Ywmin

d) If 4th bit is 1 then line intersect with **Top** boundary and

- $X_i = X_1 + (1/m)\,(Y - Y_1)$ *where* Y= Ywmax

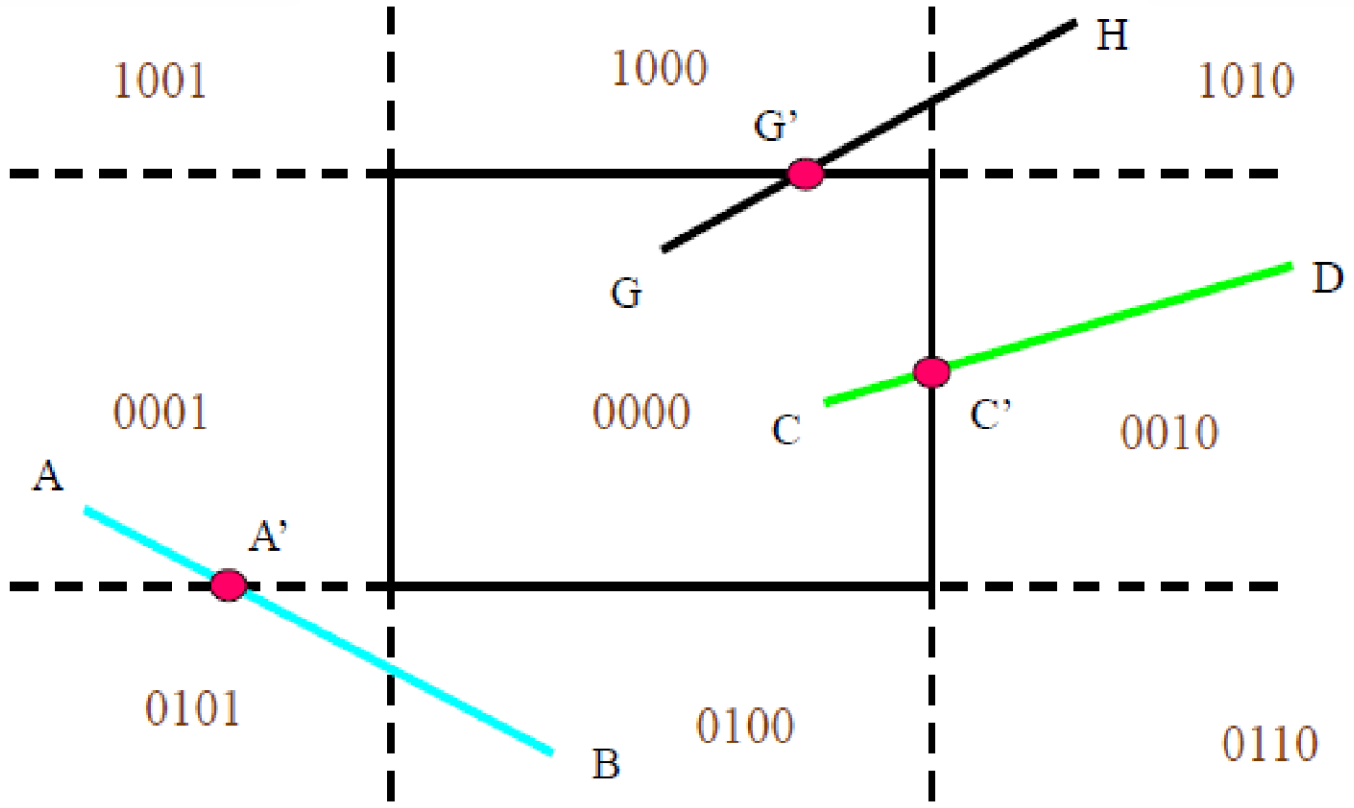
Here, (X_i , Y_i) are (X ,Y) intercepts for that the step 4 repeat step 1 through step 3 until line is completely accepted or rejected

Cohen-Sutherland Line Clipping: Example



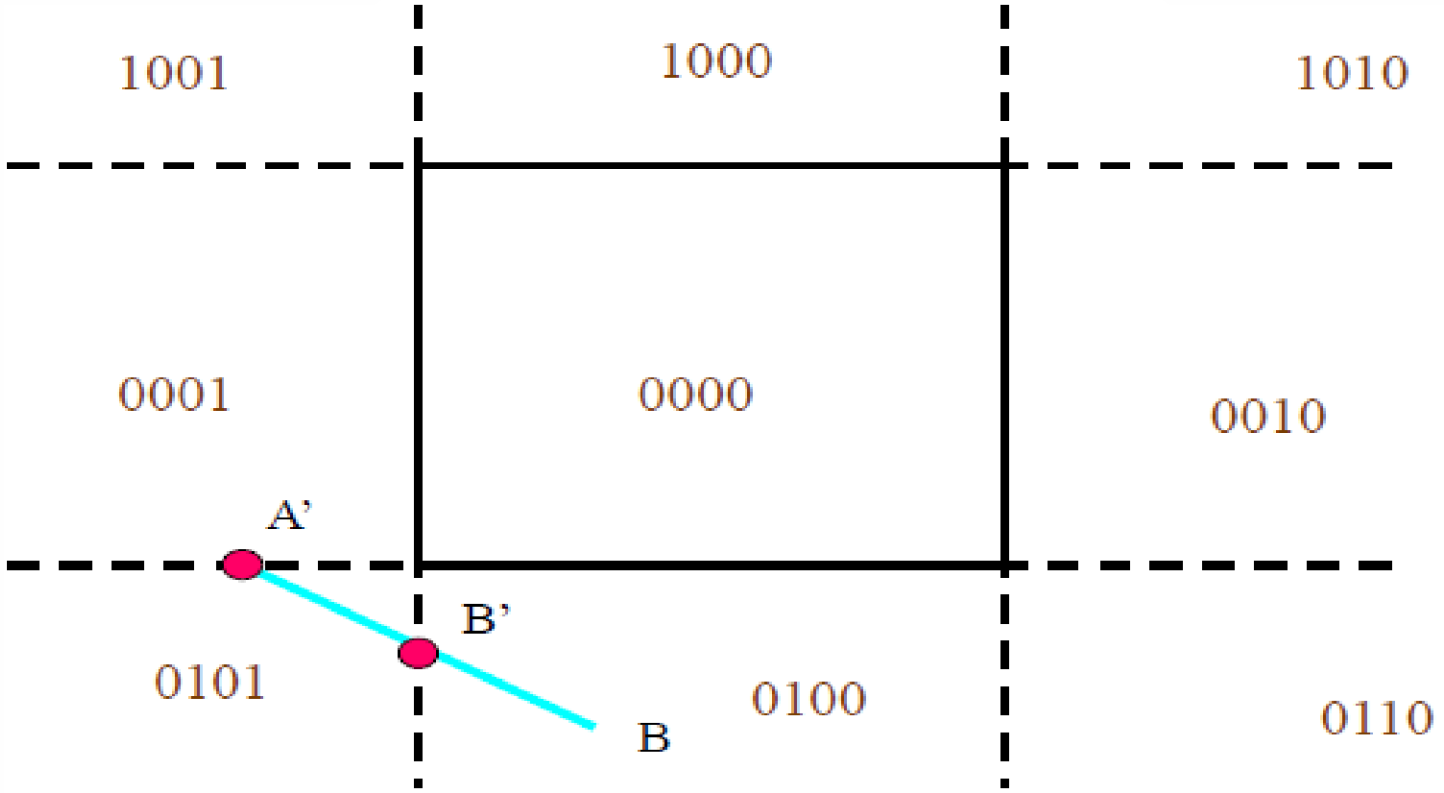
A	0001	C	0000	E	0000	G	0000	I	0110
B	0100	D	0010	F	0000	H	1010	J	0010
OR	0101	OR	0010	OR	0000	OR	1010	OR	0110
AND	0000	AND	0000	AND	0000	AND	0000	AND	0010
Clip		Clip		Accept		Clip		Reject	

Cohen-Sutherland Line Clipping: Example



A	0001	A'	0001	C	0000	C'	0000	G	0000	G'	0000
A'	0001	B	0100	C'	0000	D	1010	G'	0000	H	1010
		OR	0101	OR	0000			OR	0000		
remove		AND	0000	AND	0000	**remove**		AND	0000	**remove**	
		Clip		**accept**				**accept**			

Cohen-Sutherland Line Clipping: Example



A' 0001
B' 0100

remove

B' 0100
B 0100
OR 0100
AND 0100

reject

Cohen-Sutherland Line Clipping Algorithm

1. If both endpoints 0000 (i.e. code 1 || code 2 = 0), **ACCEPT**
2. Else If (code 1 && code 2) != 0, **REJECT**
3. Else

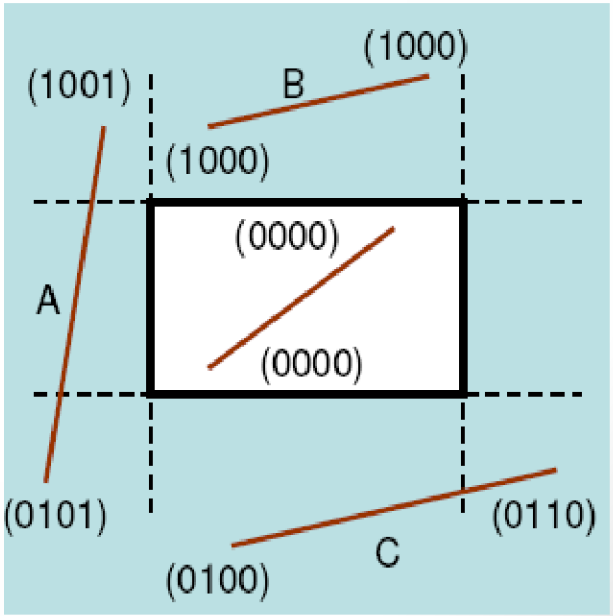
    {

        Clip line against one viewport boundary

4. Assign the new endpoint with a 4-bit region code
5. Goto 1

    }

Note: (|| is bitwise OR)
Note: (&& is bitwise AND)



Cohen-Sutherland Line Clipping Algorithm

Algorithm

Step 1 – Assign a region code for each endpoints.

Step 2 – If both endpoints have a region code 0000 then accept this line.

Step 3 – Else, perform the logical AND operation for both region codes.

Step 3.1 – If the result is not 0000, then reject the line.

Step 3.2 – Else you need clipping.

Step 3.2.1 – Choose an endpoint of the line that is outside the window.

Step 3.2.2 – Find the intersection point at the window boundary (base on region code).

Step 3.2.3 – Replace endpoint with the intersection point and update the region code.

Step 3.2.4 – Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.

Step 4 – Repeat step 1 for other lines.

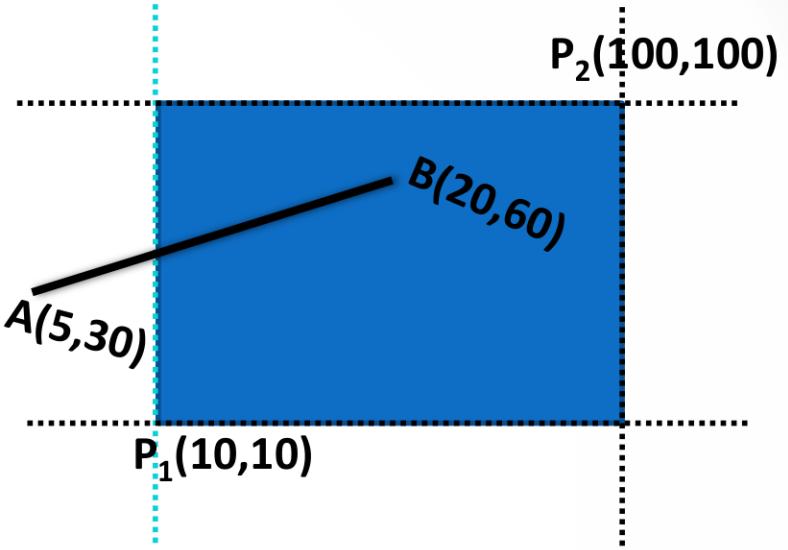
Numerical

Q. Clip a line with end point A(5,30), B(20,60) against a clip window with lower most left corner at P1(10,10) and upper right corner at P2(100,100)

Solution

Step 1: Establish the region codes for end point A,B

A(5,30)	B(20,60)
5<10 : (true)= 1	20<10: (false) =0
5>100 : (false) =0	20>100: (false)=0
30<10 : (false) =0	60<10 : (false)=0
30>100 : (false) =0	60>100: (false)=0
A(5,30) = 0001	***B(20,60)=0000***



Step 2: Visibility check

A: 0001
B: 0000
AND ——————
0000

Partial Visibility

Step 3: Intersection point with boundary

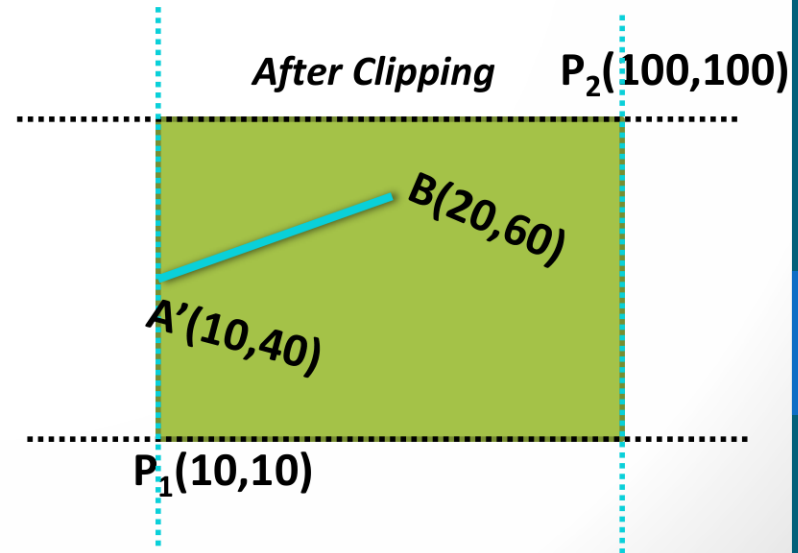
A: 0001 , Cond^{n} of 1st bit is 1

$Y_i = Y_1 + m(X - X_1)$ *where* , X= 10

= 30 + 2 (10-5)

= 40

(X,Y)=(10,40)



Line Clipping - Numerical

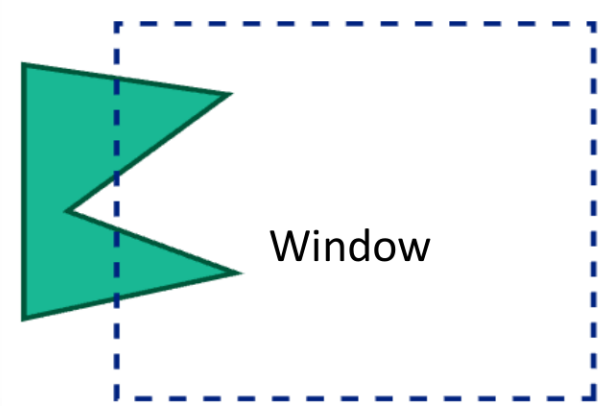
Go through your Class Copy

Assignment

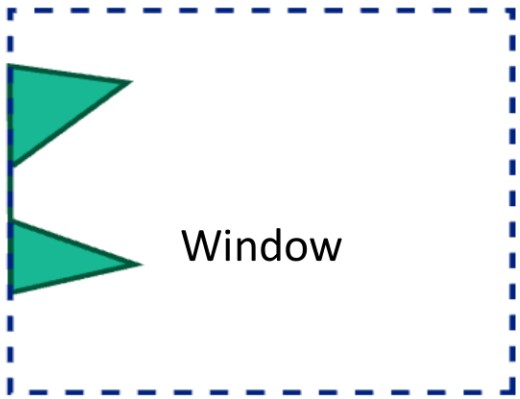
1. Explain Cohen Sutherland line clipping algorithm.
2. Let ABCD be the rectangular window with A(20,20), B(90,20), C(90,70) and D(20,70). Use Cohen Sutherland algorithm to find the region codes and clip the lines with end point P(10,30) and Q(80,80)
3. Clip a line with end point A(5,15) , B(110,60) against a clip window with lower most left corner at P1(10,10) and upper right corner at P2(100,100)
4. Clip a line with end point A(1,60) , B(20,120) against a clip window with lower most left corner at P1(10,10) and upper right corner at P2(100,100)

Polygon Clipping

- A polygon can be defined as a geometric object "consisting of a number of points (called *vertices*) and an equal number of line segments (called *sides* or *edges*).
- Polygon clipping is defined as the process of removing those parts of a polygon that lie outside a clipping window.
- Consider a general polygon that is clipped with respect to a rectangular viewing region.

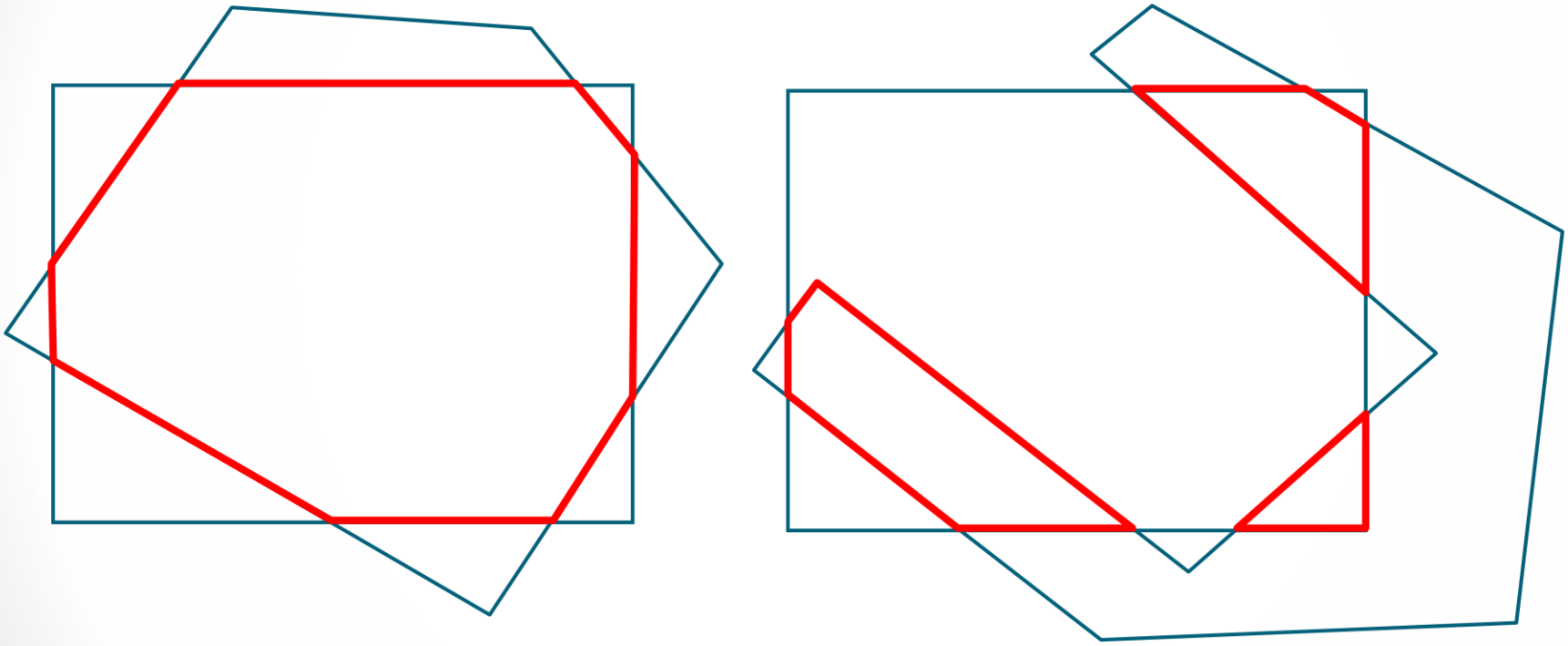


Before clipping



After clipping

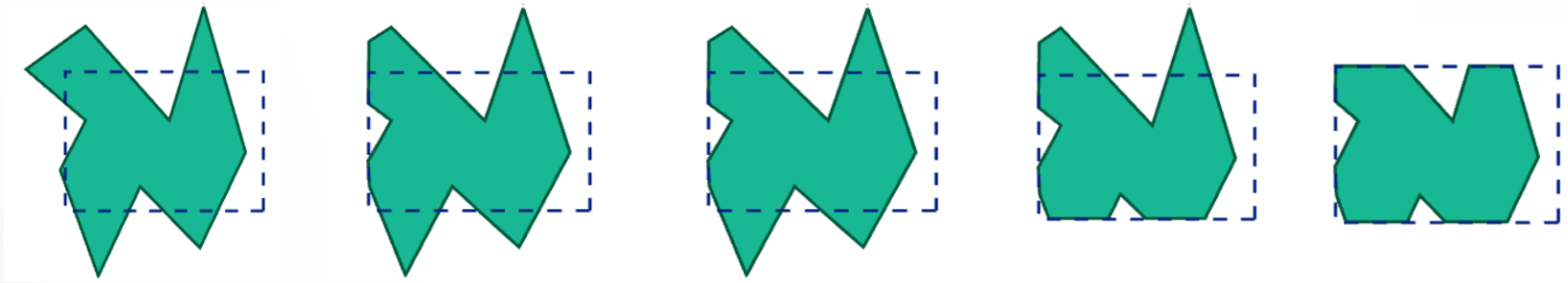
Polygon Clipping



Example

Sutherland-Hodgeman Polygon Clipping Algorithm

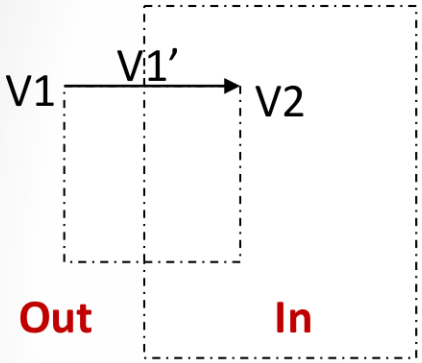
- The Sutherland-Hodgeman Polygon-Clipping Algorithms clips a given polygon successively against the edges of the given clip-rectangle.
- It starts with the initial set of polygon vertices, first clips the polygon against the left rectangle boundary of the window.
- Then successively against the right boundary, bottom boundary, and finally against the top boundary, as shown in figure.



Original Polygon | Clip Left | Clip Right | Clip Bottom | Clip Top

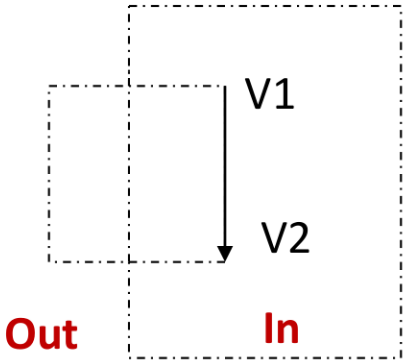
Sutherland-Hodgeman Polygon Clipping Algorithm

Case 1: Out -> In



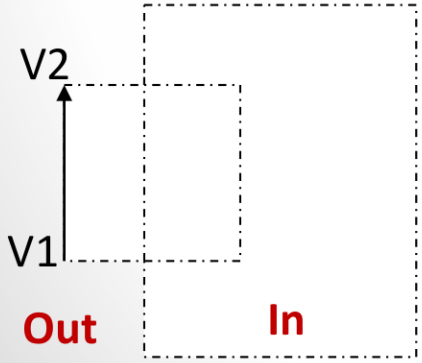
If V1 is Outside AND V2 is Inside then Save V1' and V2

Case 2: In -> In



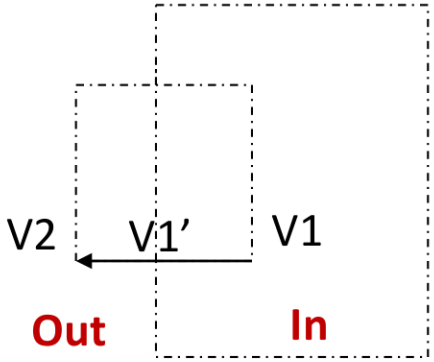
If V1 and V2 are inside then Save V2

Case 3: Out -> Out



If V1 And V2 are Outside then Save nothing

Case 4: In -> Out



If V1 is Inside AND V2 is Outside then Save V1'

Sutherland-Hodgeman Polygon Clipping Algorithm

- **Case 1:**
  - If previous vertex is outside and current vertex is inside the window boundary, then both the intersection points and the current vertex are added to the new vertex list
- **Case 2:**
  - If both vertex are inside then only the current vertex will be added to the list
- **Case 3:**
  - If the previous vertex is inside and the current vertex is outside then only the intersection point is added to the list
- **Case 4:**
  - If both the vertex are outside then intersection point are tested for visibility and then added to the vertex list

Sutherland-Hodgeman Polygon Clipping Algorithm

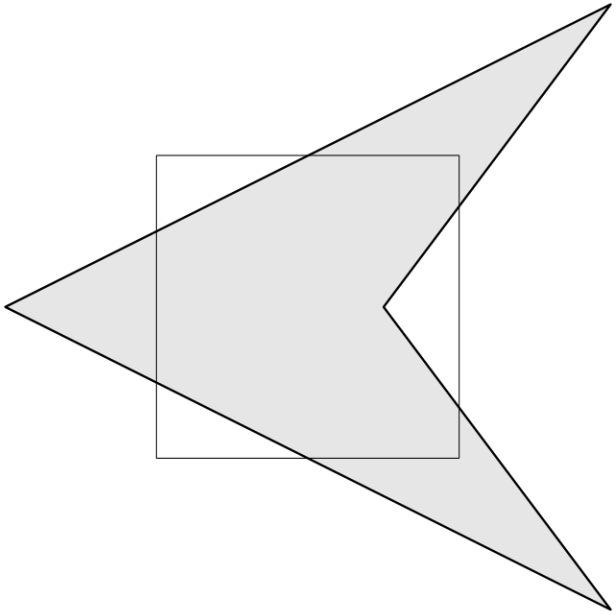
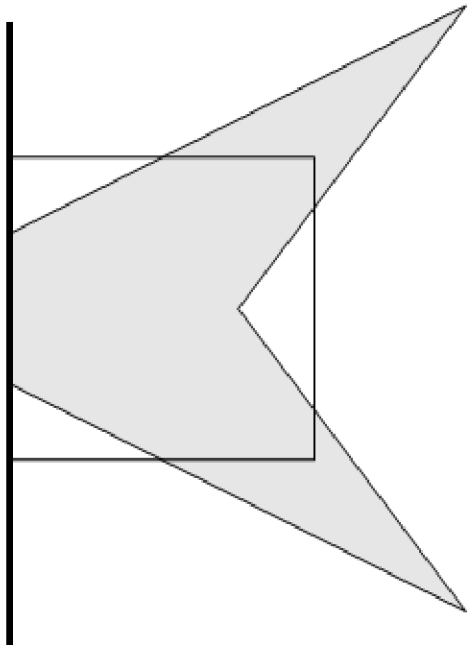


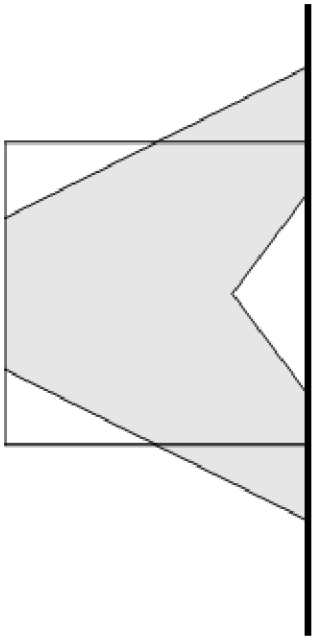
Figure: Before Clipping (Initial Condition)

Sutherland-Hodgeman Polygon Clipping Algorithm



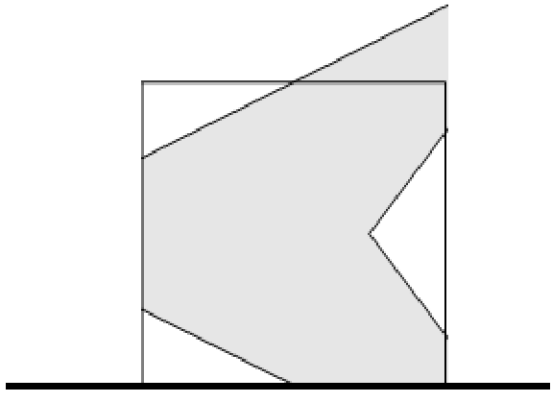
Clip Against Left Clipping Boundary

Sutherland-Hodgeman Polygon Clipping Algorithm



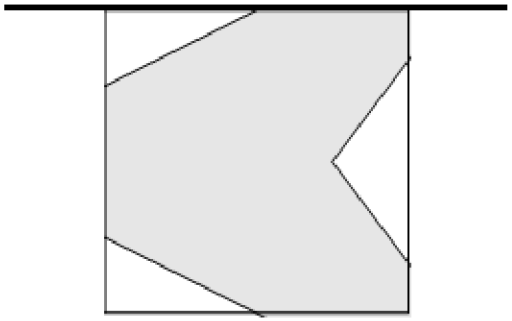
Clip Against Right Clipping Boundary

Sutherland-Hodgeman Polygon Clipping Algorithm



Clip Against Bottom Clipping Boundary

Sutherland-Hodgeman Polygon Clipping Algorithm



Clip Against Top Clipping Boundary

Sutherland-Hodgeman Polygon Clipping Algorithm

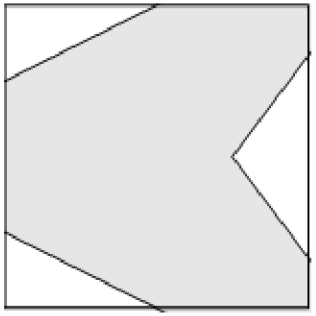


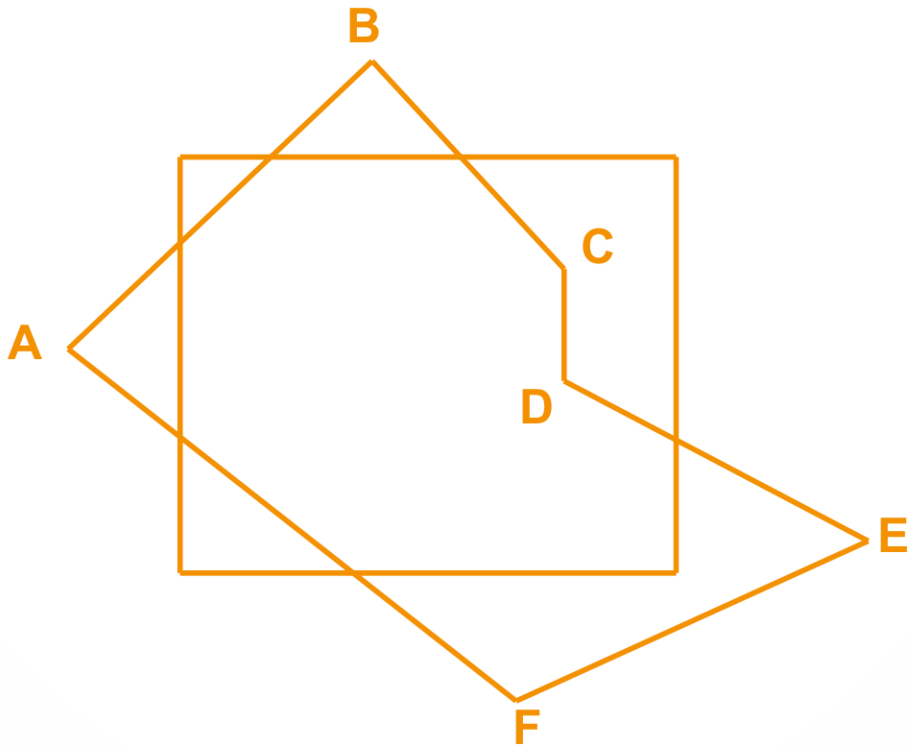
Figure: After Clipping

Steps in Sutherland-Hodgeman Polygon Clipping Algorithm

1) Read coordinates of all vertices of the Polygon.
2) Read coordinates of the clipping window.
3) Consider the left edge of the window.
4) Compare the vertices of each edge of the polygon, individually with the clipping plane.
5) Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary.
6) Repeat the steps 4 and 5 for remaining edges or the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.
7) Stop.

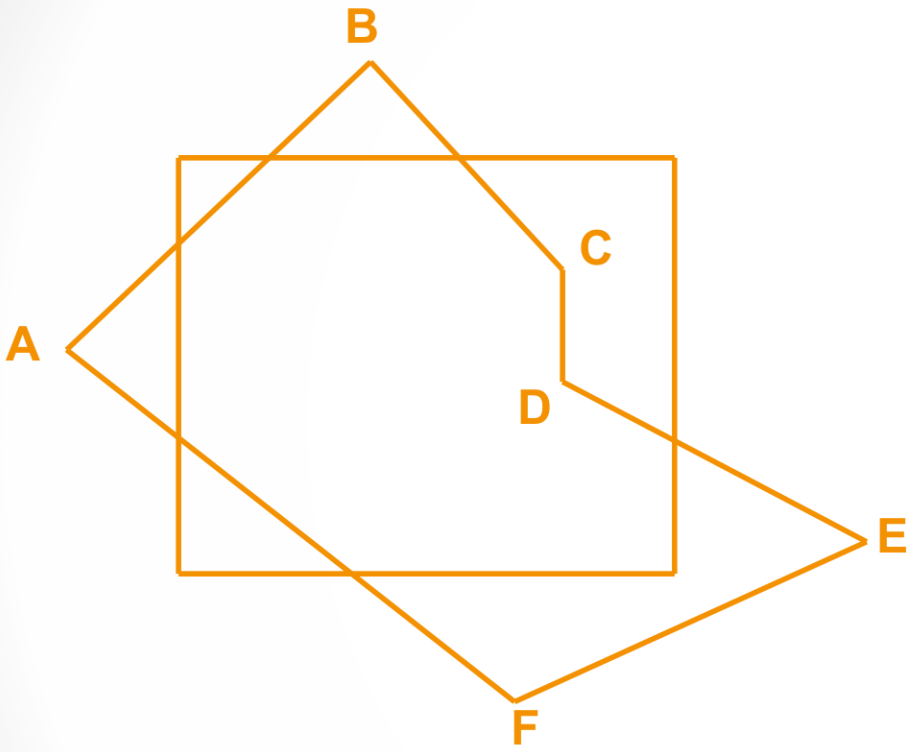
Sutherland-Hodgeman Polygon Clipping Algorithm

Example: For a polygon and clipping window shown in figure below give the list of vertices after each boundary clipping.



Input: A B C D E F

Sutherland-Hodgeman Polygon Clipping Algorithm



Input: A B C D E F

Consider a polygon side: starting vertex S; end vertex P

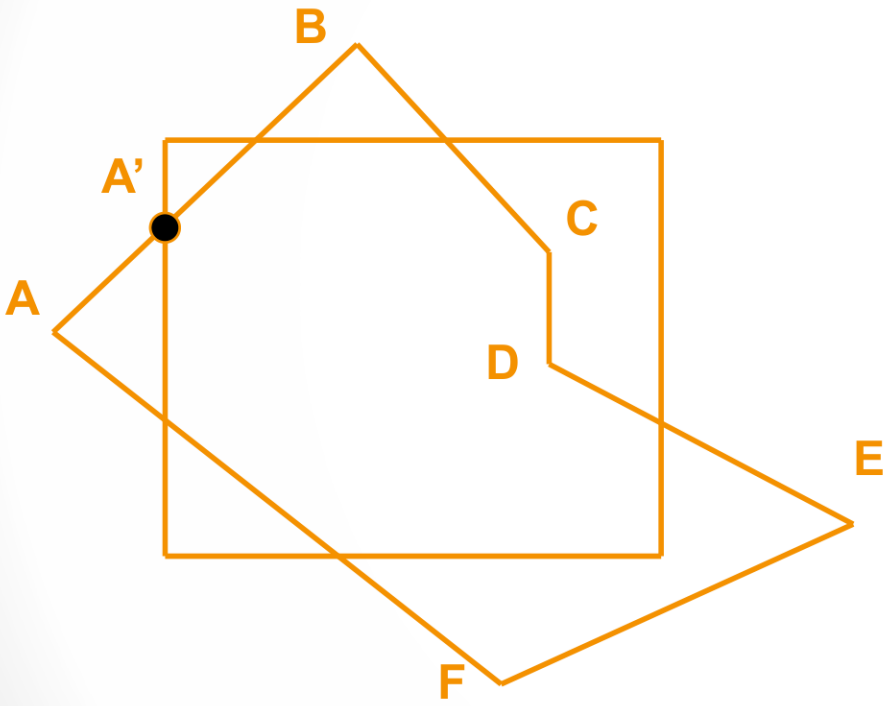
Take each edge in turn
- start with **left edge**

Take each point in turn:

(i) Input point and call it P
- thus P = A

(ii) If P is first point:
- store as P_1
- output if visible
- let S = P

Sutherland-Hodgeman Polygon Clipping Algorithm



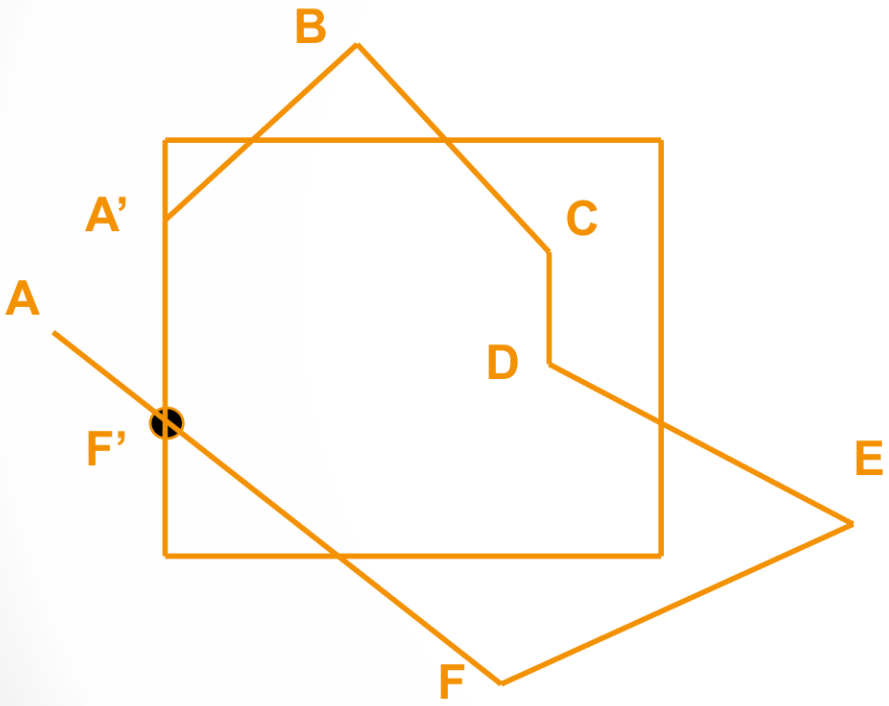
(iii) If P not first point,
then if SP crosses window edge:
- compute intersection I
- output I
output P if visible

(iv) let S = P

Output: A' B C D E F

Figure: Clip against **left edge**

Sutherland-Hodgeman Polygon Clipping Algorithm

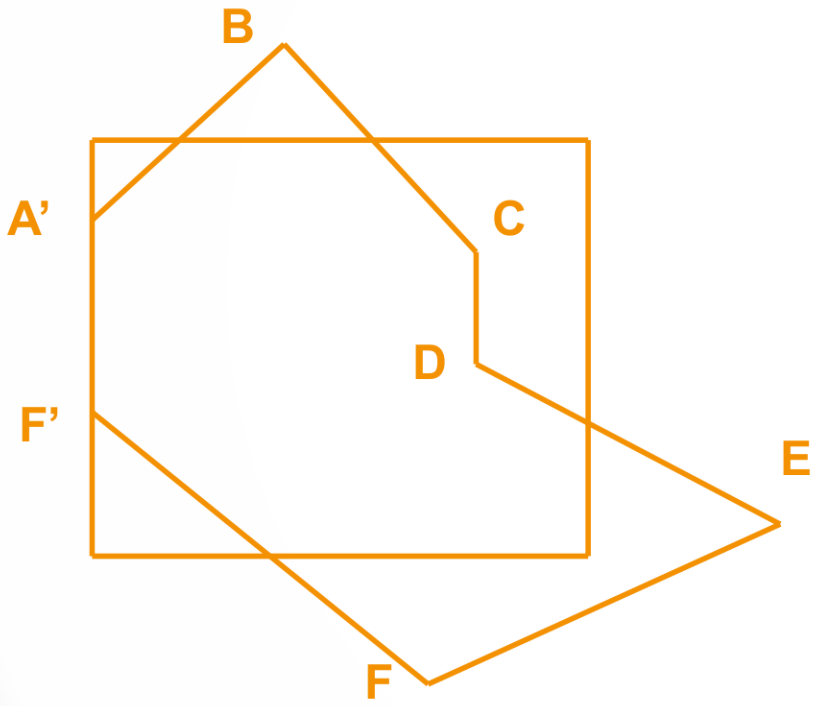


Finally, if some points have been output, then
if SP_1 crosses window edge:
- compute intersection I
- output I

Output: A' B C D E F F'

Figure: Clip against **left edge**

Sutherland-Hodgeman Polygon Clipping Algorithm



The result of clipping against the **left edge**

Sutherland-Hodgeman Polygon Clipping Algorithm

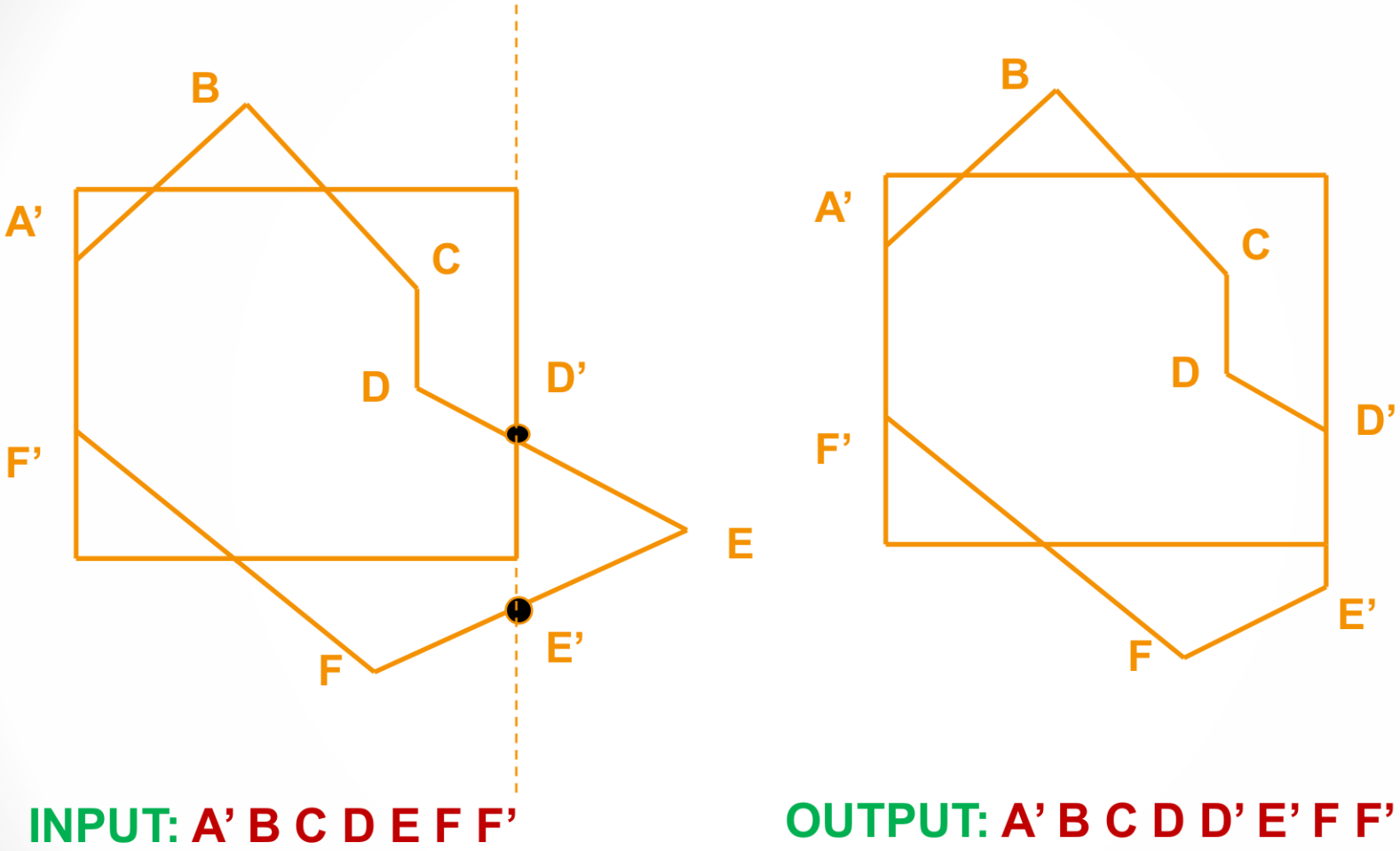
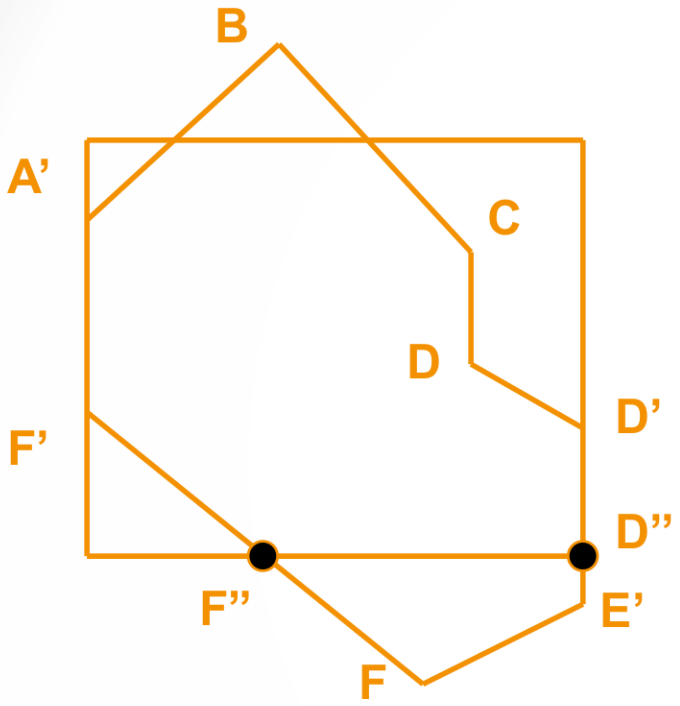
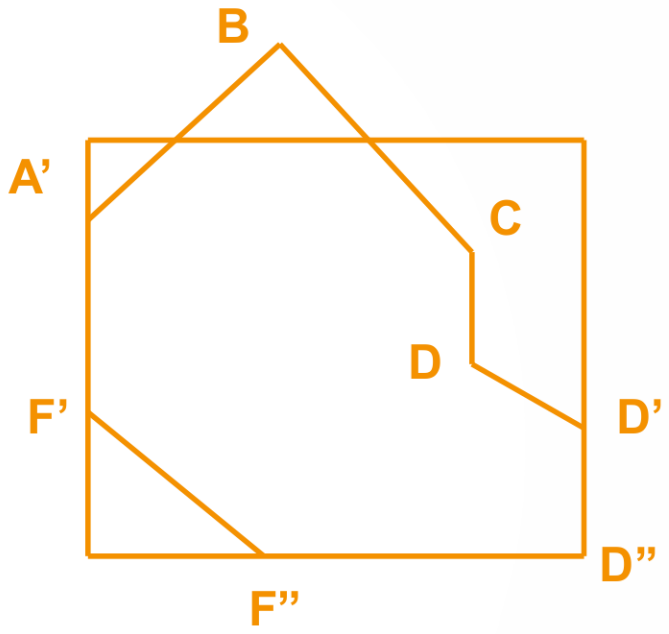


Figure: Clip against **right edge**

Sutherland-Hodgeman Polygon Clipping Algorithm



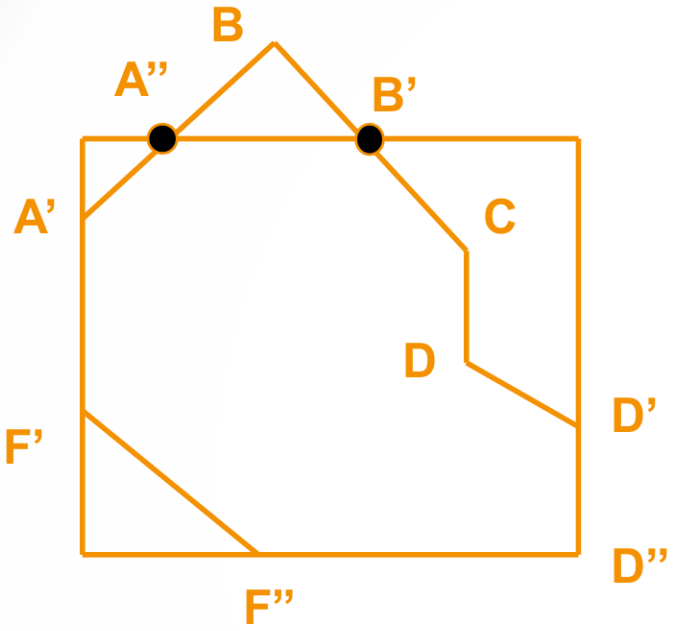


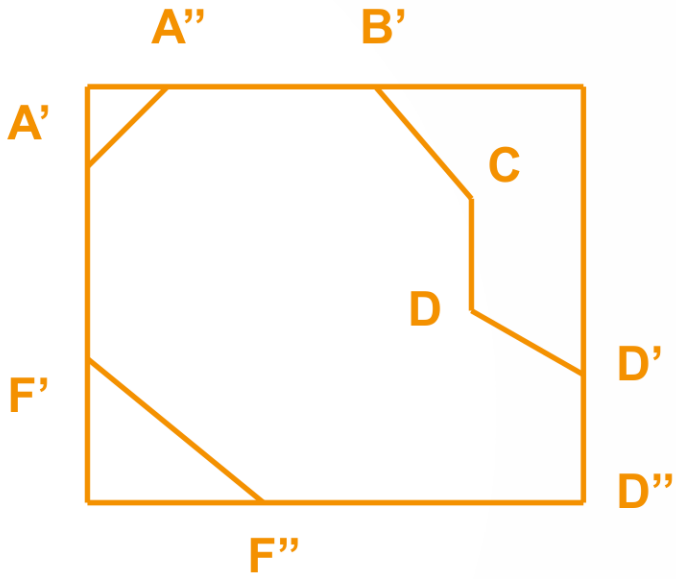
INPUT: A' B C D D' E' F F'

OUTPUT: A' B C D D' D'' F'' F'

Figure: Clip against **bottom edge**

Sutherland-Hodgeman Polygon Clipping Algorithm



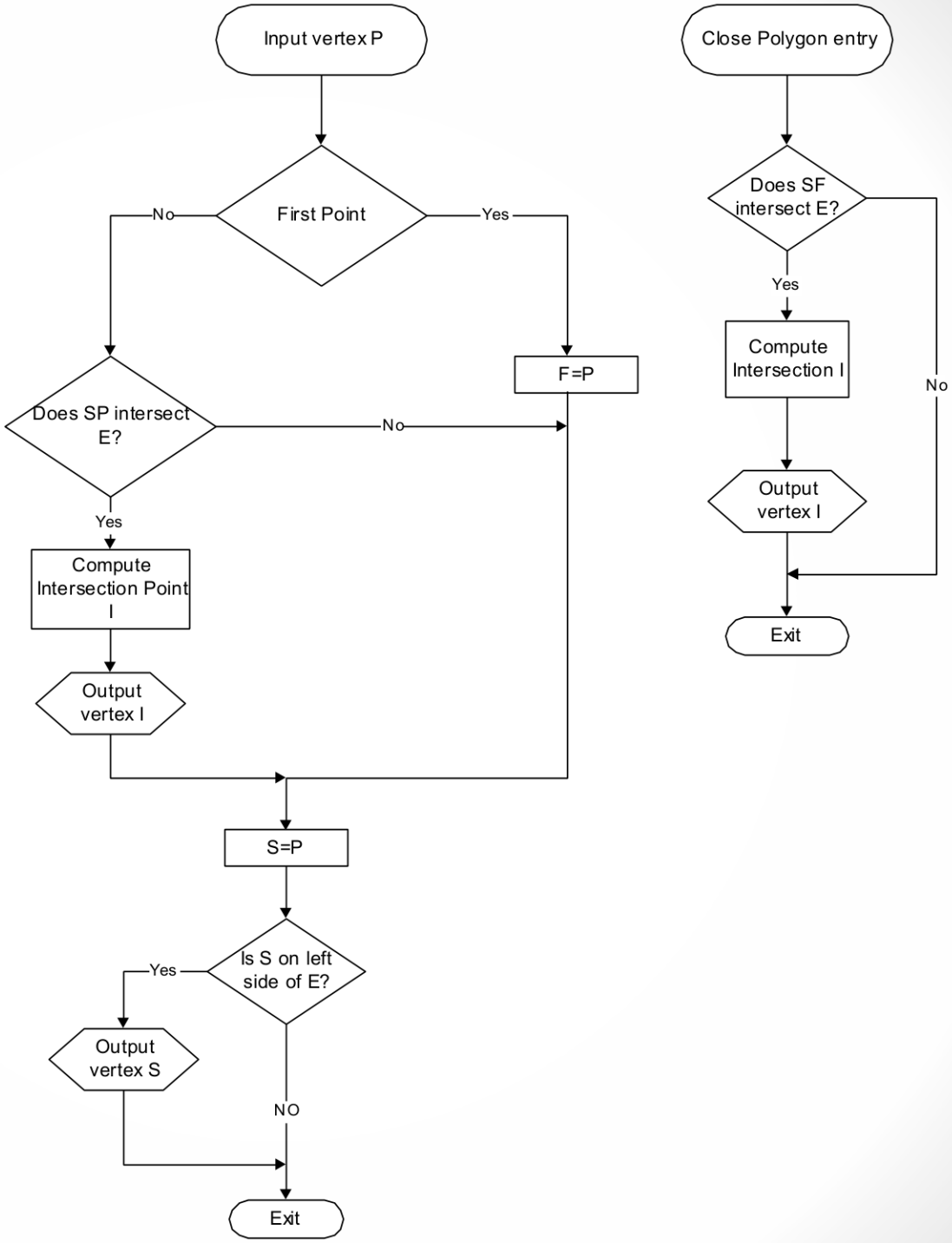


INPUT: A’ B C D D’ D” F” F’

OUTPUT: A’ A” B’ C D D’ D” F” F’

Figure: Clip against **top edge**

Sutherland-Hodgeman Polygon Clipping Algorithm

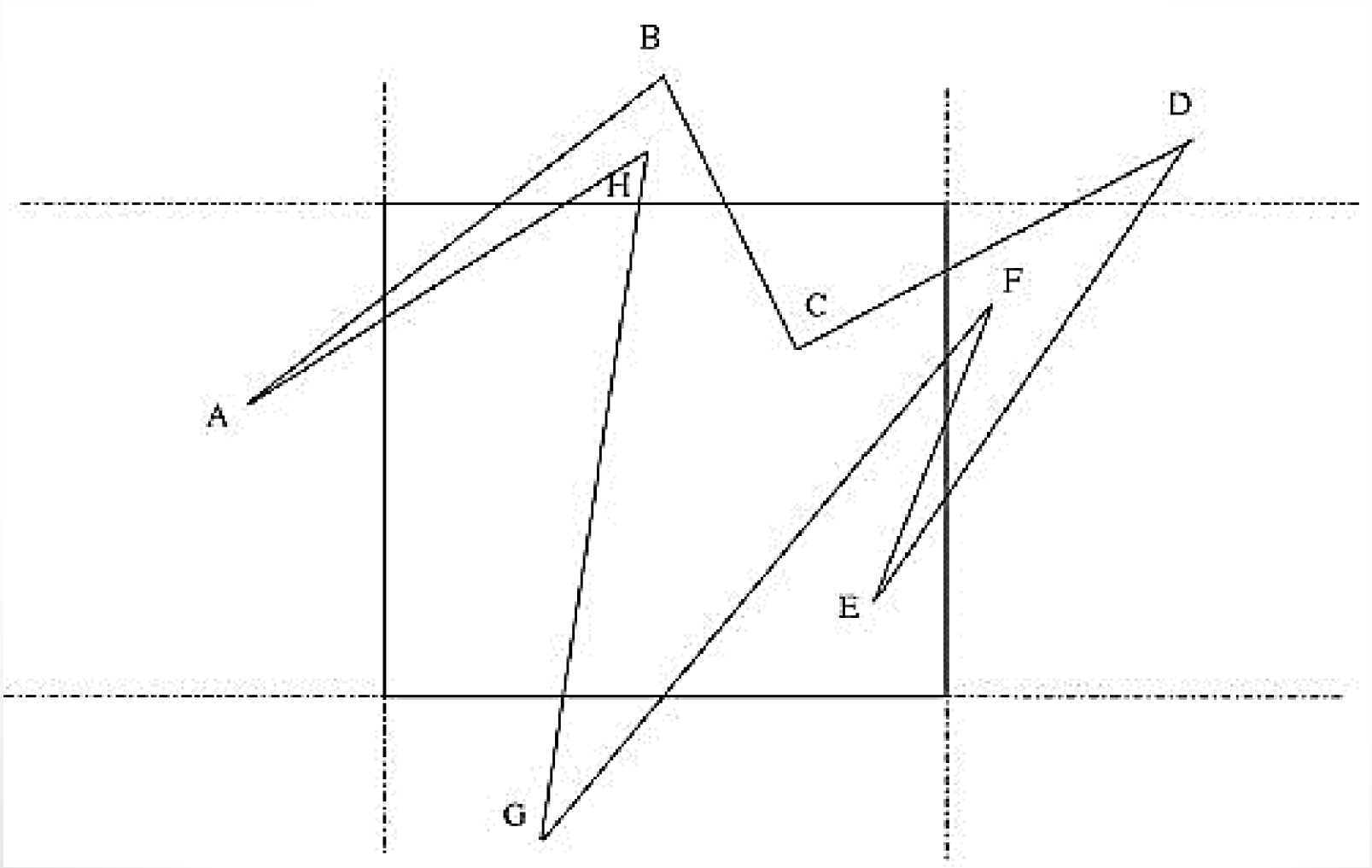


Polygon Clipping - Numerical

Go through your Class Copy

Assignment: Polygon Clipping Numerical

Show the effects of the Sutherland-Hodgeman clipping algorithm as applied to the polygon ABCDEFGH relative to the window below. Clip in the order LEFT, TOP, RIGHT, BOTTOM and give the list of vertices after each side is processed.



Board Exam Questions

1. What do you mean by window to viewport transformation? Explain 2D viewing pipeline. (2076 Batch)
2. Explain Sutherland Hodgeman polygon clipping algorithm and trace it to clip a polygon with end points P(10,30), Q(25,35), R(35,50), S(60,30), T(45,28), U(40,15) against a window whose lower left corner is at (15,25) and upper right corner is at (50,40). (2076 Batch)
3. Define window and view port. Explain 2D viewing transformation pipeline. (2075 Batch)

Board Exam Questions

4. Explain Cohen Sutherland line clipping algorithm. Let ABCD be the rectangular window with A(20,20), B(90,20), C(90,70) and D(20,70). Use Cohen Sutherland algorithm to find the region codes and clip the lines with end point P(10,30) and Q(80,80). (2075 Batch)
5. What is polygon clipping? Explain Sutherland Hodgman algorithm for polygon clipping. (2074 Batch)
6. Derive the formula for windows to viewport transformation. Given a window bordered by (0,0) at the lower left and (4,4) at the upper right. Similarly, a viewport bordered by (0,0) at the lower left and (2,2) at the upper right. If a window at position (2,4) is mapped into the viewport. What will be the position of viewport to maintain same relative placement as in window? (2074 Batch)