SRM institute of Science and Technology
Chennai

21CSE255T - COMPUTER GRAPHICS AND ANIMATION

UNIT - II

2D TRANSFORMATIONS

Prepared by

Mr. S.Prabu

Mrs.G.Bhargavi

Assistant Professor(CTech)

SRMIST-KTR

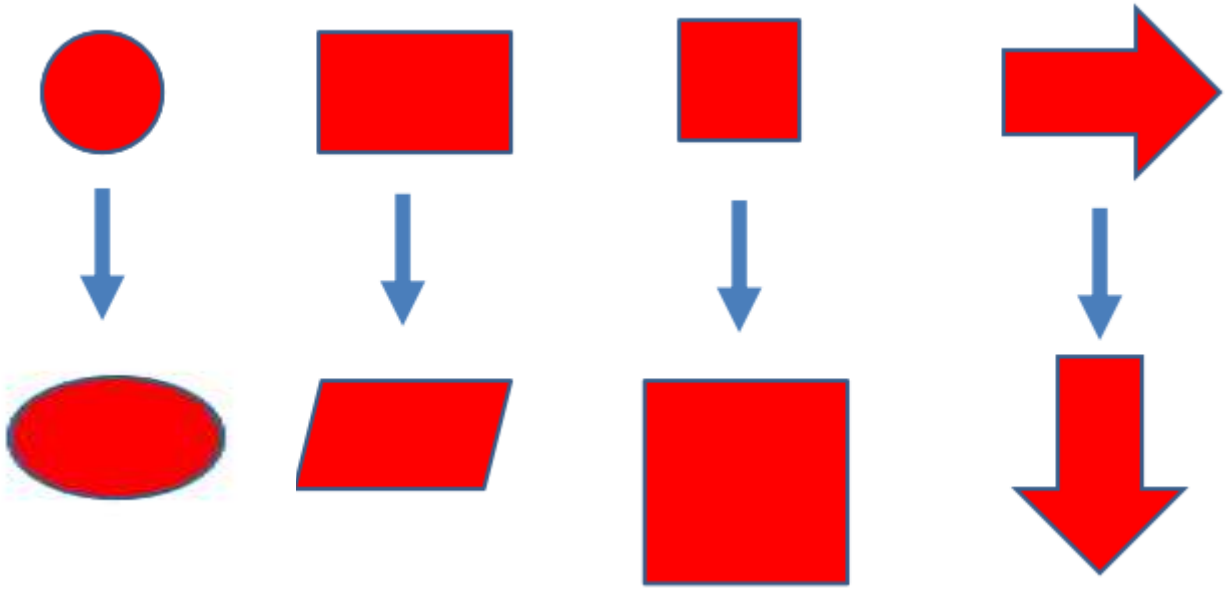
TOPICS

- Introduction to 2D Transformation
- Basic Transformation
- Matrix representation
- Composite Transformation
- Shear –Reflection
- 2D viewing
- Viewing pipeline
- Viewing functions.

- Clipping operations
- Point clipping
- Line Clipping
- Cohen-Sutherland
- Liang-Barsky
- Nicholl-Lee-Nicholl
- Nonrectangular Line Clipping
- Polygon Clipping
- Other Clipping methods

2D TRANSFORMATION

- **Introduction:**
  Changing Position, shape, size, or orientation of an object is known as transformation



2D TRANSFORMATION

- **Definition:**

  "Transformations are the operations applied to geometrical description of an object to change its position, orientation, or size are called geometric transformations".

- Transformation can be classified as
  - Basic Transformation
  - Other Transformation

2D TRANSFORMATION

The Basic Transformations:

1. Translation
2. Scaling
3. Rotation

Other Transformations:

4. Reflection
5. Shearing

2D TRANSFORMATION

- Translation

  T(*tx*, *ty*) : Translation distances

- Scale

  S(*Sx*,*Sy*) : Scale factors

- Rotation

  R(Ø) : Rotation angle

These are known as basic transformation because with combination of them we can obtain any transformatio

TRANSLATION

- Translation is a process of changing the position of an object in a straight-line path from one co- ordinate location to another.
- We can translate a 2Dpoint by adding translation distances, tx and ty.
- Suppose the original position is (x ,y)

  then new position is (x', y').
- Here x'=x + tx and y'=y + ty.
- The translation distance point (tx, ty) is called translation vector or shift vector.

TRANSLATION

- Translation equation can be expressed as single matrix equation by using column vectors to represent the coordinate position and the translation vector as

$$P' = P + T$$

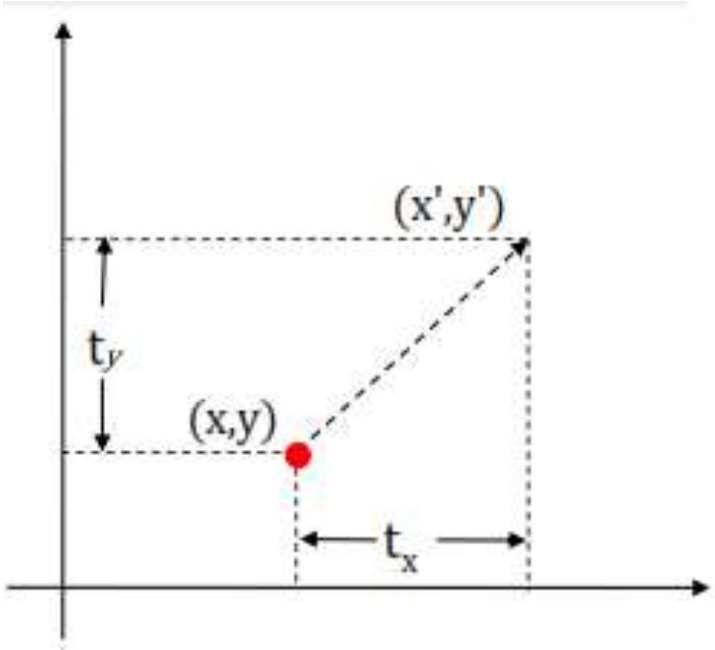
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

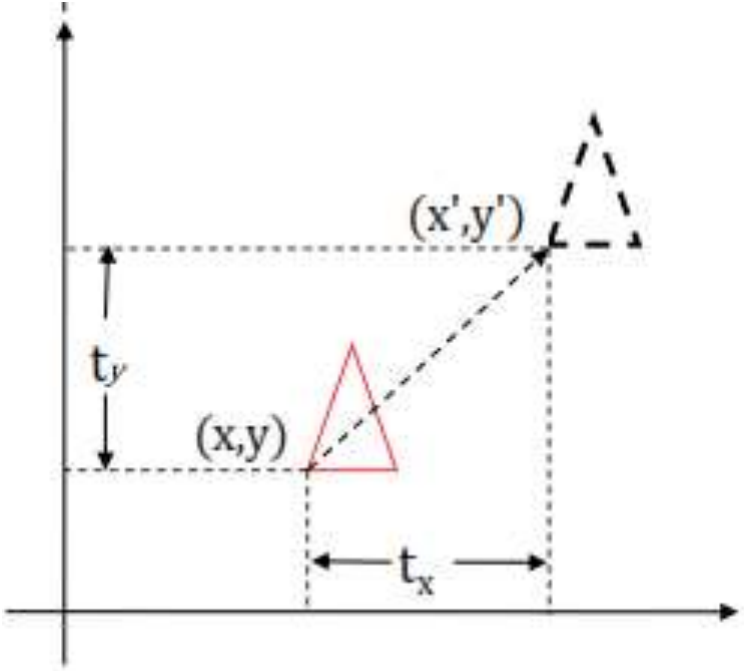
- We can also represent it in row vector form as:

$$P' = P + T$$

$$[x' \quad y'] = [x \quad y] + [t_x \quad t_y]$$

TRANSLATION





TRANSLATION

Ex. Translate a polygon with co-ordinates A(2,5) B(7,10) and C(10,2) by 3 units in X direction and 4 units in Y direction.

$$A' = A + T$$

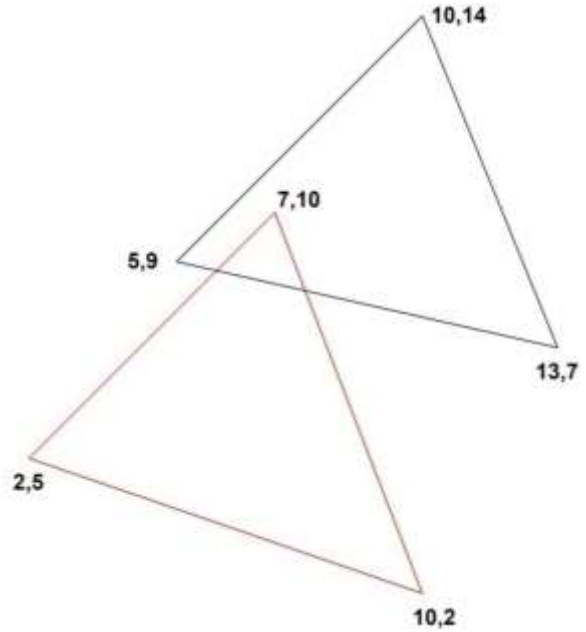
$$= \begin{bmatrix} 2 \\ 5 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 9 \end{bmatrix}$$

$$B' = B + T$$

$$= \begin{bmatrix} 7 \\ 10 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 10 \\ 14 \end{bmatrix}$$

$$C' = C + T$$

$$= \begin{bmatrix} 10 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 13 \\ 7 \end{bmatrix}$$

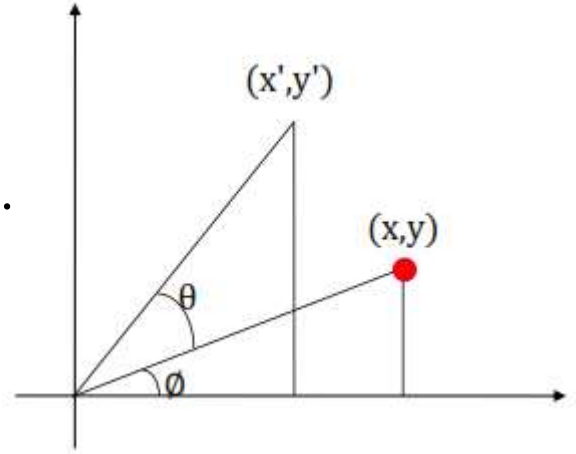


ROTATION

- Rotation is applied to an object by repositioning it along a circular path on xy plane.
- To generate a rotation, specify a rotation angle θ and the position (x_r,y_r) of the rotation point (pivot point)

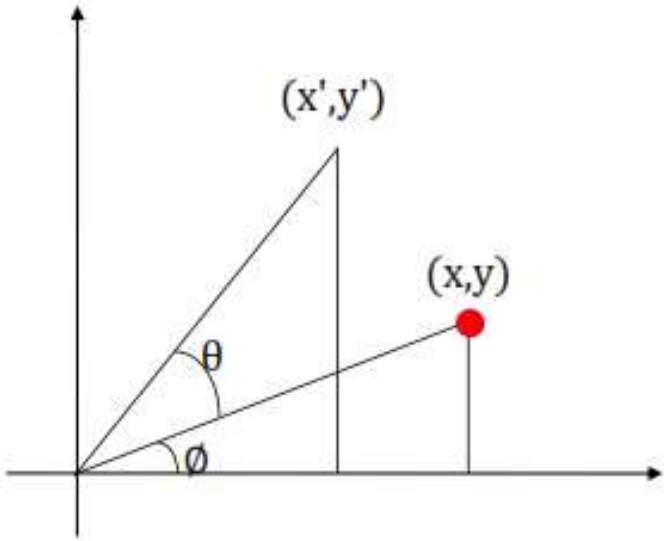
+ve rotation angle defines counter clockwise rotation.

-ve rotation angle defines clockwise rotation.



ROTATION

- We first find the equation of rotation when pivot point is at coordinate origin(0,0).
- From figure we can write.



$$x = r\cos\emptyset$$
$$y = r\sin\emptyset$$

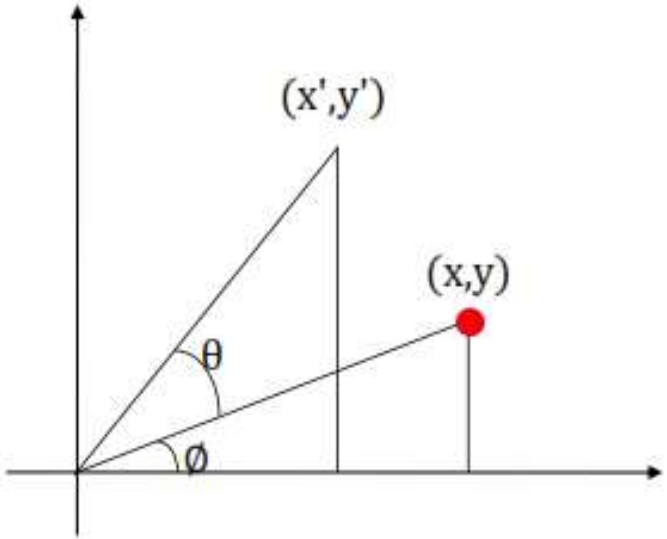
&

$$x' = r\cos(\theta+\emptyset) = r\cos\emptyset\cos\theta - r\sin\emptyset\sin\theta$$
$$y' = r\sin(\emptyset+\theta) = r\cos\emptyset\sin\theta + r\sin\emptyset\cos\theta$$

ROTATION

- Now replace $r \cos \emptyset$ with x and $r \sin \emptyset$ with y in above equation.

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

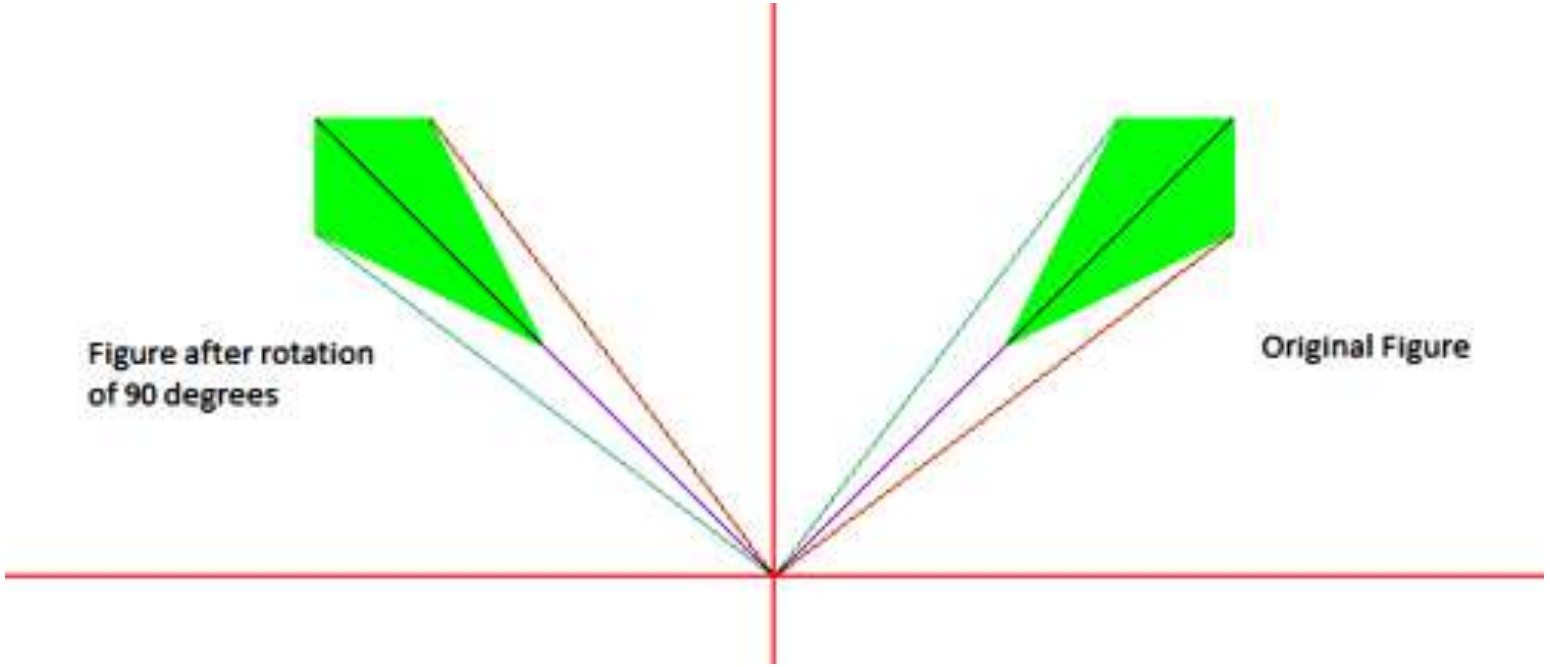


column vector matrix equation are,

$$P' = R \cdot P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

ROTATION



SCALING

- Transformation that used to alter the size of an object is known as scaling.
- This operation is carried out by multiplying coordinate value (x, y) with scale factors (s_x, s) respectively.
- Equation for scaling is given by

$$x' = x \cdot s_x \;\&\; y' = y \cdot s_y$$

- These equation can be represented in column vector matrix equation as

$$P' = S \cdot P$$

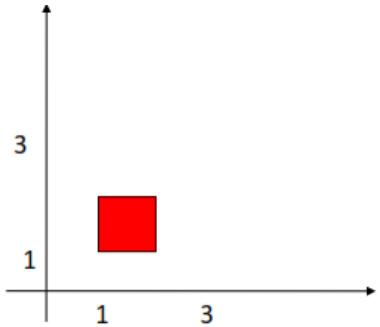
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

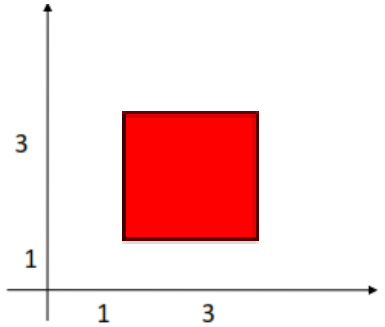
SCALING

- Normal scaling will scale as well as reposition the object.
- Scale factors less than 1 reduce the size and move object closer to origin.
- Scale factors greater than 1 enlarge the size of object and move object away from origin.
- Example scale square with opposite corner coordinate point are (1, 1) and (2, 2) by scale factor (3, 3)

$$P' = S \cdot P$$

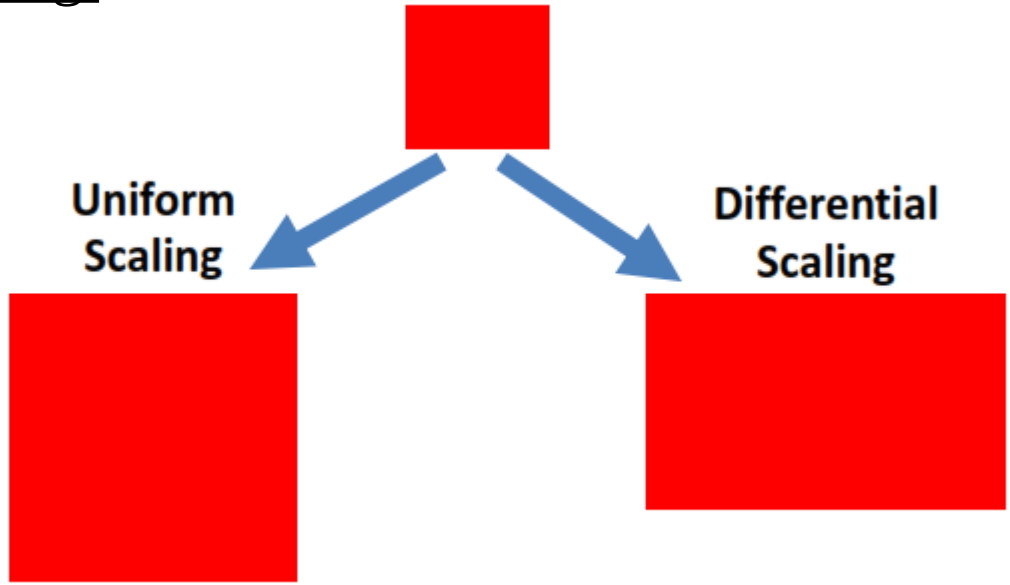
$$\begin{bmatrix} x_1' & x_2' \\ y_1' & y_2' \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 3 & 6 \end{bmatrix}$$





SCALING

- Same values of s_x and s_y will produce Uniform Scaling.
- Different values of s_x and s will produce Differential (Non Uniform) Scaling.



MATRIX REPRESENTATION & HOMOGENEOUS CO-ORDINATES

- Many graphics application involves sequence of geometric transformations.
- For efficient processing we will reformulate transformation sequences.
- We have matrix representation of basic transformation and we can express it in the general matrix form as

$$P' = M_1 \cdot P + M_2$$

where

P and P' are initial and final point position,

M_1 contains rotation and scaling terms and

M_2 contains translational terms associated with pivot point, fixed point and reposition.

HOMOGENEOUS MATRIX FOR TRANSLATION

- Let's see each representation with $\boldsymbol{h} = \mathbf{1}$

$$P' = T_{(t_x,t_y)} \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Inverse of translation matrix is obtain by putting $-\boldsymbol{t_x}$ & $-$ instead of $\boldsymbol{t_x}$ & $\boldsymbol{t_y}$

HOMOGENEOUS MATRIX FOR ROTATION

$$P' = R_{(\theta)} \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Inverse of rotation matrix is obtained by replacing $\boldsymbol{\theta}$ by $-\boldsymbol{\theta}$.

HOMOGENEOUS MATRIX FOR SCALING

$$P' = S_{(s_x, s_y)} \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Inverse of scaling matrix is obtained by replacing s_x & $_y$ by $\frac{1}{s_x}$ & $\frac{1}{s_y}$ respectively.

COMPOSITE TRANFORMATION

- In practice we need to apply more than one transformations to get desired result.
- It is time consuming to apply transformation one by one on each point of object.
- So first we multiply all matrix of required transformations which is known as composite transformation matrix.
- Than we use composite transformation matrix to transform object.
- For column matrix representation, we form composite transformations by multiplying matrices from right to left.

COMPOSITE TRANSLATION

- Two successive translations are performed as:

$$P' = T(t_{x2}, t_{y2}) \cdot \{T(t_{x1}, t_{y1}) \cdot P\}$$

$$P' = \{T(t_{x2}, t_{y2}) \cdot T(t_{x1}, t_{y1})\} \cdot P$$

$$P' = \begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

COMPOSITE TRANSLATION

$$P' = \begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = T(t_{x1} + t_{x2}, t_{y1} + t_{y2}) \cdot P$$

- Here P' and P are column vector of final and initial point coordinate respectively.
- This concept can be extended for any number of successive translations.

COMPOSITE ROTATION

- Two successive Rotations are performed as

$$P' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\}$$

$$P' = \{R(\theta_2) \cdot R(\theta_1)\} \cdot P$$

$$P' = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} \cos\theta_2\cos\theta_1 - \sin\theta_2\sin\theta_1 & -\sin\theta_1\cos\theta_2 - \sin\theta_2\cos\theta_1 & 0 \\ \sin\theta_1\cos\theta_2 + \sin\theta_2\cos\theta_1 & \cos\theta_2\cos\theta_1 - \sin\theta_2\sin\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

COMPOSITE ROTATION

$$P' = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = R(\theta_1+\theta_2) \cdot P$$

- Here P' and P are column vector of final and initial point coordinate respectively.
- This concept can be extended for any number of successive rotations

COMPOSITE SCALING

- Two successive Scalings are performed as

$$P' = S(s_{x2}, s_{y2}) \cdot \{S(s_{x1}, s_{y1}) \cdot P\}$$

$$P' = \{S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})\} \cdot P$$

$$P' = \begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

COMPOSITE SCALING

$$P' = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = S(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2}) \cdot P$$

- Here P' and P are column vector of final and initial point coordinate respectively.
- This concept can be extended for any number of successive scaling

OTHER TRANSFORMATION

- Some package provides few additional transformations which are useful in certain applications.
- Two such transformations are:
    1. Reflection
    2. Shear

REFLECTION

- A reflection is a transformation that produces a mirror image of an object.
- The mirror image for a two –dimensional reflection is generated relative to an axis of reflection by rotating the object 180o about the reflection axis.
- Reflection gives image based on position of axis of reflection. Transformation matrix for few positions are discussed here

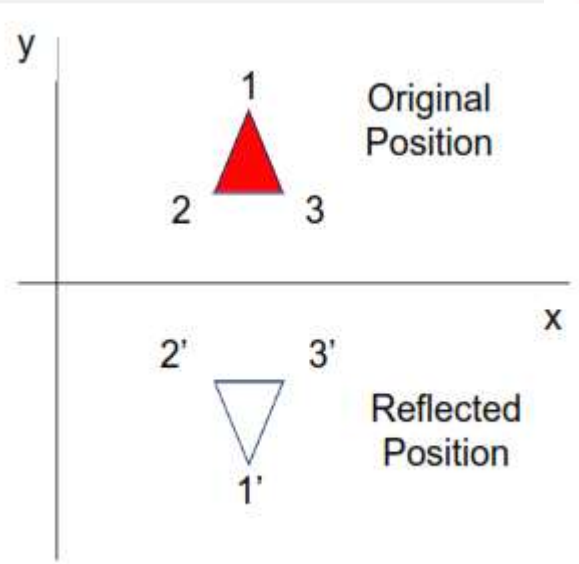
REFLECTION About an X axis

- For reflection about the line

  $\boldsymbol{y = 0}$, ***the x axis***

- Transformation matrix:

$$Ref_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



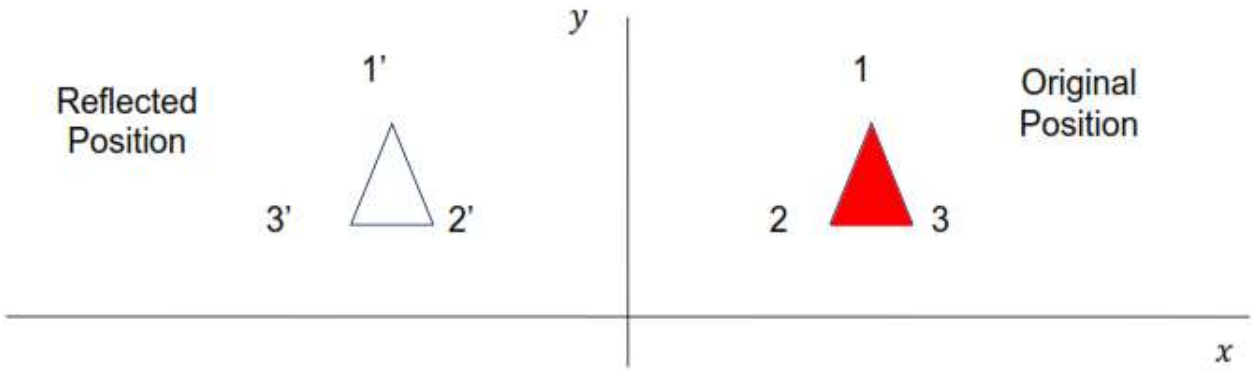
- This transformation keeps x values are same, but flips (Change the sign) y values of coordinate positions.

REFLECTION About an Y axis

- For reflection about the line
  $\mathbf{x = 0}$, ***the*** y ***axis***
- Transformation matrix:

$$Ref_y = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

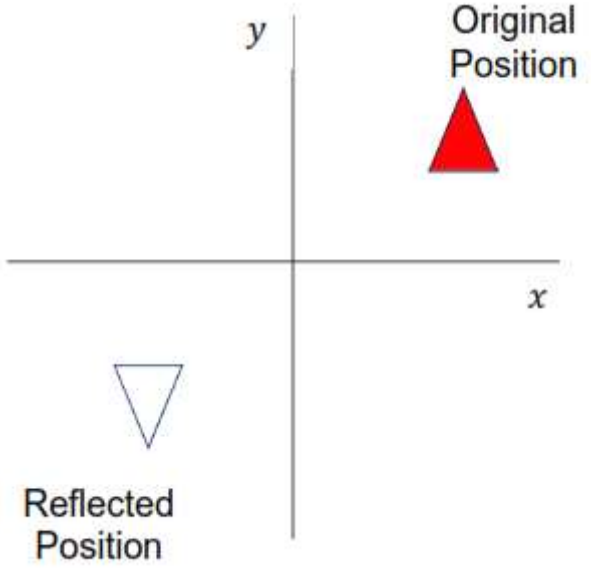
- This transformation keeps y values are same, but flips (Change the sign) x values of coordinate positions.



REFLECTION About Origin

- For reflection about the ***Origin***.
- Transformation matrix:

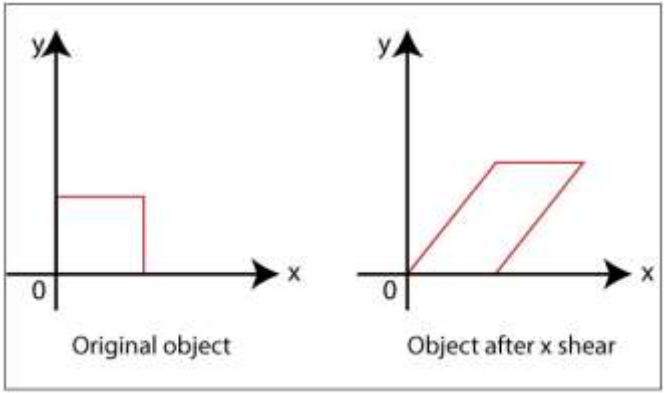
$$Ref_{origin} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



- This transformation flips (Change the sign) x and y both values of coordinate positions.

SHEAR TRANSFORMATION

- A transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called shear.



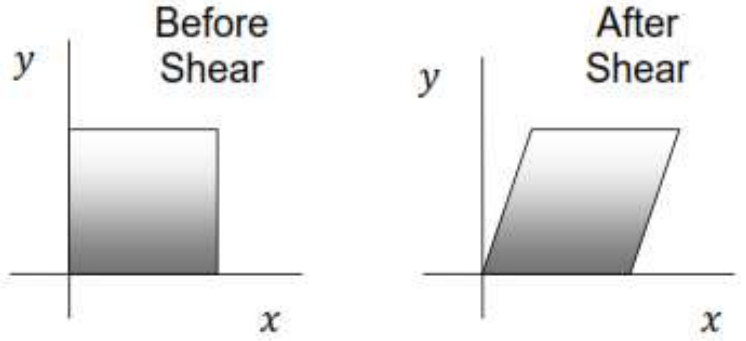
- Two common shearing transformations are those that shift coordinate x values and those that shift y values

SHEAR in X Direction

- Shear relative to $x - axis$ that is $y = 0$ line can be produced by following equation: $\boldsymbol{x' = x + sh_x \cdot y}$, $\boldsymbol{y' = y}$
- Transformation matrix for that is

$$\boldsymbol{Shear_{x-direction} = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}$$

- Here $\boldsymbol{sh_x}$ is shear parameter.
- We can assign any real value to $\boldsymbol{sh_x}$.

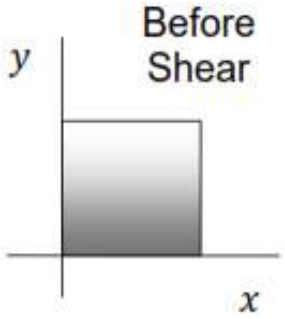


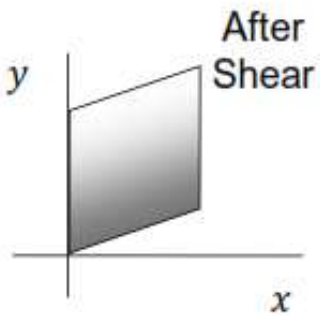
SHEAR in Y Direction

- Shear relative to $y - axis$ that is $x = 0$ line can be produced by following equation: $\boldsymbol{x' = x}$, $\boldsymbol{y' = y + sh_y \cdot x}$
- Transformation matrix for that is

$$\boldsymbol{Shear_{y-direction}} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{sh_y} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

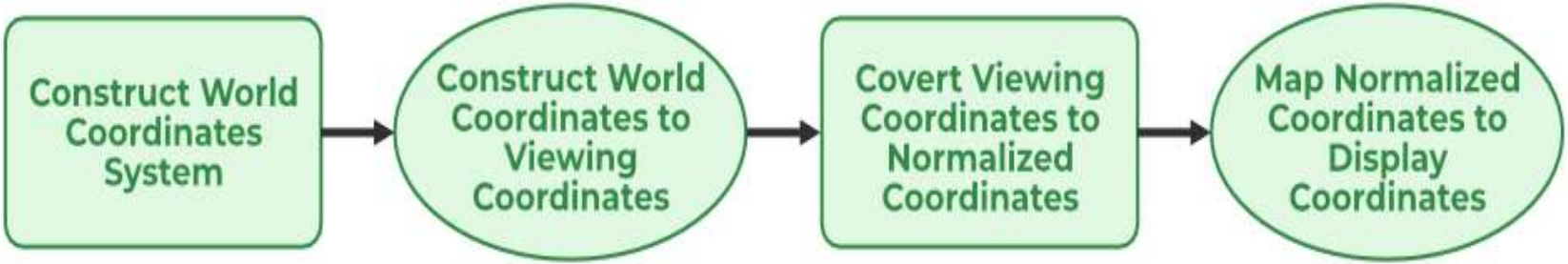
- Here $\boldsymbol{sh_y}$ is shear parameter.
- We can assign any real value to $\boldsymbol{sh_y}$.





2D VIEWING

2D viewing is a technique to map the world coordinates system (actual coordinates of the object in the real world) to device coordinates (actual coordinates of the object on the output screen like a monitor).

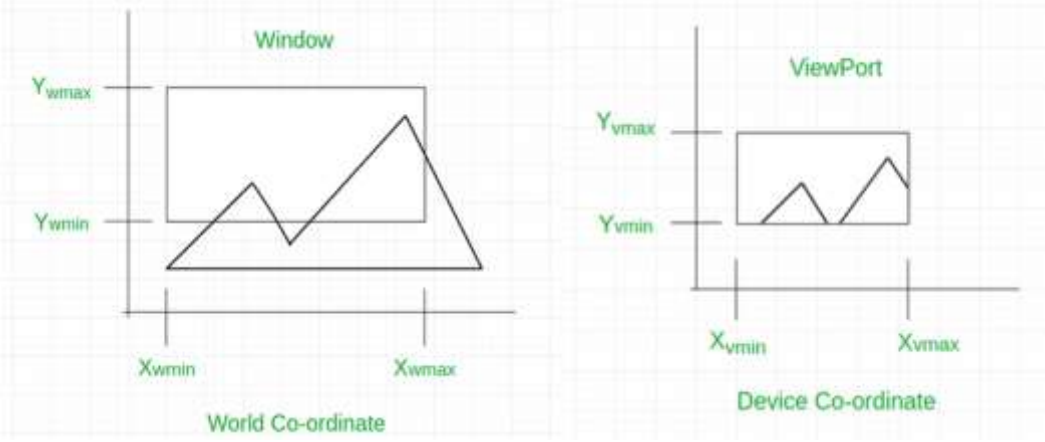


VIEWING PIPELINE

- A world coordinate area selected for display is called a **window**
- An area on a display device to which a window a mapped is called a **view port**
- The **window defines what is to be viewed**
- The **viewport defines where it is to be displayed**
- The mapping of a part of a world coordinate scene to device coordinates is referred to as a **viewing transformation.**

VIEWING PIPELINE

- Window to Viewport Transformation is the process of transforming 2D world-coordinate objects to device coordinates.
- Objects inside the world or clipping window are mapped to the viewport which is the area on the screen where world coordinates are mapped to be displayed.



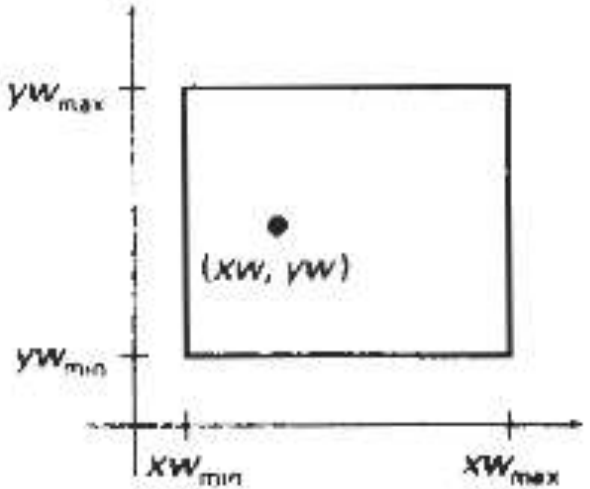
WINDOW TO VIEWPORT TRANSFORMATION

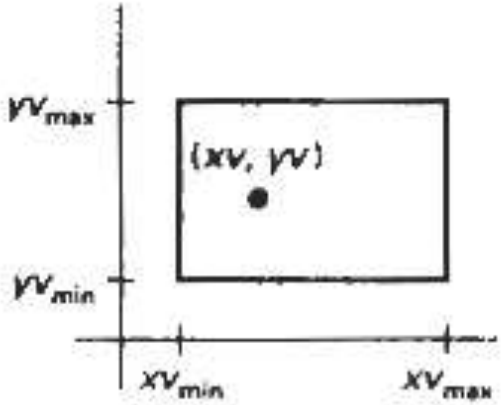
- A point at position (x_w,y_w) in a designated window is mapped to viewport coordinates $(x_{v,}y_v)$ so that relative positions in the two areas are the same.
- The figure illustrates the window to view port mapping.
- A point at position (x_w,y_w) in the window is mapped into position (x_v,y_v) in the associated view port.
- To maintain the same relative placement in view port as in window.

WINDOW TO VIEWPORT TRANSFORMATION

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

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





WINDOW TO VIEWPORT TRANSFORMATION

- The window to viewport coordinate transformation is known as workstation transformation.
- It is achieved by the following steps
  1. The object together with its window is translated until the lower left corner of the window is at the orgin.
  2. Object and window are scaled until the window has the dimensions of the viewport
  3. Translate the viewport to its correct position on the screen.
- The relation of the window and viewport display is expressed as XV- XVmin XW-XWmin

VIEWING FUNCTION

- We define a viewing reference system in a PHIGS application program with the following function:

  evaluateViewOrientationMatrix (x0, y0, xv, yv, error, viewMatrix)

- where parameters x0 and y0 are the coordinates of the viewing origin, and parameters xv and yv are the world-coordinate positions for the view up vector.

- An integer error code is generated if the input parameters are in error; otherwise, the view matrix for the world-to-viewing transformation is calculated.

VIEWING FUNCTION

- To set up the elements of a window-to-viewport mapping matrix, we invoke the function

evaluateViewMappingMatrix (xwmin, xwmax, ywmin, ywmax, xvmin, xvmax, yvmin, yvmax, error, viewMatrix)

- we can store combinations of viewing and window-viewport mappings for various workstations in a viruing table with

setVlewRepresentation (ws, viewIndex, viewMatrlx, viewMapping Matrix, xclipmin, xclipmax, yclipmin, yclipmax, clipxy)

VIEWING FUNCTION

- To set up the elements of a window-to-viewport mapping matrix, we invoke the function

evaluateViewMappingMatrix (xwmin, xwmax, ywmin, ywmax, xvmin, xvmax, yvmin, yvmax, error, viewMatrix)

- we can store combinations of viewing and window-viewport mappings for various workstations in a viruing table with

setVlewRepresentation (ws, viewIndex, viewMatrlx, viewMapping Matrix, xclipmin, xclipmax, yclipmin, yclipmax, clipxy)

VIEWING FUNCTION

- selects a particular set of options from the viewing table.

setViewIndex(viewIndex)

- At the find stage, we apply a workstation transformation by selecting a workstation window-viewport pair:

setWorkstationWindow (ws, xwsWindmin, xwsWindmax, ywsWindmin, ywsWindmax)

setWorkstationViewport (ws, xwsVPortmin, xwsVPortmax, ywsVPortmin, ywsVPortmax)

CLIPPING OPERATIONS

- **Definition:**

  "Any procedure that identifies those positions of picture is either inside or outside of a specified reason of space" is referred to as clipping algorithm or simply clipping.

- For the viewing transformation we want to display only those picture parts that are within the window.

- Everything the outside the window is discarded.

CLIPPING OPERATIONS

- **Clip Window:**

  "The region against which an object is to be clipped" is called a clip window.

- The complete world-coordinate picture can be mapped first to device coordinates, or normalized device coordinates, then clipped against the viewport boundaries.

CLIPPING OPERATIONS

Types

- There are number of clipping algorithm are used.
  - Point Clipping
  - Line Clipping
  - Polygon Clipping
  - Curve Clipping
  - Text Clipping

POINT CLIPPING

Definition:

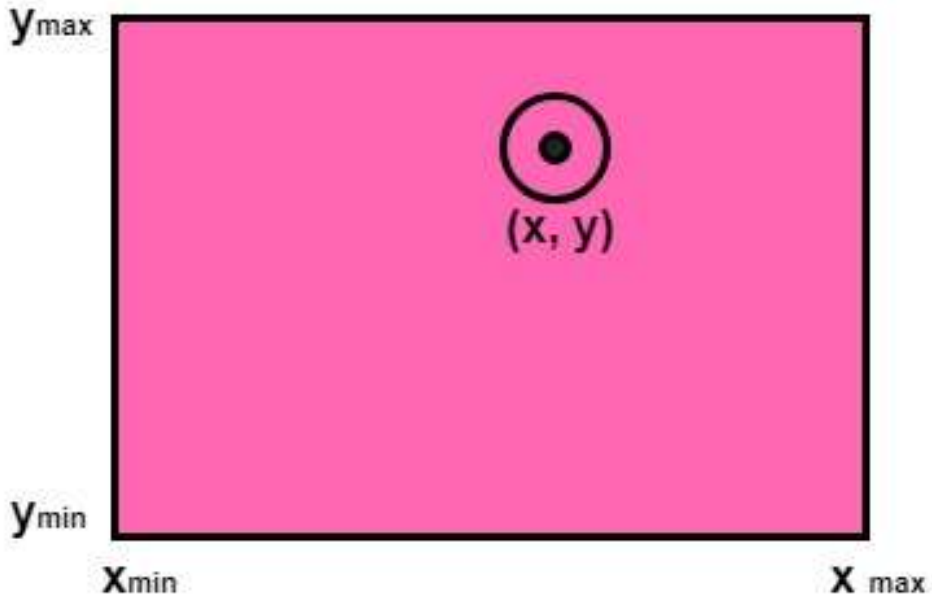
"To determine whether a point(x,y) lies inside or outside a specific region or boundary" is referred to as Point Clipping

Steps:

- Get the minimum and maximum coordinates of both viewing pane.
- Get the coordinates for a point.
- Check whether given input lies between minimum and maximum coordinate of viewing pane.
- If yes display the point which lies inside the region otherwise discard it.

POINT CLIPPING

"To determine whether a point lies inside or outside a specific region or boundary"



Conditions to be checked

$x \leq x_{max}$

$x \geq x_{min}$

$y \leq y_{max}$

$y \geq y_{min}$

LINE CLIPPING

Definition:

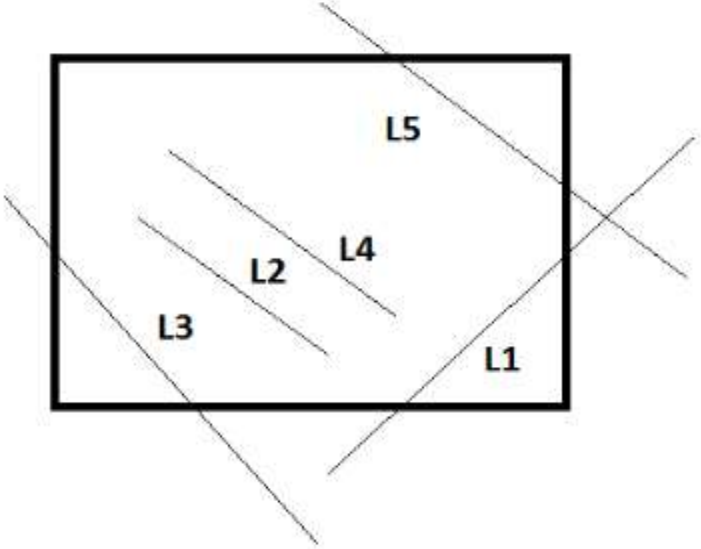
"The process of removing (clipping) lines or portions of lines outside an area of interest (a viewport)"

Steps:

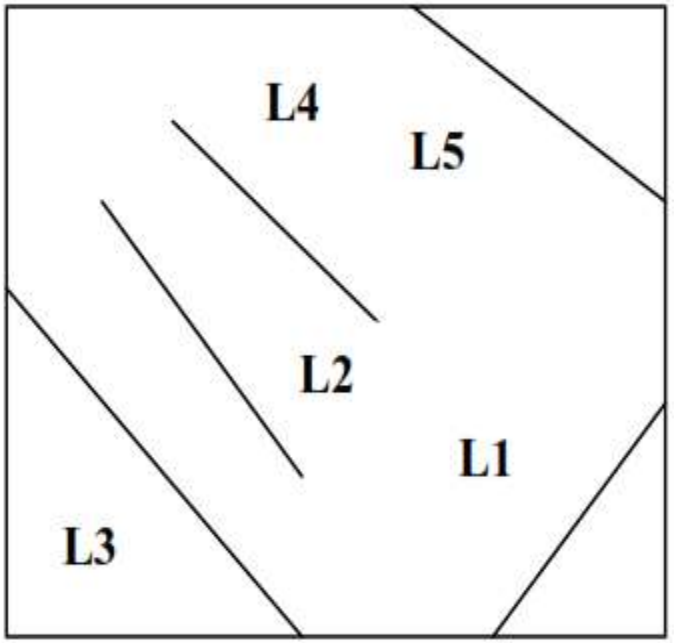
- Check the line completely inside the clipping window.
- If no, Check the line completely outside the clipping window.
- if both are no, perform intersection calculations with one or more clipping boundaries.

LINE CLIPPING

Before clipping



After Clipping



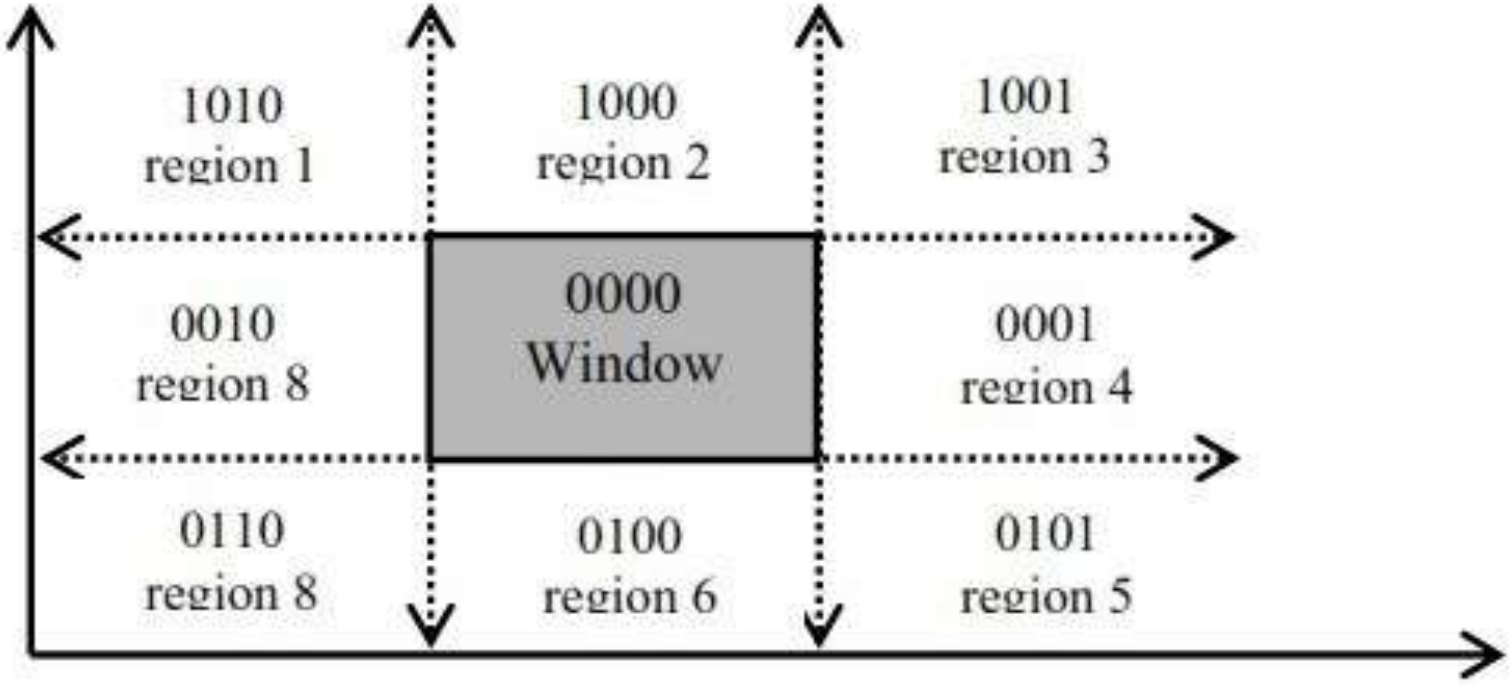
COHEN SOUTHERLAND LINE CLIPPING

- It divides a two-dimensional space into 9 regions and then efficiently determines the lines and portions of lines that are inside the given rectangular area.

Steps:

- Check the line completely inside the clipping window.
- If no, Check the line completely outside the clipping window.
- if both are no, perform intersection calculations with one or more clipping boundaries.

COHEN SOUTHERLAND LINE CLIPPING



COHEN SOUTHERLAND LINE CLIPPING

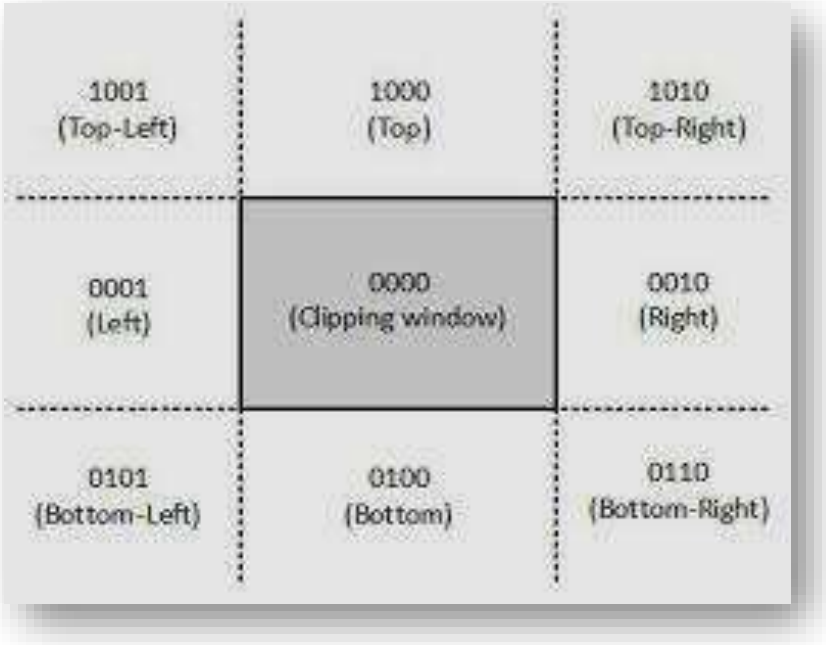
Region Code

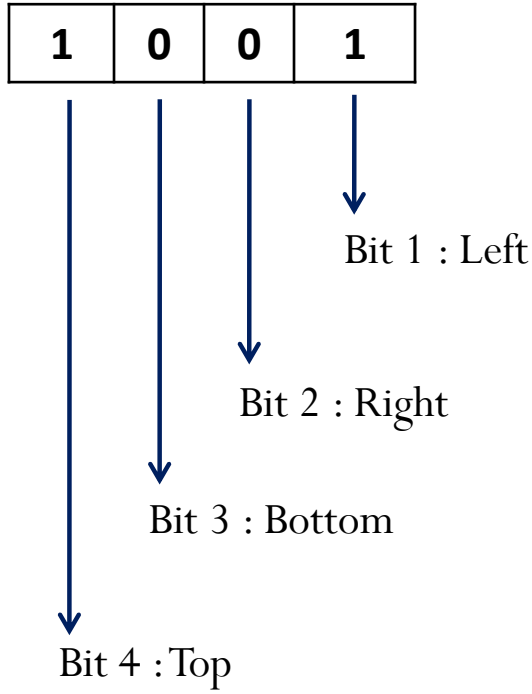
- Every line endpoint in a picture is assigned a four digit binary code called a region code.

- The central part is the viewing region or window, all the lines which lie within this region are completely visible.

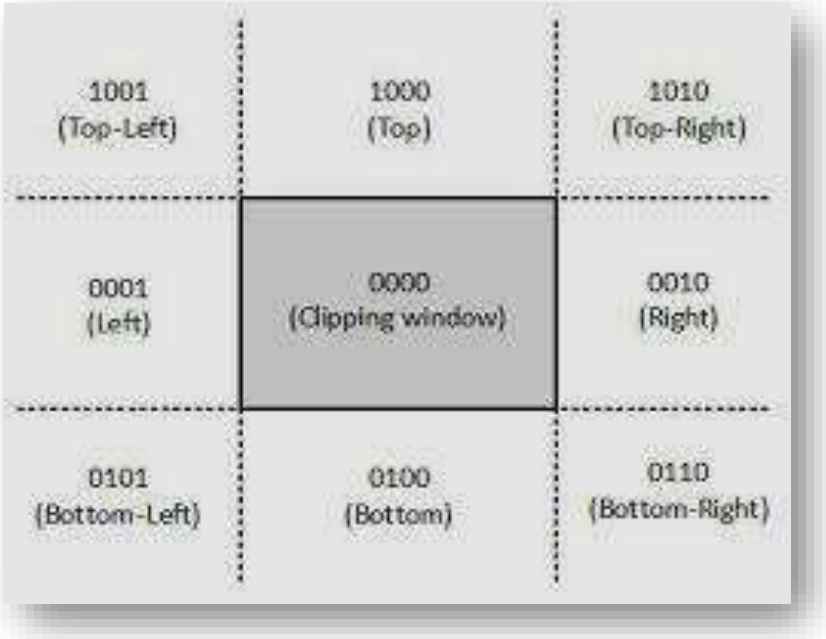
- A region code is always assigned to the endpoints of the given line.

COHEN SOUTHERLAND LINE CLIPPING





COHEN SOUTHERLAND LINE CLIPPING



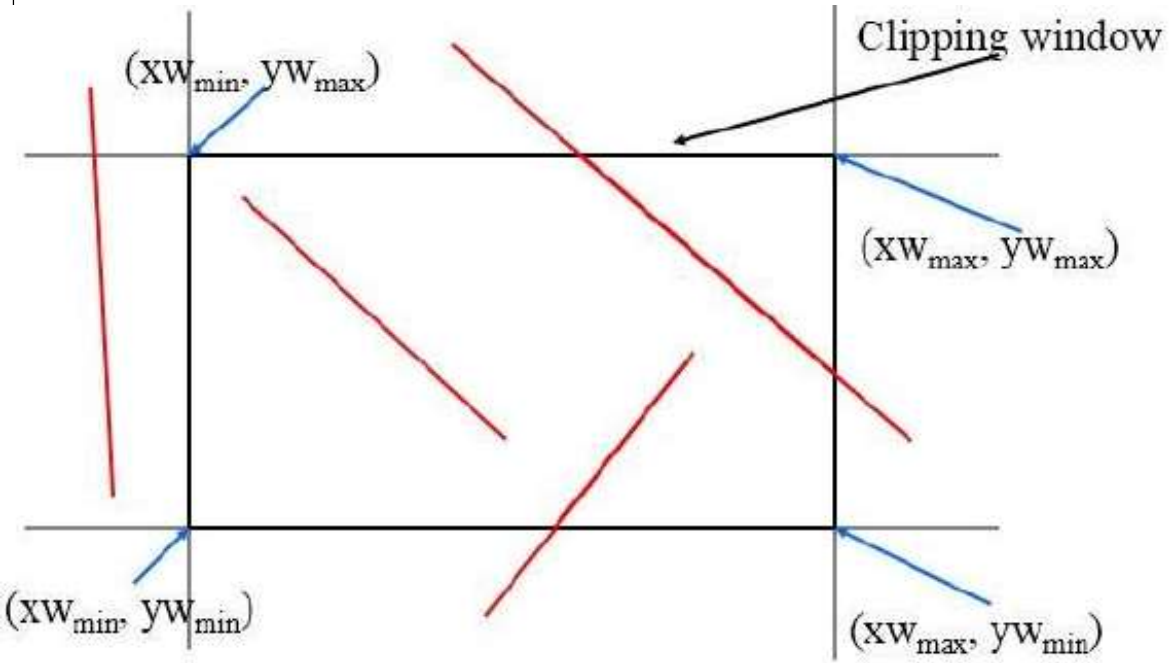
Bit1 is set to 1 if $x < xw_{min}$

bit 2 is set to 1 if $xw_{max} < x$

bit 3 is set to 1 if $y < yw_{min}$

bit 4 is set to 1 if $yw_{max} < y$.

COHEN SOUTHERLAND LINE CLIPPING



Bit1 is set to 1 if $x < xw_{min}$

bit 2 is set to 1 if $xw_{max} < x$

bit 3 is set to 1 if $y < yw_{min}$

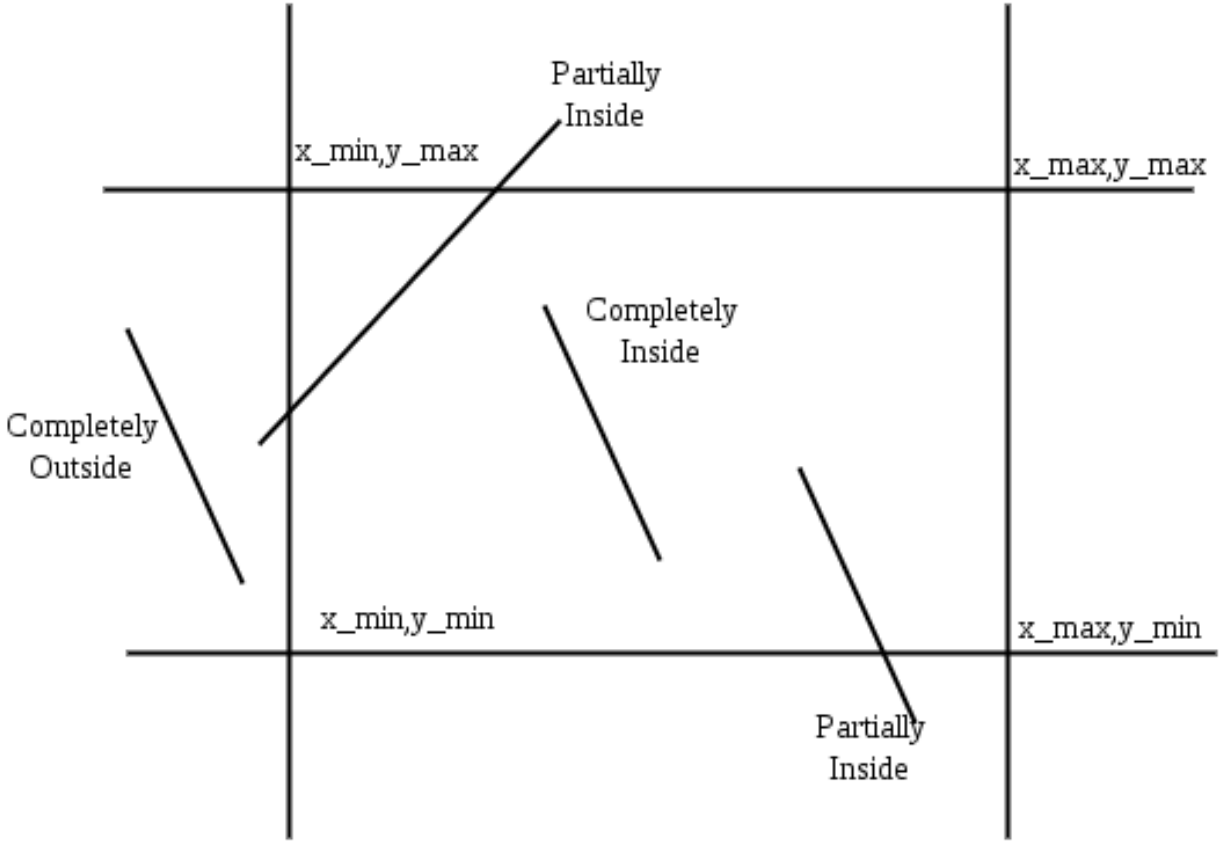
bit 4 is set to 1 if $yw_{max} < y$.

COHEN SOUTHERLAND LINE CLIPPING

There are three possible cases for any given line.

- Completely inside the window : No Clipping
- Completely outside the window : Clipping the entire line
- Partially inside the window : Find Intersection, and clip the outer portion of the line

COHEN SOUTHERLAND LINE CLIPPING



LIANG-BARSKY LINE CLIPPING

- The Liang–Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping window to determine the intersections between the line and the clip window.

- With these intersections it knows which portion of the line should be drawn.

- The parametric equation of a line can be given by,

$$x = x_1 + t(x_2 - x_1)$$
$$Y = y_1 + t(y_2 - y_1)$$

LIANG-BARSKY LINE CLIPPING

- Where, t is between 0 and 1. Then, writing the point-clipping conditions in the parametric form:

$$x_{wmin} <= X <= x_{wmax}$$
$$y_{wmin} <= Y <= y_{wmax}$$

- The above 4 in equations can be expressed as,

$$tp_k <= q_k$$

LIANG-BARSKY LINE CLIPPING

- Where, parameter p & q are defined as :

$$p1 = -\Delta x \quad q1 = x_1 - x_{wmin}$$
$$p2 = \Delta x \quad q2 = x_{wmax} - x_1$$
$$p3 = -\Delta y \quad q3 = y_1 - y_{wmin}$$
$$p4 = \Delta y \quad q4 = y_{wmax} - y_1$$

LIANG-BARSKY LINE CLIPPING

- Following observations can be:
- $p_k = 0$; Line is parallel to the window.
- $q_k < 0$; Line is completely outside the window.
- $p_k < 0$; Line is proceeds from outside to inside the window.
- $p_k > 0$; Line is proceeds from inside to outside the window.

- i.e Line intersect the clipping booundary

LIANG-BARSKY LINE CLIPPING

- Calculate the parameter $t=q_k/p_k$ for $k=1,2,3,4$.
- When $pk<0$, select the maximum values of q_k/p_k
- When $pk>0$, select the minimum values of q_k/p_k

- If $t1<t2$, then calculate the intersection points as:
- $xx_1 = x_1 + t_1 * \Delta x$
- $yy_1 = y_1 + t_1 * \Delta y$
- $xx_2 = x_1 + t_2 * \Delta x$
- $yy_2 = y_1 + t_2 * \Delta y$

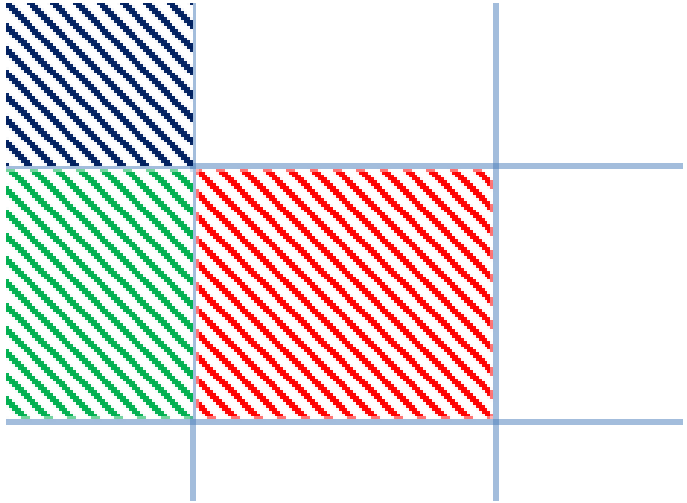
Draw the clipped line $((xx_1 , yy_1) , (xx_2 , yy_2))$

NICHOLL-LEE-NICHOL LINE CLIPPING

- These multiple intersection calculation is avoided in NLN line clipping procedure.
- By creating more regions around the clip window the NLN algorithm avoids multiple clipping of an individual line segment.
- NLN line clipping perform the fewer comparisons and divisions so it is more efficient.
- But NLN line clipping cannot be extended for three dimensions.

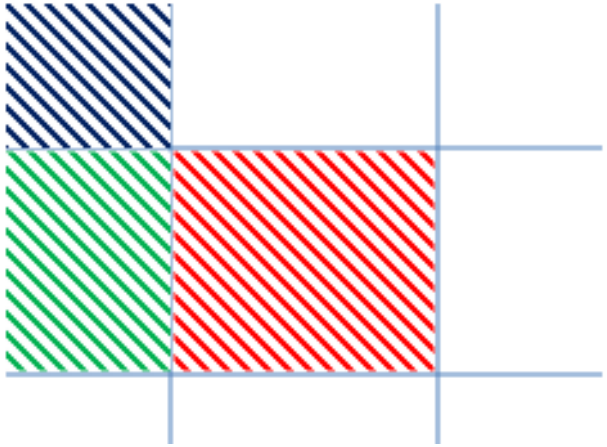
NICHOLL-LEE-NICHOL LINE CLIPPING

- For given line we find first point falls in which region out of nine region shown in figure.
- Only three region are considered which are.
  - Window region
  - Edge region
  - Corner region



NICHOLL-LEE-NICHOL LINE CLIPPING

- If point falls in other region than we transfer that point in one of the three region by using transformations.
- We can also extend this procedure for all nine regions.



NICHOLL-LEE-NICHOL LINE CLIPPING

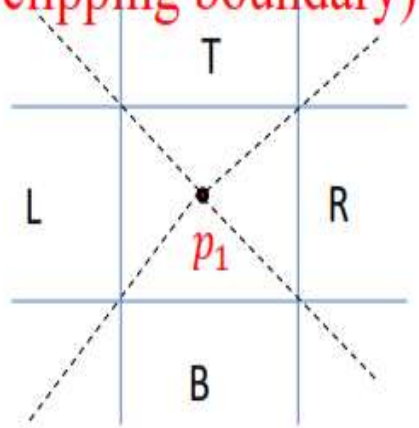
- Based on position of first point out of three region highlighted we divide whole space in new regions.

- Regions are name in such a way that name in which region p2 falls is gives the window edge which intersects the line.

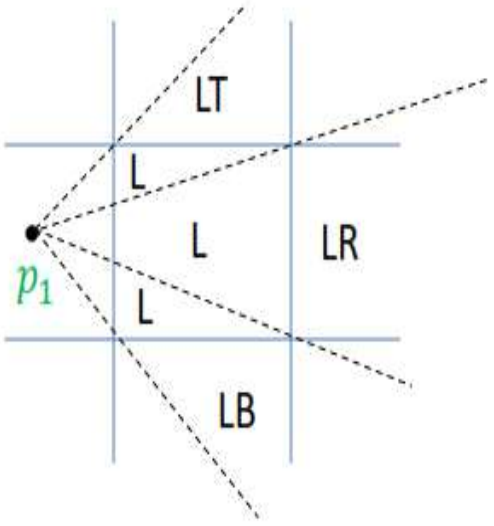
- *p*1is in window region(P1 is inside clipping boundary)

NICHOLL-LEE-NICHOL LINE CLIPPING

p_1 is in window region(P1 is inside clipping boundary)

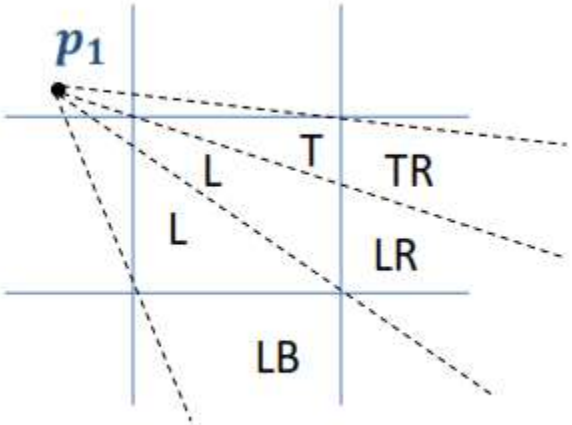


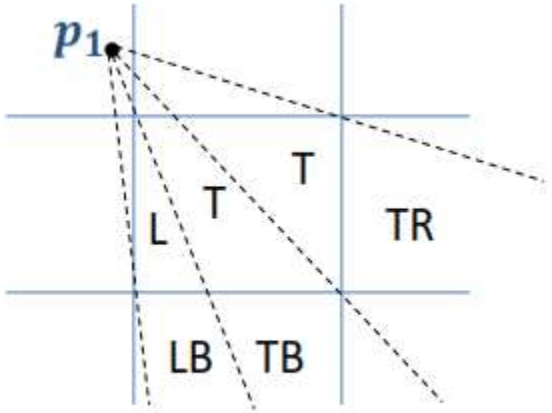
p_1 is in edge region



NICHOLL-LEE-NICHOL LINE CLIPPING

- $p1$ is in Corner region (one of the two possible sets of region can be generated)





NICHOLL-LEE-NICHOL LINE CLIPPING

Finding Region of Given Line in NLN

- For finding that in which region line p_1p_2 falls we compare the slope of the line to the slope of the boundaries:

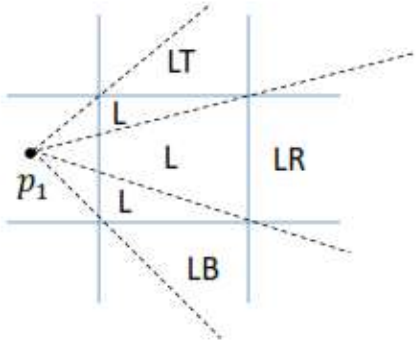
  $$slope\ \overline{p_1p_{B1}} < slope\ \overline{p_1p_2} < slope\ \overline{p_1p_{B2}}$$

  Where $\overline{p_1p_{B1}}$ and $\overline{p_1p_{B2}}$ are boundary lines.

- For example p1 is in edge region and for checking whether p2 is in region LT we use following equation.

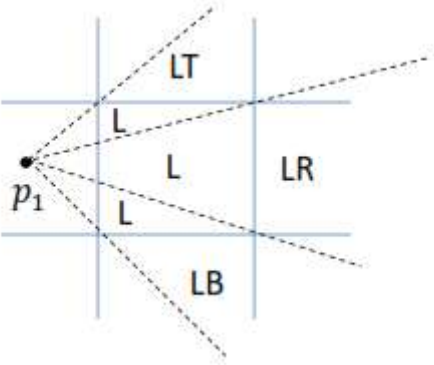
  $$slope\ \overline{p_1p_{TR}} < slope\ \overline{p_1p_2} < slope\ \overline{p_1p_{TL}}$$

  $$\frac{y_T - y_1}{x_R - x_1} < \frac{y_2 - y_1}{x_2 - x_1} < \frac{y_T - y_1}{x_L - x_1}$$



NICHOLL-LEE-NICHOL LINE CLIPPING

- After checking slope condition we need to check weather it crossing zero, one or two edges.
- This can be done by comparing coordinates of $p2$ with coordinates of window boundary.
- For left and right boundary we compare x coordinates and for top and bottom boundary we compare y coordinates.
  - If line is not fall in any defined region than clip entire line.
  - Otherwise calculate intersection.



NICHOLL-LEE-NICHOL LINE CLIPPING

Intersection Calculation in NLN

- After finding region we calculate intersection point using parametric equation which are:

$$x = x_1 + (x_2 - x_1)t$$

$$y = y_1 + (y_2 - y_1)t$$

- For left or right boundary $x = x_l$ *or* x_r respectively, with $t = (x_{l/r} - x_1)/(x_2 - x_1)$, so that y can be obtain from parametric equation as below:

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x_L - x_1)$$

- Keep the portion which is inside and clip the rest.

NICHOLL-LEE-NICHOL LINE CLIPPING

Intersection Calculation in NLN

- After finding region we calculate intersection point using parametric equation which are:

$$x = x_1 + (x_2 - x_1)t$$
$$y = y_1 + (y_2 - y_1)t$$

- For left or right boundary $x = x_1$ or x_r respectively, with $t = (x_{1/r} - x_1) / (x_2 - x_1)$, so that y can be obtain from parametric equation as below.

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x_L - x_1)$$

- Keep the portion which is inside and clip the rest.

NICHOLL-LEE-NICHOL LINE CLIPPING

Intersection Calculation in NLN

- Similarly for top or bottom boundary $y = y_t$ or y_b respectively, and $t = (y_{t/b} - y_1) / (y_2 - y_1)$, so that we can calculate x intercept as follow:

$$x = x_1 + \frac{x_2 - x_1}{y_2 - y_1}(y_T - y_1)$$

NON RECTANGULAR LINE CLIPPING

- In some applications, it is often necessary to clip lines against arbitrarily shaped polygons.
- Algorithms based on parametric line equations, such as the Liang-Barsky method can be extended easily convex polygon windows.
- We do this by modifying the algorithm to include the parametric equations for the boundaries oi the clip region.

NON RECTANGULAR LINE CLIPPING

- Circles or other curved-boundary clipping regions are also possible.
- But less commonly used.

- They are slower because intersection calculations involve nonlinear curve equations.

- At the first step, lines can be clipped against the bounding rectangle.

- Lines that can be identified as completely outside the bounding rectangle are discarded.

NON RECTANGULAR LINE CLIPPING

- To identify inside lines, we can calculate the distance of line endpoints from the circle center.

- If the square of this distance for both endpoints of a line is less than or equal to the radius squared, we can save the entire line.

- The remaining lines arc then processed through the intersection calculations, which must solve simultaneous circle-line equations

NON RECTANGULAR LINE CLIPPING

Splitting Concave Polygons

- We can identify a concave polygon by calculating the cross products of successive edge vectors in order around the polygon perimeter.

- If the z component of some cross products is positive while others have a negative z component, we have a concave polygon.

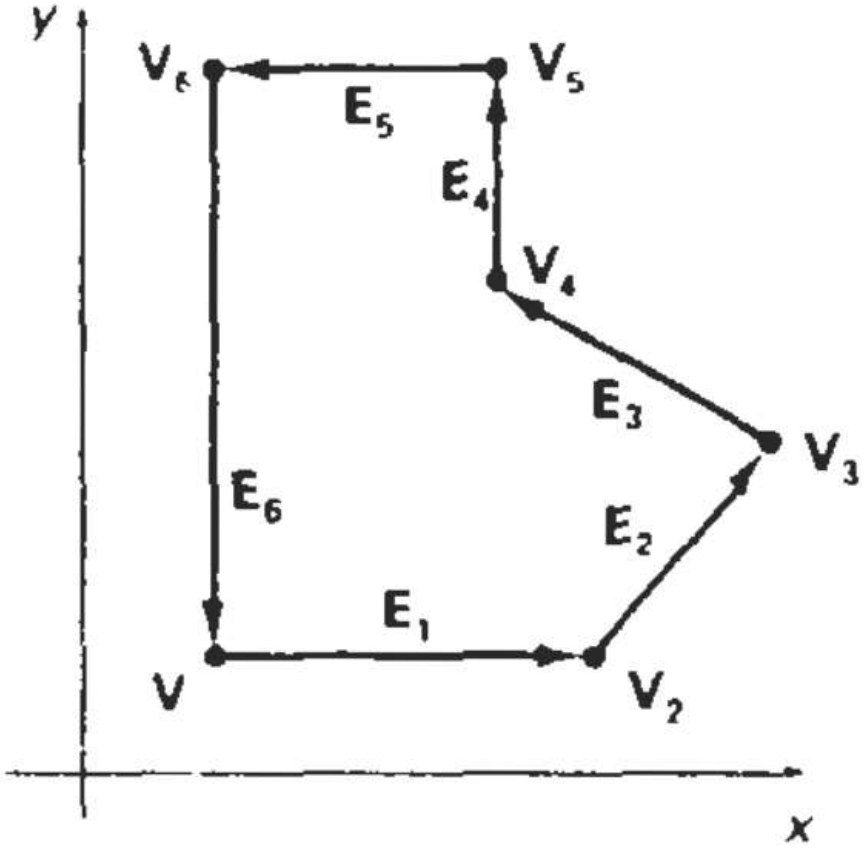
- Otherwise, the polygon is convex.

NON RECTANGULAR LINE CLIPPING

Splitting Concave Polygons

- This is assuming that no series of three successive vertices are collinear, in which case the cross product of the two edge vectors for these vertices is zero.
- If all vertices are collinear, we have a degenerate polygon (a straight line).
- The Figure(in next slide) illustrates the edge vector cross-product method for identifying concave polygons.

NON RECTANGULAR LINE CLIPPING



$(E_1 \times E_2)_z > 0$

$(E_2 \times E_3)_z > 0$

$(E_3 \times E_4)_z < 0$

$(E_4 \times E_5)_z > 0$

$(E_5 \times E_6)_z > 0$

$(E_6 \times E_1)_z > 0$

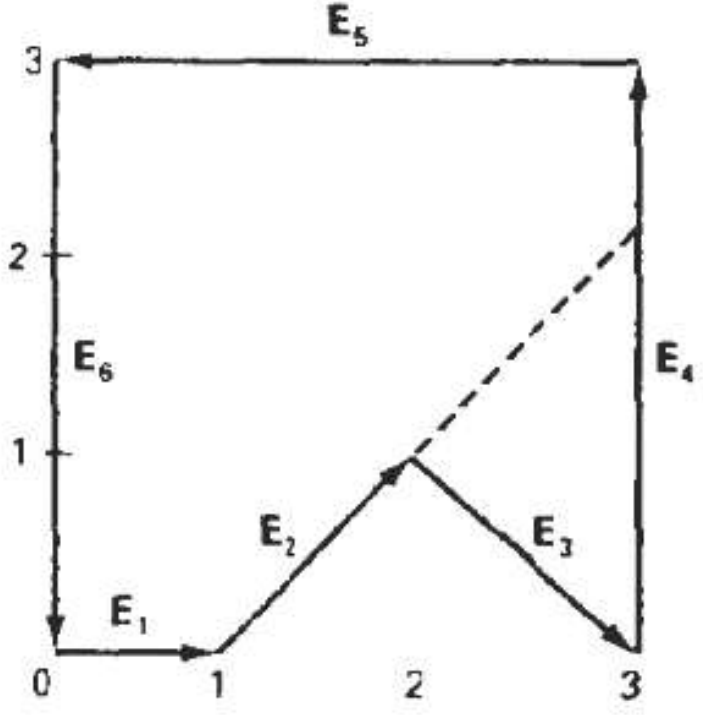
SPLITTING CONCAVE POLYGONS

Vector Method

- A vector method for splitting a concave polygon in the xy plane is to calculate the edge-vector cross products in a counter clockwise order.
- And note the sign of the z component of the cross products.
- If any z component turns out to be negative (as in Fig. Previous slide), the polygon is concave and we can split it along the line of the first edge vector in the crossproduct pair.

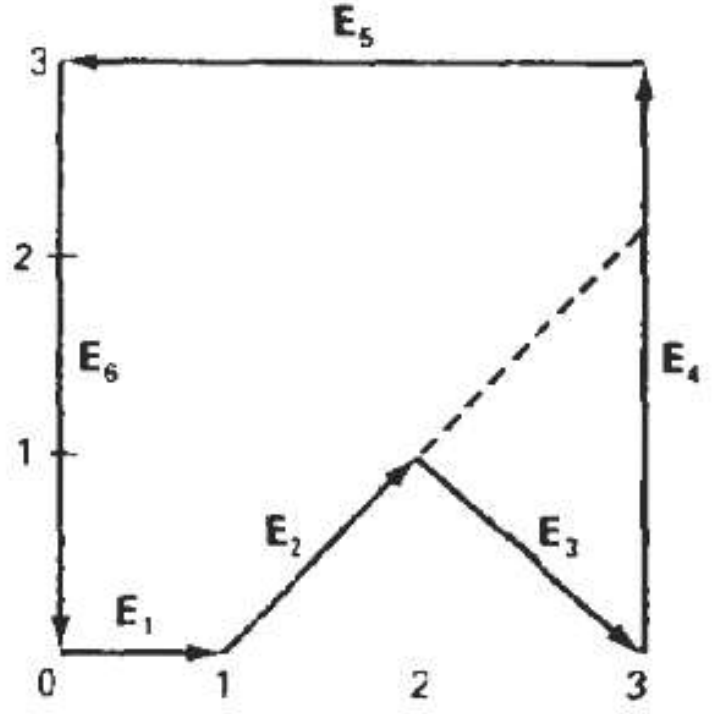
SPLITTING CONCAVE POLYGONS

- The figure shows a concave polygon with six edges.



SPLITTING CONCAVE POLYGONS

- Edge vector for this polygon can be expressed as



$E_1 = (1, 0, 0)$, $E_2 = (1, 1, 0)$

$E_3 = (1, -1, 0)$, $E_4 = (0, 2, 0)$

$E_5 = (-3, 0, 0)$, $E_6 = (0, -2, 0)$

SPLITTING CONCAVE POLYGONS

- Where the z component is 0, since all edges are in the xy plane.
- The cross product $E_{ix}E_j$ for two successive edge vectors is a vector perpendicular to the xy plane with z component equal to

$$E_{ix}E_{jy} - E_{jx}E_{iy}$$

$$\mathbf{E}_1 \times \mathbf{E}_2 = (0, 0, 1), \qquad \mathbf{E}_2 \times \mathbf{E}_3 = (0, 0, -2)$$

$$\mathbf{E}_3 \times \mathbf{E}_4 = (0, 0, 2), \qquad \mathbf{E}_4 \times \mathbf{E}_5 = (0, 0, 6)$$

$$\mathbf{E}_5 \times \mathbf{E}_6 = (0, 0, 6), \qquad \mathbf{E}_6 \times \mathbf{E}_1 = (0, 0, 2)$$

SPLITTING CONCAVE POLYGONS

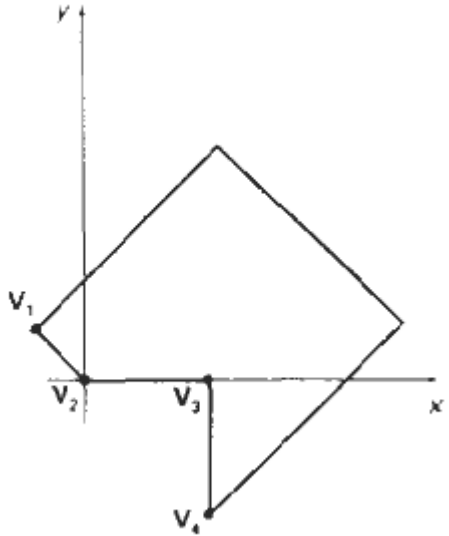
- Since the cross product E2 X E3 has a negative z component, we split the polygon along the line of vector E2.
- The line equation for this edge has a slope of 1 and a y intercept of -1.
- We then determine the intersection of this line and the other polygon edges to split the polygon into two pieces.
- No other edge cross products are negative, so the two new polygons are both convex.

SPLITTING CONCAVE POLYGONS

- Splitting a concave polygon using the rotational method.

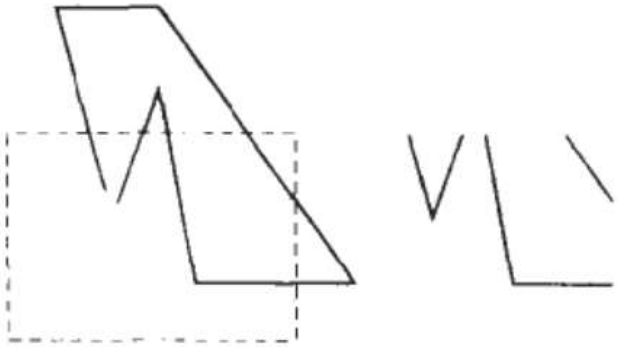
- After rotating V3 onto the x axis, we find that V4, is below the x axls.

- So we split the polygon along the line of V2 V3



POLYGON CLIPPING

- To clip polygons, we need to modify the line-clipping procedures.
- A polygon boundary processed with a line clipper may be displayed as a series of unconnected line segments depending on the orientation of the polygon to the clipping window.



POLYGON CLIPPING

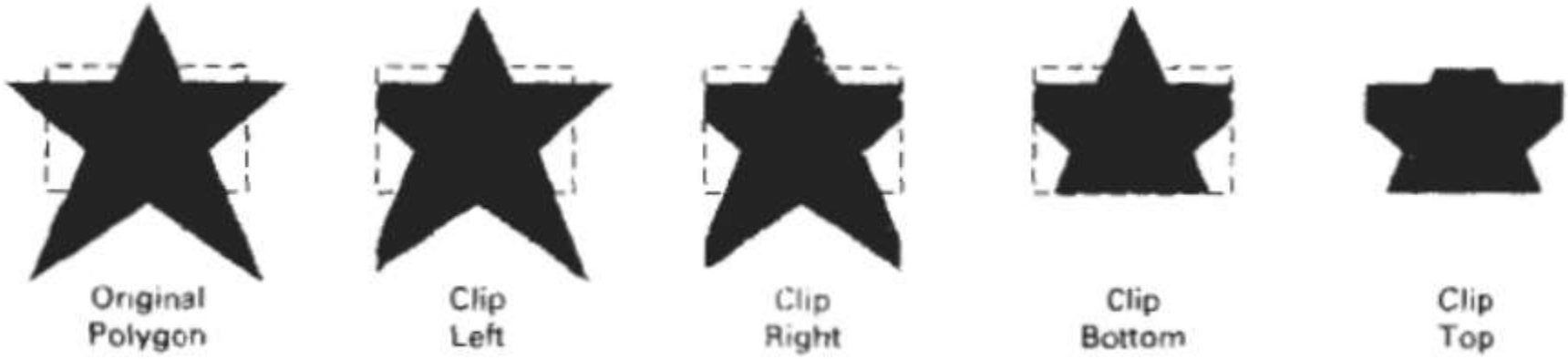
- For polygon clipping, we require an algorithm that will generate one or more closed areas that are then scan converted for the appropriate area fill.
- The output of a polygon clipper should be a sequence of vertices that defines the clipped polygon boundaries.

SUTHERLAND-HODGEMAN POLYGON CLIPPING

- We can correctly clip a polygon by processing the polygon boundary as a whole against each window edge.
- This could be accomplished by processing all polygon vertices against each clip rectangle boundary in turn.
- 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 k successively passed to a right boundary

SUTHERLAND-HODGEMAN POLYGON CLIPPING

- The new set of vertices could then k successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper, as in Fig.



SUTHERLAND-HODGEMAN POLYGON CLIPPING

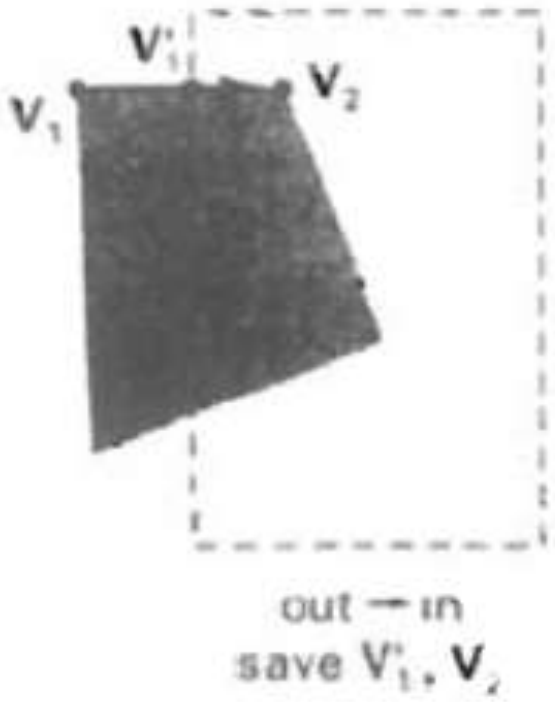
- At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper.

- There are four possible cases when processing vertices in sequence around the perimeter of a polygon.

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

SUTHERLAND-HODGEMAN POLYGON CLIPPING

- 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.

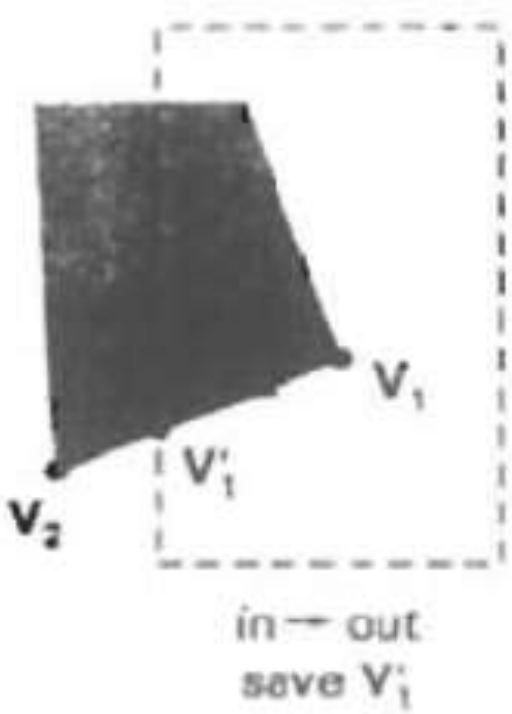


SUTHERLAND-HODGEMAN POLYGON CLIPPING

- If both input vertices are inside the window boundary, only the second vertex is added to the output vertex list.

SUTHERLAND-HODGEMAN POLYGON CLIPPING

- 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.



SUTHERLAND-HODGEMAN POLYGON CLIPPING

- If both input vertices are outside the window boundary, nothing is added to the output list.

OTHER CLIPPING METHODS

- Various parametric line-clipping methods have also been adapted to polygon clipping. And they are particularly well suited for clipping against convex polygon clipping windows.
  - Curve Clipping
  - Text Clipping

CURVE CLIPPING

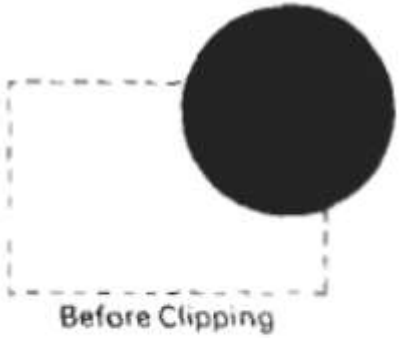
- Areas with curved boundaries can be clipped with methods similar to those discussed in the line clipping.
- Curve-clipping procedures will involve nonlinear equations, however, and this requires more processing than for objects with linear boundaries.
- The bounding rectangle for a circle or other curved object can be used first to test for overlap with a rectangular clip window.

CURVE CLIPPING

- If the bounding rectangle for the object is completely inside the window, we save the object.
- If the rectangle is determined to be completely outside the window, we discard the object.
- In either case, there is no further computation necessary.
- For a circle, we can use the coordinate extents of individual quadrants and then octants for preliminary testing before calculating curve-window intersections.
- For an ellipse, we can test the coordinate extents of individual quadrants.

CURVE CLIPPING

- Following figure illustrates circle clipping against a rectangular window.

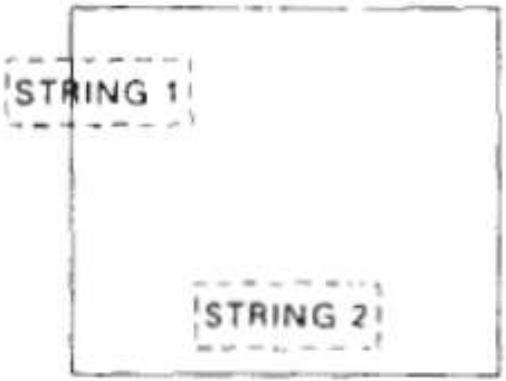


TEXT CLIPPING

- If the bounding rectangle for the object is completely inside the window, we save the object.
- If the rectangle is determined to be completely outside the window, we discard the object.
- In either case, there is no further computation necessary.
- For a circle, we can use the coordinate extents of individual quadrants and then octants for preliminary testing before calculating curve-window intersections.
- For an ellipse, we can test the coordinate extents of individual quadrants.

TEXT CLIPPING

- The simplest method for processing character strings relative to a window boundary is to use the **all-or-none string-clipping** strategy shown in Fig.

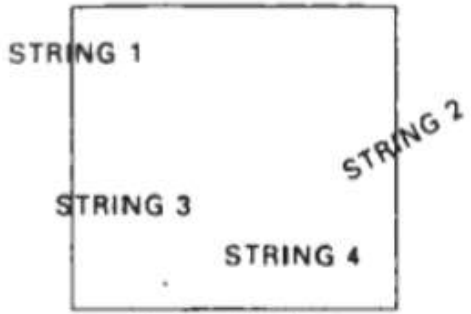


Before Clipping

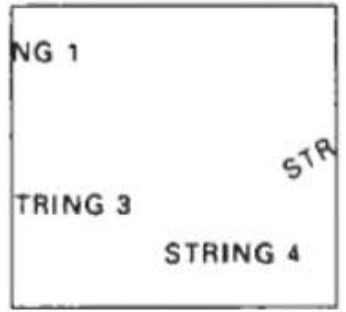
After Clipping

TEXT CLIPPING

- An alternative to rejecting an entire character string that overlaps a window boundary is to use the **all-or-none character-clipping** strategy.
- Here we discard only those characters that are not completely inside the window.



Before Clipping

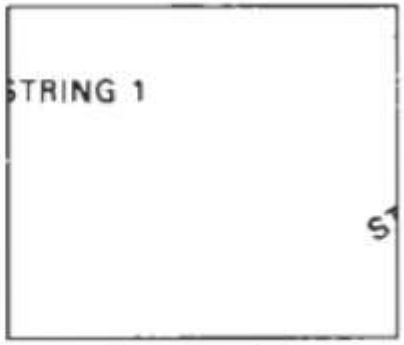


After Clipping

TEXT CLIPPING

- Final method is to **clip the components of individual characters**.
- We now treat characters in much the same way that we treated lines.
- If an individual character overlaps a clip window boundary, we clip off the parts of the character that are outside the window.

Before Clipping



After Clipping