

MODULE III

Two Dimensional transformations-Homogeneous coordinate systems-Matrix formulation and concatenation of transformations.

Windowing concepts-two dimensional clipping.

Two Dimensional transformations

Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation.

Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

Homogenous Coordinates

To perform a sequence of transformation such as translation followed by rotation and scaling, we need to follow a sequential process:

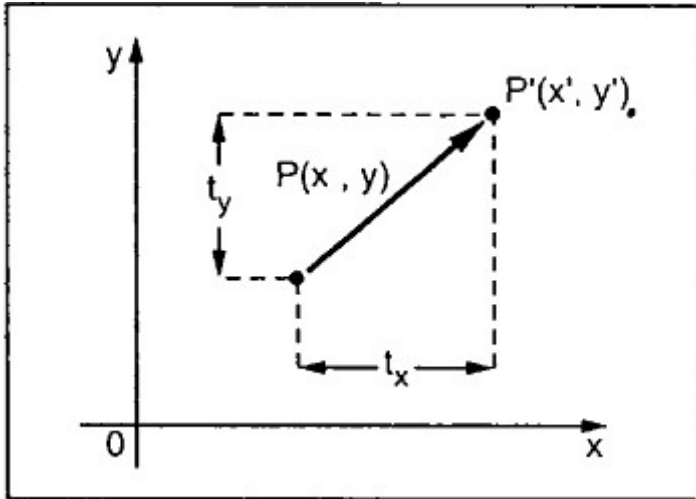
1. Translate the coordinates,
2. Rotate the translated coordinates, and then
3. Scale the rotated coordinates to complete the composite transformation.

To shorten this process, we have to use 3×3 transformation matrix instead of 2×2 transformation matrix. To convert a 2×2 matrix to 3×3 matrix, we have to add an extra dummy coordinate W.

In this way, we can represent the point by 3 numbers instead of 2 numbers, which is called **Homogenous Coordinate** system. In this system, we can represent all the transformation equations in matrix multiplication. Any Cartesian point $P(X, Y)$ can be converted to homogenous coordinates by $P' (X_h, Y_h, h)$.

Translation

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate (t_x, t_y) to the original coordinate (X, Y) to get the new coordinate (X', Y') .



From the above figure, you can write that:

$$X' = X + t_x$$

$$Y' = Y + t_y$$

The pair (t_x, t_y) is called the translation vector or shift vector. The above equations can also be represented using the column vectors.

$$P = \begin{bmatrix} X \\ Y \end{bmatrix}$$

$$P' = \begin{bmatrix} X' \\ Y' \end{bmatrix}$$

$$T = \begin{bmatrix} t_x & t_y \end{bmatrix}$$

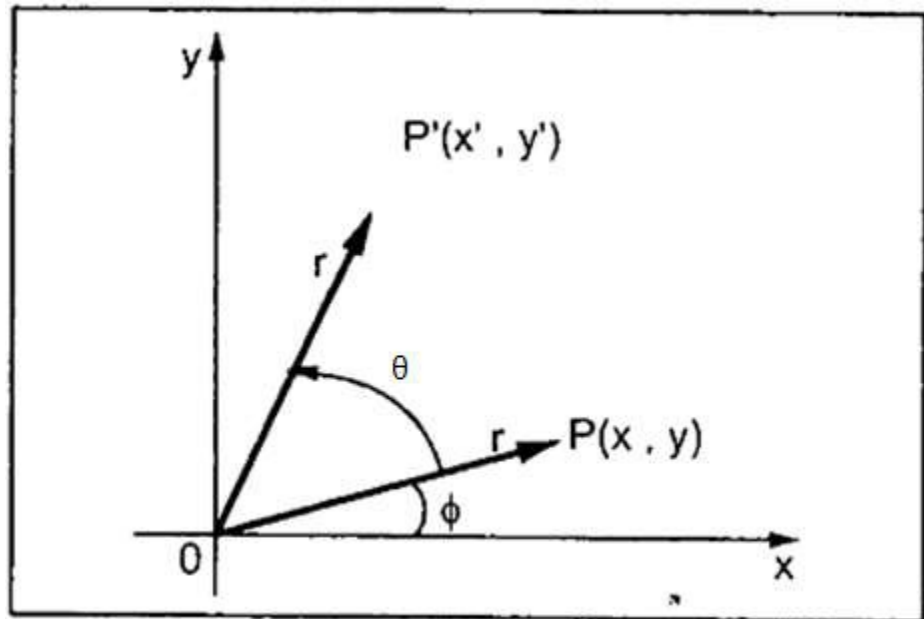
We can write it as:

$$P' = P + T$$

Rotation

In rotation, we rotate the object at particular angle θ (theta) from its origin. From the following figure, we can see that the point $P(X, Y)$ is located at angle Φ from the horizontal X coordinate with distance r from the origin.

Let us suppose you want to rotate it at the angle θ . After rotating it to a new location, you will get a new point $P' (X', Y')$.



Using standard trigonometric the original coordinate of point P(X, Y) can be represented as:

$$X = r \cos \Phi \dots\dots (1)$$

$$Y = r \sin \Phi \dots\dots (2)$$

Same way we can represent the point P' (X', Y') as:

$$X' = r \cos (\Phi + \theta) = r \cos \Phi \cos \theta - r \sin \Phi \sin \theta \dots\dots (3)$$

$$Y' = r \sin (\Phi + \theta) = r \cos \Phi \sin \theta + r \sin \Phi \cos \theta \dots\dots (4)$$

Substituting equation (1) & (2) in (3) & (4) respectively, we will get

$$X' = X \cos \theta - Y \sin \theta$$

$$Y' = X \sin \theta + Y \cos \theta$$

Representing the above equation in matrix form,

$$\begin{bmatrix} X' & Y' \end{bmatrix} = \begin{bmatrix} X & Y \end{bmatrix} \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$$

OR

$$P' = P \cdot R$$

Where R is the rotation matrix

$$R = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$$

The rotation angle can be positive and negative.

For positive rotation angle, we can use the above rotation matrix. However, for negative angle rotation, the matrix will change as shown below:

$$\begin{aligned} R &= \begin{bmatrix} \cos(-\theta) & \sin(-\theta) \\ -\sin(-\theta) & \cos(-\theta) \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (\because \cos(-\theta) = \cos \theta \text{ and } \sin(-\theta) = -\sin \theta) \end{aligned}$$

Scaling

To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

Let us assume that the original coordinates are (X, Y), the scaling factors are (S_x , S_y), and the produced coordinates are (X' , Y'). This can be mathematically represented as shown below:

$$X' = X \cdot S_x \quad \text{and} \quad Y' = Y \cdot S_y$$

The scaling factor S_x , S_y scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix} \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

OR

$$P' = P \cdot S$$

Where S is the scaling matrix. The scaling process is shown in the following figure.

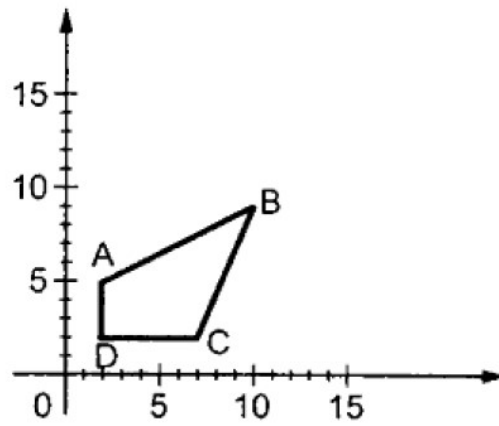


Figure: Before scaling process

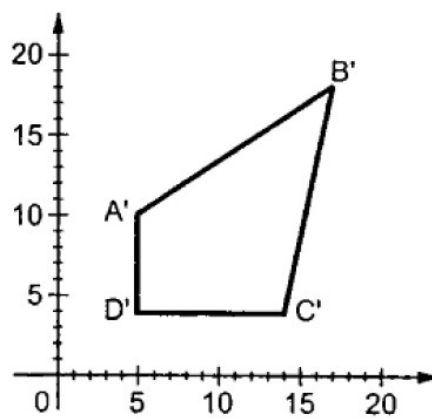


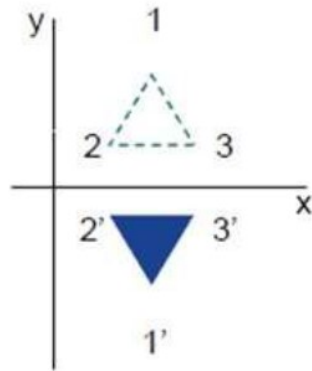
Figure: After Scaling Process

If we provide values less than 1 to the scaling factor S , then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object.

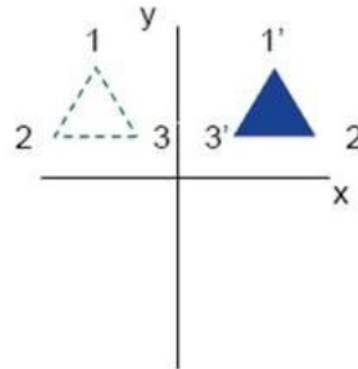
Reflection

Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180° . In reflection transformation, the size of the object does not change.

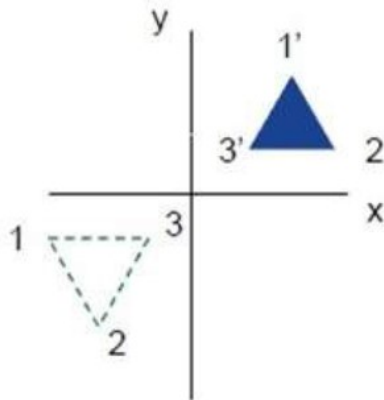
The following figures show reflections with respect to X and Y axes, and about the origin respectively.



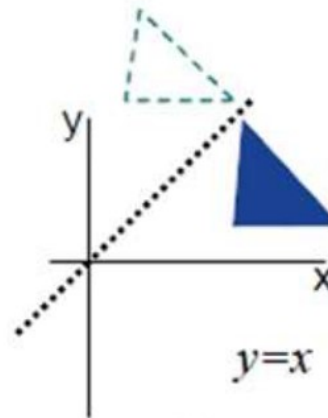
(a)



(b)



(c)



(d)

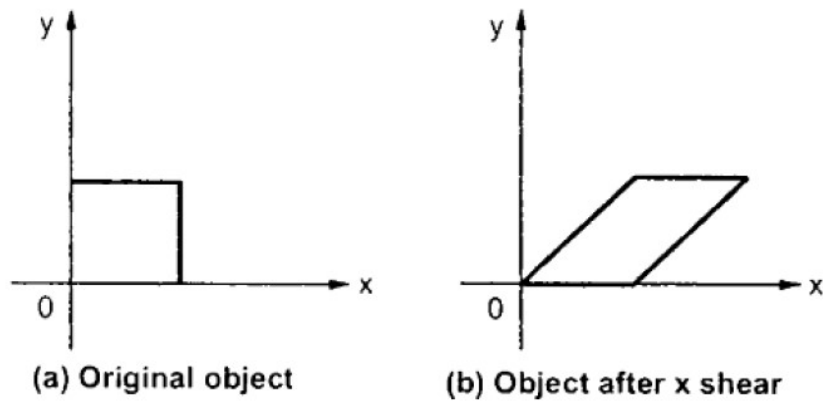
Figure: Reflection about line $y=x$

Shear

A transformation that slants the shape of an object is called the shear transformation. There are two shear transformations **X-Shear** and **Y-Shear**. One shifts X coordinates values and other shifts Y coordinate values. However, in both the cases, only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as **Skewing**.

X-Shear

The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.



The transformation matrix for X-Shear can be represented as:

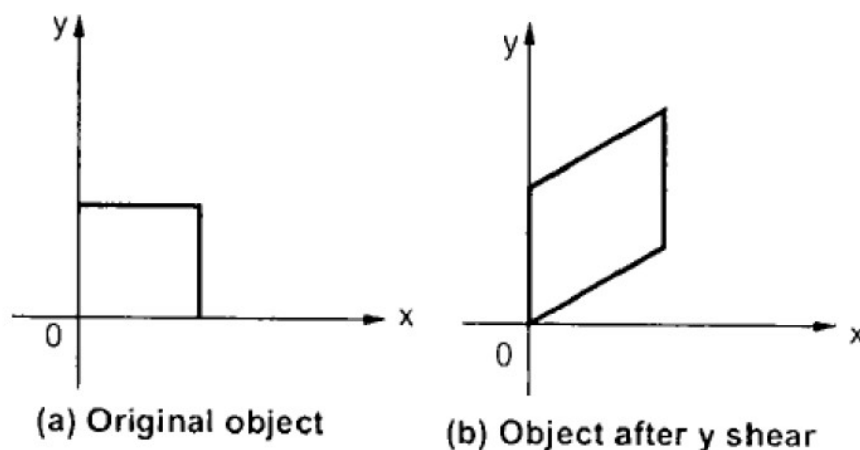
$$X_{Sh} = \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$X' = X + Sh_x \cdot Y$$

$$Y' = Y$$

Y-Shear

The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into lines which slopes up or down as shown in the following figure.



The Y-Shear can be represented in matrix form as:

$$Y_{Sh} = \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Y' = Y + Sh_y \cdot X$$

$$X' = X$$

Composite Transformation

If a transformation of the plane T1 is followed by a second plane transformation T2, then the result itself may be represented by a single transformation T which is the composition of T1 and T2 taken in that order. This is written as $T = T1 \cdot T2$.

Composite transformation can be achieved by concatenation of transformation matrices to obtain a combined transformation matrix.

A combined matrix:

$$[T][X] = [X] [T1] [T2] [T3] [T4] \dots [Tn]$$

Where [Ti] is any combination of

- Translation
- Scaling
- Shearing
- Rotation
- Reflection

The change in the order of transformation would lead to different results, as in general matrix multiplication is not cumulative, that is $[A] \cdot [B] \neq [B] \cdot [A]$ and the order of multiplication. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformation, one after another.

For example, to rotate an object about an arbitrary point (Xp, Yp), we have to carry out three steps:

1. Translate point (Xp, Yp) to the origin.
2. Rotate it about the origin.
3. Finally, translate the center of rotation back where it belonged.

Windowing Concepts

The formal mechanism for displaying views of a picture on an output device is as follows:

Typically, a graphics package allows a user to specify which part of a defined picture is to be displayed and where that part is to be placed on the display device. Any convenient Cartesian

coordinate system, referred to as the world-coordinate reference frame, can be used to define the picture. For a two-dimensional picture, a view is selected by specifying a subarea of the total picture area. A user can select a single area for display, or several areas could be selected for simultaneous display or for an animated panning sequence across a scene. The picture parts within the selected areas are then mapped onto specified areas of the device coordinates. When multiple view areas are selected, these areas can be placed in separate display locations, or some areas could be inserted into other, larger display areas. Transformations from world to device coordinates involve translation, rotation, and scaling operations, as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area.

A world-coordinate area selected for display is called a window. An area on a display device to which a window is mapped is called a viewport. The window defines *what* is to be viewed; the viewport defines *where* it is to be displayed. Often, windows and viewports are rectangles in standard position, with the rectangle edges parallel to the coordinate axes. Other window or viewport geometries, such as general polygon shapes and circles, are used in some applications, but these shapes take longer to process. In general, the mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation.

Sometimes the two-dimensional viewing transformation is simply referred to as the *window-to-viewport transformation* or the *windowing transformation*.

But, in general, viewing involves more than just the transformation from the window to the viewport. Following Figure illustrates the mapping of a picture section that falls within a rectangular window onto a designated angular viewport.

In computer graphics terminology, the term *window* originally referred to an area of a picture that is selected for viewing.

Clipping

Usually only a specified region of a picture needs to be displayed. This region is called the clip window. An algorithm which identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a clipping algorithm.

Different clipping algorithms are

- Point clipping

- Line clipping
- Area clipping(polygon clipping)
- Curve clipping
- Text clipping

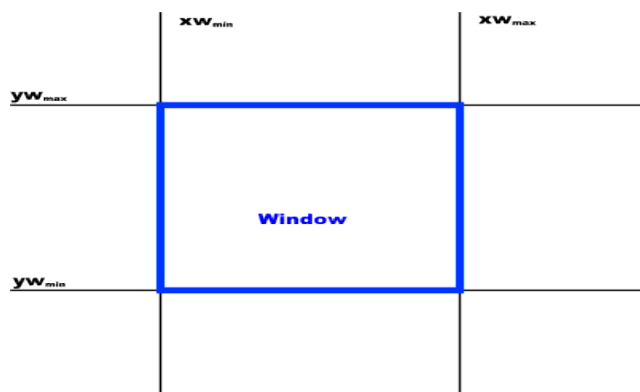
Point clipping

A point (x, y) lies inside the rectangular clip window (xmin, ymin; xmax, ymax) if and only if the following inequalities hold:

$$x_{min} \leq x \leq x_{max}$$

$$y_{min} \leq y \leq y_{max}$$

Where the edges of the clip window (xw_{min}, xw_{max}, yw_{min}, yw_{max}) can be either the world-coordinate window boundaries or viewport boundaries. If any one of these four inequalities is not satisfied, the point is clipped (not saved for display).



Line Clipping

Lines intersecting a rectangular clip region are always clipped to a single line segment. Figure shows examples of clipped lines.

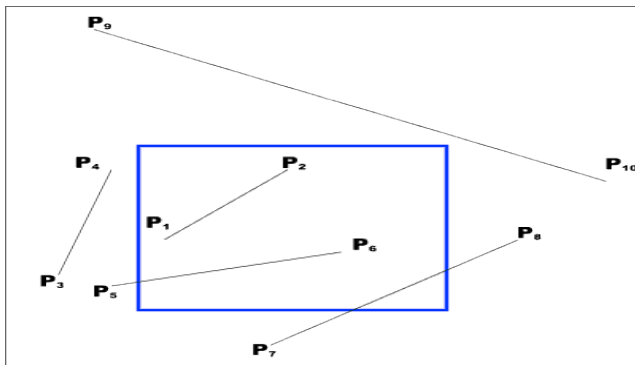
Clipping end points

Before discussing clipping lines, consider clipping of individual points. If the x coordinate boundaries of the clip rectangle are at x_{min} and x_{max} and the y coordinate boundaries are at y_{min} and y_{max}, the following conditions must be satisfied for a point at (x, y) to be inside the clip rectangle.

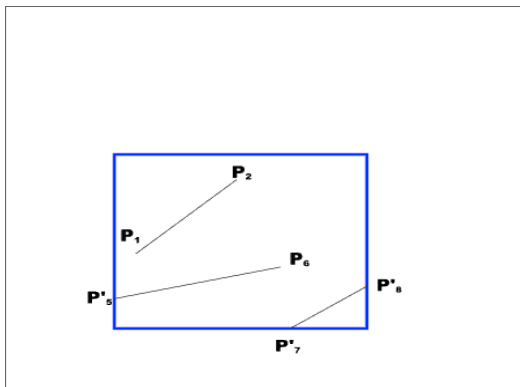
$$x_{min} \leq x \leq x_{max}$$

$$y_{min} \leq y \leq y_{max}$$

If any of the above conditions do not hold, the point is outside the clip rectangle.



Before Clipping



After Clipping

Cohen-Sutherland Line Clipping

The algorithm has 2 parts. The first part determines whether **the line lies entirely on the screen, then it will be accepted and if not whether the line can be rejected** if lying entirely **off the screen**. If it satisfies none of these 2 tests, then the line is divided in to 2 parts and these 2 tests are applied to each part. The algorithm depends on the fact that every line is entirely on the screen. Or can be divided so that one part can be rejected.

We are extending the edges of the screen so that they divide the space occupied by the unclipped picture in to 9 regions. **Each of these regions has a 4 bit code called region code.**

Clip rectangle extended into a plane divided into 9 regions .Each region is defined by a unique 4-bit string

- bit1 = 1: left of left edge ($X < X_{min}$)
- 2nd bit = 1: right of right edge ($X > X_{max}$)
- 3rd bit = 1: below bottom edge ($Y < Y_{min}$)
- fourth bit = 1: above top edge ($Y > Y_{max}$)

The frame buffer itself, in the center, has code 0000.

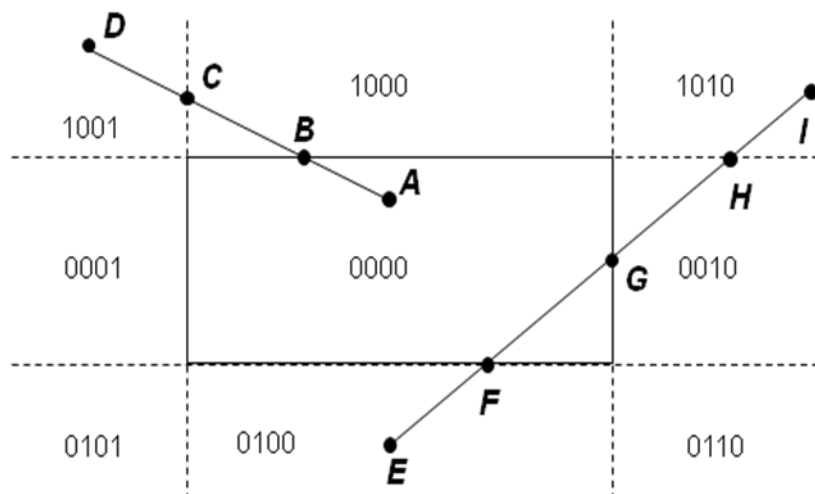
```

1001 | 1000 | 1010
-----+-----+-----
0001 | 0000 | 0010
-----+-----+-----
0101 | 0100 | 0110

```

For each line segment:

1. each end point is given the 4-bit code of its region
2. repeat until acceptance or rejection
 - if both codes are 0000 -> trivial acceptance means the particular line segment is completely inside the clip window. Save the line
 - if bitwise logical AND of codes is not 0000 -> trivial rejection , means the particular line segment is completely outside the clip window. Reject the line
 - Otherwise calculate intermediate points and repeat above steps



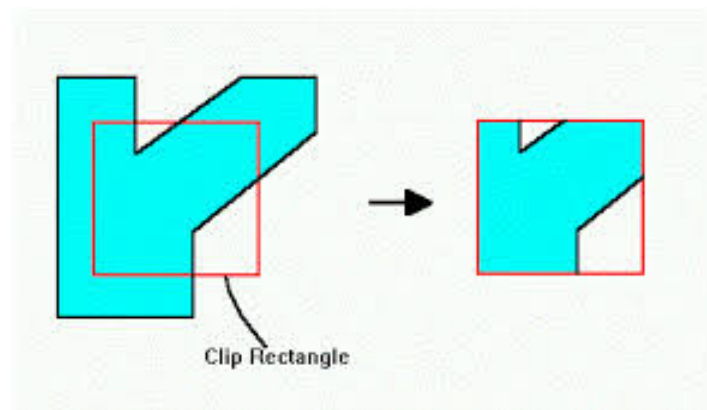
Region code bit values can be determined with the following two steps:

- 1) Calculate the differences between end point coordinates and clipping boundaries
- 2) Use the resultant sign bit of each difference, calculation to set the corresponding value in the region code.

Bit 1 is the sign bit of $x - x_{wmin}$, bit 2 is the sign bit of $x_{wmax} - x$, bit 3 is the sign bit of $y - y_{wmin}$, and bit 4 is the sign bit of $y_{wmax} - y$.

Polygon Clipping (Sutherland Hodgman Algorithm)

Polygon is an object having closed outlines bounded by straight edges. An example of polygon clipping is shown below.



A polygon can also be clipped by specifying the clipping window. Sutherland Hodgman polygon clipping algorithm is used for polygon clipping. In this algorithm, all the vertices of the polygon are clipped against each edge of the clipping window.

Beginning with the initial set of polygon vertices, we could first clip the polygon against the **left rectangle boundary** to produce a new sequence of vertices. The new set of vertices could then be successively passed to a **right boundary clipper**, **bottom boundary clipper** and a **top boundary clipper**. There are four possible cases when processing vertices in sequence around the perimeter of a polygon.

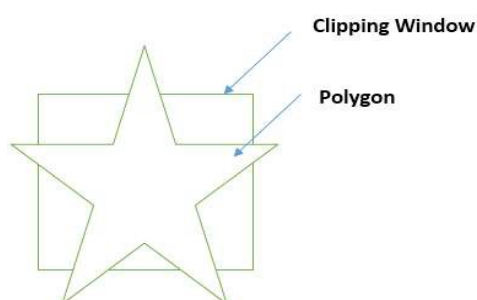


Figure: Polygon before clipping

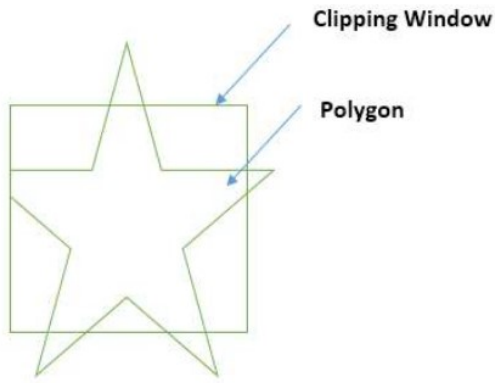


Figure: Clipping Left Edge

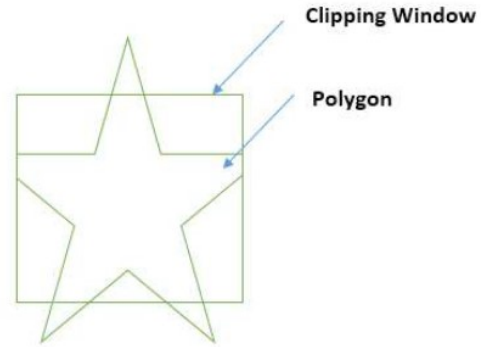


Figure: Clipping Right Edge

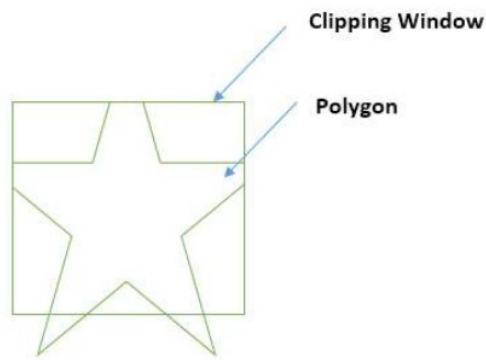


Figure: Clipping Top Edge

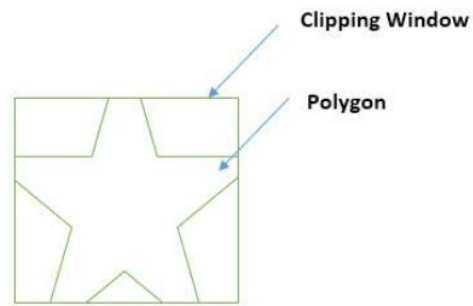
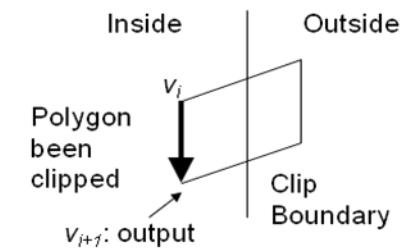


Figure: Clipping Bottom Edge

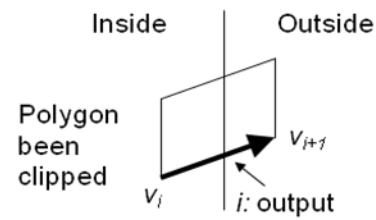
As each pair of adjacent polygon vertices is passed to a window boundary clipper, we make the following tests:

- 1) If the first vertex is outside the window boundary and the second vertex is inside, both the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list.
- 2) If both input vertices are inside the window boundary, only the second vertex is added to the output vertex list.
- 3) If the first vertex is inside the window boundary and the second vertex is outside, only the edge intersection with the window boundary is added to the output vertex list.
- 4) If both input vertices are outside the window boundary, nothing is added to the output list.

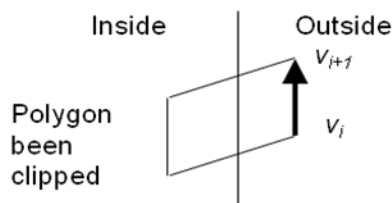
Sutherland-Hodgeman Algorithm(cont.)



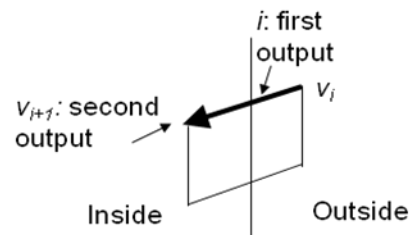
Case 1



Case 2



Case 3
(no output)



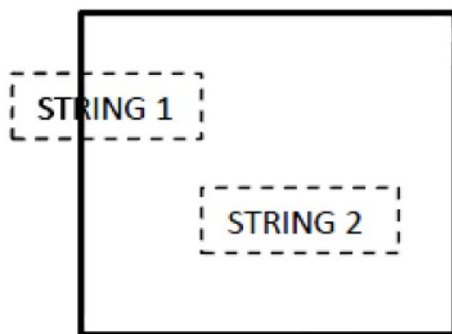
Case 4

Text Clipping

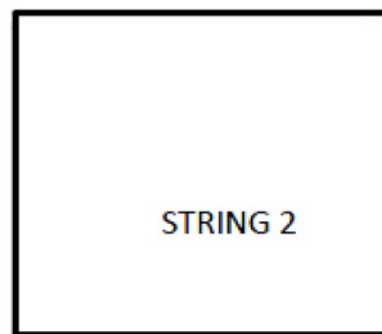
Various techniques are used to provide text clipping in a computer graphics. It depends on the methods used to generate characters and the requirements of a particular application. There are three methods for text clipping which are listed below:

1. All or none string clipping
2. All or none character clipping
3. Text clipping

The following figure shows all or none string clipping:



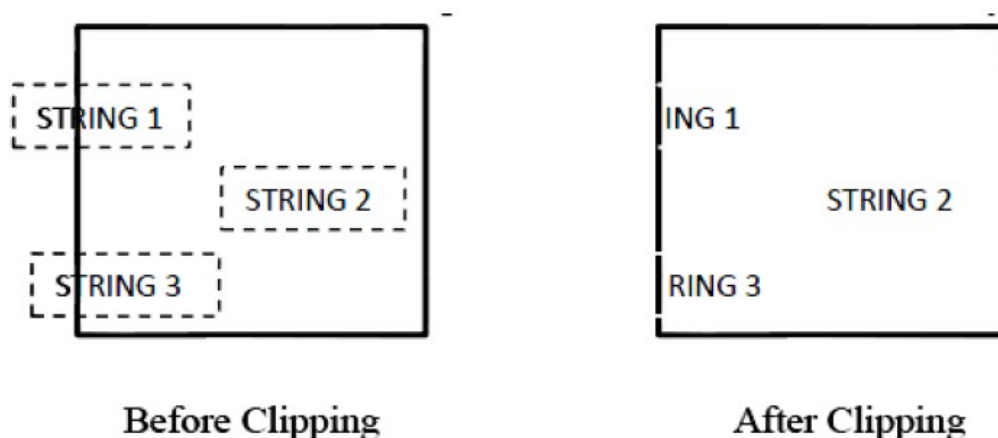
Before Clipping



After Clipping

In all or none string clipping method, either we keep the entire string or we reject entire string based on the clipping window. As shown in the above figure, STRING2 is entirely inside the clipping window so we keep it and STRING1 being only partially inside the window, we reject.

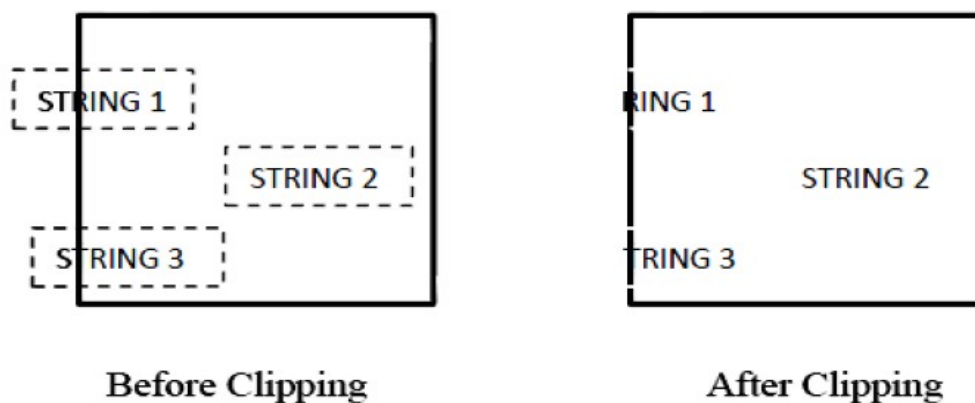
The following figure shows all or none character clipping:



This clipping method is based on characters rather than entire string. In this method if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then:

- You reject only the portion of the string being outside
- If the character is on the boundary of the clipping window, then we discard that entire character and keep the rest string.

The following figure shows the text clipping:



This clipping method is based on characters rather than the entire string. In this method if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then

- You reject only the portion of string being outside.
- If the character is on the boundary of the clipping window, then we discard only that portion of character that is outside of the clipping window.