

COMPUTER GRAPHICS

References

1. M.Berger," Computer Graphic with Pascal ", B/C Publishing Company,1984.
2. J.D.Foley & A.Dametal," Introduction to Computer Graphic ", Addison – wesly, 1993.
3. D.Hearn & M.p.Baker," Computer Graphics ", 2nd Ed., Prentice – Hall,1994.
4. Computer Graphics Tutorials Point Simply Easy Learning.(www.tutorialspoint.com).
5. اساسيات الرسم بالحاسوب، دكتور نضال العبادي، الطبعة الاولى، 2018.
6. Fundamentals of Computer Graphics 4th Edition by Steve Marschner (Author), Peter Shirley (Author) 2016.
7. J.D Foley & A. Dametal, "Introduction to Computer Graphic", Addison-Wesly,1993.
8. D. Hearn & M.P. Baker," Computer Graphics ", 2nd Ed., Prentice-Hall, 1994.
9. B.E. Johnson," 3D Modeling and Animation ", design, images& text 1976-2012.
10. <https://www.javatpoint.com/computer-graphics-cathode-ray-tube>
- 11- Internet.

Part One

Introduction to Computer Graphics

Overview

Computer graphic can be defined as the creation and manipulation of graphic image (generation, representation, manipulation, processing) by computer.

Computer graphic started as a technique to enhance the display of information generated by a computer. This ability to interpret and represent numerical data in pictures has significantly increased the computer's ability to represent information to the user in a clear and understandable form.

Application of Graphics

1. Management may be displayed as charts and diagrams (data are converted into bar charts, pie charts and graphs).
2. Maps can be created for all kinds of geographic information.
4. Simulation.
5. Video games provide.
6. Computer graphics user interfaces (GUIs).
7. Photo Enhancement - Sharpening blurred photos.
8. Medical imaging - MRIs, CAT scans, etc. - Non-invasive internal examination.

Mode: divided to:

1. **Text mode:** deal with characters, numbers and symbols.
2. **Graphics mode:** deal with pixels.

Picture Elements

1. **Pixel:** is the smallest addressable screen element. It is the smallest piece of the display screen which we can control.
2. **Line:** has patron or type. **Specification:** color, lighting, type, width.

Types of line:



3. **Curve:** depends on start point and end point and angle.

1. Graphic System

1.1 Cathode Ray Tube:

The primary output device in a graphical system is the video monitor. The main element of a video monitor is the Cathode Ray Tube (CRT). CRT is a technology used in traditional computer monitors and televisions. *The image on CRT display is created by firing electrons from the back of the tube of phosphorus located towards the front of the screen. Once the electron heats the phosphorus, they light up, and they are projected on a screen.* The color you view on the screen is produced by a blend of red, blue and green light.

Components of CRT

- 1. Electron Gun:** Electron gun consisting of a series of elements (heater) and a cathode. The electron gun creates a source of electrons which are focused into a narrow beam directed at the face of the CRT.
- 2. Control Electrode:** It is used to turn the electron beam on and off.
- 3. Focusing system:** It is used to create a clear picture by focusing the electrons into a narrow beam.
- 4. Deflection Yoke:** It is used to control the direction of the electron beam towards specified positions on the phosphor-coated screen.
- 5. Phosphorus-coated screen:** The inside front surface of every CRT is coated with phosphors. Phosphors glow when a high-energy electron beam hits them. *Phosphorescence* is the term used to characterize the light given off by a phosphor after it has been exposed to an electron beam.

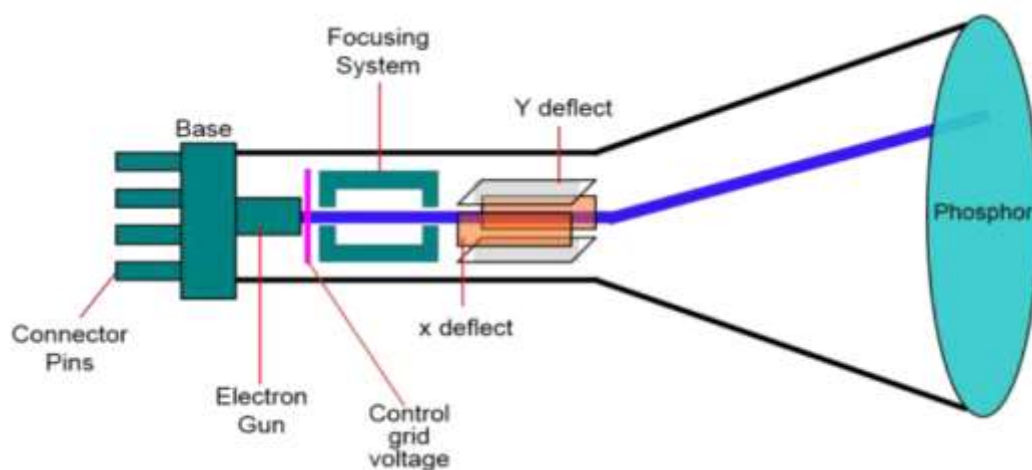


Figure 1: Cathode Ray Tube (CRT)

1.2 Random Scan and Raster Scan Display:

1.2.1 Random Scan Display

Random Scan System uses an electron beam which operates like a pencil to create a line image on the CRT screen. The picture is constructed out of a sequence of straight-line segments. Each line segment is drawn on the screen by directing the beam to move from one point on the screen to the next, where its x & y coordinates define each point. After drawing the picture, the system cycles back to the first line and design all the lines of the image 30 to 60 time each second. The *electron beam is directed only to the part of the screen where the picture is to be drawn rather than scanning from left to right and top to bottom as in raster scan*. The process is shown in fig.2:

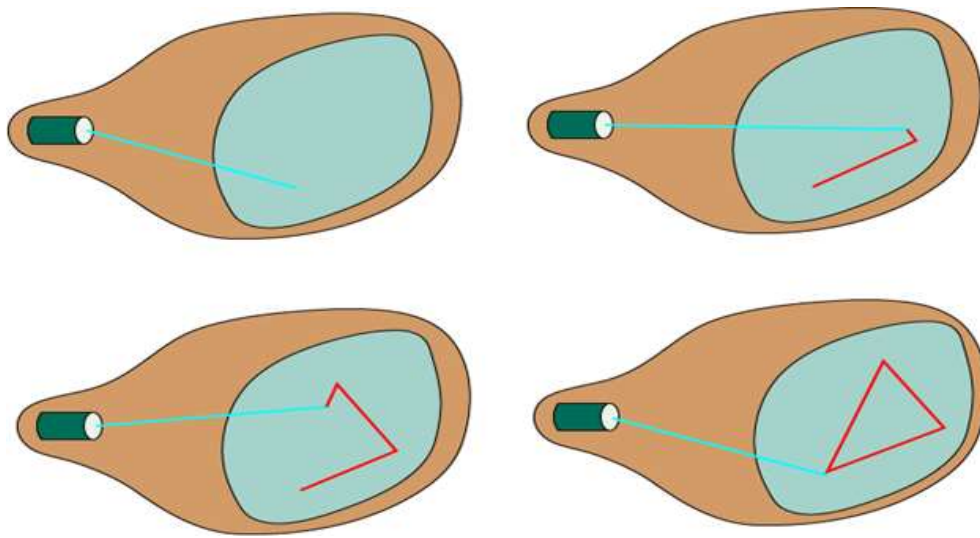
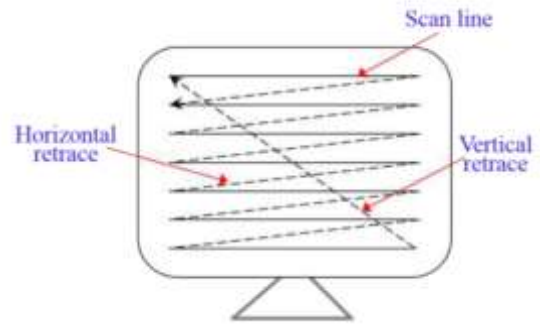


Figure 2: Random-scan monitors

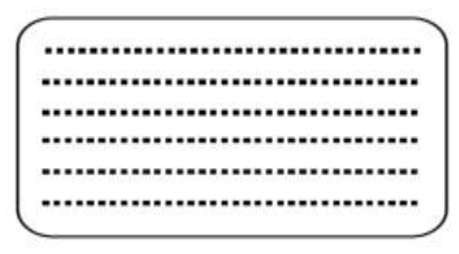
1.2.2 Raster Scan Display

A Raster Scan Display is based on intensity of pixels in the form of a rectangular box called *Raster on the screen*. Information of on and off pixels is stored in Frame buffer. Televisions in our house are based on Raster Scan Method. The raster scan system can store information of each pixel position. Raster Scan provides a refresh rate of 60 to 80 frames per second.

Frame Buffer is also known as *Raster or bit map*. In Frame Buffer the positions are called picture elements or *pixels*. Beam refreshing is of two types. **First** is horizontal retracing and **second** is vertical retracing. When the beam starts from the top left corner and reaches the bottom right scale, it will again return to the top left side called at *vertical retrace*. Then it will again more horizontally from top to bottom call as horizontal retracing shown in fig.3:

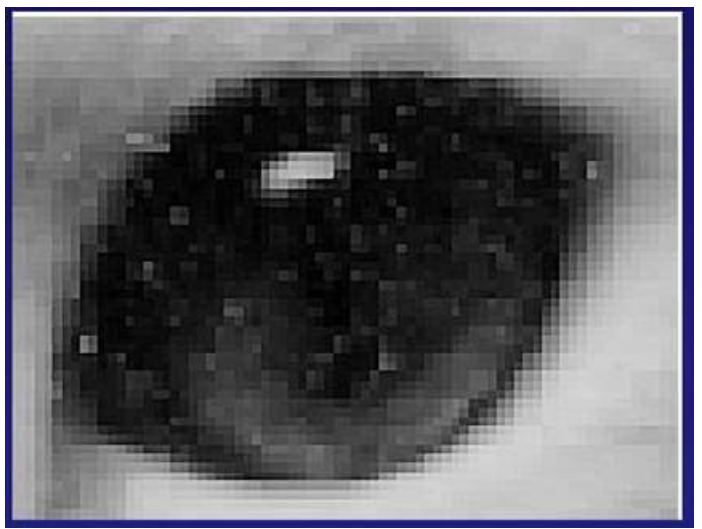
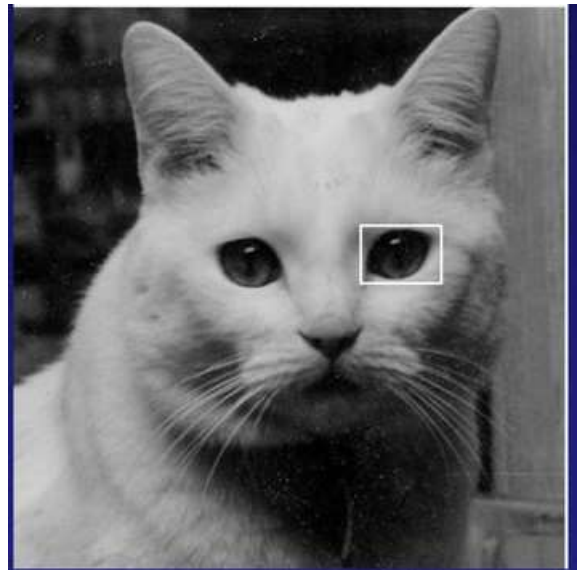


A



B

Figure 3:
A: Raster Scan
B: Raster Display Image



1.2.3 Color CRT Monitor

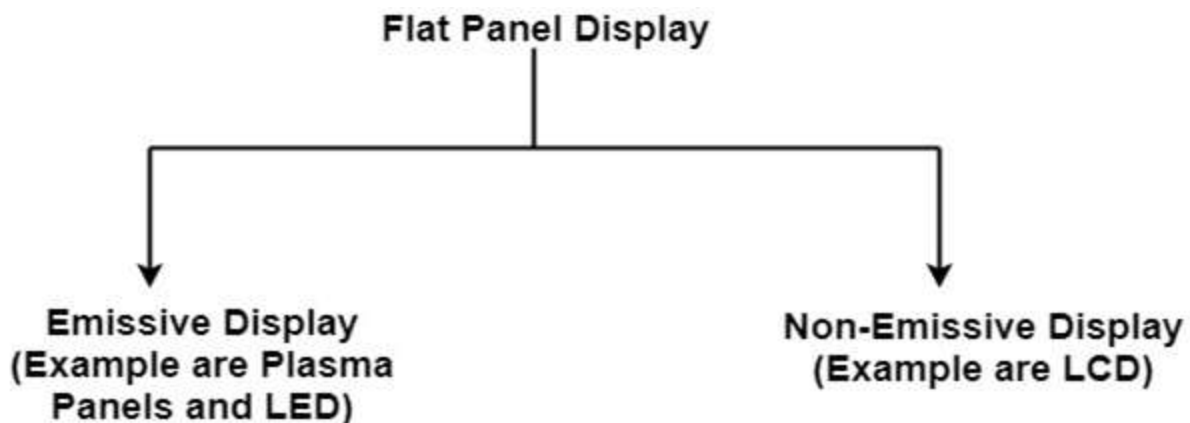
Each electron gun in a RGB monitor, has an assigned number of bit that determines the intensities of the red, green and blue phosphors, with one bit per gun eight color are possible ($2^3=8$).

R	G	B	Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	cyan
1	0	0	red
1	0	1	magenta
1	1	0	Yellow
1	1	1	white

1.3 Flat Panel Display:

The *Flat-Panel display* refers to a class of video devices that have reduced volume, weight and power requirement compare to CRT.

Example: Small T.V. monitor, calculator, pocket video games, laptop computers, an advertisement board in elevator.



1. **Emissive Display:** The emissive displays are devices that convert electrical energy into light. Examples are Plasma Panel and LED (Light Emitting Diodes).

2. **Non-Emissive Display:** The Non-Emissive displays use optical effects to convert sunlight or light from some other source into graphics patterns. Examples are LCD (Liquid Crystal Device).

1.3.1 Plasma Panel Display

Plasma-Panels are also called as *Gas-Discharge Display*. It consists of an array of small lights. Lights are fluorescent in nature. The essential components of the plasma-panel display are:

1. **Cathode:** It consists of fine wires. It delivers negative voltage to gas cells. The voltage is released along with the negative axis.
2. **Anode:** It also consists of fine wires. It delivers positive voltage. The voltage is supplied along positive axis.
3. **Fluorescent cells:** It consists of small pockets of gas liquids when the voltage is applied to this liquid (neon gas) it emits light.
4. **Glass Plates:** These plates act as capacitors. The voltage will be applied, the cell will glow continuously.

The gas will glow when there is a significant voltage difference between horizontal and vertical wires. The voltage level is kept between 90 volts to 120 volts. Plasma level does not require refreshing. Erasing is done by reducing the voltage to 90 volts.

Each cell of plasma has two states, so cell is said to be stable. Displayable point in plasma panel is made by the crossing of the horizontal and vertical grid. The resolution of the plasma panel can be up to 512 * 512 pixels.

1.3.2 LED (Light Emitting Diode)

In an LED, a matrix of diodes is organized to form the pixel positions in the display and picture definition is stored in a refresh buffer. Data is read from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light pattern in the display.

1.3.3 LCD (Liquid Crystal Display)

Liquid Crystal Displays are the devices that produce a picture by passing polarized light from the surroundings or from an internal light source through a liquid-crystal material that transmits the light.

LCD uses the liquid-crystal material between two glass plates; each plate is the right angle to each other between plates liquid is filled. One glass plate consists of rows of conductors arranged in vertical direction. Another glass plate is consisting of a row of conductors arranged in horizontal direction. The pixel position is determined by the intersection of the vertical & horizontal conductor. This position is an active part of the screen.

Liquid crystal display is temperature dependent. It is between zero to seventy degree Celsius. It is flat and requires very little power to operate.

2. Look-Up Table

Image representation is essentially the description of pixel colors. There are three primary colors: R (red), G (green) and B (blue). We may allocate 3 bits for each pixel, with one bit for each primary color. The 3-bit representation allows each primary to vary independently between two intensity levels: 0 (off) or 1 (on). Hence each pixel can take on one of the eight colors.

Bit 1:r	Bit 2:g	Bit 3:b	Color name
0	0	0	Black اسود
0	0	1	Blue ازرق
0	1	0	Green اخضر
0	1	1	Cyan شذري
1	0	0	Red احمر
1	0	1	Magenta بنفسجي
1	1	0	Yellow اصفر
1	1	1	White ابيض

If a fourth bit is used for the brightness' or intensity of the displayed color it results in ($2^4=16$) possible colors. When the brightness bit equals one another set of eight bright colors is produced.

Note: if True Color is 24 bit then displayed color = 0 to $2^{24}-1$.

In RGB 24 bit → Red (8 bit = 0 to 2^8-1), Green (8 bit = 0 to 2^8-1), Blue (bit= 0 to 2^8-1)

3. Screen Clarity

The screen clarity depends on three qualities:

3.1 Resolution

The number of scan lines and the number of pixels on a line or dots per unit area. The more pixels per square inch, and the better of resolution.

- Low resolution 300 scans lines & 400 pixels (lines).
- High resolution more than 1000 scan lines & 1000 pixels (lines).

3.2 Dot Pitch

It is the amount of space between the center of adjacent pixels, the closer the dots, the crisper the image. For crisp images, dot pitch should be less than 0.31 millimeter.

3.3 Refresh Rate

It is the number of times per second that the pixels are recharged so that their glow remains bright. In general, displays are refreshed 45 to 100 times per second.

4. The graphic display

The graphic display consists of three components:

- a) Frame buffer.
- b) Display controller.
- c) Scan conversion algorithms.

a) Frame buffer

Each screen pixel corresponds to a particular entry in a two – dimensional array residing in memory. This memory is called a **frame buffer** or **bit map** see **figure 4**.

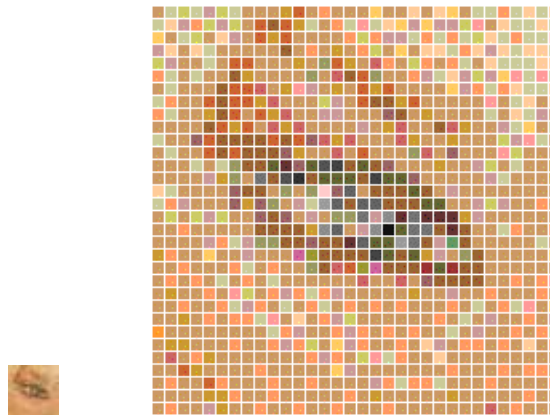


Figure 4: Bit Map

The number of **rows** in the frame buffer array equal the number of **raster lines** on the display screen, and the number of **columns** in this array equals the **number of pixels** on each raster line. The current trends is to have the frame buffer accessible to CPU of the main memory, to allowing rapid of the storage image.

To display a pixel on the screen, a specific value is placed into the corresponding memory location in the frame buffer array. In figure (5), a value of **1** placed in a location in the frame buffer results in the corresponding (black) pixel displayed on the screen.

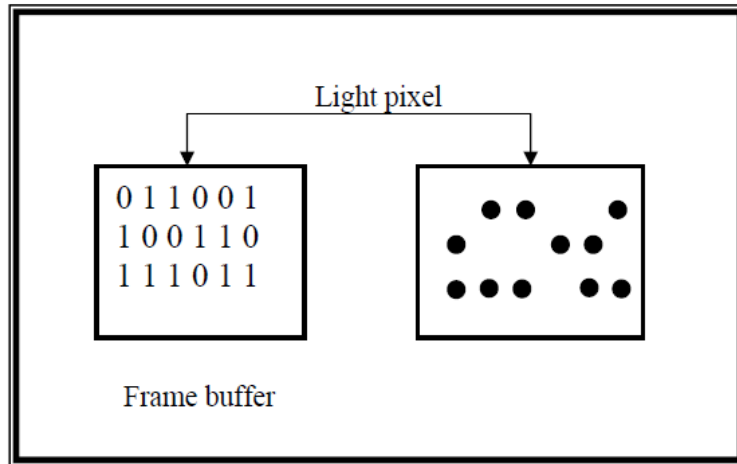


Figure 5: Bit Plane of Frame Buffer

Number of pixels depending on screen type, so that we have many types likes:

- 1) Monochrome 2 color (black, green).
- 2) CGA (Color Graphic Adapter) 16 color 320 X 200 .
- 3) EGA (Enhanced Graphic Adapter) 16 color 640 X 480 .
- 4) VGA (Video Graphic Adapter) 256 color 640 X 480 .
- 5) SVGA (Super Video Graphic Adapter) 256 color 1024 X 768 .

Each pixel and corresponding memory location in the frame buffer is accessed by an (X,Y) integer coordinate pair. The X value refers to the rows; the Y value refers to the columns position.

b) Display controller

The hardware device reads the contents of frame buffer into a video buffer which then *converts the digital representation of a string of pixel values into analog voltage signals* that are sent serially to the video screen. Whenever the display controller encounters a value of **1** in a single – bit – plane frame buffer a high – voltage. Signal is sent to the CRT which turns on the corresponding screen pixel.

c) Scan conversion

Images are usually defined in terms of equation for example $A + B = 5$ or graphic descriptions, such as “draw a line from A to B“. *Scan conversion* is the *process of converting this abstract representation of an image into the appropriate pixel values in the frame buffer.*

Part Two

Drawing Elementary Figures and Line Drawing Algorithm

2. Scan Conversion Definition

It is a *process of representing graphics objects a collection of pixels*. The graphics objects are continuous. The pixels used are discrete. Each pixel can have either *on* or *off* state.

The circuitry of the video display device of the computer is capable of converting binary values (0, 1) into a pixel *on* (1) and pixel *off* (0) information. Using this ability graphics computer represent picture having discrete dots. Any model of graphics can be reproduced with a dense matrix of dots or points.

Examples of objects which can be scan converted

1. Point
2. Line
3. Sector
4. Arc
5. Ellipse
6. Rectangle
7. Polygon
8. Characters
9. Filled Regions

The process of converting is also called as *rasterization*. The algorithms implementation varies from one computer system to another computer system. Some algorithms are implemented using the software. Some are performed using hardware or firmware. Some are performed using various combinations of hardware, firmware and software.

2.1 Plotting Point

The term *pixel* is a short form of the *picture element*. It is also called a *point or dot*. *It is the smallest picture unit accepted by display devices*. A picture is constructed from hundreds of such pixels. Pixels are generated using commands. Lines, circle, arcs, characters; curves are drawn with closely spaced pixels. To display the digit or letter matrix of pixels is used.

The closer the dots or pixels are, the better will be the quality of picture. Closer the dots are, crisper will be the picture. Picture will not appear jagged and unclear if pixels are closely spaced. So the quality of the picture is directly proportional to the density of pixels on the screen.

Pixels are also defined as the *smallest addressable unit or element of the screen*. Each pixel can be assigned an address as shown in fig.6:

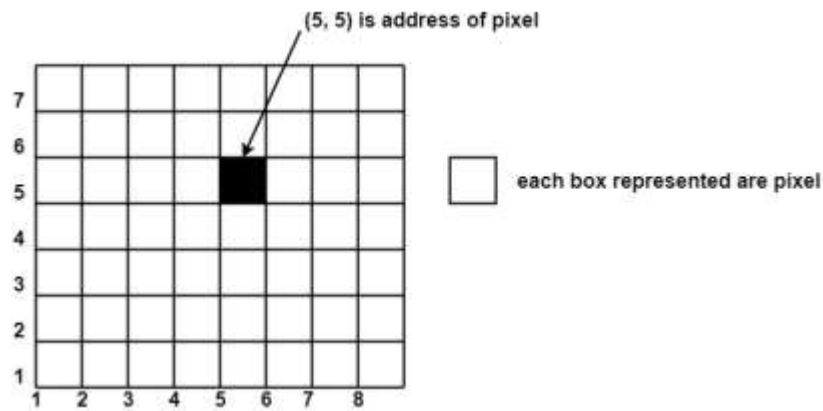


Figure 6: Address of a pixel

Each pixel has some co-ordinate value. The *coordinate* is represented using row and column.

$P(5,5)$ used to represent a pixel in the 5th row and the 5th column. Each pixel has some intensity value which is represented in memory of computer called a **frame buffer (refresh buffer)**. This memory is a *storage area for storing pixels values using which pictures are displayed*. It is also called as **digital memory**. Inside the buffer, image is stored as a pattern of binary digits either 0 or 1. So there is an array of 0 or 1 used to represent the picture. In black and white monitors, *black* pixels are represented using 1's and *white* pixels are represented using 0's. In case of systems having *one bit per pixel* frame buffer is called a *bitmap*. In systems with *multiple bits per pixel* it is called a *pixmap*.

Each pixel is accessed by a positive integer (x,y) coordinate pair, the value **x** start at origin 0, and increase from left to right, the **y** value start at 0 increases from top to bottom, as in figure (7).

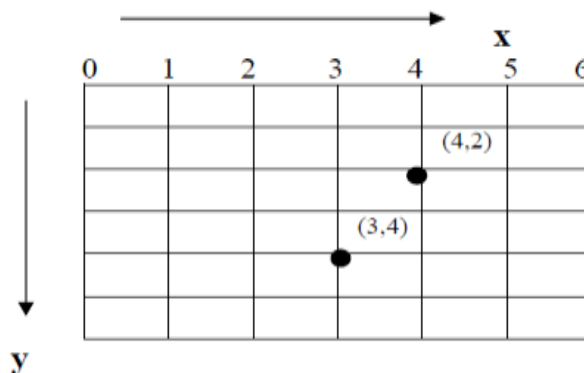
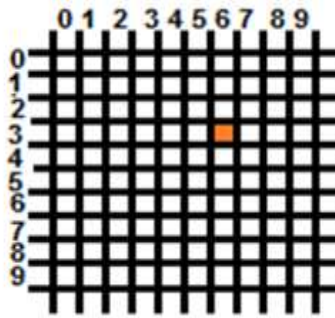


Figure 7: Screen of Positive Integer (x,y) Coordinate

To draw a point on the display screen, a point plotting procedure is required. We assume the availability of the command:

Putpixel (x , y , color)

Ex.: Putpixel (6 , 3 , 3)



The drawing on the screen starts from top to down, and from left to right. The pixel coordinates on the screen (VGA 640×480) is shown in figure 8 below:

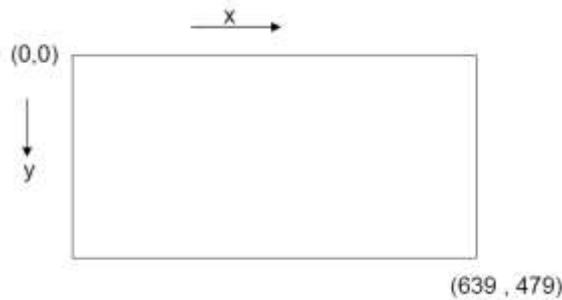


Figure 8: Pixel coordinates on the screen (VGA 640×480)

2.2 Line Drawing Algorithms

A straight line may be defined by two endpoints & an equation. To draw a line, you need two points between which you can draw a line. In fig.9 the two endpoints are described by (x_1,y_1) and (x_2,y_2) . The equation of the line is used to determine the x, y coordinates of all the points that lie between these two endpoints.

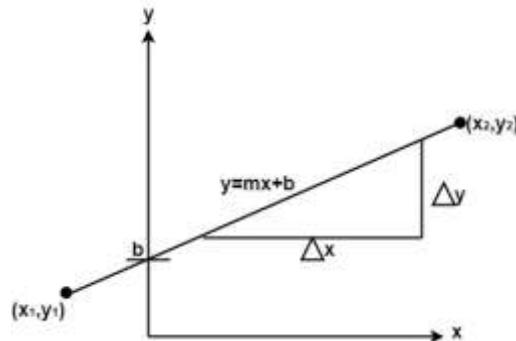


Figure 9: Two endpoints of a line

Using the equation of a straight line, $y = mx + b$ where $m = \frac{\Delta y}{\Delta x}$ and b = the y intercept, we can find values of y by incrementing x from $x = x_1$, to $x = x_2$. By scan-converting these calculated x , y values, we represent the line as a sequence of pixels.

In order to draw a line, it is necessary to determine which pixels lie nearest the line and provide the best approximation to the desired line. The accuracy and quality of the displayed line depends on the resolution of the display device. High resolution displays draw lines that look straight and continue and start and end accurately. Lower resolution displays may draw lines with gaps figure 10.

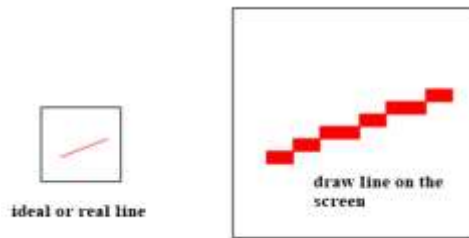


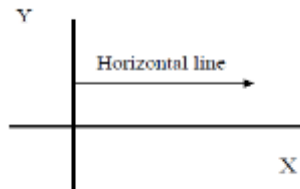
Figure 10: Low and high resolution line

2.2.1 Draw Horizontal Line

To draw horizontal line, the y value is fixed and the value x varies. The following codes draw horizontal line from (x_{start}, y) to (x_{end}, y) .

```
for x := xstart to xend do
  putpixel ( x , y , white ) ;
```

If $x_{start} > x_{end}$ then (to) in the (for) loop must be replaced by $down to$.

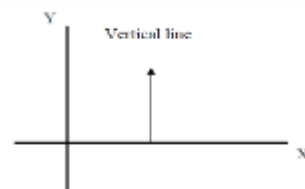


2.2.2 Draw Vertical Line

To draw vertical line, the x value is fixed and y value varies. The following codes draw a vertical line from (x, y_{start}) to (x, y_{end}) .

```
for y := ystart to yend do
  putpixel ( x , y , white ) ;
```

If $y_{start} > y_{end}$ then (to) in the (for) loop must be replaced by $down to$.



2.2.3 Draw Diagonal Line

To draw a diagonal line with a **slope equal to +1**, we need only repeatedly *increment by one* unit **both** the **x** and **y** values from the starting to the ending pixels. **The slope is defined as the change in y value divided by change in x values.**

$$\text{Slope} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$$

The following codes draw a **diagonal** line:

```
x := xstart ;
y := ystart ;
i := 0 ;
while ( x + i ) <= xend do
  begin
    putpixel ( x + i , y + i , white ) ;
    i := i + 1 ;
  end ;
```

To draw a line with a slope -1 , replace $y+i$ by $y-i$ in the code above.

Example

Show the tracing to draw a diagonal line from (0 , 0) to (3 , 3).

Solution:

Point 1 (0 , 0) , point 2 (3 , 3).

1	(x+i , y+i)
0	(0,0)
1	(1,1)
2	(2,2)
3	(3,3)
4	stop

Note: if the slope is zero then line is a horizontal $dy(y_2 - y_1) = (y_1 - y_2) = 0$ and if slope is undefined then line is a vertical $dx(x_2 - x_1) = (x_1 - x_2) = 0$ but if slope line is not equaled +1 or -1 then use one of the three algorithms below to draw diagonal lines:

- A) Direct use of line equation $Y = mx + b$.
- B) Simple DDA (Digital Differential Analyzer)
- C) Bresenham's algorithm

2.3 Arbitrary Line Algorithm

The display screen can be illuminated only at pixels locations; therefore a raster scan display has a staircase effect that only approximates the actual line as shown in figure 11:

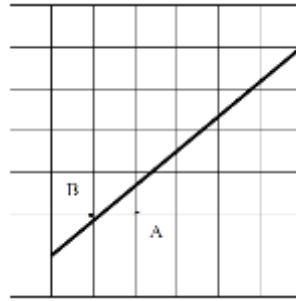


Figure 11: Arbitrary Slopee Line

2.3.A Using line equation “ $Y=mX+b$ ” (Direct Method)

In this method, draw a line between two points by drawing a group of pixels using the command **putpixel (x, y, color)**, with substituting in straight line equation’s : $Y = m \times X + b$, where (m) is the slope and (b) is a constant which represents the y interrupt figure 12. The constant **b** is found as: if the endpoints of line are (x_1, y_1) (x_2, y_2) then the $b = y_1 - mx_1$ or $b = y_2 - mx_2$.

Direct method for drawing lines can be shown in **algorithm** below:

Input $X_{start}, Y_{start}, X_{end}, Y_{end}$

Output: Arbitrary line

```
{
m=  $\frac{y_{end}-y_{start}}{x_{end}-x_{start}}$ 
b=  $Y_{start} - m * X_{start}$ ;
for x=  $X_{start}$  to  $X_{end}$ 
{
y= $m*x+b+0.5$ ;
putpixel(x,y,color); } }
```

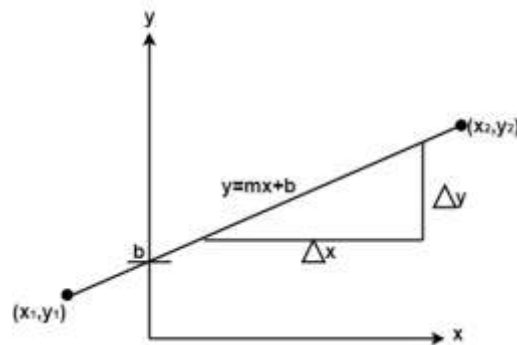


Figure 12: Drawing line using straight line equation

Example: A line with starting point as (0, 0) and ending point (6, 18) is given. Calculate value of intermediate points and slope of line.

Solution: P₁ (0,0) P₇ (6,18)

$$x_1=0$$

$$y_1=0$$

$$x_2=6$$

$$y_2=18$$

$$M = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{18 - 0}{6 - 0} = \frac{18}{6} = 3$$

The equation of line is

$$y = m x + b$$

$$y = 3x + b \dots \dots \dots \text{equation (1)}$$

put value of x from initial point in equation (1), i.e., (0, 0) x =0, y=0

$$0 = 3 * 0 + b$$

$$0 = b \Rightarrow \mathbf{b=0}$$

put b = 0 in equation (1)

$$y = 3x + 0$$

$$\mathbf{y = 3x}$$

Now calculate intermediate points

$$\text{Let } x = 1 \Rightarrow y = 3 * 1 \Rightarrow y = 3$$

$$\text{Let } x = 2 \Rightarrow y = 3 * 2 \Rightarrow y = 6$$

$$\text{Let } x = 3 \Rightarrow y = 3 * 3 \Rightarrow y = 9$$

$$\text{Let } x = 4 \Rightarrow y = 3 * 4 \Rightarrow y = 12$$

$$\text{Let } x = 5 \Rightarrow y = 3 * 5 \Rightarrow y = 15$$

$$\text{Let } x = 6 \Rightarrow y = 3 * 6 \Rightarrow y = 18$$

So points are:

$$P_1 (0,0)$$

$$P_2 (1,3)$$

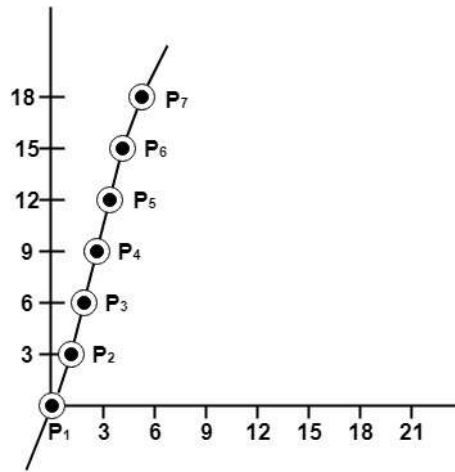
$$P_3 (2,6)$$

$$P_4 (3,9)$$

$$P_5 (4,12)$$

$$P_6 (5,15)$$

$$P_7 (6,18)$$



Example: trace the line equation to draw the line that endpoints are (1, 5), (7, 2), and draw the line in screen coordinate.

Solution:

$$dx=7-1=6; \quad dy=2-5= -3; \quad m= -3/6= -1/2 =-0.5; \quad b=5- (-0.5)*1=5.5$$

dx>0 then increment X

X	Y	Point(X,Y)	Plot(X,Y)
1	$1*-0.5+5.5$	(1,5)	(1,5)
2	$2*-0.5+5.5$	(2,4.5)	(2,5)
3	$3*-0.5+5.5$	(3,4)	(3,4)
4	$4*-0.5+5.5$	(4,3.5)	(4,4)
5	$5*-0.5+5.5$	(5,3)	(5,3)
6	$6*-0.5+5.5$	(6,2.5)	(6,3)
7	$7*-0.5+5.5$	(7,2)	(7,2)

2.3.B DDA Algorithm <https://www.youtube.com/watch?v=Qyt1ccpm1hY>

Digital Differential Analyzer (DDA) algorithm is an incremental method of scan conversion of line. In this method calculation is performed at each step but by using results of previous steps. It work on the principle of simultaneously incrementing **x** and **y** by small steps proportional to the first derivatives of **x** and **y**. The *slope* a line between the two points (x1,y1), (x2,y2) is given by $m=(y2-y1)/(x2-x1)$.

The algorithm starts with the initial values $x=x1$ & $y=y1$, the coordinates are then incremented by Δy and Δx respectively to find the *next points coordinates*. The value of **x** and **y** are *rounded to integers* and a pixel is set at that point. This step is repeated until the second end point (x2, y2) is reached.

```

// Input: x1, y1, x 2, y2.
// Output: Arbitrary line
1: dx=x2-x1:dy=y2-y1
// approximate the line length
2: if (abs(dx)>abs(dy)) then length=abs(dx)
   else length=abs(dy)
// select the larger of Δy or Δx to be raster unit
3: xinc=dx/length: yinc=dy/length
4: x=x1: y=y1
5: for i=0 to length
   5.1: putpixel(x, y,color)
   5.2: x=x +xinc : y=y +yinc
6: next i

```

Explain DDA algorithm

1- Read *start* and *end* coordinates

$(X_0, Y_0), (X_n, Y_n)$

Step1: calculate $\Delta X, \Delta Y, M$

$$\Delta X = X_n - X_0$$

$$\Delta Y = Y_n - Y_0$$

$$M = \Delta X / \Delta Y$$

Step2: find number of points between start and end coordinates.

If $(\text{abs}(\Delta X) > \text{abs}(\Delta Y))$

Length = k = $\text{abs}(\Delta X)$

Else

k = $\text{abs}(\Delta Y)$

Step3: current point (X_p, Y_p) , next point (X_{p+1}, Y_{p+1}) , find the next point by following 3 cases:

Case1: if $M < 1$

$$X_{p+1} = (1 + X_p)$$

$$Y_{p+1} = (M + Y_p)$$

Case2: if $M = 0$

$$X_{p+1} = (1 + X_p)$$

$$Y_{p+1} = (1 + Y_p)$$

Case3: if $M > 1$

$$X_{p+1} = (1/m + X_p)$$

$$Y_{p+1} = (1 + Y_p)$$

Example

Start point (5,6), end point(8,12)

Solution

1- $\Delta X=8-5=3$

$\Delta Y=12-6=6$

$M=\Delta Y/\Delta X = 6/3=3$

2- Calculate number of points

$|\Delta X| < |\Delta Y|$

$3 < 6$

Length = $k = \Delta Y = 6$

3- **M > 1** satisfied

$X_{p+1}=1/m + X_p$

$Y_{p+1}=1+Y_p$



$X_{p+1}=1/2 + X_p$

$Y_{p+1}=1+Y_p$

X_0	Y_0	X_{P+1}	Y_{P+1}	Round(X_{p+1}, Y_{p+1})
5	6	5.5	7	(6,7)
		$0.5+5.5=6$	8	(6,8)
		$0.5+6=6.5$	9	(7,9)
		$0.5+6.5=7$	10	(7,10)
		$0.5+7=7.5$	11	(8,11)
		$0.5+7.5=8$	12	(8,12)
			stop	

Where:

X_p = previous

X_{p+1} = next point

Example: Draw a line from (0,0) to (5,5) using DDA.

Solution:

$x_1=0, y_1=0, x_2=5, y_2=5.$

$abs(x_2 - x_1)=5, abs(y_2 - y_1)=5.$

Length=5.

$$\Delta x = \frac{(x_2 - x_1)}{length} = \frac{5}{5} = 1$$

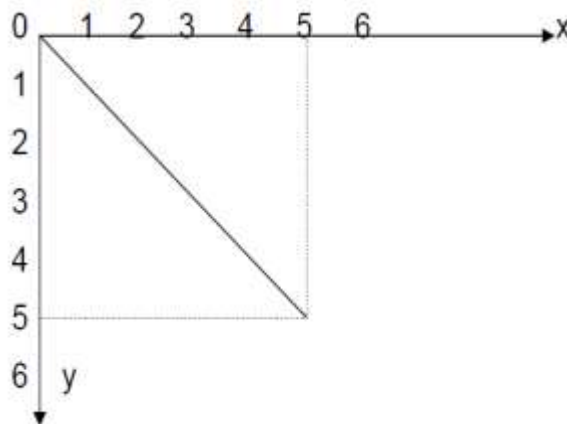
$$\Delta y = \frac{(y_2 - y_1)}{length} = \frac{5}{5} = 1$$

$x=x_1+0.5 \times \text{sign}(\Delta x)=0+0.5 \times \text{sign}(1)=0.5$

$y=y_1+0.5 \times \text{sign}(\Delta y)=0+0.5 \times \text{sign}(1)=0.5$

Incrementing through the main loop yields:

i	plot	x	y
		0.5	0.5
1	(1,1)		
		1.5	1.5
2	(2,2)		
		2.5	2.5
3	(3,3)		
		3.5	3.5
4	(4,4)		
		4.5	4.5
5	(5,5)		
		5.5	5.5
6	stop		



Example: Trace the simple DDA algorithm to draw a line between the two points (23,33), (29,40).

Solution:

$$dx=29-23=6$$

$$dy=40-33=7$$

$$\text{Length}=7$$

$$X_{\text{inc}}=6/7 =0.857$$

$$Y_{\text{inc}}=7/7=1.$$

X	Y	Plot(X)	Plot(Y)
23	33	23	33
23.857	34	24	34
24.714	35	25	35
25.571	36	26	36
26.429	37	26	37
27.286	38	27	38
28.143	39	28	39

Example: If a line is drawn from (2, 3) to (6, 15) with use of DDA. How many points will needed to generate such line?

Solution: P1 (2,3) P11 (6,15)

$$x_1=2$$

$$y_1=3$$

$$x_2= 6$$

$$y_2=15$$

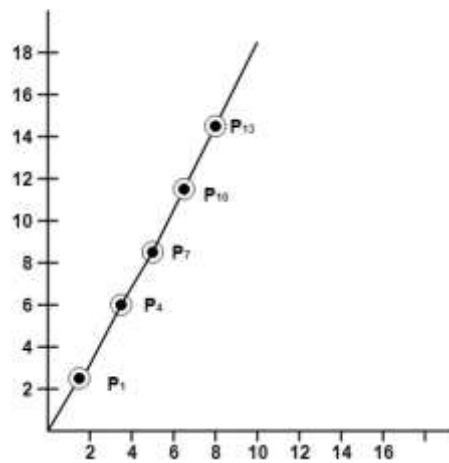
$$dx = 6 - 2 = 4$$

$$dy = 15 - 3 = 12$$

$$m = dy/dx = 12/4 = 3$$

For calculating next value of x takes $x = x + \frac{1}{m}$

$P_1(2, 3)$	point plotted
$P_2(2\frac{1}{3}, 4)$	point plotted
$P_3(2\frac{2}{3}, 5)$	point not plotted
$P_4(3, 6)$	point plotted
$P_5(3\frac{1}{3}, 7)$	point not plotted
$P_6(3\frac{2}{3}, 8)$	point not plotted
$P_7(4, 9)$	point plotted
$P_8(4\frac{1}{3}, 10)$	point not plotted
$P_9(4\frac{2}{3}, 11)$	point not plotted
$P_{10}(5, 12)$	point plotted
$P_{11}(5\frac{1}{3}, 13)$	point not plotted
$P_{12}(5\frac{2}{3}, 14)$	point not plotted
$P_{13}(6, 15)$	point plotted



2.3.C Bresenham's Line Drawing Algorithm

It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly. In this method, next pixel selected is that one who has the least distance from true line. *This algorithm is designed so that each iteration changes one of the coordinate values by ± 1 .* The other coordinate *may or may not change* depending on the value of an **error** term maintained by the algorithm. This **error term** record the **distance** measure perpendicular to the axis of greatest movement between the exact path of the line and the actual dots generated.

If the x-axis is the axis of greatest movement, then at each **iteration the x coordinates of the line is incremented**, and **the slope of the line by dy/dx is added to the error term** fig.(13.b.) Whether to increment the y coordinate of the current point, a **positive e value indicates that the exact path of the line lies above the current point**, therefore the y coordinate is incremented, and 1 is decremented from e .

<https://www.youtube.com/watch?v=ujyCaZJIDcg>

If e is negative the y coordinate **value is left unchanged**, and vice verse if y-axis is greatest movement. <https://www.youtube.com/watch?v=h3gDB89h0os&t=24s> (example)

Note:-If X-axis is greatest movement the initial $e = dy/dx$. But If Y-axis is greatest movement the initial $e = dx/dy$

For example, as shown in figure (13.a), from position (2, 3) you need to choose between (3, 3) and (3, 4). You would like the point that is closer to the original line.

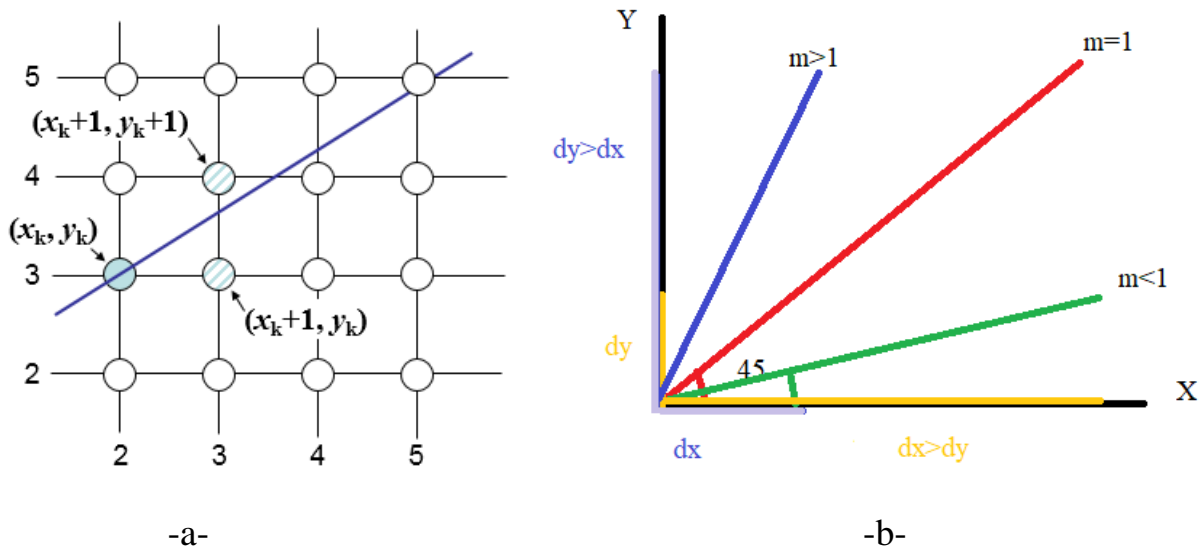


Figure 13: a: Choose between two
b: the slope

As an example, a line in the first quadrant (line with slope between 0 and 1) see figure 14.

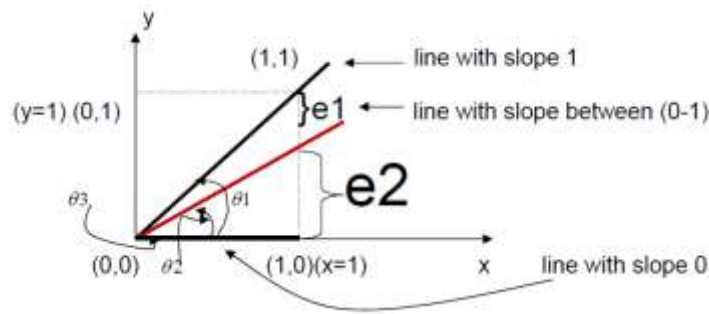
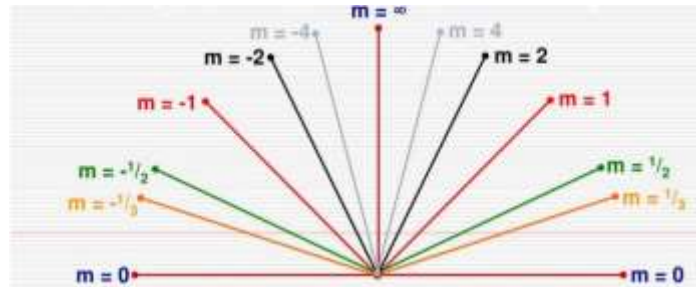


Figure 14: Slope between 0 and 1



This algorithm is **X-axis greatest** movement and **slope is positive**.

- 1: $dx=x_2-x_1$: $dy=y_2-y_1$: $e=(dy/dx)-0.5$: $x=x_1$: $y=y_1$;
- 2: for $i=0$ to $|dx|$
 - 2.1: plot pixel (x, y)
 - 2.2: if $(e>0)$ then if $(y_1>y_2)$ $y=y-1$
else
 $y=y+1$
 $e=e-1$
 - 2.3: if $(x_1>x_2)$ then $x=x-1$
else $x=x+1$
 - 2.4: **$e=e+ (dy/dx)$**
- 3: next i

Conclusion:

- 1- If the slope of the required line through $(0,0) > 0.5$, then it is intercept with line $x=1$, and will be closer to the line $y=1$ than to the line $y=0$. Hence, the screen point at $(1,1)$ is better to represent the path of the line than that at $(1,0)$.
- 2- If the slope of the required line through $(0,0) < 0.5$ then the opposite is true.
- 3- For the slope= 0.5 , the algorithm chooses point $(1,1)$ to plot.

Example:

By using Bresenham algorithm draw a line by used the following points (0 , 0) and (5 , 3).

Solution:

$x=0, y=0, \Delta x=5, \Delta y=3 \quad m=3/5=0.6 \quad e=dy/dx - 0.5$

i	plot	x	y	e
		0	0	$0.1 \rightarrow 0.6-0.5$
1	(0 , 0)		1	-0.9
		1		-0.3
2	(1 , 1)	2		0.3
3	(2 , 1)		2	-0.7
		3		-0.1
4	(3 , 2)	4		0.5
5	(4 , 2)		3	-0.5
		5		0.1
6	stop			

Example: Trace Bresenham algorithm to draw a line between the two points (50, 65), (59, 68).

Solution:

$dx= 59-50= 9$

$dy= 68-65= 3$

$m= dy/dx =3/9 = 0.333$

$e=0.333 - 0.5 = -0.167$

X	Y		e
50	65	-0.167	+m(2.4)
51	65	0.166	-1+m (2.4+2.2)
52	66	-0.501	+m (2.4)
53	66	-0.168	+m(2.4)
54	66	0.165	-1+m(2.4+2.2)
55	67	-0.502	+m(2.4)
56	67	-0.169	+m(2.4)
57	67	0.164	-1+m(2.4+2.2)
58	68	-0.503	

Note: This algorithm uses in slope of line positive but in negative slope replaces $e > 0$ to $e < 0$ and $e = e - 1$ to $e = e + 1$ and coordinate $y = y + 1$ to $y = y - 1$ Or $x = x + 1$ to $x = x - 1$ to obtain algorithms:

```

1: dx=x2-x1: dy=y2-y1: e=(dy/dx)+0.5: x=x1: y=y1;
2: for i=0 to |dx|
  2.1: plot pixel (x, y)
  2.2: if (e<0) then if (y1>y2) y=y-1
        else y=y+1
            e=e+1
  2.3: if(x1>x2) then x=x-1
        else x=x+1
  2.4: e=e+ (dy/dx)
3: next i

```

Example: Trace the Bresenhams algorithm to draw a line between the two points (1, 5), (7, 2).

Solution:

$dx = 7 - 1 = 6;$

$dy = 2 - 5 = -3$

$m = dy/dx = -3/6 = -0.5$

{Negative slope} uses algorithm modify of Bresenhams

$e = -0.5 + 0.5 = 0$

X	Y	e	
1	5	0	+m
2	5	-0.5	+1+m
3	4	0	+m
4	4	-0.5	+1+m
5	3	0	+m
6	3	-0.5	+1+m
7	2	0	

Example: Trace the line where end points (0,3), (2,-2) by Bresnham's Method.

Solution:

$$dx = 2 - 0 = 2, \quad dy = -2 - 3 = -5,$$

$|dy| > |dx| \rightarrow Y$ is Greatest move: Change for each step,
 $m = (dx/dy) = 2/-5 = -0.4 \rightarrow$

$m < 0$ then $e < 0$ then change X otherwise un-change X because less move,
 $e = (dx/dy) + 0.5 = 0.1, X = 0, Y = 3$

X	Y	e	Status of e
0	3	0.1	+(dx/dy)
0	2	-0.3	+1+(dx/dy)
1	1	0.3	+(dx/dy)
1	0	-0.1	+1+(dx/dy)
2	-1	0.5	+(dx/dy)
2	-2	0.1	Finish

$dx > 0$ increment X
 $dy < 0$ decrement Y

Example: Starting and Ending position of the line are (1, 1) and (8, 5). Find intermediate points.

Solution:

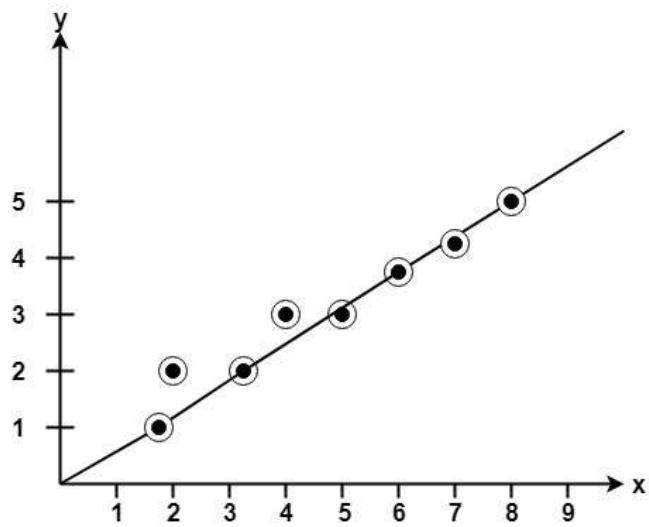
$$\begin{aligned} x_1 &= 1 \\ y_1 &= 1 \\ x_2 &= 8 \\ y_2 &= 5 \end{aligned}$$

$$\begin{aligned} dx &= x_2 - x_1 = 8 - 1 = 7 \\ dy &= y_2 - y_1 = 5 - 1 = 4 \end{aligned}$$

$$\begin{aligned} I_1 &= 2 * \Delta y = 2 * 4 = 8 \\ I_2 &= 2 * (\Delta y - \Delta x) = 2 * (4 - 7) = -6 \end{aligned}$$

$$d = I_1 - \Delta x = 8 - 7 = 1 \quad // \text{ distance variable}$$

x	y	d=d+I ₁ or I ₂
1	1	d+I ₂ =1+(-6)= -5
2	2	d+I ₁ =-5+8=3
3	2	d+I ₂ =3+(-6)= -3
4	3	d+I ₁ =-3+8=5
5	3	d+I ₂ =5+(-6)= -1
6	4	d+I ₁ =-1+8=7
7	4	d+I ₂ =7+(-6)=1
8	5	



Bresenham Algorithm(2)

$D_A \rightarrow \text{top}$

$D_B \rightarrow \text{bottom}$

$$P_0 = 2\Delta y - \Delta x$$

$$\Delta x = x_2 - x_1$$

$$\Delta y = y_2 - y_1$$

$$M = \Delta y / \Delta x$$

1) Calculate decision variable **OR** error term **e**

$$e = D_B - D_A$$

e < 0 \rightarrow when $D_B < D_A$

e >= 0 \rightarrow $D_B \geq D_A$

2) Find slope M, decision parameter **$P_0 \rightarrow 2\Delta y - \Delta x$**

M < 1		OR	M >= 1	
P < 0	P >= 0		P < 0	P >= 0
$X_{i+1} = X_i + 1$	$X_{i+1} = X_i + 1$		$X_{i+1} = X_i$	$X_{i+1} = X_i + 1$
$Y_{i+1} = Y_i$	$Y_{i+1} = Y_i + 1$		$Y_{i+1} = Y_i + 1$	$Y_{i+1} = Y_i + 1$
$P_1 = P_0 + 2\Delta Y$	$P_1 = P_0 + 2\Delta Y - 2\Delta X$		$P_1 = P_0 + 2\Delta X$	$P_1 = P_0 + 2\Delta X - 2\Delta Y$

Algorithm:

Step 1: Start

Step 2: Declare x1, y1, x2, y2.

Step 3: Calculate $dx = x_2 - x_1$
 $dy = y_2 - y_1$

Step 4: Calculate slope, $m = dy / dx$.

Step 5: For **m < 1**: Calculate initial decision variable: $P = 2dy - dx$.

Step 6: while (x1 <= x2)

if (P < 0):

$$x_k = x_k + 1$$

$$P = P + 2dy$$

$$y_k = y_k$$

else :

$$x_k = x_k + 1$$

$$P = P + 2dy - 2dx$$

$$y_k = y_k + 1$$

Step 7: Plot (x_k, y_k)

Step 8: End

Example

Start point (1,1) and end point (5,3)

Solution

$$\Delta Y=2 \quad \Delta X=4 \quad M= \Delta Y/ \Delta X = 0.5$$

$M < 1$

1) $P_0 \rightarrow 2\Delta y - \Delta x$

$$P_0=2*2-4=0 \rightarrow P \geq 0$$

$$X_{i+1}=X_i+1$$
$$Y_{i+1}=Y_i+1$$

$$X_{i+1}=1+1=2$$

$$Y_{i+1}=1+1=2$$

$$P_1=P_0+2\Delta Y-2\Delta X$$

$$P_1=0+2*2-2*4= -4 \rightarrow P < 0$$

2) $X_{i+1}=X_i+1$

$$Y_{i+1}=Y_i$$

$$X_{i+1}=2+1=3$$

$$Y_{i+1}=2$$

$$P_1=P_0+2\Delta Y$$

$$P_1= -4+2*2=0 \rightarrow P \geq 0$$

3) $X_{i+1}=X_i+1$

$$Y_{i+1}=Y_i+1$$

$$X_{i+1}=3+1=4$$

$$Y_{i+1}=2+1=3$$

$$P_1=P_0+2\Delta Y-2\Delta X$$

$$P_1=0+2*2-2*4= -4 \rightarrow P < 0$$

4) $X_{i+1}=X_i+1$

$$Y_{i+1}=Y_i$$

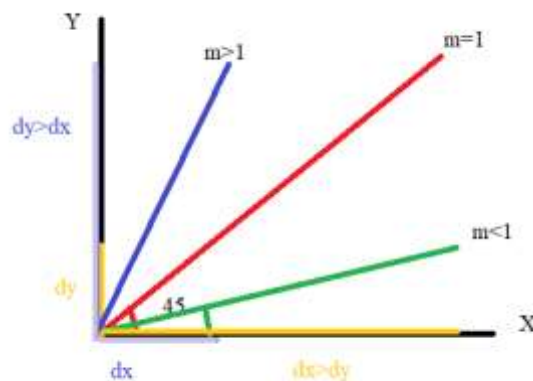
$$X_{i+1}=4+1=5$$

$$Y_{i+1}=3$$

P	X _i	Y _i	X _{i+1}	Y _{i+1}
0	1	1	2	2
-4	2	2	3	2
0	3	2	4	3
-4	4	3	5	3
Stop			Stop	

Bresenham's Line Algorithm for $|m = \Delta y / \Delta x| < 1$

1. Read the line end points (x_1, y_1) and (x_2, y_2) such that they are not equal.
// If equal, then plot that point and exit
2. $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
3. // Initialize starting point
 $x = x_1$
 $y = y_1$
4. $e = 2\Delta y - \Delta x$ //Initialize value of decision variable or error to compensate for nonzero intercepts
5. $i = 1$ //Initialize counter
6. Plot (x, y)
7. While $(e \geq 0)$ //when $e \geq 0$, error is initialized with $e = e - 2*\Delta x$. This is continued till error is negative.
In each iteration y is incremented by 1.
 $\{ y = y + 1$
 $e = e - 2\Delta x \}$
// when $e < 0$, error is initialized to $e = e + 2*\Delta y$. In both the cases x is incremented by 1
 $x = x + 1$
 $e = e + 2\Delta y$
8. $i = i + 1$
9. If $(i \leq \Delta x)$ then goto step 6
10. Stop



Example Consider the line from (5, 5) to (13, 9), use the Bresenham's algorithm to rasterize the line.

Solution: Evaluating steps 1 through 4 in the Bresenham's algorithm we have,

$$\Delta x = |13-5|=8$$

$$\Delta y = |9-5|=4$$

$$x=5, y=5.$$

$$e=2\Delta y-\Delta x$$

$$= 2*4-8 =0$$

Tabulating the results of each iteration in the step 5 through 10

i	plot	x	y	e
		5	5	0
1	(5,5)	6	6	-16
				-8
2	(6,6)	7	6	0
3	(7,6)	8	7	-16
				-8
4	(8,7)	9	7	0
5	(9,7)	10	8	-16
				-8
6	(10,8)	11	8	0
7	(11,8)	12	9	-16
				-8
8	(12,9)	13	9	0
9	(13,9)	14	10	-16
				-8

Part Three

Circle, Ellipse Drawing Algorithms

3. Circle Drawing Algorithms

Circles are the most used curves to elementary graphics. Circle is an eight-way symmetric figure. The shape of circle is the same in all quadrants. In each quadrant, there are two octants. If the calculation of the point of one octant is done, then the other seven points can be calculated easily by using the concept of eight-way symmetry. For drawing, circle considers it at the origin. If a point is $P1(x, y)$, then the other seven points will be as shown in figure 15. So we will calculate only 45° arc. From which the whole circle can be determined easily.

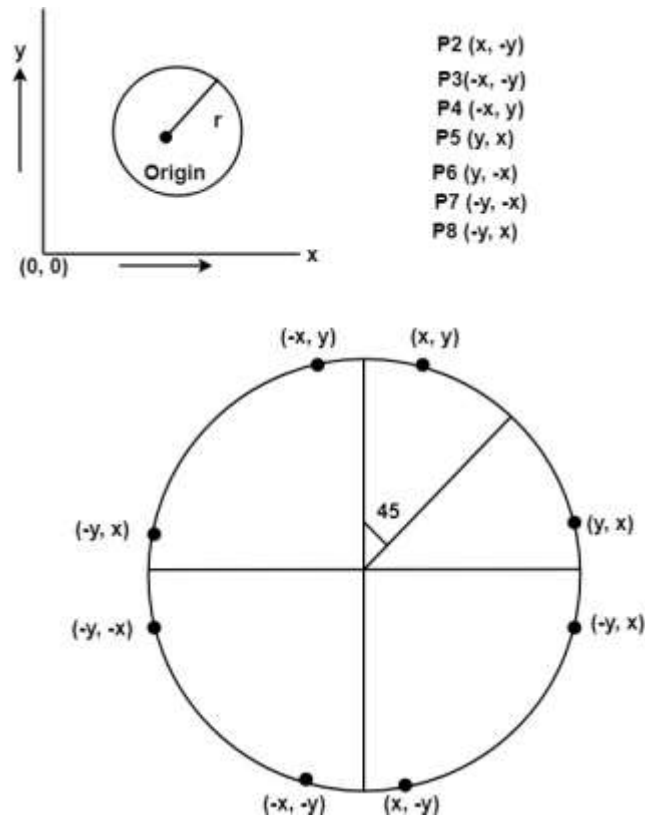


Figure 15: Eight-way symmetric of a circle

If we want to display circle on screen then the *putpixel* function is used for eight points as shown below:

```
putpixel (x, y, color)
putpixel (x, -y, color)
putpixel (-x, y, color)
putpixel (-x, -y, color)
putpixel (y, x, color)
putpixel (y, -x, color)
putpixel (-y, x, color)
putpixel (-y, -x, color)
```

There are two standard methods of mathematically defining a **circle centered at the origin**.

3.1 Defining a circle using Polynomial Method (Circle Equation)

3.2 Defining a circle using Polar Co-ordinates

3.1 Defining a circle using Polynomial Method (Circle Equation)

A circle is specified by the coordinate of its center (x_c, y_c) and its radius (r) .

• The circle equation is:

$$(x-x_c)^2 + (y-y_c)^2 = r^2 \dots\dots\dots (1)$$

If the *center* of the circle is at the *origin* $(0,0)$ fig. 16 the equation is:

$$x^2 + y^2 = r^2 \dots\dots\dots (2)$$

Solving equation (1) for y obtain $y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$

Solving equation (2) for y obtain $y = \pm \sqrt{r^2 - (x)^2}$

- $x, y \rightarrow$ point on the boundary
- $x_c, y_c \rightarrow$ the center of the circle
- $r \rightarrow$ radius

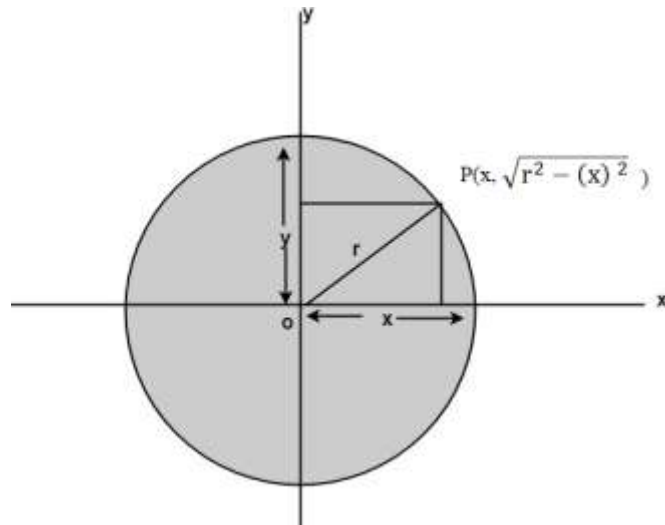


Figure 16: Center of the circle is at the origin $(0,0)$

Note: to draw a circle increment x by one unit from $-r$ to r and so the above equation to solve for the two y value at each step (convert to integer).

Example circle has center (100,-20) & radius 6 units find circle points.

Solution

x (-6 to 6)	$y1 = +\sqrt{r^2 - (x)^2}$	$y2 = -\sqrt{r^2 - (x)^2}$	$x+x_c$	$y1+y_c$	$y2+y_c$
-6	0	0	94	-20	-20
-5	$y1 = +\sqrt{6^2 - (-5)^2}$	$Y2 = -\sqrt{6^2 - (-5)^2}$	95	$-20 + \sqrt{11}$	$-20 - \sqrt{11}$
-4					
-3					
-2					
-1					
0					
1					
.					
.					
6					

This method of drawing a circle is inefficient because:

1. We are not taking advantage of symmetry of the circle.
2. The amount of processing time required to perform the squaring and square root operation repeatedly.

The algorithm in circle equation is:

```

1: x:= -r : dt:= 1/r //increment unit in circle
2: while (x<= r)
2.1: y1:= +sqrt(r*r-x*x) : y2:= -sqrt(r*r-x*x);
2.2: putpixel(xc +x, yc+y1,color) : putpixel(xc +x, yc+y2,color)
2.3 x=x+dt
3: goto 2

```

Example: Draw a circle by using circle equation, when R=5, X_c=0 , Y_c=0.

Solution:

for x: = -R To + R

$$y = \sqrt{R^2 - x^2}$$

for x:= -5 to +5

$$y = \sqrt{5^2 - x^2}$$

X	Y	Plot
-5	0	(-5,0)
-4	-3,+3	(-4,-3),(-4,3)
-3	-4,+4	(-3,-4),(-3,4)
-2		
-1		
0	-5,+5	(0,-5),(0,5)
1		
2		
3	-4,+4	(3,-4),(3,4)
4	-3,+3	(4,-3),(4,3)
5	0	(5,0)

3.2 Defining a circle using Polar Co-ordinates

The second method of defining a circle makes use of *polar coordinates* as shown in fig.17:

$$x = r \cos \theta \quad y = r \sin \theta$$

Where:

θ = current angle

r = circle radius

x = x coordinate

y = y coordinate

By this method, θ is stepped from 0 to $\frac{\pi}{4}$ & each value of x & y is calculated.

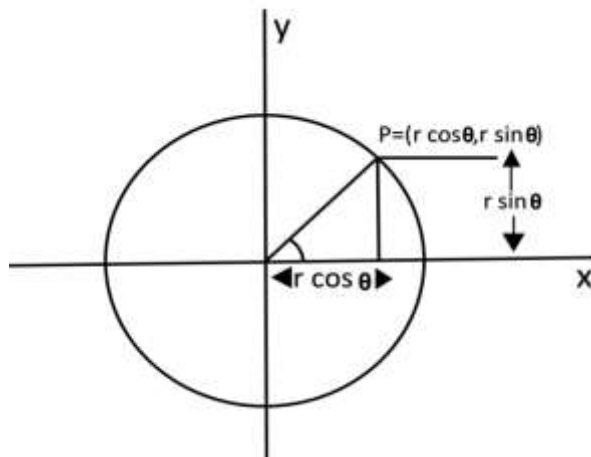


Figure 17: Polar coordinates

Polar coordinates Algorithm

Step1: Set the initial variables:

r = circle radius

(h, k) = coordinates of the circle center

i = step size

$\theta_{\text{end}} = (22/7)/4$

$\theta = 0$

Step2: if $\theta > \theta_{\text{end}}$ then stop.

Step3: Compute

$$x = r * \cos \theta \quad y = r * \sin \theta$$

Step4: Plot the eight points, found by symmetry i.e., the center (h, k) , at the current (x, y) coordinates.

Plot $(x + h, y + k)$ Plot $(-x + h, -y + k)$

Plot $(y + h, x + k)$ Plot $(-y + h, -x + k)$

Plot $(-y + h, x + k)$ Plot $(y + h, -x + k)$

Plot $(-x + h, y + k)$ Plot $(x + h, -y + k)$

Step5: increment $\theta = \theta + i$

Step6: goto step (2).

Or

1: $dt = 1/r$: $th = 0$: $pi = 22/7.0$

2: while $(th \leq 2 * pi)$

2.1: $x = r * \cos(th)$: $y = r * \sin(th)$

2.2: putpixel($x_c + x, y_c + y, color$)

2.3: $th = th + dt$

3: goto 2

Example trace the algorithm that uses the polar representation to generate eight points of the circle centered at $(300, 150)$ with a radius of 5 units.

Solution $dt = 1/r = 1/5 = 0.2$; $x = r * \cos(\Theta)$; $y = r * \sin(\Theta)$;

Θ	x	y	$x+x_c$	$y+y_c$	Plot x	Plot y
0	5	0	305	150	305	150
0.2	4.9	0.993	304.9	150.993	305	151
0.4	4.605	1.947	304.605	151.947	304	152
...	...					

3.2.1 Incremental drawing of circles

A circle is specified by the coordinates of its center (x_c, y_c) and its radius r as shown in figure 18. The most familiar equation of the circle is:

$$(x - x_c)^2 + (y - y_c)^2 = r^2, \quad y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$$

If the center of the circle is at the **origin (0,0)** the above equation reduces to:

$$x^2 + y^2 = r^2, \quad y = \sqrt{r^2 - x^2} \quad \text{for } y \geq 0.$$

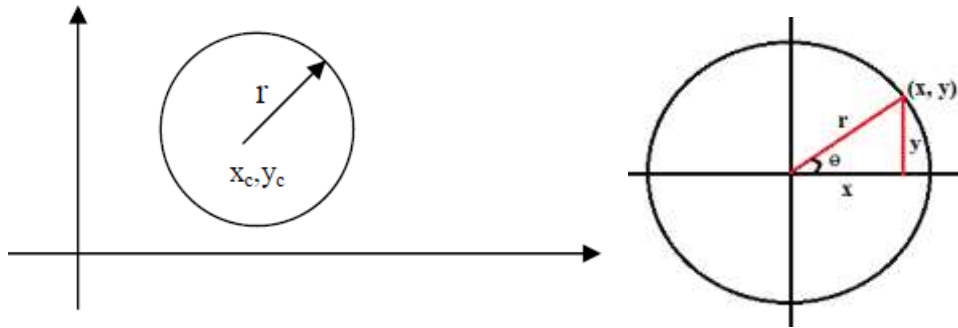


Figure 18: Circle Coordinates and Radius

Where:

x, y → are coordinates of the circle center point.

r → is the radius of the circle.

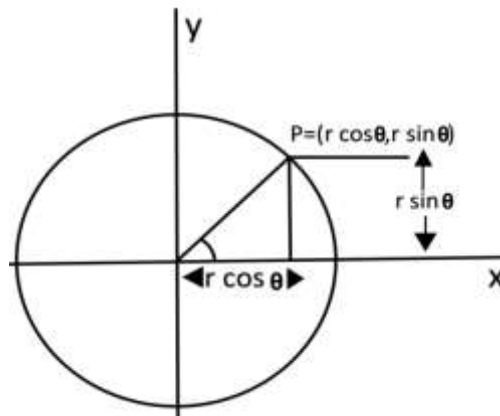
حساب احداثيات اي نقطة على محيط دائرة نصف قطرها $r =$ باستخدام دوال الجيب والجيب تمام على اعتبار ان المركز في نقطة الاصل نستخدم المعادلات ادناه:

$$\cos(\theta) = x / r$$

$$x = r \cdot \cos(\theta)$$

$$\sin(\theta) = y / r$$

$$y = r \cdot \sin(\theta)$$



One method of **eliminating the problem** of plotting points evenly spaced around the circle is to use **polar representation** of a circle:

$$x = x_c + r \cos \theta, \quad y = y_c + r \sin \theta$$

Where: $\theta \rightarrow$ is measured in radians from 0 to 2π

arc length = $r \times \theta$, r =radius (constant)

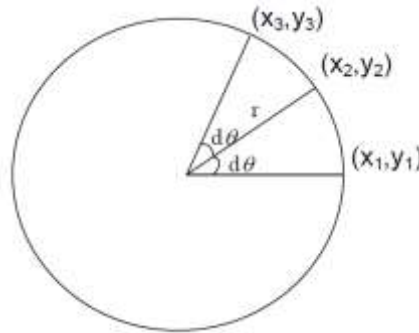


Figure 19: Circle Coordinates with equal spacing (arc length)

In this method we depend on angles to draw the circle, since it propose the first angle $\theta=0$, and end angle is (360). The change in angle ($d\theta$) must be small value $d\theta=1/r$.

Equal increments of $\theta = d\theta$ which is result in equal spacing (arc length) between successively plotted points as shown in figure 19. The **incremental equations** are:

$$x_2 = x_1 \cos d\theta - y_1 \sin d\theta$$

$$y_2 = y_1 \cos d\theta + x_1 \sin d\theta$$

To generate a circle we start at $x_1=0$, $y_1=r$, and fixed angle increment $d\theta$, we can compute all points on the circle by calculating **cos dθ** and **sin dθ** only once.

Circle may be drawn using the command putpixel() and the command circle(x,y,r). *The method of drawing a circle by increment the x values by one unit from (-r) to (+r) and use circle equation to solve y values at each step is inefficient for the following reasons:*

1-The large amount of processing time required to perform squaring and square root operations repeatedly.

2- The resulted circle is dense and flat near the y-axis, and has large gaps near x-axis. Figure 20:

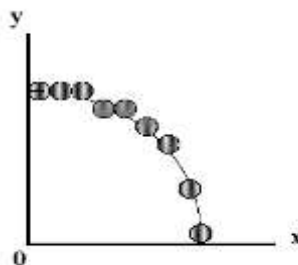


Figure 20: Dense and flat near the y-axis

The algorithm of Incremental drawing of circles is:

```

1: x=r : y=0: th=0 : dt=1/r : ct=cos(dt) : st=sin(dt)
2: while (th<= 2*pi)
  2.1: plot pixel(x +xc, y +yc)
  2.2: t=x
      x=x*ct - y*st; ( Xn+1 =Xn*cos(ΔΘ) –Yn*sin(ΔΘ))
      y=y*ct + t*st; (Yn+1 =Yn*cos(ΔΘ)+Xn*sin(ΔΘ))
  2.3: th+=dt; // represent counter
3: goto 2 // if not reach 360°
  
```

4. Symmetric of circle points

A circle is a symmetrical figure. Any circle-generating algorithms may be used to plot eight points for each value that the algorithm calculates. As shown in the figure 21, only one octant of the circle has to be generated. A successive reflection obtains the other parts.

if a point (x,y) lies on the circle $x^2+y^2=r^2$ centered at the origin, then so do seven other points: $(-x,y)$, $(x,-y)$, $(-x,-y)$, (y,x) , $(-y,x)$, $(y,-x)$, $(-y,-x)$ as shown in figure 21:

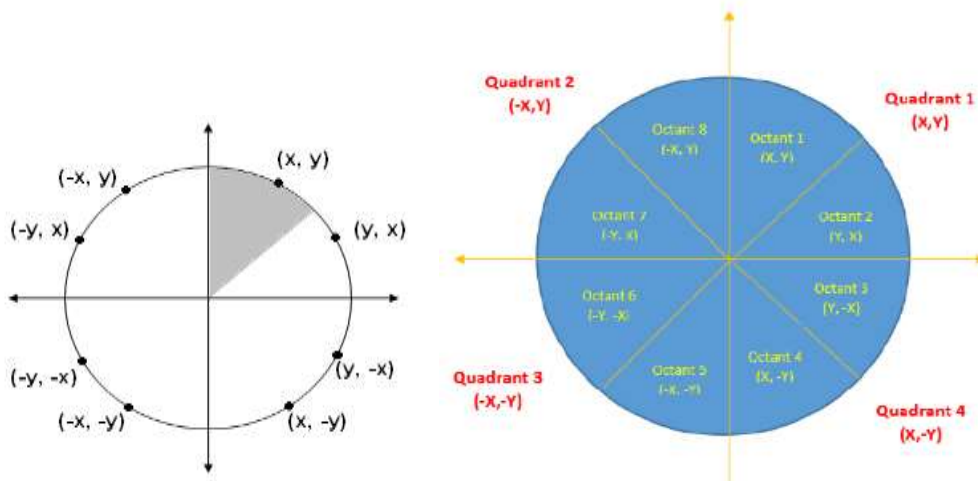


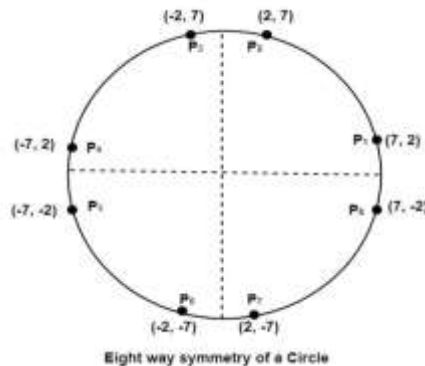
Figure 21: Eight Points in the Circumference of the Circle

$(x, y) \Rightarrow (x + x_c, y + y_c)$	First quadratic
$(y, x) \Rightarrow (y + x_c, x + y_c)$	
$(-x, y) \Rightarrow (-x + x_c, y + y_c)$	Second quadratic
$(-y, x) \Rightarrow (-y + x_c, x + y_c)$	
$(-x, -y) \Rightarrow (-x + x_c, -y + y_c)$	Third quadratic
$(-y, -x) \Rightarrow (-y + x_c, -x + y_c)$	
$(x, -y) \Rightarrow (x + x_c, -y + y_c)$	Forth quadratic
$(y, -x) \Rightarrow (y + x_c, -x + y_c)$	

To find the symmetric points on a circle centered at (x_c, y_c) add x_c to the first coordinate and y_c to the second coordinate for each of the eight points.

Example: Let we determine a point $(2, 7)$ of the circle then other points will be $(2, -7)$, $(-2, -7)$, $(-2, 7)$, $(7, 2)$, $(-7, 2)$, $(-7, -2)$, $(7, -2)$

These seven points are calculated by using the property of reflection. The reflection is accomplished by reversing x, y co-ordinates.



5. A Symmetric in increment method of circles

By advantage of symmetric points in circle the modified algorithm is:

Input: x_c, y_c, r .

Output: circle.

```

{
dθ= 1/r;
ct=cos(dθ);
st=sin(dθ);
x=0; y=r;
while ( y ≥ x)
{
putpixel(floor(xc+x),floor(yc+y),color);
putpixel(floor(xc-x),floor(yc +y),color);
putpixel(floor(xc +x),floor(yc -y),color);
putpixel(floor(xc -x),floor(yc -y),color);
putpixel(floor(xc +y),floor(yc +x),color);
putpixel(floor(xc -y),floor(yc +x),color);
putpixel(floor(xc +y),floor(yc -x),color);
putpixel(floor(xc -y),floor(yc -x),color);
xtemp=x;
x= x × ct – y × st;
y= y × ct + xtemp ×st;
}

```

6. Circle Generation – Bresenham’s Algorithm

One of the most efficient and easiest circle algorithms is Bresenham. To begin, only one octant of the circle need be generated. The other parts can be obtained by successive reflections. This is illustrated in Fig. 22. If the first octant (0 to 45°) is generated, the second octant can be obtained by reflection through the line $y=x$ to yield the first quadrant. The results in the first quadrant are reflected through the line $x=0$ to obtain those in the second quadrant.

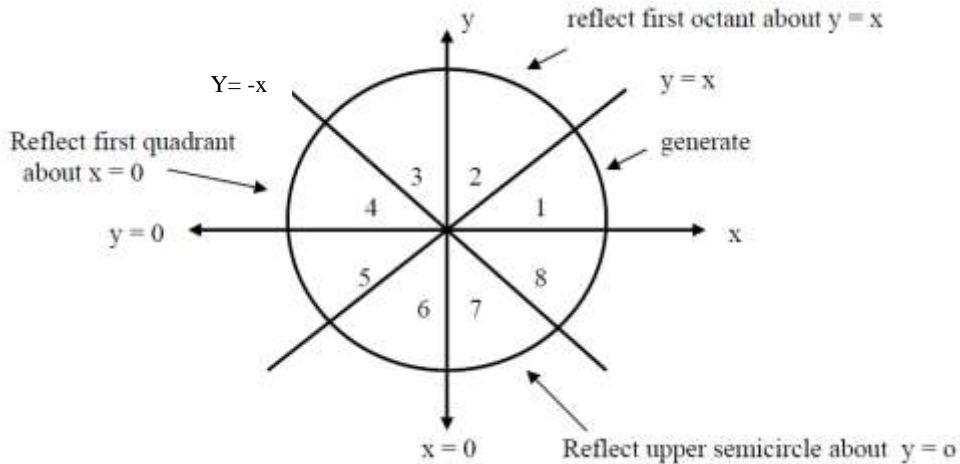


Figure 22: Generation of a Complete Circle from the First Octant

The combined result in the upper semicircle are reflected through the line $y=0$ to complete the circle. Bresenham’s Algorithm is consider the *first quadrant of an origin- centered circle*. If the algorithm begins at $x=0$, $y=R$, then for **clockwise generation** of the circle y is a monotonically decreasing function of x in the first quadrant. Here the clockwise generation starting at $x=0$, $y=R$ is chosen. The center of the circle is (0,0). See Figure 23.

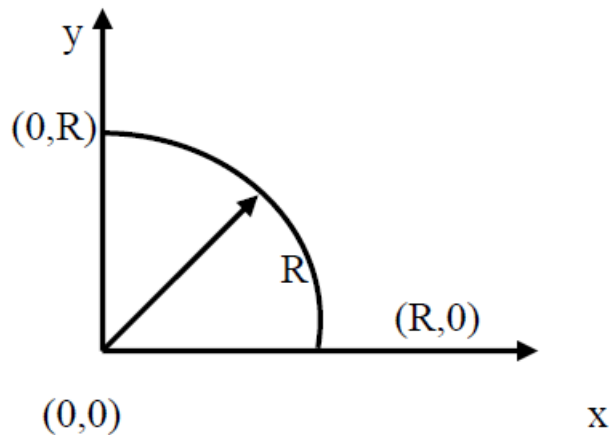


Figure 23: First Quadrant of a Circle

The circle equation, when the center (x_c, y_c) , and the radius R , is:

$$(x-x_c)^2 + (y-y_c)^2 = R^2$$

And when the center of this circle is the origin $(0, 0)$, then the equation:

$$x^2 + y^2 = R^2 \quad \text{Because of } x_c, y_c = 0.$$

We cannot display a continuous arc on the raster display. Instead, we have to choose the nearest pixel position to complete the arc.

We have put the pixel at (X, Y) location and now need to decide where to put the next pixel: at $N(X+1, Y)$ or at $S(X+1, Y-1)$. Figure 24. (N...out, S... in).

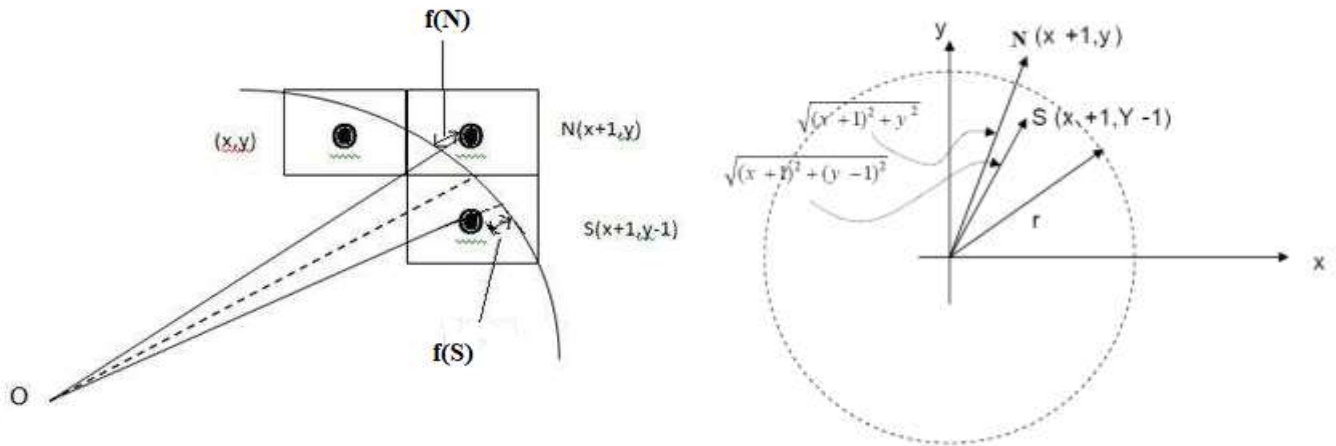


Figure 24: Choosing the Next Pixel

This can be decided by the decision parameter d .

- If $d \leq 0$, then $N(X+1, Y)$ is to be chosen as next pixel. $\sqrt{(x+1)^2 + y^2}$
- If $d > 0$, then $S(X+1, Y-1)$ is to be chosen as the next pixel. $\sqrt{(x+1)^2 + (y-1)^2}$

Algorithm

Input: r, x_c, y_c

Output: circle.

```
{
x=0; y=r; d=3-2r;
while ( x ≤ y ) {
putpixel(x+xc,y+yc,color); // plot the other 7 points
if (d<0)
d=d+4x+6;
else { d=d+4(x-y)+10; y--; }
x++
} }
```

7. Scan Converting an Ellipse

The ellipse is also a symmetric figure like a circle but is four-way symmetry rather than eight-way. Stretching a circle in one direction produces an ellipse Figure 25.

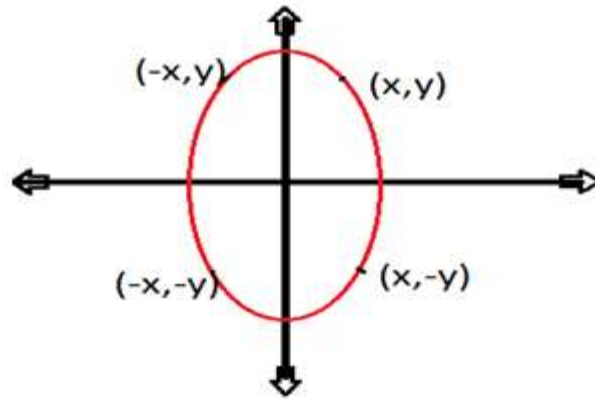


Figure 25: 4-way symmetry Ellipse

There two methods of defining an Ellipse:

1. Polynomial Method of defining an Ellipse
2. Trigonometric method of defining an Ellipse

7.1 Polynomial Method of defining an Ellipse

An ellipse is defined as the set of points such that the sum of the distances from two fixed positions (*foci*) is the same for all points. If the distances to the two *foci* from any point $P = (x, y)$ on the ellipse are labeled d_1 and d_2 , then the general equation of an ellipse can be stated as

$$d_1 + d_2 = d_3 + d_4 = \text{constant}$$

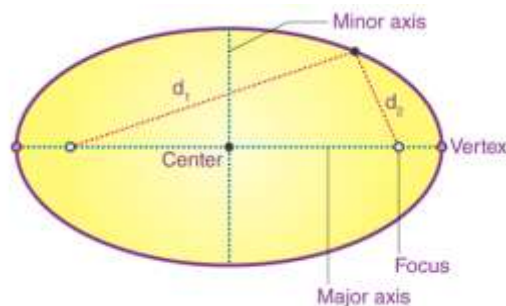


Figure 26: Polynomial Description of Ellipse

Ellipse equations are greatly simplified if the major and minor axes are oriented to align with the coordinate axes. In Fig.27, we show an ellipse in "standard position" with major and minor axes oriented parallel to the x and y-axes. Parameter r_x for this example labels the *semi major*

axis, and parameter r_y labels the *semi minor axis*. The equation of the ellipse can be written in terms of the ellipse center coordinates and parameters r_x and r_y as:

$$\frac{(x-x_c)^2}{r_x^2} + \frac{(y-y_c)^2}{r_y^2} = 1$$

Where (x_c, y_c) = ellipse center

r_x =length of major axis (radius x).

r_y =length of minor axis (radius y).

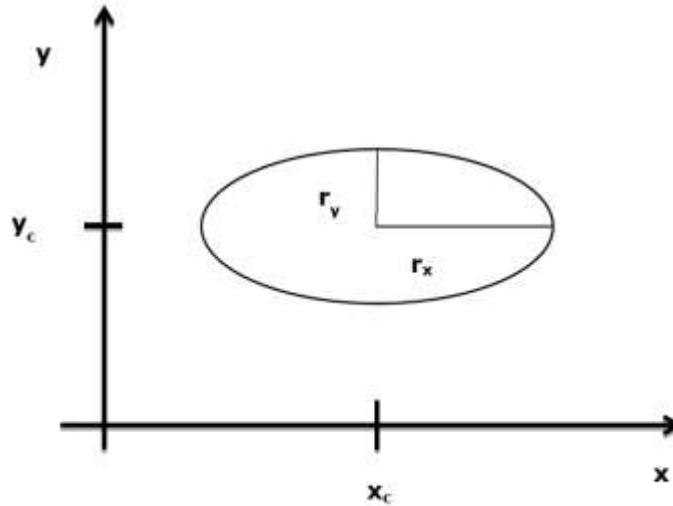


Figure 27: Ellipse centered at (x_c, y_c) with semi-major axis r_x and semi-minor axis r_y

When the polynomial method is used to define an ellipse, the value of x is incremented from $-a$ to a . For each step of x , each value of y is found by evaluating the expression:

$$y = y_c \pm b1 \sqrt{1 - \frac{(x - x_c)^2}{a1^2}}$$

Algorithm:

- 1: $dt=1/((r_x+r_y)/2) : x= -r_x;$
- 2: while $(x \leq r_x)$
 - 2.1: $y1= +b*((1- (x^2)) / (r_x^2))^{(1/2)}$
 - 2.2: $y2= - b*((1- (x^2)) / (r_x^2))^{(1/2)}$
 - 2.3: plot pixel($x_c +x, y_c+ y1$) : plot pixel($x_c +x, y_c+ y1$) // add 2 points
 - 2.4: $x=x+dt$
- 3: goto 2

Note if $a=b$ then polynomial convert same as circle equation

7.2 Trigonometric method of defining an Ellipse

The polar equation for an ellipse centered at (x_c, y_c) and r_x is the radius on the x-axis and r_y is the radius on the y-axis.

$$x = x_c + r_x * \cos(\Theta)$$

$$y = y_c + r_y * \sin(\Theta)$$

The angle Θ assumes value from 0 to 2π radius, the values of r_x and r_y affect the shape of the ellipse, if $r_y > r_x$ the ellipse is longer in the y direction, if $r_x = r_y$ the equation produces a circle.

Then algorithm is:

```
1: dt=1/ ((r_x +r_y)/2) : th=0 : pi=22/7.0
2: do while (th<=2*Pi)
    2.1: x= x_c+r_x*cos(th) :    y= y_c+r_y*sin(th)
    2.2: plot pixel(x,y), 8
    2.3: th=th+dt
3: goto 2
```

Part Four

2D

Transformation

4.1 Two Dimensional Geometric Transformations

Geometric transformations mean changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotations, reflection and shearing. 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.

2D Transformations are:

1. Homogeneous coordinates
2. Matrices
3. Transformations
4. Geometric Transformations

4.1.1 Homogeneous coordinates

There are three types of co-ordinate systems:

1. Cartesian Co-ordinate System
 - ✓ Left Handed Cartesian Co-ordinate System(Clockwise)
 - ✓ Right Handed Cartesian Co-ordinate System (Anti Clockwise)
2. Polar Co-ordinate System
3. Homogeneous Co-ordinate System

We can always change from one co-ordinate system to another.

- A point (x, y) can be re-written in *homogeneous coordinates* as (x_h, y_h, h)
- The homogeneous parameter h is a non-zero value such that:
$$X = X_h/h \quad Y = Y_h/h$$
- We can then write any point (x, y) as (hx, hy, h)
- We can conveniently choose $h = 1$ so that (x, y) becomes $(x, y, 1)$

4.1.2 Matrices

• **Definition:** A matrix is an $(n \times m)$ array of scalars, arranged conceptually in n rows and m columns, where n and m are positive integers. We use A, B, and C to denote matrices.

• If $n = m$, we say the matrix is a *square matrix*.

• We often refer to a matrix with the notation:

$\mathbf{A} = [\mathbf{a}(i,j)]$, where $a(i,j)$ denotes the scalar in the i^{th} row and the j^{th} column

• Note that the text uses the typical mathematical notation where the i and j are subscripts. We'll use this alternative form as it is easier to type and it is more familiar to computer scientists.

❖ **Scalar-matrix multiplication:**

$$\alpha A = [\alpha a(i,j)]$$

❖ **Matrix-matrix addition:** A and B are both $n \times m$

$$C = A + B = [a(i,j) + b(i,j)]$$

❖ **Matrix-matrix multiplication:** A is $n \times r$ and B is $r \times m$

$$C = AB = [c(i,j)] \text{ where } c(i,j) = \sum_{k=1}^r a(i,k) b(k,j)$$

❖ Each point $P(x,y)$ in the *homogenous matrix* form is represented as

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad 3 \times 1$$

❖ Recall matrix multiplication takes place:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ h & i & j \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a*x+b*y+c*z \\ d*x+e*y+f*z \\ h*x+i*y+j*z \end{bmatrix}$$

4.1.3 Transformations

A **transformation** is a function that maps a point (or vector) into another point (or vector).

A– Geometric Transformations: In Geometric transformation an object itself is moved relative to a stationary coordinate system or background. The mathematical statement of this view point is described by geometric transformation applied to each point of the object.

B– Coordinate Transformation: The object is held stationary while coordinate system is moved relative to the object. These can easily be described in terms of the opposite operation performed by Geometric transformation.

Types of Transformations:

1. Translation
2. Scaling
3. Rotating
4. Reflection
5. Shearing

1. Translation

A translation moves an object to a different position on the screen. A point (x, y) is translated to a new position (x', y') by move it **H** units in the horizontal direction and **V** units in the vertical direction. The translation pair (H, V) is called as *shift vector*. Here the object is positioned from one coordinate location to another. Figure 15.

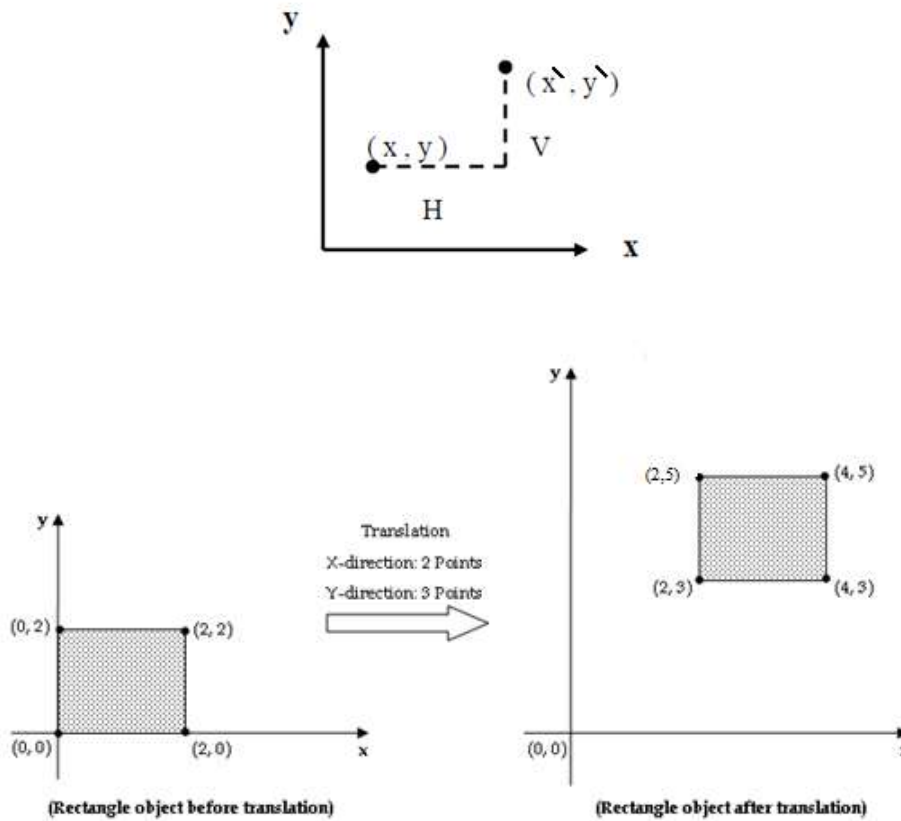


Figure 15: Horizontal and vertical displacement

Mathematically this can be represented as:

$$\begin{aligned} X' &= X + H \\ Y' &= Y + V \end{aligned}$$

The **H** and **V** represent the horizontal and vertical displacement or distance that the point has moved. If **H** is positive, the point moves to the right, if **H** is negative the point moves to the left, similarly, a point **V** moves the point up, a negative **V** moves it down. To move object we must translate every point describing the object.

To translate an object in an image we must translate every point defining the object. All point, are displaced the same distance and the object is draw using these transformed points.

Note1: using coordinate system the translating factor are:

If $H > 0$ then point moves to the right.

If $H < 0$ then point moves to the left.

If $V > 0$ then point moves to the up.

If $V < 0$ then point moves to the down.

Note2: using coordinate system in screen then translating factor are:

If $H > 0$ then point moves to the right.

If $H < 0$ then point moves to the left.

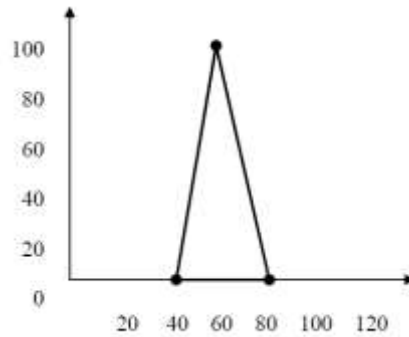
If $V > 0$ then point moves to the down.

If $V < 0$ then point moves to the up.

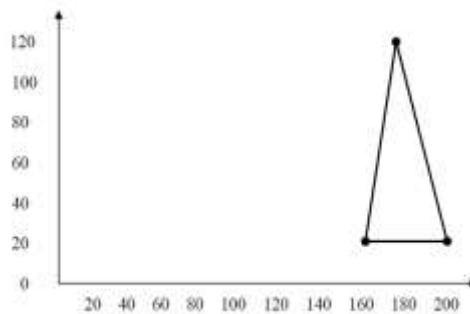
Example

Consider a triangle defined by its three vertices $(40, 0)$, $(80, 0)$, $(60, 100)$ be translated 120 units to the right and 20 units up.

Solution:



$H = 120$, $V = 20$. The **new** vertices are: $(160, 20)$, $(200, 20)$, $(180, 120)$



$$\begin{bmatrix} 40 & 0 \\ 80 & 0 \\ 60 & 100 \end{bmatrix} \xrightarrow{\substack{\text{rightup} \\ (120, 20)}} \begin{bmatrix} 160 & 20 \\ 200 & 20 \\ 180 & 120 \end{bmatrix}$$

before

after

Matrix for Translation:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ tx & ty & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

2. Scaling

To change the size of an object, scaling transformation is used. We can change the size of an object, or the entire image, by *multiplying* the distance between points by an enlargement or reduction factor. This factor is called the “**scaling factor**“, and the operation that changes the size is called **scaling**. If the scaling is greater than **1**, the object is enlarge, if the factor is less than **1**, the object is made smaller, a factor of 1 has no effect on the object. Whenever scaling is performed, there is one point that remains at the same location. This is called the **fixed point** of the scaling transformation.

$$x' = x * S_x$$

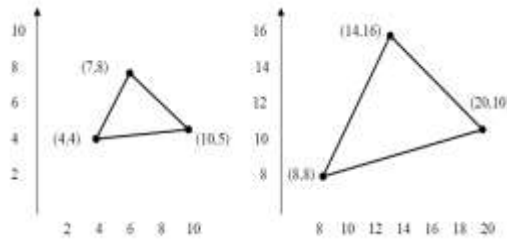
$$y' = y * S_y$$

Example

Scale the triangle (4 , 4) , (7 , 8) , (10 , 5) by $S_x = 2$ and $S_y = 2$, about the origin point.

Solution:

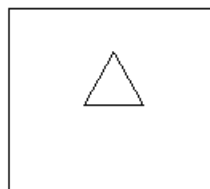
The new points are: (8 , 8) , (14 , 16) , (20 , 10)



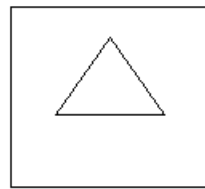
Enlargement: If $T1 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$, if (x1 y1) is original position and $T1$ is translation vector then (x2 y2) are coordinated after scaling

$$[x_2 y_2] = [x_1 y_1] \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = [2x_1 2y_1]$$

The image will be enlarged two times



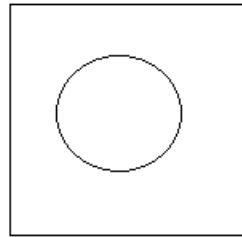
Original Image



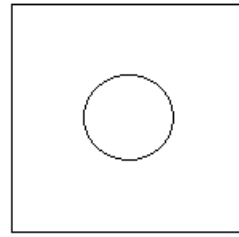
Enlarge Image

Reduction: If $T1 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$. If (x_1, y_1) is original position and $T1$ is translation vector, then (x_2, y_2) are coordinates after scaling

$$[x_2, y_2] = [x_1, y_1] \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} = [0.5x_1, 0.5y_1]$$



Original Image



Reduction Image

Matrix for Scaling:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Rotation

It is a process of changing the angle of the object. Rotation can be clockwise or anticlockwise. For rotation, we have to specify the angle of rotation and rotation point. Rotation point is also called a pivot point. It is point about which object is rotated. In rotation, we rotate the object at particular angle θ (theta) from its origin. After the object has been rotated, it is still the same distance away from the pivot point; however its orientation has been changed.

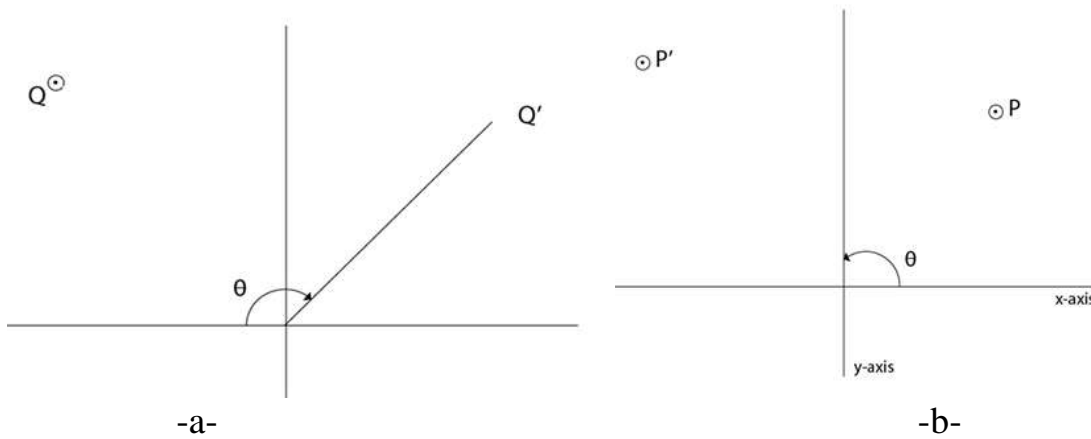
Types of Rotation:

1. Clockwise
2. Counterclockwise (Anticlockwise)

The **positive value** of the pivot point (rotation angle) rotates an object in a *counter-clockwise* (anti-clockwise) direction. Figure 16.a.

The **negative value** of the pivot point (rotation angle) rotates an object in a *clockwise* direction. Figure 16.b.

When the object is rotated, then every point of the object is rotated by the same angle.



**Figure 16: Rotation (a- clockwise Q original position, Q' after rotation, θ angle rotation
b- counterclockwise P original position, P' after rotation, θ angle rotation)**

Any point (x, y) can be represented by its *radial distance*, r , from the origin and its angle, ϕ , of the x – axis as show in figure 16.

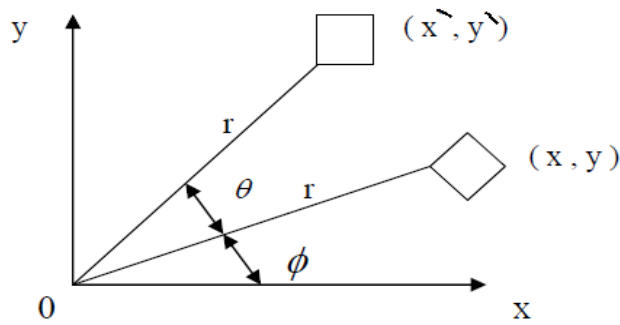


Figure 16: Rotation about origin

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

$$\left. \begin{aligned} x &= r * \cos (\phi) \\ y &= r * \sin (\phi) \end{aligned} \right\} \dots\dots\dots (1) \quad \text{where } r \text{ is the length}$$

If (x, y) is rotated an angle θ in the **counterclockwise** direction. The transformed point (x', y') is represented as :

$$\left. \begin{aligned} x &= r * \cos (\phi + \theta) \\ y &= r * \sin (\phi + \theta) \end{aligned} \right\} \dots\dots\dots (2)$$

**$\sin (A + B) = \sin A . \cos B + \sin B . \cos A$
 $\cos (A + B) = \cos A . \cos B - \sin A . \sin B$**

Using the laws of sines and cosines from trigonometry, the equation (2) become:

$$\left. \begin{aligned} x' &= r * \cos(\theta) * \cos(\theta) - r * \sin(\theta) * \sin(\theta) \\ y' &= r * \sin(\theta) * \cos(\theta) + r * \cos(\theta) * \sin(\theta) \end{aligned} \right\} \dots\dots\dots (3)$$

From the definition of x and y, the equation (3) reduces to:

$$\left. \begin{aligned} x' &= x * \cos(\theta) - y * \sin(\theta) \\ y' &= y * \cos(\theta) + x * \sin(\theta) \end{aligned} \right\} \dots\dots\dots (4) \quad \text{عكس عقارب الساعة}$$

So to rotate a point (x , y) through a **clockwise** angle θ about the origin of the coordinate system we write:-

$$\left. \begin{aligned} x' &= x \cos(\theta) + y \sin(\theta) \\ y' &= -x \sin(\theta) + y \cos(\theta) \end{aligned} \right\} \dots\dots\dots (5) \quad \text{باتجاه عقارب الساعة}$$

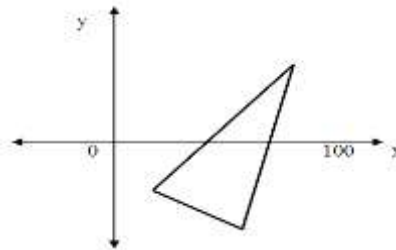
Example

The triangle (10 , 0) , (30 , 0) , (50 , 80) rotate 45° clockwise about the origin.

Solution:

$$\begin{aligned} x' &= x \cos(\theta) + y \sin(\theta) \\ y' &= -x \sin(\theta) + y \cos(\theta) \end{aligned}$$

(7.07 , - 7.07) , (21.21 , - 21.21) , (91.93 , 21.12)



Rotate about the origin

Rotation about an arbitrary point (not origin): If we want to rotate an object or point about an arbitrary point (pivot point), first of all, we translate the point about which we want to rotate to the origin. Then rotate point or object about the origin, and at the end, we again translate it to the original place (i.e. translate, rotate, translate) figure 17.

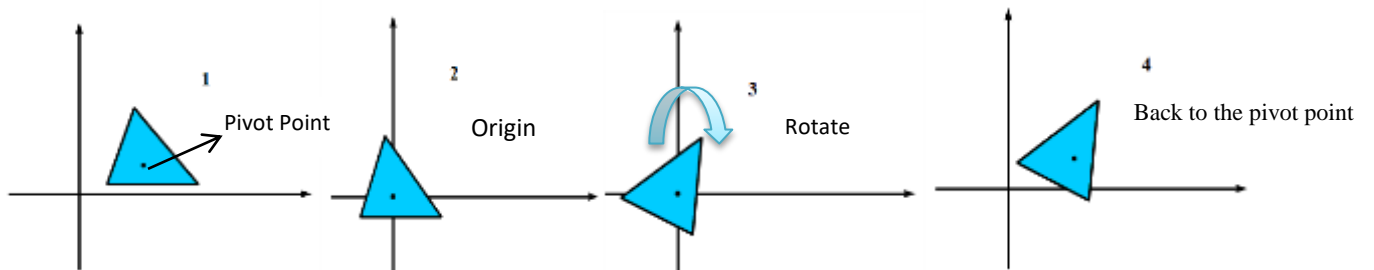


Figure 17: Rotation about an arbitrary (pivot) point

To rotate an object an angle (θ) about a pivot point (x_p, y_p) **other than the origin**, we perform the following three steps:

Step 1: Translate

Translate the pivot point (x_p, y_p) to the origin. Every point (x, y) defining the object is translated to a new point (x', y') where:

$$\begin{aligned}x' &= x - x_p \\y' &= y - y_p\end{aligned}$$

Step 2: Rotate

Use these translated points (x', y'), θ degree about the origin to obtain the new point (x'', y'') where:

$$\begin{aligned}x'' &= x' * \cos(\theta) - y' * \sin(\theta) \\y'' &= y' * \cos(\theta) + x' * \sin(\theta)\end{aligned}$$

By substituting for x' and y' :

$$\begin{aligned}x'' &= (x - x_p) * \cos(\theta) - (y - y_p) * \sin(\theta) \\y'' &= (y - y_p) * \cos(\theta) + (x - x_p) * \sin(\theta)\end{aligned}$$

Step 3: Translate

Translate the center of rotation back to the pivot point (x_p, y_p).

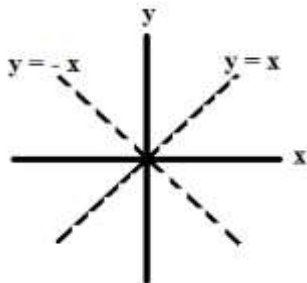
$$\begin{aligned}x' &= r * \cos(\theta) * \cos(\theta) - r * \sin(\theta) * \sin(\theta) \\y' &= r * \sin(\theta) * \cos(\theta) + r * \cos(\theta) * \sin(\theta) \\x' &= x * \cos(\theta) - y * \sin(\theta) \\y' &= y * \cos(\theta) + x * \sin(\theta) \\x''' &= x'' + x_p \\y''' &= y'' + y_p\end{aligned}$$

By substituting for x'' and y'' :

$$\begin{aligned}x''' &= (x - x_p) * \cos(\theta) - (y - y_p) * \sin(\theta) + x_p \\y''' &= (y - y_p) * \cos(\theta) + (x - x_p) * \sin(\theta) + y_p\end{aligned}$$

4. 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. Table (1) shows types of reflection.



Coordinate Rules for Reflection

If (a, b) is reflected on the *x-axis*, its image is the point $(a, -b)$

If (a, b) is reflected on the *y-axis*, its image is the point $(-a, b)$

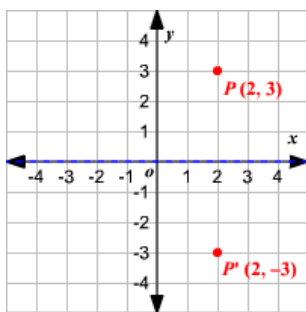
If (a, b) is reflected on the line $y = x$, its image is the point (b, a)

If (a, b) is reflected on the line $y = -x$, its image is the point $(-b, -a)$

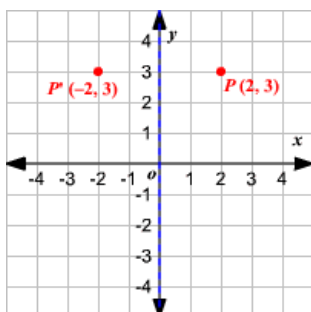
If (a, b) is reflected *about the origin* point, its image is the point $(-a, -b)$

Reflection	Reflection Rule	In Words	What it looks like on the graph
Over the x-axis	$(x,y) \rightarrow (x,-y)$	Negate the y coordinates	Image is directly above or below the original
Over the y-axis	$(x,y) \rightarrow (-x,y)$	Negate the x coordinates	Image is left or right of the original
Over $y=x$	$(x,y) \rightarrow (y,x)$	Swap the x and y coordinates	$y=x$ passes through the plane at a 45 degree angle. Image is above or below this line from the original
Origin	$(x,y) \rightarrow (-x,-y)$	Negate both the x and y coordinates	Image is rotated 180 degrees
$y = -x$	$(x,y) \rightarrow (-y,-x)$	reverse the order of the coordinates and reverse the signs	$(-2, 6)$ would become $(-6,2)$

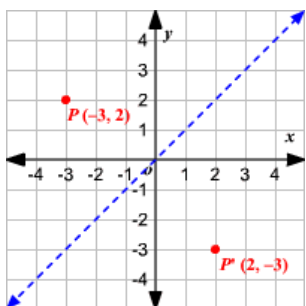
Reflection in the x-axis: The rule for a reflection over the x-axis is $(x,y) \rightarrow (x,-y)$.



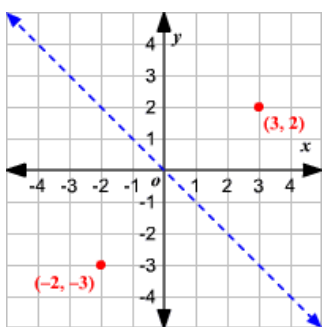
Reflection in the y-axis: The rule for a reflection over the y-axis is $(x,y) \rightarrow (-x,y)$.



Reflection in the line y=x: The rule for a reflection in the line $y=x$ is $(x,y) \rightarrow (y,x)$.



Reflection in the line y=-x: The rule for a reflection in the origin is $(x,y) \rightarrow (-y,-x)$.



Example Find the image of the point $(3, 2)$ that has undergone a reflection across

- a) the x-axis,
- b) the y-axis,
- c) the line $y = x$, and
- d) the line $y = -x$.

Write the notation to describe the reflection.

Solution:

- a) Reflection across the **x-axis**: $r_x\text{-axis}(3,2) \rightarrow (3,-2)$
- b) Reflection across the **y-axis**: $r_y\text{-axis}(3,2) \rightarrow (-3,2)$
- c) Reflection across the line **$y = x$** : $r_{y=x}(3,2) \rightarrow (2,3)$
- d) Reflection across the line **$y = -x$** : $r_{y=-x}(3,2) \rightarrow (-2,-3)$

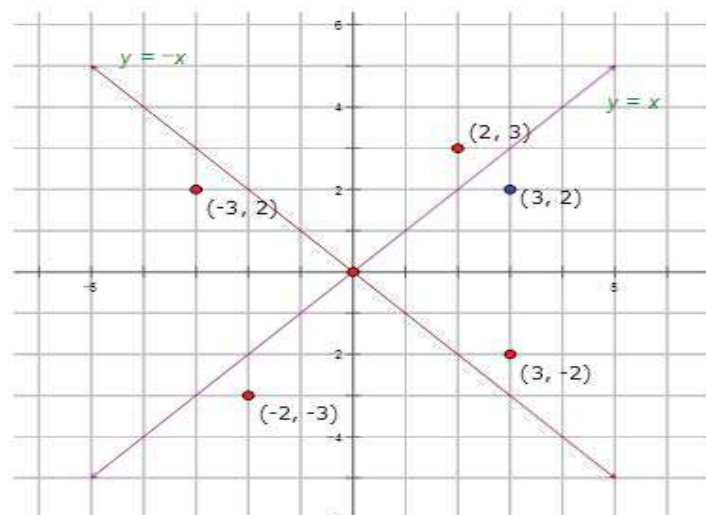
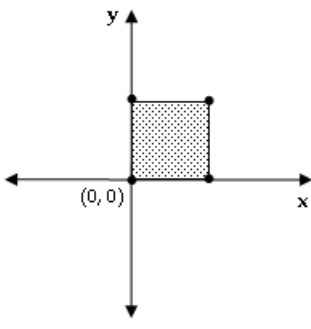
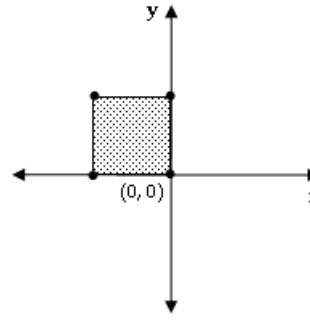
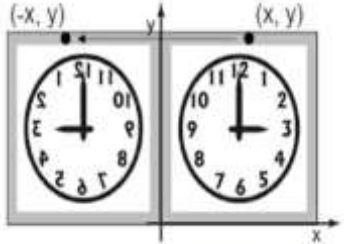
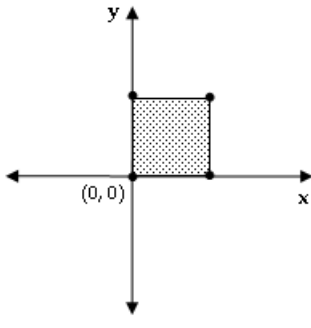
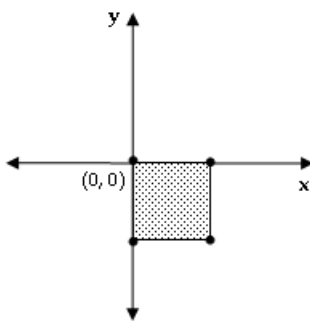
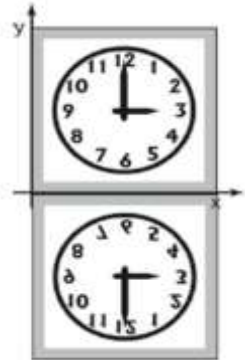
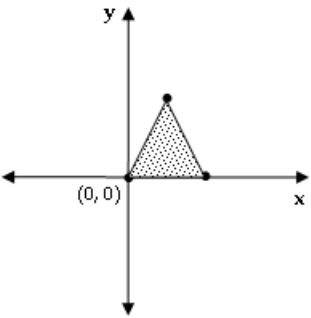
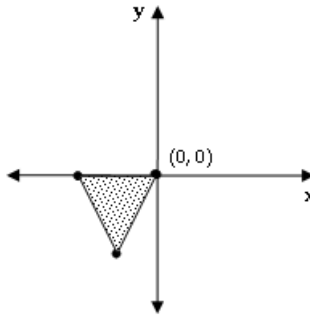
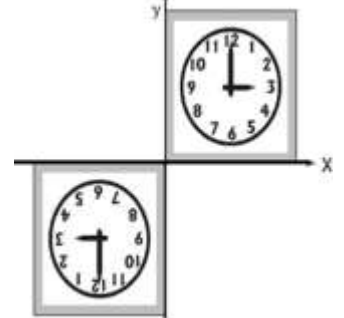
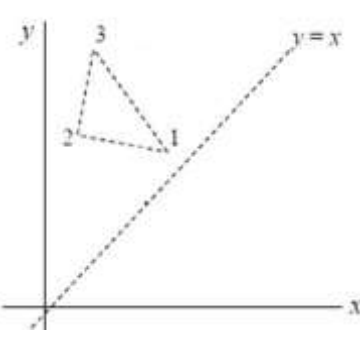
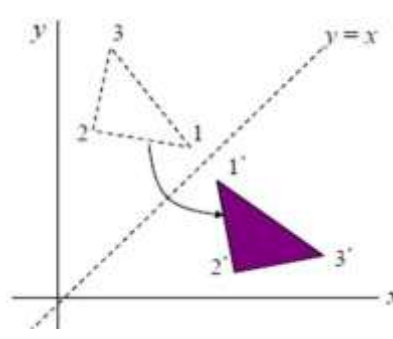
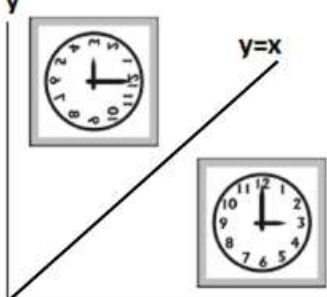
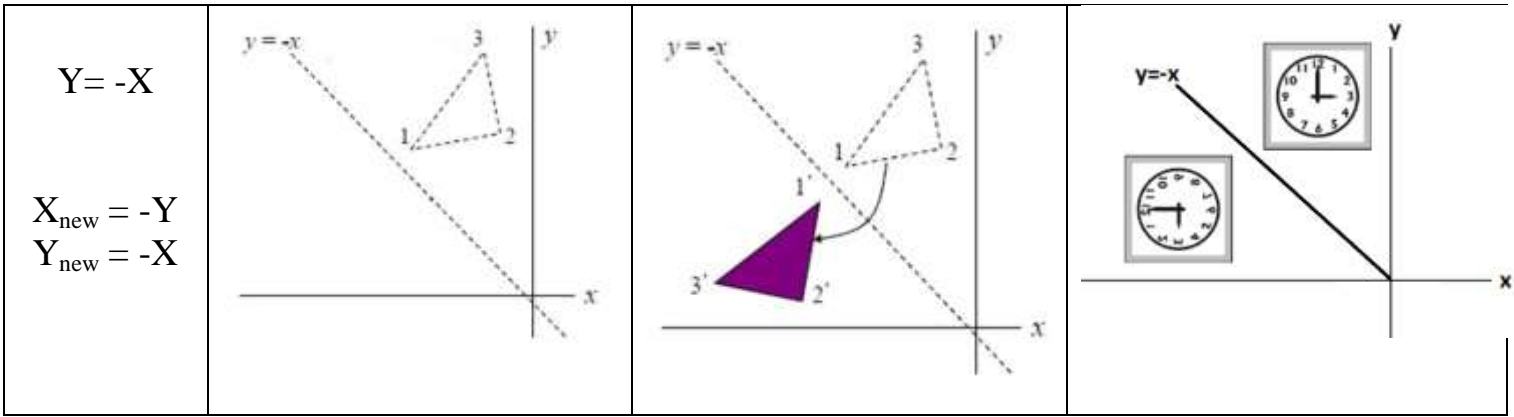


Table 1: Types of Reflection

Reflection	Original Object	Reflected Object	Example
<p>About the Y axis</p> <p>$X_{\text{new}} = -X$</p> <p>$Y_{\text{new}} = Y$</p>			
<p>About the X axis</p> <p>$X_{\text{new}} = X$</p> <p>$Y_{\text{new}} = -Y$</p>			
<p>About the origin</p> <p>$X_{\text{new}} = -X$</p> <p>$Y_{\text{new}} = -Y$</p>			
<p>$Y=X$</p> <p>$X_{\text{new}} = Y$</p> <p>$Y_{\text{new}} = X$</p>			



5. 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 shift X coordinate values and other shifts Y coordinate values. In both the cases, only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as **Skewing**. The shear can be in one direction or in two directions.

Shearing in the X-direction (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 figure 18.

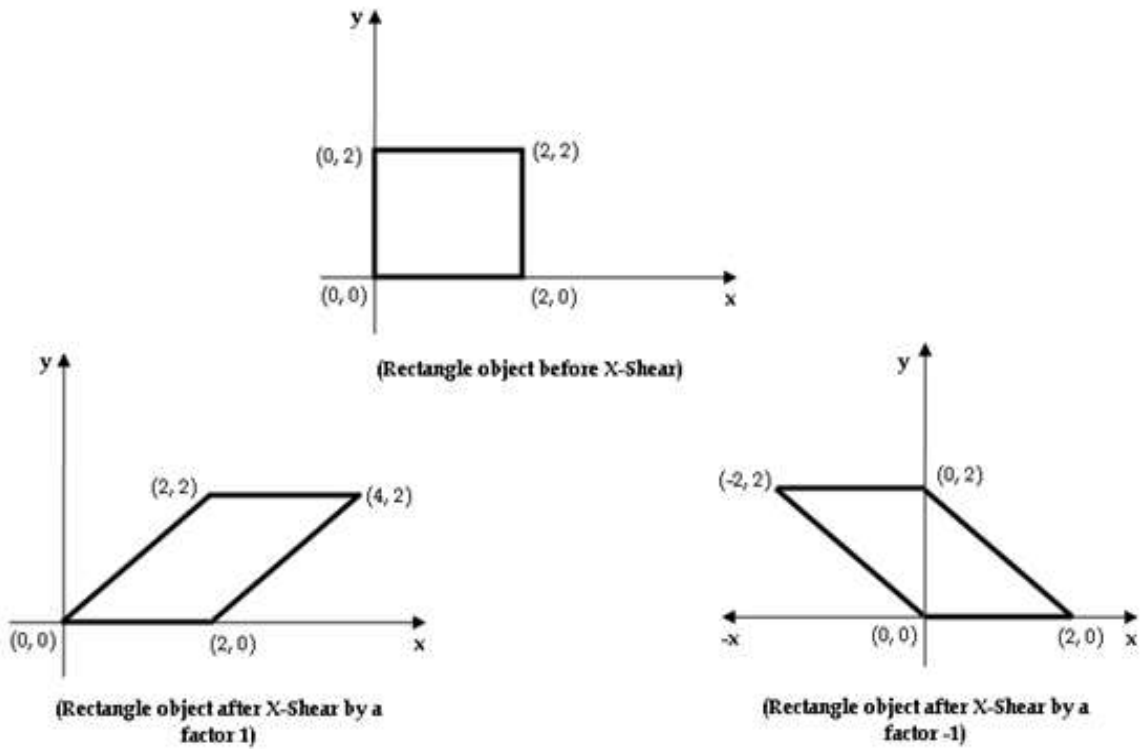


Figure 18: X-Shear Transformation

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

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

$$\hat{X} = X + Sh_x * Y$$

$$\hat{Y} = Y$$

Shearing in the Y-direction (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 figure 19.

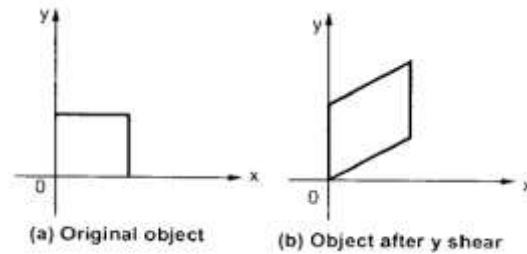


Figure 19: Y-Shear Transformation

The homogeneous matrix Y-Shear can be represented as:

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

$$\hat{Y} = Y + Sh_y * X$$

$$\hat{X} = X$$

Shearing in X-Y directions: Here layers will be slide in both *x* as well as *y* direction. The sliding will be in horizontal as well as vertical direction. The shape of the object will be distorted figure 20. The homogeneous matrix of shear in both directions is given by:

$$\begin{bmatrix} 1 & Sh_y & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

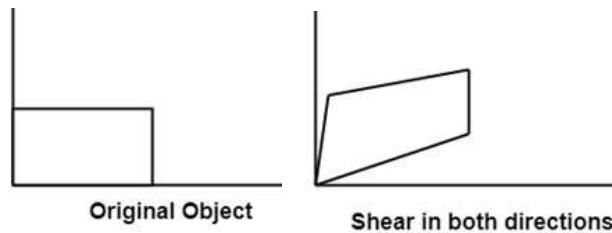


Figure 20: X-Y Shear Transformation

Example :

Shear the object (1,1), (3,1), (1,3), (3,3) with

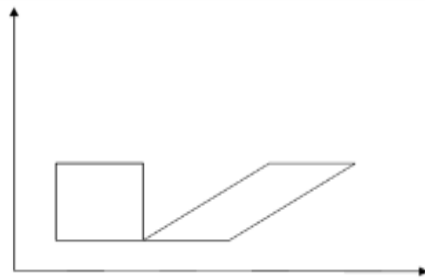
a: $sh_x = 2$

b: $sh_y = 2$

Solution

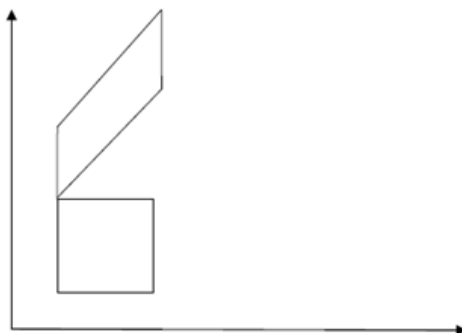
a: sh_x

$$\begin{bmatrix} 1 & 1 & 1 \\ 3 & 1 & 1 \\ 1 & 3 & 1 \\ \mathbf{3} & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 1 \\ 5 & 1 & 1 \\ 7 & 3 & 1 \\ \mathbf{9} & 3 & 1 \end{bmatrix}$$



b: sh_y

$$\begin{bmatrix} 1 & 1 & 1 \\ 3 & 1 & 1 \\ 1 & 3 & 1 \\ 3 & \mathbf{3} & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 1 \\ 3 & 7 & 1 \\ 1 & 5 & 1 \\ 3 & \mathbf{9} & 1 \end{bmatrix}$$



4.2 Matrix Representation of Transformations

Translation of point by the change of coordinate cannot be combined with other transformation by using simple matrix application. Such a combination is essential if we wish to rotate an image about a point other than origin by translation, rotation again translation.

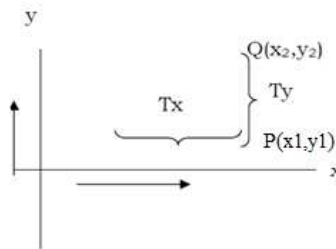
To combine these three transformations into a single transformation, **homogeneous** coordinates are used. In homogeneous coordinate system, two-dimensional coordinate positions (x, y) are represented by triple-coordinates. Here we perform translations, rotations, scaling to fit the picture into proper position.

For two-dimensional geometric transformation, we can choose homogeneous parameter h to any non-zero value. For our convenience take it as **one**. Each two-dimensional position is then represented with homogeneous coordinates $(x, y, 1)$.

4.2.1 Homogeneous Coordinates

The matrix representation of the transformation: Translation, Scaling, Rotation.

1. Translation



Consider a point $P(x_1, y_1)$ to be translated to another point $Q(x_2, y_2)$. If we know the point value (x_2, y_2) , we can directly shift to Q by displaying the pixel (x_2, y_2) . Suppose we only know that we want to shift by a distance of T_x along x axis and T_y along Y axis. Then the coordinates can be derived by $x_2 = x_1 + T_x$ and $Y_2 = y_1 + T_y$.

Suppose we want to shift a triangle with coordinates at $A(20,10)$, $B(30,100)$ and $C(40,70)$. The shifting to be done by 20 units along x axis and 10 units along y axis. Then the new triangle will be at $A_1(20+20, 10+10)$ $B_1(30+20, 10+10)$ $C_1(40+20, 70+10)$.

$$\text{In the matrix form } [x_2 \ y_2 \ 1] = [x_1 \ y_1 \ 1] \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$$

$$[x' \ y' \ 1] = [x \ y \ 1] \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ H & V & 1 \end{bmatrix}$$

2. Scaling

$$[x' \ y' \ 1] = [x \ y \ 1] \times \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. Rotation

(a) Counterclockwise direction

$$[x' \ y' \ 1] = [x \ y \ 1] \times \begin{bmatrix} \cos(x) & \sin(x) & 0 \\ -\sin(x) & \cos(x) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) Clockwise direction

$$[x' \ y' \ 1] = [x \ y \ 1] \times \begin{bmatrix} \cos(x) & -\sin(x) & 0 \\ \sin(x) & \cos(x) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4. Shearing

$$x_{sh} = [x' \ y' \ 1] = [x \ y \ 1] \times \begin{bmatrix} 1 & 0 & 0 \\ sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$y_{sh} = [x' \ y' \ 1] = [x \ y \ 1] \times \begin{bmatrix} 1 & sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5. Reflection

A- Reflection about X – axis:

$$x' = x$$

$$y' = -y$$

$$Ref_x = [x' \ y' \ 1] = [x \ y \ 1] \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

B- Reflection about Y – axis:

$$x' = -x$$

$$y' = y$$

$$\text{Ref}_y = [x' \ y' \ 1] = [x \ y \ 1] \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

C- Reflection about the origin (0, 0):

$$x = -x$$

$$y = -y$$

$$\text{Ref}_{\text{origin}} = [x' \ y' \ 1] = [x \ y \ 1] \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

D- Reflection about the line $y = x$:

$$x = y$$

$$y = x$$

$$\text{Ref}_{y=x} = [x' \ y' \ 1] = [x \ y \ 1] \times \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

E- Reflection about the line $y = -x$:

$$x = -y$$

$$y = -x$$

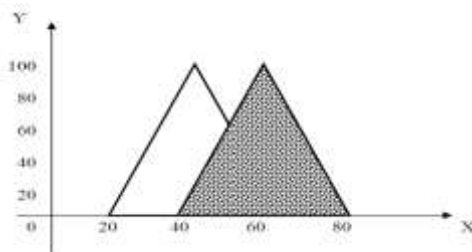
$$\text{Ref}_{y=-x} = [x' \ y' \ 1] = [x \ y \ 1] \times \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example

Consider a triangle defined by its three vertices (40,100), (20, 0), (60, 0) be translated 20 units to the right, using matrix representation.

Solution:

$$\begin{bmatrix} 40 & 100 & 1 \\ 20 & 0 & 1 \\ 60 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 20 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 60 & 100 & 1 \\ 40 & 0 & 1 \\ 80 & 0 & 1 \end{bmatrix}$$



Example

Rotate the triangle $(7, 8)$, $(4, 4)$, $(10, 5)$ 90° counterclockwise about the point $(7, 8)$, using matrix representation.

Solution:

1) Translate:-

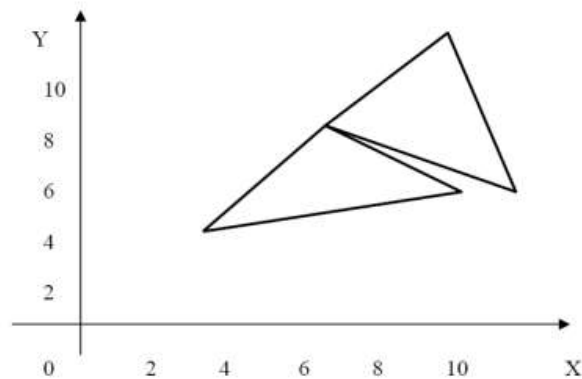
$$\begin{bmatrix} 7 & 8 & 1 \\ 4 & 4 & 1 \\ 10 & 5 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -7 & -8 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ -3 & -4 & 1 \\ 3 & -3 & 1 \end{bmatrix}$$

2) Rotate: -

$$\begin{bmatrix} 0 & 0 & 1 \\ -3 & -4 & 1 \\ 3 & -3 & 1 \end{bmatrix} \times \begin{bmatrix} \cos(90) & \sin(90) & 0 \\ -\sin(90) & \cos(90) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 4 & -3 & 1 \\ 3 & 3 & 1 \end{bmatrix}$$

3) Translate: -

$$\begin{bmatrix} 0 & 0 & 1 \\ 4 & -3 & 1 \\ 3 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 7 & 8 & 1 \end{bmatrix} = \begin{bmatrix} 7 & 8 & 1 \\ 11 & 5 & 1 \\ 10 & 11 & 1 \end{bmatrix}$$

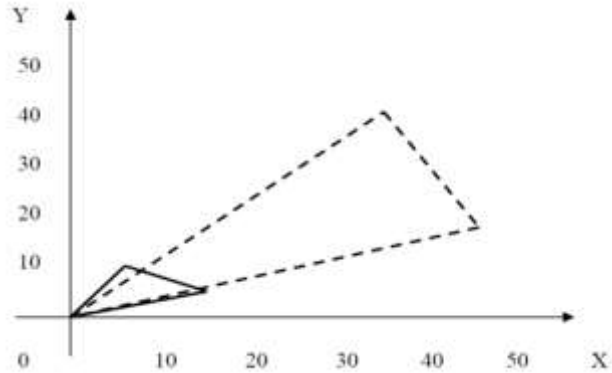


Example

Magnify the triangle $(0, 0)$, $(8, 10)$, $(12, 4)$, 4 times its size, using matrix representation.

Solution:

$$\begin{bmatrix} 0 & 0 & 1 \\ 8 & 10 & 1 \\ 12 & 4 & 1 \end{bmatrix} \times \begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 32 & 40 & 1 \\ 48 & 16 & 1 \end{bmatrix}$$



Example

Reflect the shape $(20, 70)$, $(40, 50)$, $(60, 70)$, $(40, 90)$, about:

1- X – axis 2- Y- axis 3- origin $(0,0)$ 4- $y = x$ 5- $y = -x$, by used matrix representation, and draw the result.

Solution

1- X – axis

$$x' = x$$

$$y' = -y$$

$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 20 & -70 & 1 \\ 40 & -50 & 1 \\ 60 & -70 & 1 \\ 40 & -90 & 1 \end{bmatrix}$$

2- Y- axis:

$$x' = -x$$

$$y' = y$$

$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -20 & 70 & 1 \\ -40 & 50 & 1 \\ -60 & 70 & 1 \\ -40 & 90 & 1 \end{bmatrix}$$

3- Origin (0,0)

$$x = -x$$

$$y = -y$$

$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -20 & -70 & 1 \\ -40 & -50 & 1 \\ -60 & -70 & 1 \\ -40 & -90 & 1 \end{bmatrix}$$

4- $y = x$

$$x = y$$

$$y = x$$

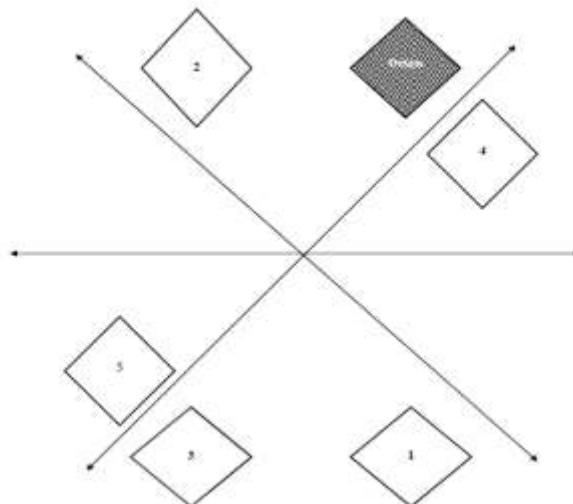
$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 70 & 20 & 1 \\ 50 & 40 & 1 \\ 70 & 60 & 1 \\ 90 & 40 & 1 \end{bmatrix}$$

5- $y = -x$

$$x = -y$$

$$y = -x$$

$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -70 & -20 & 1 \\ -50 & -40 & 1 \\ -70 & -60 & 1 \\ -90 & -40 & 1 \end{bmatrix}$$



Part Five

Mapping (Normalized Device Coordinates) (Window, Viewport) & Clipping

5.1 Two-Dimensional Viewing and Clipping

It is defined as *a process for displaying views of a two-dimensional picture on an output device*:

– Specify which parts of the object to display (clipping window, or world window, or viewing window) Figure 28. Where:

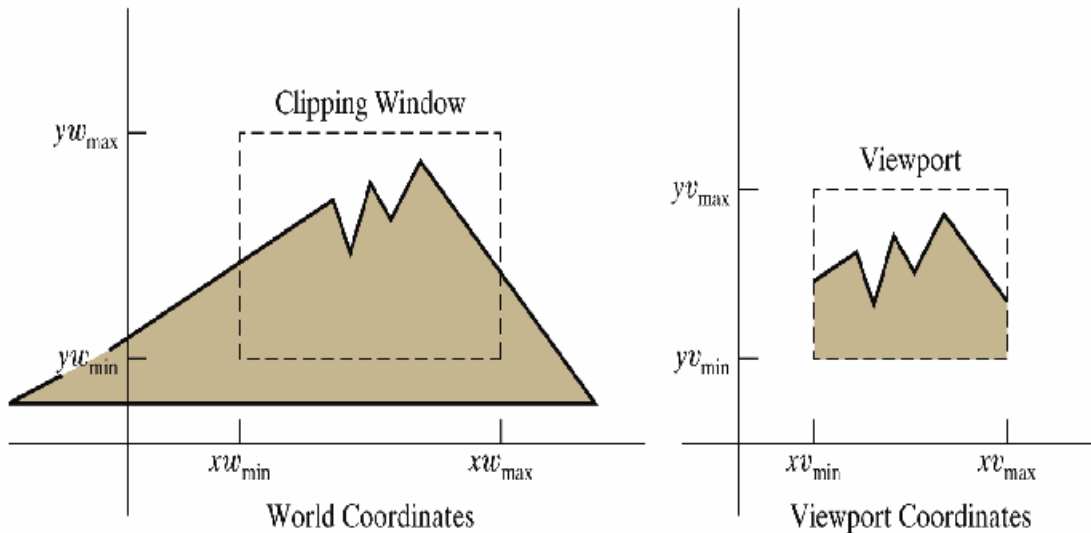


Figure 28: Clipping window, world window and viewing window

– **World Co – Ordinate System** is a right handed Cartesian coordinate system in which *picture is actually defined*.

– **Physical Device Co – Ordinate System** is a coordinate system that correspond to output device or work stations where image to be displayed.

– **Normalized Co – Ordinate System** It is a right handed coordinate system in which display area of virtual display device correspond to the unit square (1×1).

– **Window** or Clipping window is the selected section of a scene that is displayed on a display window. It is a finite region from World Coordinate System.

– **View port** is the window where the object is viewed on the output device. It is a finite region from Device Coordinate System.

The primary use of clipping in computer graphics is to remove objects, lines, or line segments that are outside the viewing pane. The viewing transformation is insensitive to the position of points relative to the viewing volume – especially those points behind the viewer – and it is necessary to remove these points before generating the view.

The method for selecting and enlarging portions of a drawing enclosed in a rectangular region is called **Windowing**. The rectangular region is called a **window**. The technique for not showing that part of the drawing which one is not interested in is called **clipping**. Figure 29.

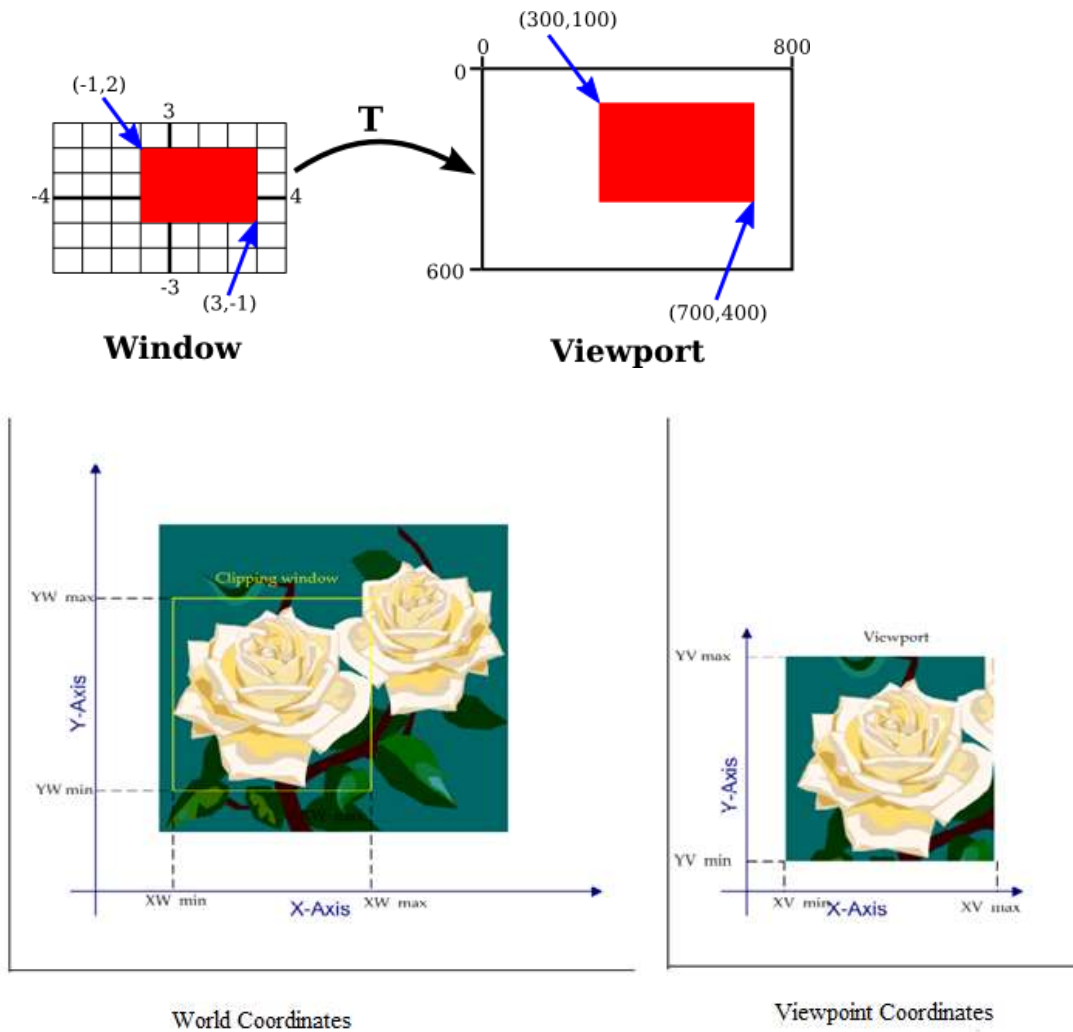


Figure 29: A Window and Viewing mapping

A *window* is specified by four world coordinate: **XWmin**, **XWmax**, **YWmin**, and **YWmax**. Similarly, a viewport is described by four normalized device coordinates: **XVmin**, **XVmax**, **YVmin**, and **YVmax**. Figure 29.

The objective of *window-to-viewport* mapping is to *convert the world coordinates (WX, WY) of an arbitrary point to its corresponding normalized device coordinates (VX, VY)*. In order to maintain the same relative placement of the point in the viewport as in the window, we require:

$$\frac{xv - xv_{\min}}{xv_{\max} - xv_{\min}} = \frac{xw - xw_{\min}}{xw_{\max} - xw_{\min}} \dots\dots\dots \text{equation 1}$$

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

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

$$xv = xv_{\min} + (xw - xw_{\min}) * sx$$

$$yv = yv_{\min} + (yw - yw_{\min}) * sy \dots\dots\dots \text{equation 2}$$

Where scaling factors are:

$$sx = \frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}}$$

$$sy = \frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}}$$

Note: the *coordinate* of window is *world coordinate* but the **view port** is the coordinate x = 0 to 1 and y = 0 to 1.

Equation (1) and Equation (2) can also be derived with a set of transformation that converts the *window or world coordinate area* **into** the *viewport or screen coordinate area*. This conversation is performed with the following sequence of transformations:

1. Perform a *scaling* transformation using a fixed point position (xwmin,ywmin) that scales the window area to the size of the viewport.
2. *Translate* the scaled window area to the position of the viewport. Relative proportions of objects are maintained if the scaling factors are the same (sx=sy).

1- Viewing Transformation: Mapping from object space to image space,

1. Change the window size to become the size of the view port (**Scaling**).
2. Position the window at the desired location on the screen (**Translate**) by moving the Lower-left corner of the window to the view port Lower-left corner location. To do this we need 2 steps:

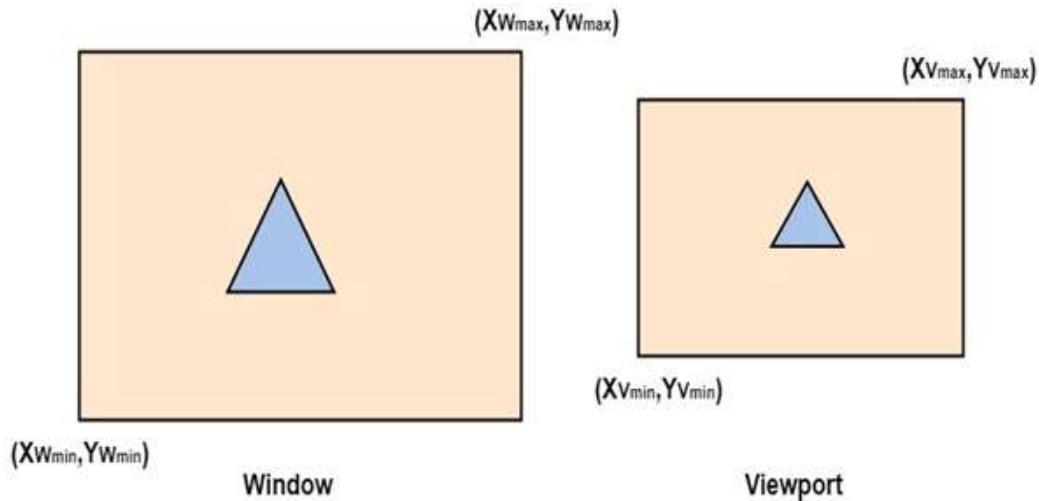
Step 1:

Move the corner to the origin (to perform the necessary scaling without disturbing the corner the corner's position).

Step 2:

Move it to the view port corner location.

Matrix Representation of the three steps of Transformation:



Step1: Translate window to origin 1

$$T_x = -X_{W_{min}} \quad T_y = -Y_{W_{min}}$$

Step2: Scaling of the window to match its size to the viewport

$$S_x = (X_{V_{max}} - X_{V_{min}}) / (X_{W_{max}} - X_{W_{min}})$$

$$S_y = (Y_{V_{max}} - Y_{V_{min}}) / (Y_{W_{max}} - Y_{W_{min}})$$

Step3: Again translate viewport to its correct position on screen.

$$T_x = X_{V_{min}}$$

$$T_y = Y_{V_{min}}$$

Above three steps can be represented in matrix form:

$$VT = T * S * T_1$$

T = Translate window to the origin

S = Scaling of the window to viewport size

T₁ = Translating viewport on screen.

$$T = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X_{W_{min}} & -Y_{W_{min}} & 1 \end{vmatrix}$$

$$S = \begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$T_1 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ X_{V_{min}} & Y_{V_{min}} & 1 \end{vmatrix}$$

Viewing Transformation = T * S * T₁

Example

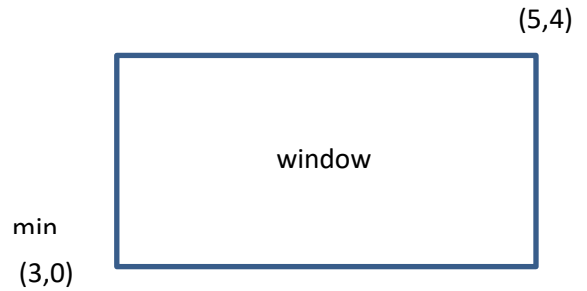
A *window* has left and right boundaries of 3 and 5 and lower and upper boundaries of 0 and 4. The *view port* is the upper-right quadrant of the screen with boundaries at 0.5 and 1.0 for both X and Y direction, find the viewing transformation?

1- Translate

$$T_x = -X_{w_{min}} \quad T_y = -Y_{w_{min}}$$

The first translation matrix would be:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix} \text{ To origin}$$



2- Scaling

$$S_x = (X_{v_{max}} - X_{v_{min}}) / (X_{w_{max}} - X_{w_{min}})$$

$$S_y = (Y_{v_{max}} - Y_{v_{min}}) / (Y_{w_{max}} - Y_{w_{min}})$$

The length of the window is:

$$5 - 3 = 2 \quad \text{in the X direction}$$

$$4 - 0 = 4 \quad \text{in the Y direction}$$

The length of the view port is $1.0 - 0.5 = 0.5$ in the X direction

The X scale factor is $S_x = 0.5/2 = 0.25$

In the Y direction is $S_y = 0.5/4 = 0.125$

The scaling transformation matrix is:

$$\begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3- Translate

$$T_x = X_{v_{min}}$$

$$T_y = Y_{v_{min}}$$

Finally to position the view port requires a translation:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{bmatrix}$$

The viewing transformation is then: $T * S * T1$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{bmatrix} = \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ -0.25 & 0.5 & 1 \end{bmatrix}$$

In general the viewing transformation is:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -WXL & -WYL & 1 \end{bmatrix} \begin{bmatrix} \frac{(VXH-VXL)}{(WXH-WXL)} & 0 & 0 \\ 0 & \frac{(VYH-VYL)}{(WYH-WYL)} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ VXL & VYL & 1 \end{bmatrix} =$$

$$\begin{bmatrix} \frac{(VXH-VXL)}{(WXH-WXL)} & 0 & 0 \\ 0 & \frac{(VYH-VYL)}{(WYH-WYL)} & 0 \\ VXL-WXL \frac{(VXH-VXL)}{(WXH-WXL)} & VYL-WYL \frac{(VYH-VYL)}{(WYH-WYL)} & 1 \end{bmatrix}$$

V: View port

W: Window

X: Position of a vertical boundary

Y: Horizontal boundary

H: High boundary

L: Low boundary

2 – Clipping

Is a process which divided each element of the picture into its *visible* and *invisible* portions. Visible portion is selected. An invisible portion is discarded.

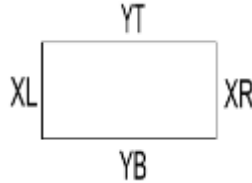
- If we imagine a box about a portion of the object so we could display what is enclosed in the box such a box called a **Window**.
- If we do not wish to use the entire screen for display, we can imagine a box on the screen and have the image confined to that box such a box in the screen space is called a **view port**.
- When the window is hanged a different part of the object at the same position is displayed.

- If we change the view port, we see the same part of the object drawn at a different place on the display.

The process of clipping determines which elements of the picture lie inside the window and so are visible.

2-1 Rectangular Clipping Windows

The clipping window assumes to be rectangles whose sides are aligned with the coordinate area. The X extent is measured from X min to X max and the Y extent is measured from Y min to Y max.



There are three types of clipping:

- 1 – Point
- 2 – Line
- 3 – Polygon

a) Point Clipping

Point Clipping is used to determining, whether the point (x,y) is inside the window or not. For this following conditions are checked:

1. $x \leq x_{max}$
2. $x \geq x_{min}$
3. $y \leq y_{max}$
4. $y \geq y_{min}$

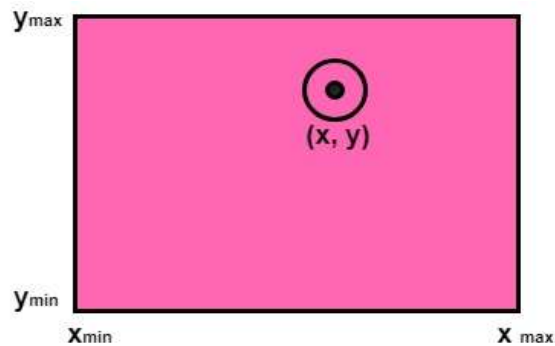


Figure 30: Point Clipping

The (x, y) is coordinate of the point. If anyone from the above inequalities is false, then the point will fall outside the window and will not be considered to be visible. Figure 30.

Algorithm of Point Clipping:

Step 1: First, we set the value of x_{\min} and x_{\max} coordinates for the window.

Step 2: Now, set the coordinates of a given point (P, Q) .

Step 3: Now check the above mention condition.

Step 4: If

Point coordinates lie between the (x_{\min}, x_{\max}) and (y_{\min}, y_{\max})

Then

{Display the point in the view pane}

Else

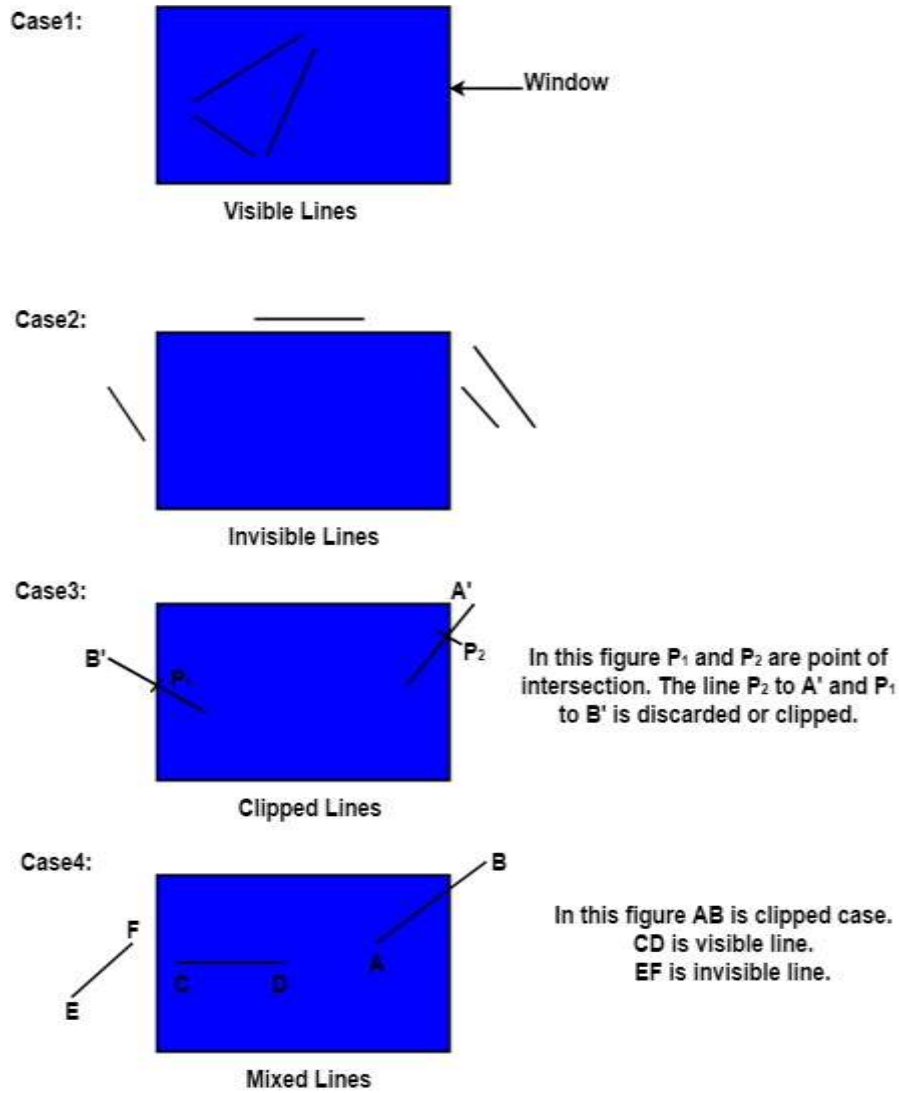
{Remove the point}

Step 5: Stop.

b) Line Clipping

Lines are of three types:

1. **Visible:** A line or lines entirely inside the window is considered visible
2. **Invisible:** A line entirely outside the window is considered invisible
3. **Clipped:** A line partially inside the window and partially outside is clipped. For clipping point of intersection of a line with the window is determined.



The concept of line clipping is same as point clipping. In line clipping, we will cut the portion of line which is outside of window and keep only the portion that is inside the window see figure 31.

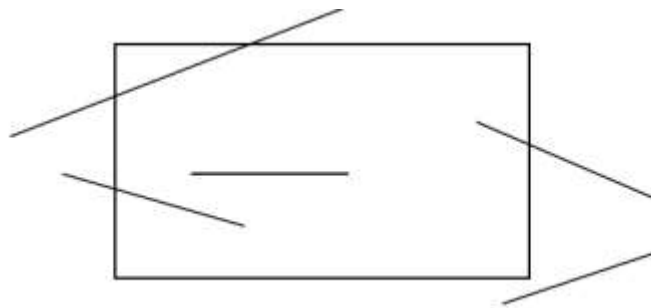


Figure 31: Line Clipping

How to decide which of the lines, or more precisely which part of every line is to be displayed. Since we know the coordinates of the screen,

- i) Any line whose end points lie within the screen coordinate limits will have to display fully (because we cannot have a straight line whose end points are within the screen and any other middle point in outside).
- ii) Any line whose end points lie totally outside the screen coordinates will have to examine to see if any intermediate point is inside the screen boundary.
- iii) Any line whose one end point lies inside the boundary will have to be *identified*.

In case of (ii) and (iii), we should decide up to what point, the line segment can be displayed. Simply finding the *intersection* of the line with the screen boundary can do this.

We will use 4-bits to divide the entire region (Cohen Sutherland). These 4 bits represent the *Top, Bottom, Right, and Left* of the region as shown in the figure 32. Here, the **TOP** and **LEFT** bit is set to 1 because it is the **TOP-LEFT** corner. i.e. TBRL code 1001 (T=1, not Bottom so B=0, not Right so R=0, L=1,).

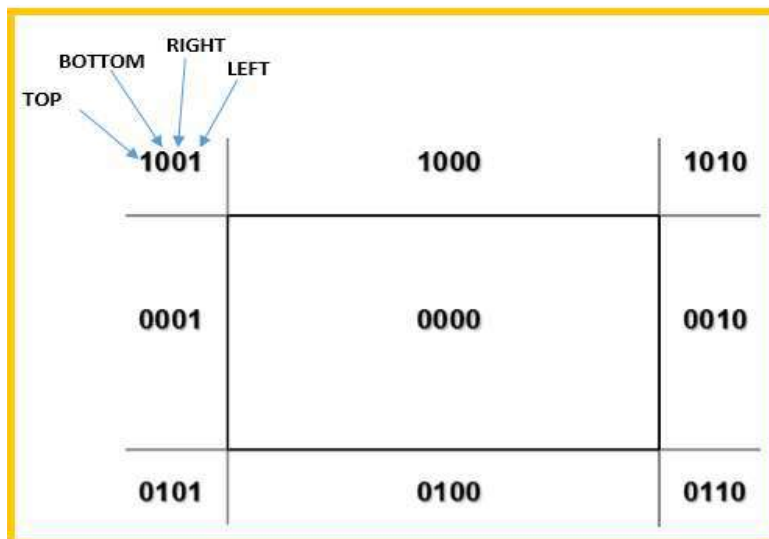


Figure 32: 4-bits Code

- First bit on** if $x < x_l$ to the left of window.
- Second bit on** if $x > x_r$ to the right of window.
- Third bit on** if $y < y_b$ to the below of window.
- Fourth bit on** if $y > y_t$ to the top of window.

The conditions can be checked by simply comparing the screen coordinate values with the coordinates of the endpoints of the line.

If for a line, both end points have the bit pattern of 0000, the line can be displayed as it is (trivially).

Otherwise, the pattern of 1's will indicate as to with respect to which particular edge the intersection of the line is to be verified.

For example if one of the points of a straight line shows 1000, then its intersecting section with regard to the top edge needs to be computed (since the point is above the top edge). If for the same line, the other point returns 0010, then since a segment of the line is beyond the right edge, the intersection with the right edge is to be computed.

Line Segment Clipping

Line clipping process is two phases:

- 1: Identify those lines which intersect the window and so need to be clipped.
- 2: Perform the clipping.

All line segments fall into one of the following clipping categories figure 33.

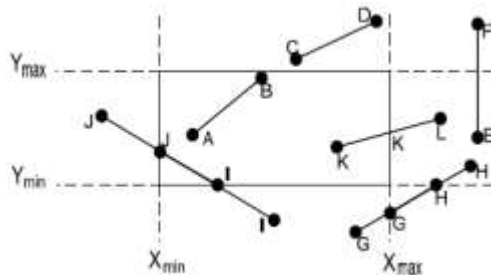


Figure 33: Clipping categories

1 – Visible:

Both endpoints of the line segment lie within the window (Line AB).

When the **OR** operation between two endpoint codes is FALSE (0000), the line is **inside** the clipping window, and we save it.

2 – Not visible:

The line segment definitely lies outside the window (Line CD and EF). This will occur if the line segment from (X_1, Y_1) to (X_2, Y_2) satisfies any one of the following four inequalities

$$X_1, X_2 > X_{\max} \quad Y_1, Y_2 > Y_{\max} \quad X_1, X_2 < X_{\min} \quad Y_1, Y_2 < Y_{\min}$$

Doing **Bitwise AND** between the two outcodes of the line segment endpoints, if the result is **NOT** 0000, then the line endpoints share the same region and the line segment does not cross the clipping window so it is **rejected**.

3 – Clipping candidate:

The line is in neither category 1 nor 2 (Line GH, IJ, and KL). If endpoint is above the window ($y > Y_{\max}$) or endpoint is below the window ($y < Y_{\min}$) **the Y clipped is equaled boundaries of window** and **X clipped is equaled $X = (y - y_1) / m + x_1$** .

Note: if $m = 0$ then $X = X_1$ suppose "down or X_2 suppose up" if endpoint is right of window ($x > X_{\max}$) or endpoint is left of window ($x < X_{\min}$) **the X clipped is equaled boundaries of window** and **Y clipped is equaled $y = (X - x_1) / m + y_1$** .

Line clipping is performed by using the line clipping algorithm. The line *clipping algorithms* are:

1. Cohen Sutherland Line Clipping Algorithm
2. Midpoint Subdivision Line Clipping Algorithm
3. Liang-Barsky Line Clipping Algorithm

A- Cohen Sutherland Line Clipping Algorithm

1. Read 2 end points of line as $p1(x1,y1)$ and $p2(x2,y2)$
2. Read 2 corner points of the clipping window (left-top and right-bottom) as $(wx1,wy1)$ and $(wx2,wy2)$
3. Assign the region codes for 2 endpoints $p1$ and $p2$ using following steps:-
initialize code with 0000

Set bit 1 if $x < wx1$

Set bit 2 if $x > wx2$

Set bit 3 if $y < wy2$

Set bit 4 if $y > wy1$

4. Check for visibility of line
 - a. If region codes for both endpoints are zero then line is completely visible. Draw the line go to step 9.
 - b. If region codes for endpoints are not zero and logical ANDing of them is also nonzero then line is invisible. Discard the line and move to step 9.
 - c. If it does not satisfy 4.a and 4.b then line is partially visible.
5. Determine the intersecting edge of clipping window as follows:-
 - a. If region codes for both endpoints are nonzero find intersection points $p1'$ and $p2'$ with boundary edges.
 - b. If region codes for any one end point is non zero then find intersection point $p1'$ or $p2'$.
6. Divide the line segments considering intersection points.
7. Reject line segment if any end point of line appears outside of any boundary.
8. Draw the clipped line segment.
9. Stop.

Example: Suppose window (-30,40), (40, -40) check with clipping line **Line1(70,0),(0,70)** & line2 (-50,10),(0,-30) & line3 (50,70),(60,-70)

Solution:- Xmin= -30, Xmax=40, Ymin= -40, Ymax=40

Line1 → (70,0) [0010] because $X > X_{max}$

Line1 → (0,70) [0001] because $Y > Y_{max}$

Finally [0010] AND [0001] → [0000] 'need Clipping

Line2 → (-50,10) [1000] because $X < X_{min}$

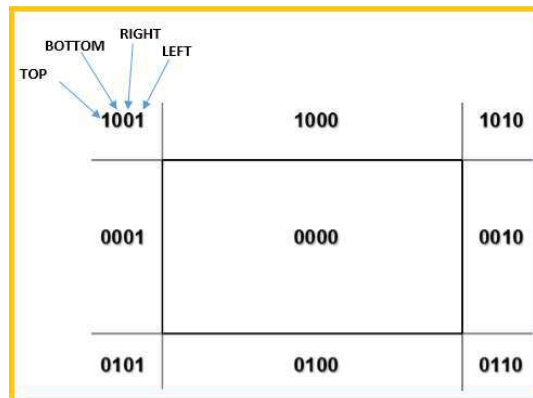
Line2 → (0,-30) [0000] point inside

Finally [1000] AND [0000] → [0000] 'need Clipping

Line3 → (50,70) [0011] because $X > X_{max}$ & $Y > Y_{max}$

Line3 → (50,-70) [0110] because $X > X_{max}$ & $Y < Y_{min}$

Finally [0011] AND [0110] → [0010] it line is outside {not visible}



Note: If result AND (bitwise)=0 then need clipping {case.1 , case. 3} otherwise is fully external of window Coordinate {not visible case. 2}

NOTES

- Line Visible → And = 0 , OR = 0
- Line Invisible → And < > 0 , OR < > 0
- Otherwise Line needs Clipping.
- Point line Clipping must be flag is 0000

Simple visibility algorithm

Check for totally visible lines.

If $((x_b < x_L) \text{ OR } (x_b > x_R))$ then 1.

If $((x_e < x_L) \text{ OR } (x_e > x_R))$ then 1.

If $((y_b < y_B) \text{ OR } (y_b > y_T))$ then 1.

If $((y_e < y_B) \text{ OR } (y_e > y_T))$ then 1.

Draw line

Go to 3

1 - Check for totally invisible lines

if $((x_b < x_L) \text{ AND } (x_e < x_L))$ then 2

if $((x_b > x_L) \text{ AND } (x_e > x_L))$ then 2

if $((y_b < y_L) \text{ AND } (y_e < y_B))$ then 2

if $((y_b > y_T) \text{ AND } (y_e > y_T))$ then 2

The line is partially visible or diagonally crosses the corner; determine the intersections go to 3

2 line is invisible

3 next line

Now, let us study how this clipping algorithm works. For the sake of simplicity we will tackle all the cases with the help of example lines 11 to 15 shown in Figure 34. Each line segment represents a case.

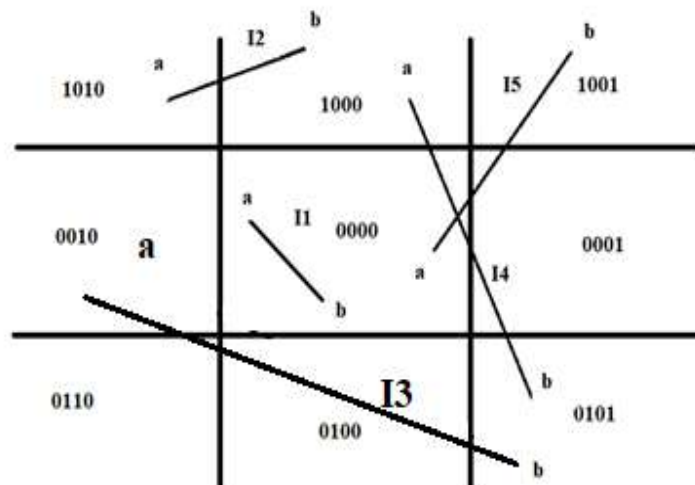


Figure 34:
Various cases of Cohen Sutherland Line Clippings

Note, that in *Figure 34*, line l_1 is completely visible, l_2 and l_3 are completely invisible; l_4 and l_5 are partially visible.

Case 1: $l_1 \rightarrow$ **Completely visible**, (both points lie inside the window)

Case 2: l_2 and $l_3 \rightarrow$ **Invisible** , rejection

Case 3: l_4 and $l_5 \rightarrow$ **partially visible** (partially inside the window)

Now, let us examine these three cases with the help of this algorithm:

Case 1: (Trivial acceptance case) *if the TBLR bit codes of the end points P, Q of a given line is 0000 then line is completely visible.* Here this is the case as the end points a and b of line l_1 are: a (0000), b (0000). If this acceptance test is *failed* then, the line segment PQ is passed onto Case 2.

Case 2: (Rejection Case) *if the logical intersection (AND) of the bit codes of the end points P, Q of the line segment is $\neq 0000$ then line segment is not visible or is rejected.*

Note that, in *Figure 34*, line 2 is completely on the top of the window but line 3 is neither on top nor at the in bottom plus, either on the LHS nor on the RHS of the window. We use the standard formula of logical ANDing to test the non visibility of the line segment.

So, to test the visibility of line 2 and 3 we need to calculate the logical intersection of end points for line 2 and line 3.

line l2: bit code of end points are 1010 and 1000

logical intersection of end points = $(1010) \wedge (1000) = 1000$

as logical intersection $\neq 0000$. So line 2 will be invisible.

line l3: end points have bit codes 0010 and 0101 now logical intersection = 0000, i.e., $0010 \wedge 0101 = 0000$ from the *Figure 34*, the line is invisible. Similarly in line 4 one end point is on top and the other on the bottom so, logical intersection is 0000 but then it is partially visible, same is true with line 5. These are special cases and we will discuss them in case 3.

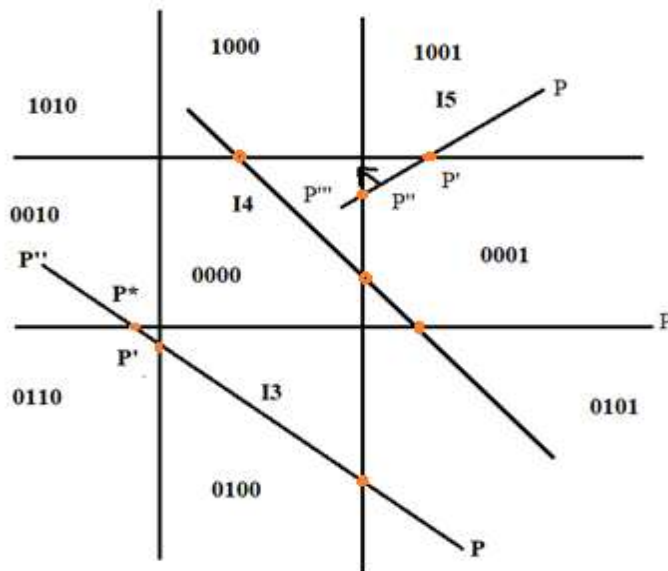


Figure 35: Cohen Sutherland line clipping

Case 3: Suppose for the line segment PQ, both the trivial acceptance and rejection tests *failed* (i.e., Case 1 and Case 2 conditions do not hold, this is the case for 13, 14 and 15 line segments) shown in Figure35. For such non-trivial Cases the algorithm is processed, as follows.

Since, both the bitcodes for the end points P, Q of the line segment cannot be equal to 0000. Let us assume that the starting point of the line segment is P whose bit code is not equal to 0000. For example, for the line segment 15 we choose P to be the bitcodes 1001.

Now, scan the bitcode of P from the first bit to the fourth bit and find the position of the bit at which the bit value 1 appears at the first time. For the line segment 15 it appears at the very first position. If the bit value 1 occurs at the first position then proceed to intersect the line segment with the UP edge of the window and assign the first bit value to its point of intersection as 0.

Similarly, if the bit value 1 occurs at the second position while scanning the bit values at the first time then intersect the line segment PQ with the Down edge of the window and so on. This point of intersection may be labeled as P'. Clearly the line segment PP' is outside the window and therefore rejected and the new line segment considered for clipping will be P'Q.

The coordinates of P' and its remaining new bit values are computed. Now, by taking P as P', again we have the new line segment PQ which will again be referred to Case 1 for clipping.

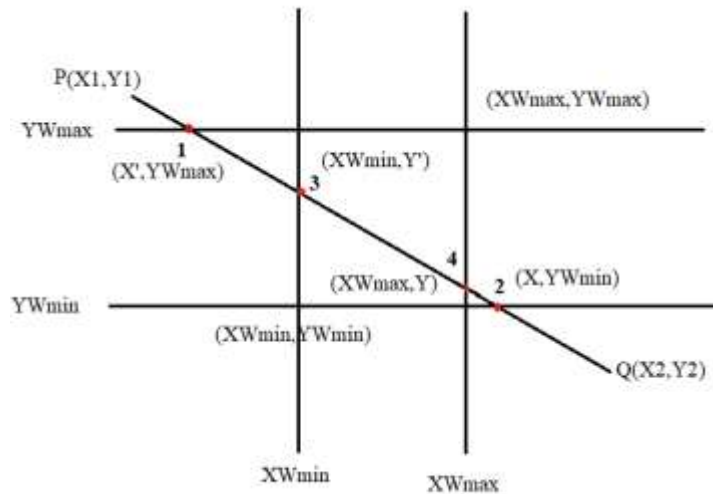


Figure 36: Line Clipping – Geometrically

Geometrical study of the above type of clipping (it helps to find point of intersection of line PQ with any edge) figure 36.

Let (x_1, y_1) and (x_2, y_2) be the coordinates of P and Q respectively.

1) **Top case/above**

if $y_1 > yw_{max}$ then 1st bit of bit code = 1 (signifying above) else bit code = 0

2) **Bottom case/below case**

if $y_1 < yw_{min}$ then 2nd bit = 1 (i.e. below) else bit = 0

3) **Left case:** if $x_1 < xw_{min}$ then 3rd bit = 1 (i.e. left) else 0

4) **Right case:** if $x_1 > xw_{max}$ then 4th bit = 1 (i.e. right) else 0

Similarly, the bit codes of the point Q will also be assigned.

1) **Top/above case:**

equation of top edge is: $y = yw_{max}$. The equation of line PQ is

$$y - y_1 = m (x - x_1) \quad \text{where,} \quad m = (y_2 - y_1) / (x_2 - x_1)$$

The coordinates of the point of intersection will be (x, yw_{max})

∴ equation of line between point P and intersection point is

$$(yw_{max} - y_1) = m (x - x_1)$$

rearrange we get

$$x = x_1 + (1/m)(yw_{\max} - y_1) \text{ ----- (A)}$$

Hence, we get coordinates (x, yw_{\max}) i.e., coordinates of the intersection.

2) **Bottom/below edge** start with $y = yw_{\min}$ and proceed as for above case.

∴ equation of line between intersection point (x', yw_{\min}) and point Q i.e. (x_2, y_2) is

$$(yw_{\min} - y_2) = m(x' - x_2)$$

rearranging that we get,

$$x' = x_2 + (1/m)(yw_{\min} - y_2) \text{ ----- (B)}$$

The coordinates of the point of intersection of PQ with the bottom edge will be

$$(x_2 + (1/m)(yw_{\min} - y_2), yw_{\min})$$

3) **Left edge:** the equation of left edge is $x = xw_{\min}$.

Now, the point of intersection is (xw_{\min}, y) .

Using 2 point from the equation of the line we get

$$(y - y_1) = m(xw_{\min} - x_1)$$

$$\text{Rearranging that, we get, } y = y_1 + m(xw_{\min} - x_1) \text{ ----- (C)}$$

Hence, we get value of xw_{\min} and y both i.e. coordinates of intersection point is given by

$$(xw_{\min}, y_1 + m(xw_{\min} - x_1))$$

4) **Right edge:** proceed as in left edge case but start with $x = xw_{\max}$.

Now point of intersection is (xw_{\max}, y') .

Using 2 point form, the equation of the line is

$$(y' - y_2) = m(xw_{\max} - x_2)$$

$$y' = y_2 + m(xw_{\max} - x_2) \text{ ----- (D)}$$

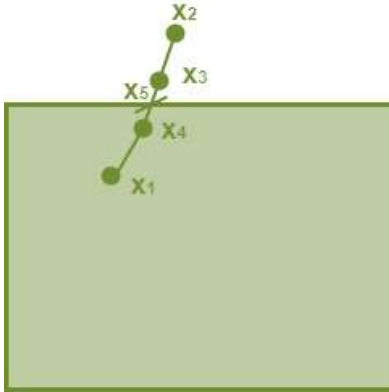
The coordinates of the intersection of PQ with the right edge will be

$$(xw_{\max}, y_2 + m(xw_{\max} - x_2))$$

B- MidPoint Subdivision Line Clipping Algorithm:

It is used for clipping line. The line is divided in two parts. Mid points of line is obtained by dividing it in two short segments. Again division is done, by finding midpoint. This process is continuing until line of visible and invisible category is obtained. Let (x_i, y_i) are midpoint

$$xm = \frac{x_1+x_2}{2} \quad ym = \frac{Y_1+Y_2}{2}$$



Step1: find $(x_1+x_2)/2$ i.e. $x_3=(x_1+x_2)/2$

Step2: find $x_4=(x_3+x_1)/2$

Step3: find $x_5=(x_3+x_4)/2$

x_5 lie on point of intersection of boundary of window.

Algorithm of midpoint subdivision Line Clipping:

Step1: Calculate the position of both endpoints of the line

Step2: Perform OR operation on both of these endpoints

Step3: If the OR operation gives 0000

then

Line is guaranteed to be visible

else

Perform AND operation on both endpoints.

If $AND \neq 0000$

then the line is invisible

else

$AND=0000$

then the line is clipped case.

Step4: For the line to be clipped. Find midpoint

$$X_m = (x_1 + x_2) / 2$$

$$Y_m = (y_1 + y_2) / 2$$

X_m is midpoint of X coordinate.

Y_m is midpoint of Y coordinate.

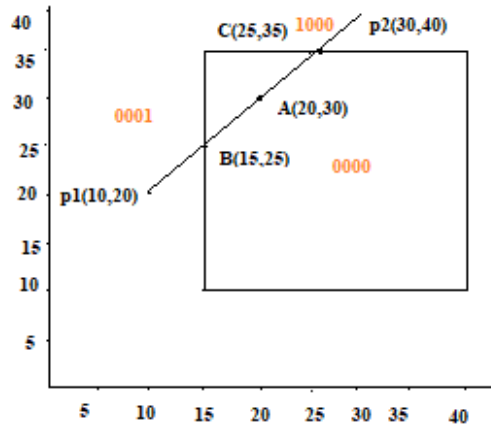
Step5: Check each midpoint, whether it nearest to the boundary of a window or not.

Step6: If the line is totally visible or totally rejected not found then repeat step 1 to 5.

Step7: Stop

Example

Window coordinates xmin, xmax, ymin, ymax as (15,40,10,35), clip the line p1p2 where P1(10,20) & p2(30,40) using midpoint subdivision algorithm .



Solution

1-Ap1

$$x = \frac{10+30}{2} = 20$$

$$y = \frac{20+40}{2} = 30$$

A(20,30)

A=0000

P1=0001 &

0000



midpoint **A**

2-p1A

$$x = \frac{30}{2} = 15$$

$$y = \frac{50}{2} = 25$$

B(15,25)

P1=0001 &
B=0001

0001 → REJECTED

3-BA

$$x = \frac{30}{2} = 15$$

$$y = \frac{50}{2} = 25$$

B(15,25)

A=0000 &
B=0000

0000 → midpoint **B**

4-Ap2

$$x = \frac{50}{2} = 25$$

$$y = \frac{70}{2} = 35$$

C(25,35)

A=0000 &
P2=1000

0000 → midpoint **C**

5-AC

A=0000 &
C=0000

0000 → VISIBLE

6-Cp2

C=1000 &
P2=1000

1000 → REJECTED

Line segment	Region code	AND	Result	Action
P1p2	0001 1000	0000	Partially visible	Find midpoint A
P1A	0001 0000	0000	Partially visible	Midpoint B
P1B	0001 0001	0001	Invisible	Rejected
BA	0000 0000	–	Completely visible	Draw the line
Ap2	0000 1000	0000	Partially visible	Midpoint C
AC	0000 0000	–	Completely visible	Draw the line
Cp2	1000 1000	1000	Invisible	Rejected

3- Intersection Point

If M is the slope of the line segment between points (X1, Y1) and (X2, Y2) then if $X1 \neq X2$

$$M = \frac{Y2 - Y1}{X2 - X1}$$

Then any point (X, Y) on the line is

$$M = \frac{Y - Y1}{X - X1}$$

- If the line segment crosses a left or right window edge then $X1 \neq X2$ and M has non denominator.
- If the line crosses a top or bottom window edge then $Y1 \neq Y2$ and the reciprocal of the slope $1/m$ has a nonzero denominator.

The slope M is obtained from the two given endpoints.

- If we are testing against a left or right direction the X value is known (the left or right edge value).

The X value is substituted into the equation for Y.

$$Y = M * (X - X1) + Y1$$

For top and bottom the Y value known then:

$$X = 1/M * (Y - Y1) + X1$$

C: Polygon Clipping Algorithm

A polygon can also be clipped by specifying the clipping window. Sutherland Hodgeman 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.

First the polygon is clipped against the left edge of the polygon window to get new vertices of the polygon. These new vertices are used to clip the polygon against right edge, top edge, bottom edge of the clipping window as shown in the figure 37.

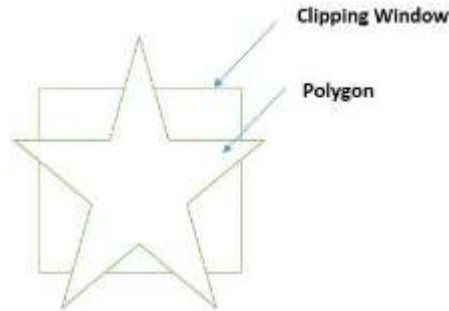


Figure 37: Polygon Before Clipping

While processing an edge of a polygon with clipping window, an intersection point is found if edge is not completely inside clipping window and the a partial edge from the intersection point to the outside edge is clipped. The figures 38-41 show left, right, top and bottom edge clippings:

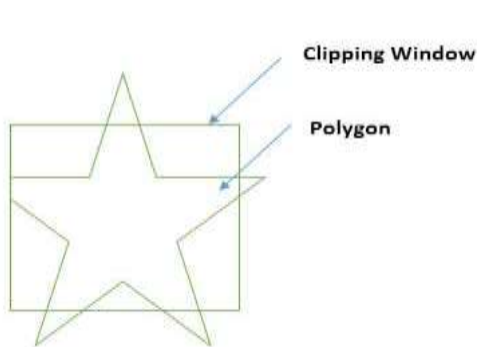


Figure 38: Clipping Left Edge

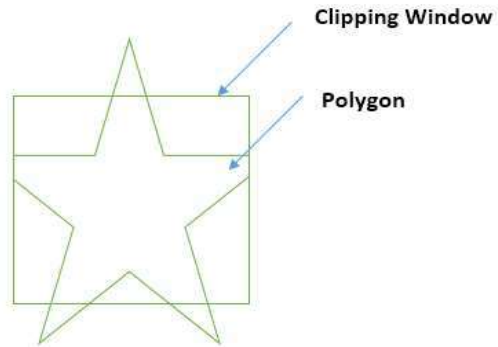


Figure 39: Clipping Right Edge

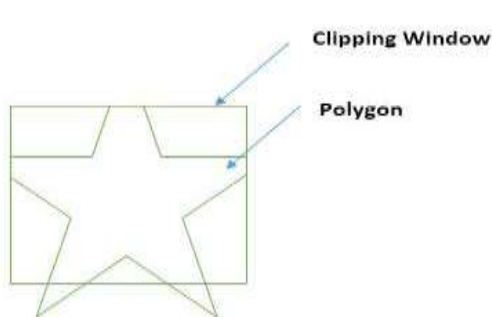


Figure 40: Clipping Top Edge

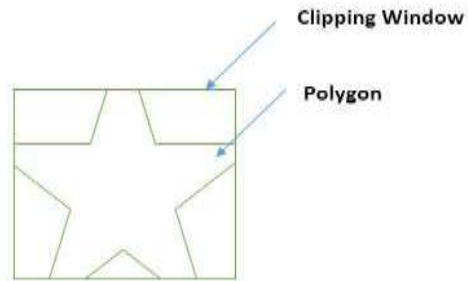


Figure 41: Clipping Bottom Edge

1. If the first vertex is an outside the window, the second vertex is inside the window. Then second vertex is added to the output list. The point of intersection of window boundary and polygon side (edge) is also added to the output line.
2. If both vertexes are inside window boundary. Then **only** second vertex is added to the output list.
3. If the first vertex is inside the window and second is an outside window. The edge which intersects with window is added to output list.
4. If both vertices are the outside window, then nothing is added to output list.

The input polygon is: V1, V2, V3, V4 (set of vertices)

When we make **Left** Clipping the result is: V2 P1 P2 V1

When we make **Right** Clipping the result is: P2V1V2P1

When we make **Top** Clipping the result is: P1 P2 V1V2

When we make **Bottom** Clipping the result is: V1V2 P1P2

We apply the operation 4 times (Bottom, right, top, left)

Left :- V2 P1 P2 V1

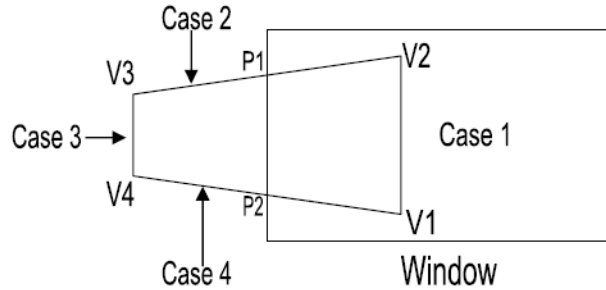
Bottom: - P1 P2 V1 V2

Right :- P2 V1 V2 P1

Top :- P1 P2 V1 V2

- Clips a polygon against each edge of the window.
- For each edge it inputs a list of vertices and outputs a new list of vertices.
- The input list is a sequence of consecutive vertices of the polygon obtained from the previous edge clipping.

Example1



There are 4 possible cases:

Case 1:

First and second V1, V2 inside the window, **V2** is sent to the output list.

Case 2:

First vertex V2 inside and the second vertex V3 outside the window, the intersection point (**P1**) of the side of the polygon joining the vertices and the edge is added to the output list.

Case 3:

Both vertices V3, V4 outside the window and no point are output.

Case 4:

First vertex V4 outside the window and the second vertex V1 inside the window, the intersection point (**P2**) and the second vertex **V1** are added to the output list.

The result of this left clipping is the transformation of

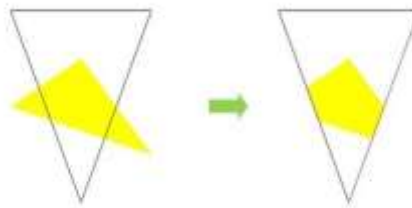
input list {V1, V2, V3, V4} to the output list {V2, P1, P2, V1}.

Example2

Input: Polygon: (100,150), (200,250), (300,200)

Clipping Area: (100,300), (300,300), (200,100)

Output: (242, 185) (166, 166) (150, 200) (200, 250) (260, 220)



5.2 Aspect Ratio

Aspect Ratio: is the ratio of the horizontal width to the vertical height.

- The horizontal and vertical plots of an equal number of pixels have different lengths.
- The ratio is a consequence of non – square pixels and rectangular display screen.

Example

If we plot eight pixels horizontally on the display screen and then we measure the line, we find that it is 0.3 cm wide. If we plot eight pixels vertically on the display screen and then we measure the line, we find that its 0.4 cm height.

The $A.R = 0.4 / 0.3 = 1.33$

New number of pixels = Old number of pixels * AR

New number of pixels = $8 * 1.33 \rightarrow = 10.64 \approx 11$ pixels.

$11 - 8 = 3$ pixels will be added to the horizontal line. Figure 42.

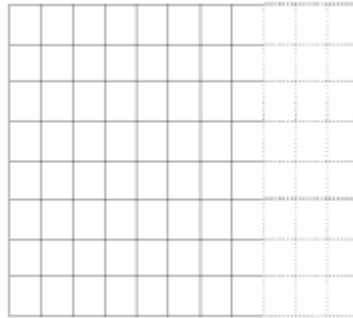


Figure 42: pixels added to the horizontal line

5.3 The Normalized Device Co-ordinate

Different display devices may have different screen sizes as measured in pixels. If we wish our program to be *devices independent* we should specify the coordinates in some units other than pixels and then use the interpreter to *convert these coordinates* to the appropriate values for particular display, we are using device independent units are called the *normalized device coordinates*. We will consider using only a square portion of the device.

Windows in World Coordinates Space (WCS) will be mapped to viewports that are specified within a unit square in NDC space. Map viewports from NDC coordinates to the screen, figure 43.

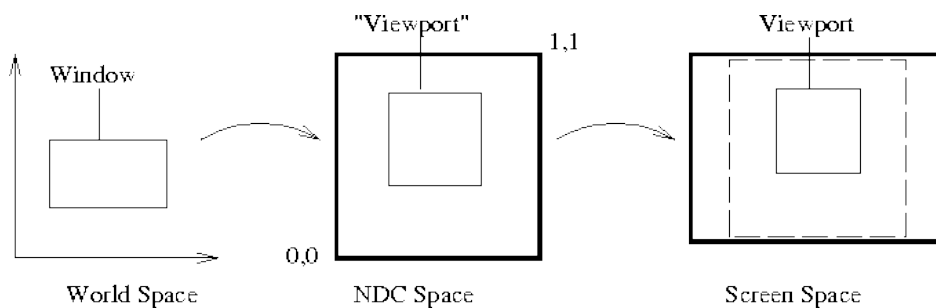


Figure 43: Mapping from WCS to Screen Space

For 2D drawing, the visible range of the display window is from $[-1,-1]$ to $[1,1]$. In other words, you need to transform your points to this range so that they will be visible. This is called *Normalized Device Coordinate* (NDC) system, Figure 44.

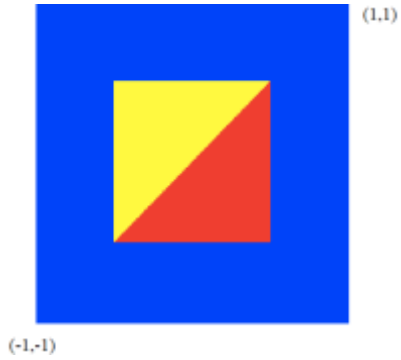


Figure 44: Normalized Device Coordinates

A *pixel in a window* is referenced as two integers (i,j) . This is called the *screen coordinate* (SC) System, figure 45.



Figure 45: Screen Coordinates

Just do a linear mapping from $[-1,-1] \times [1,1]$ to $[0,0] \times [I_{\max}, J_{\max}]$. Assume (x,y) is in NDC, (i,j) is in SC, then

$$i = (x - (-1))/2.0 * I_{\max}$$

$$j = (y - (-1))/2.0 * J_{\max}$$

If you do not want to use the entire window, you can define a sub-area called ‘*Viewport*’ as your drawing area. Your drawing will only show up in the viewport. Your points will be mapped from NDC to viewport, figure 46.

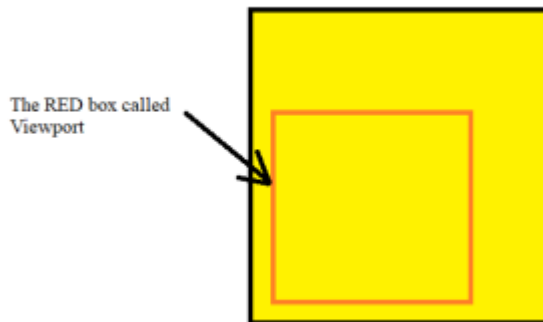


Figure 46: Mapping from NDC to viewport

Convert the vertex coordinates from the normalized device coordinates (NDC) to the screen space, the NDC has the range of (-1,1) in both X and Y for everything that is visible.

Just do a linear mapping from $[-1,-1] \times [1,1]$ to $[I_{min}, J_{min}] \times [I_{max}, J_{max}]$.

Assume (x,y) is in NDC, (i,j) is in SC, then

$$i = (x - (-1))/2.0 * (I_{max}-I_{min}) + I_{min}$$

$$j = (y - (-1))/2.0 * (J_{max}-J_{min}) + J_{min}$$

Viewport: the rectangular region in the screen for displaying the graphical objects (contents of the window). Viewport is defined using the screen coordinate system in pixels $[I_{min}, J_{min}]$, Figure 47. That is done when we want to display different views of the image in different regions of the screen. Using the same window and several viewports, the same object can be placed in different regions. Choosing different windows and viewports results in a screen with multiple images.

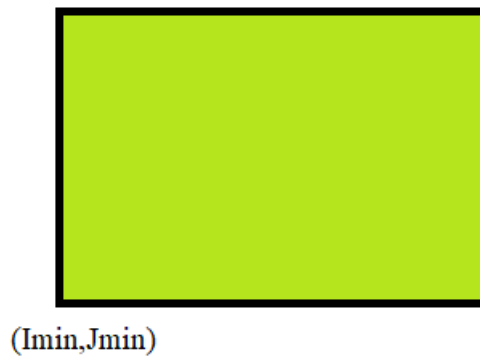
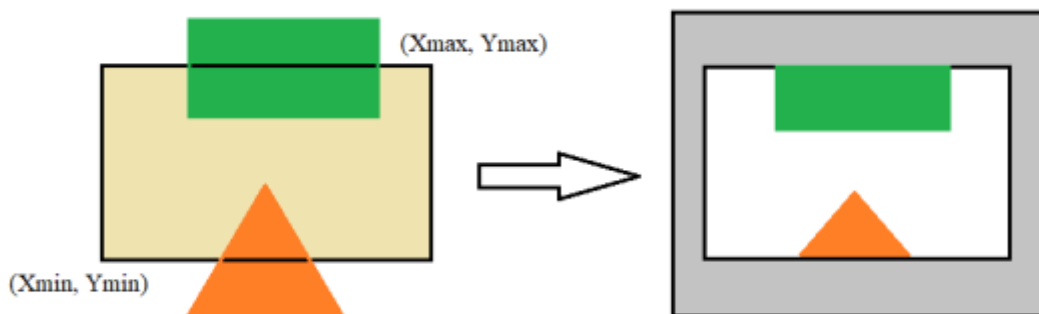


Figure 47: Viewport defined using screen coordinate system

As we discussed, only points that are mapped to the $[-1,-1] - [1,1]$ in NDC space are visible, figure 48.



$$i = (x - (-1))/2.0 * (I_{max}-I_{min}) + I_{min}$$

$$j = (y - (-1))/2.0 * (J_{max}-J_{min}) + J_{min}$$

Figure 48: Viewport Mapping

Windowing: is the capability of displaying part of the World Coordinate System (WCS) image, enclosed in a rectangular region. That is when the image described in the world coordinate system is too complicated to be viewed clearly on the screen and the user may want to view (and enlarge) only a portion of the image. Adjusting the size of the window has the effect of enlarging or shrinking or even distorting a portion of the image or the entire image.

World coordinate system: is a user-defined coordinate system chosen for a specific application. These screen independent coordinate can have a large or small numeric range, negative values and fraction.

NDC (Normalized Device Coordinate): We need to use a device independent coordinate system called (NCD) to describe the viewport. This (NCD) better than using the coordinate of the display screen to describe the viewport. The (NCD) system allows the application programmer to write graphics programs independent of the resolution of the display screen

Point (-1, -1) in NDC is located at the bottom left corner (Y up), figure 49.

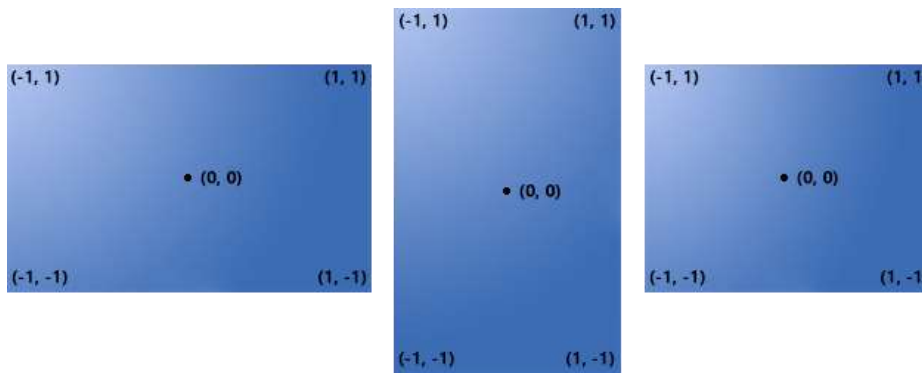


Figure 49: Normalized Device Coordinates Resize with the Screen Size

- 1) Framebuffer Coordinate (Viewport coordinate):** when we write into attachment or read from attachment or copy/bit between attachments, we use framebuffer coordinate to specify the location. The **origin (0, 0)** is located at the **top-left corner (Y down)**, figure 50.

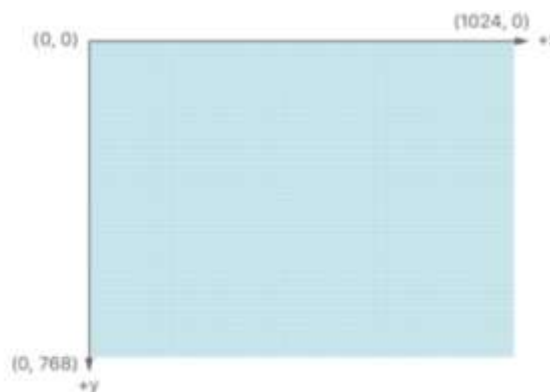


Figure 50: Framebuffer Coordinate

2) **Texture Coordinate:** when we upload texture into memory or sample from texture, we use texture coordinate. The **origin (0, 0)** is located at the **top-left corner (Ydown)**. Figure 51.



Figure 51: Texture Coordinate

5.4 Window to Viewport mapping

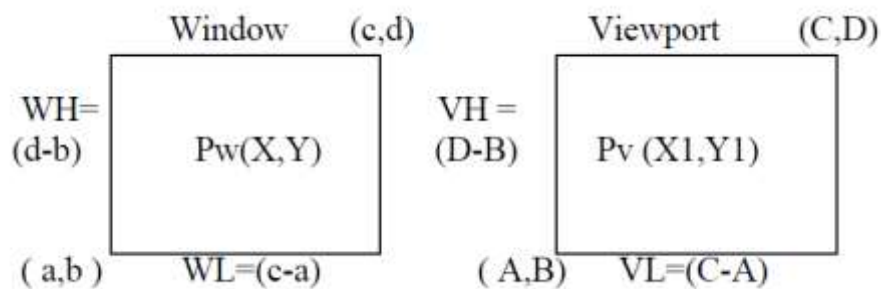
It is a transformation function that converts from Window coordinate system to (NCD). It produces on a specific view of the image.

To change the view we must change the window and/or the viewport and record this mapping. The window and the viewport are described by the coordinates of the lower left vertex.

5.5 Window to viewport mapping algorithm

We take the coordinates of the points of the object in a designated window and transform them to give the coordinates of the corresponding points in the viewport on the screen.

This mapping requires three steps:



1- Window Shift

Shift the *lower left corner* of the window to the origin:

$$W = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -a & -b & 1 \end{bmatrix}$$

2- Scale the window dimensions to the dimensions of the viewport.

The scaling involves a factor of

$$\frac{(C - A)}{(c-a)} \quad \text{in the X direction}$$

$$\frac{(D - B)}{(d-b)} \quad \text{in the Y direction}$$

The matrix of the local *scaling* is:

$$S = \begin{array}{|c|c|c|} \hline \frac{(C - A)}{(c-a)} & 0 & 0 \\ \hline 0 & \frac{(D - B)}{(d-b)} & 0 \\ \hline 0 & 0 & 1 \\ \hline \end{array}$$

3- Viewport shift

Shift the *lower left corner* of the viewport from the screen origin to its proper position.

$$V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ A & B & 1 \end{bmatrix}$$

Multiplying the three matrices in order, we get a single matrix for the window to viewport mapping:

$$M = W * S * V$$

Or by using the equation

$$X1 = (X - a) * (VL/WL) + A$$

$$Y1 = (Y - b) * (VH/WH) + B$$

Where (X, Y) is the point in the window and to be mapped to the point in the viewport (X1, Y1)

Example

Find the normalization transformation that maps a window whose lower left corner is at (1,3) and upper right corner is at (3,5) onto a viewport that has lower left corner at (0.2,0.5) and upper right corner (0.8,0.9) .What position of point p(2.5,3.5) in window onto viewport coordinate?

Solution The matrix for the *window shift* is:

$$W = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ -1 & -3 & 1 \end{bmatrix} \quad \text{back to origin (0,0)}$$

The *scaling matrix* is

$$S = \begin{bmatrix} (0.8-0.2)/(3-1) & 0 & 0 \\ 0 & (0.9-0.5)/(5-3) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The matrix for the *viewport shift*

$$V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0.2 & 0.5 & 1 \end{bmatrix}$$

When we multiply these three matrices together (remembering that order matters) we get

$$M = W S V$$

$$= \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.2 & 1 \\ -0.1 & -0.1 & 1 \end{bmatrix}$$

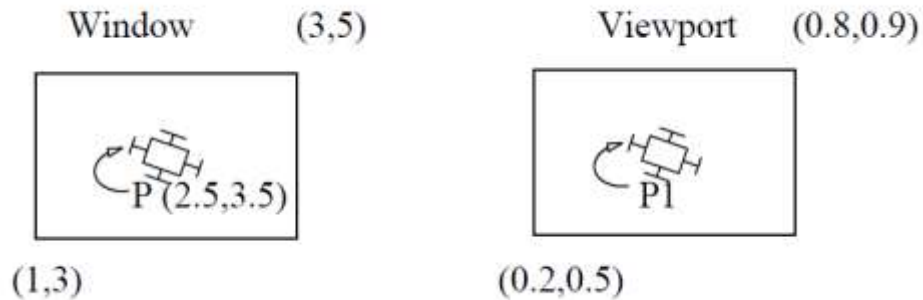
The point P has coordinate (2.5, 3.5) so its homogeneous vector is [2.5 3.5 1]. We multiply this by the matrix M and obtain:

$$[2.5 \quad 3.5 \quad 1] \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.2 & 1 \\ -0.1 & -0.1 & 1 \end{bmatrix} = [0.65 \quad 0.6 \quad 1]$$

Thus the coordinate of P* in the viewport are (0.65, 0.6).

Example: Point P is to be displayed on part of a screen as indicated in the figure. Describe the sequence of transformation that will transform the window shown to the viewport indicated and give the matrix for each.

Calculation the coordinates of P where it appears on the screen



OR

Find the normalization transformation that maps a window whose lower left corner is at(1,3) and upper right corner is at (3,5) onto a viewport that has a lower left corner (0.2,0.5) and upper right corner (0.8,0.9), what position of point p in the window is (2.5,3.5) onto viewport coordinate?

Solution

$$W = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -3 & 1 \end{bmatrix}$$

$$S = \begin{array}{|c|c|c|} \hline \frac{(0.8 - 0.2)}{(3 - 1)} & 0 & 0 \\ \hline 0 & \frac{(0.9 - 0.5)}{(5 - 3)} & 0 \\ \hline 0 & 0 & 1 \\ \hline \end{array}$$

$$V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & 0.5 & 1 \end{bmatrix}$$

So the single matrix M will be:

$$M = \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.2 & 0 \\ -0.1 & -0.1 & 1 \end{bmatrix}$$

P1 (in viewport) = P (in window) * M

$$= [2.5 \ 3.5 \ 1] \times \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.2 & 0 \\ -0.1 & -0.1 & 1 \end{bmatrix}$$

OR by using the equation

$$X1 = (2.5-1) * (0.6/2) + 0.2 = 0.65$$

$$Y1 = (3.5-3) * (0.9-0.5)/(5-3) + 0.5 = 0.6$$

Part Six

Three – Dimensional Transformations

6. Three – Dimensional Transformations

In the 2D system, we use only two coordinates X and Y but in 3D, an extra coordinate Z is added. Furthermore, 3D graphics components are now a part of almost every personal computer and, although traditionally intended for graphics-intensive software such as games, they are increasingly being used by other applications. The techniques used in computer graphics to display 3-D world are a mathematical model instead of lens to create the image.

1- Coordinates Systems

A three- dimensional coordinate system can be view as an extension of the two dimension coordinate system. The third – dimension **depth** is represented by the **Z – axis** figure 52. A point can be described by triple (**X , Y , Z**) of coordinate values.

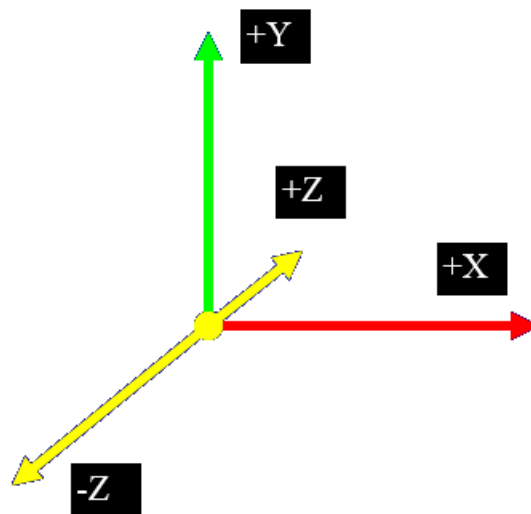


Figure 52: Three- Dimensional Coordinates

2. 3D Transformations

A transformation is the process of mapping points to other positions. The manipulation viewing and creation of 3-D images require the use of 3-D geometric and coordinate transformations. Just as in the case of 2D, we represent the transformation operations as a series of matrix operations. Since in the 2-dimensional case we were representing a point (x,y) as a tuple [x y 1], in the 3-dimensional, a point (x,y,z) will be associated with *homogeneous* row vector [x,y, z, 1]. We can represent all three dimensional linear transformation by multiplication of 4*4 matrix.

2.1 Translation

It is *the movement of an object from one position to another position*. Translation is done using translation vectors. There are three vectors in 3D instead of two. These vectors are in x, y, and z directions. Translation in the x-direction is represented using T_x . The translation is y-direction is represented using T_y . The translation in the z- direction is represented using T_z .

If P is a point having co-ordinates in three directions (x, y, z) is translated, then after translation its coordinates will be (x¹ y¹ z¹) after translation. T_x T_y T_z are translation vectors in x, y, and z directions respectively.

$$\begin{aligned}
 x^1 &= x + T_x \\
 y^1 &= y + T_y \\
 z^1 &= z + T_z
 \end{aligned}$$

Three-dimensional transformations are performed by transforming each vertex of the object. If an object has five corners, then the translation will be accomplished by translating all five points to new locations. Figure 53 shows the effect of translation:

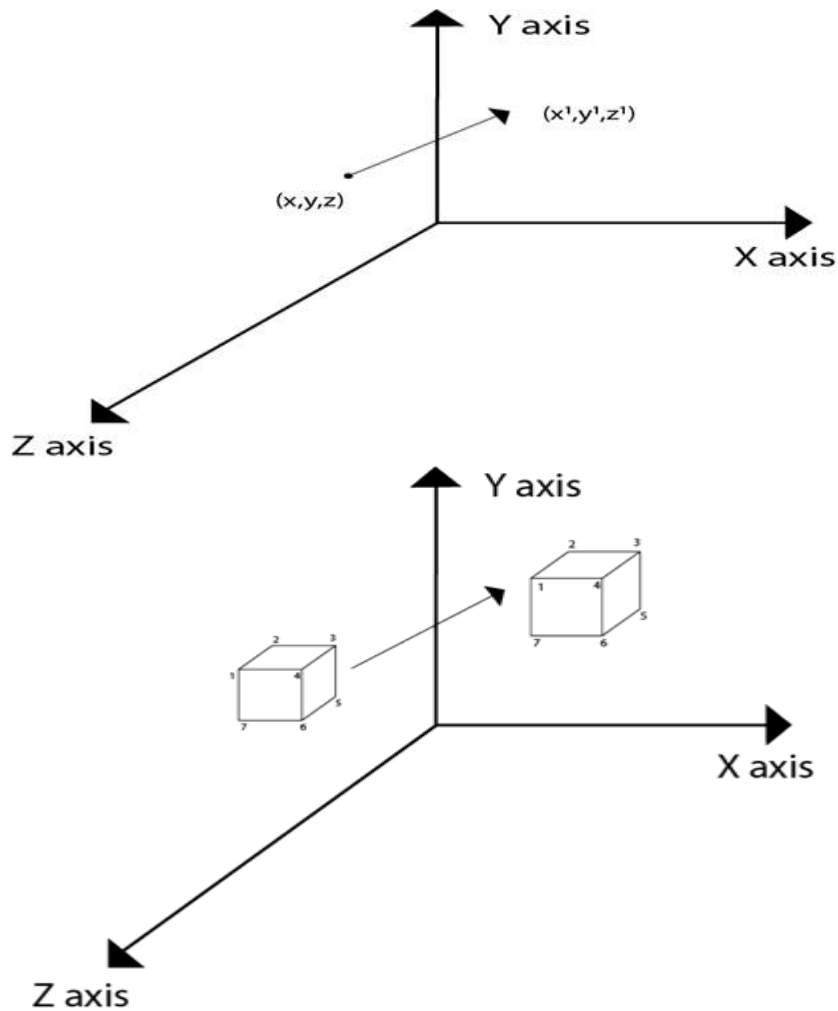


Figure 53: 3D Translation

A point can be translated in 3D by adding translation coordinate (T_x, T_y, T_z) to the original coordinate (X, Y, Z) to get the new coordinate (X', Y', Z').

Matrix for translation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

$$P' = P.T$$

$$\begin{aligned} [x' \ y' \ z' \ 1] &= [x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix} \\ &= [x+T_x \quad y+T_y \quad z+T_z \quad 1] \end{aligned}$$

A point (x, y, z) is translated to a new position (x', y', z') by moving it T_x units in the X – direction and by T_y units in the Y – direction and T_z units in Z – direction. Mathematically this can be represented as:-

$$\begin{aligned} x' &= x + T_x \\ y' &= y + T_y \\ z' &= z + T_z \end{aligned}$$

Example

Move a point P that is located at $(3, 4, 5)$ to a new location with distance $(2, 4, 6)$ units.

$$\begin{aligned} \text{Given, } P &= [3, 4, 5, 1] \\ T_x &= 2, \quad T_y = 4, \quad T_z = 6 \end{aligned}$$

We apply the translation transformation and obtain the result as:

$$[3 \ 4 \ 5 \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 4 & 6 & 1 \end{bmatrix} = [3+2+4+4+5+6+1] = [5 \ 8 \ 11 \ 1]$$

Hence the result is the point P being shifted to a new position P'.

2.2 Scaling

Scaling is used to change the size of an object. The size can be increased or decreased. Scaling can be achieved by multiplying the original coordinates of the object with the *scaling factor* to get the desired result. If the scaling factor is *greater than 1*, the object is enlarged, if the factor is *less than 1*, the object is made smaller, a factor of *1* has no effect on the object. Whenever scaling is performed, there is one point that remains at the same location. This is called **fixed point** of the scaling transformation. In the scaling, three factors are required S_x , S_y and S_z .

S_x = Scaling factor in x-direction

S_y = Scaling factor in y-direction

S_z = Scaling factor in z-direction

Figure 54. The letter S denotes the basic scaling matrix.

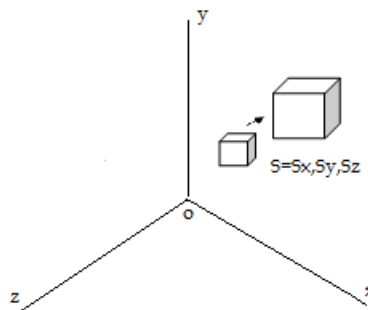


Figure 54: 3D Scaling

a) To scale an object from **origin point** we used the following matrix.

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

b) To scale an object from **fixed point** (x_p, y_p, z_p), figure 55, we perform the following three steps:

step1: Translate the fixed point (x_p, y_p, z_p) to the origin. Every point (x, y, z) is moved to a new point (x_p, y_p, z_p):

$$X_1 = X - X_p$$

$$Y_1 = Y - Y_p$$

$$Z_1 = Z - Z_p$$

step2: Scale these translated points with the origin as the fixed points:

$$X_2 = X_1 \times S_x$$

$$Y_2 = Y_1 \times S_y$$

$$Z_2 = Z_1 \times S_z$$

step3: The fixed point is translated to its original position (x_p, y_p, z_p):

$$X_3 = X_2 + X_p$$

$$Y_3 = Y_2 + Y_p$$

$$Z_3 = Z_2 + Z_p$$

$$[x_1 \ y_1 \ z_1 \ 1] = [x \ y \ z \ 1] \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -X_p & -Y_p & -Z_p & 1 \end{bmatrix} \times \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ X_p & Y_p & Z_p & 1 \end{bmatrix}$$

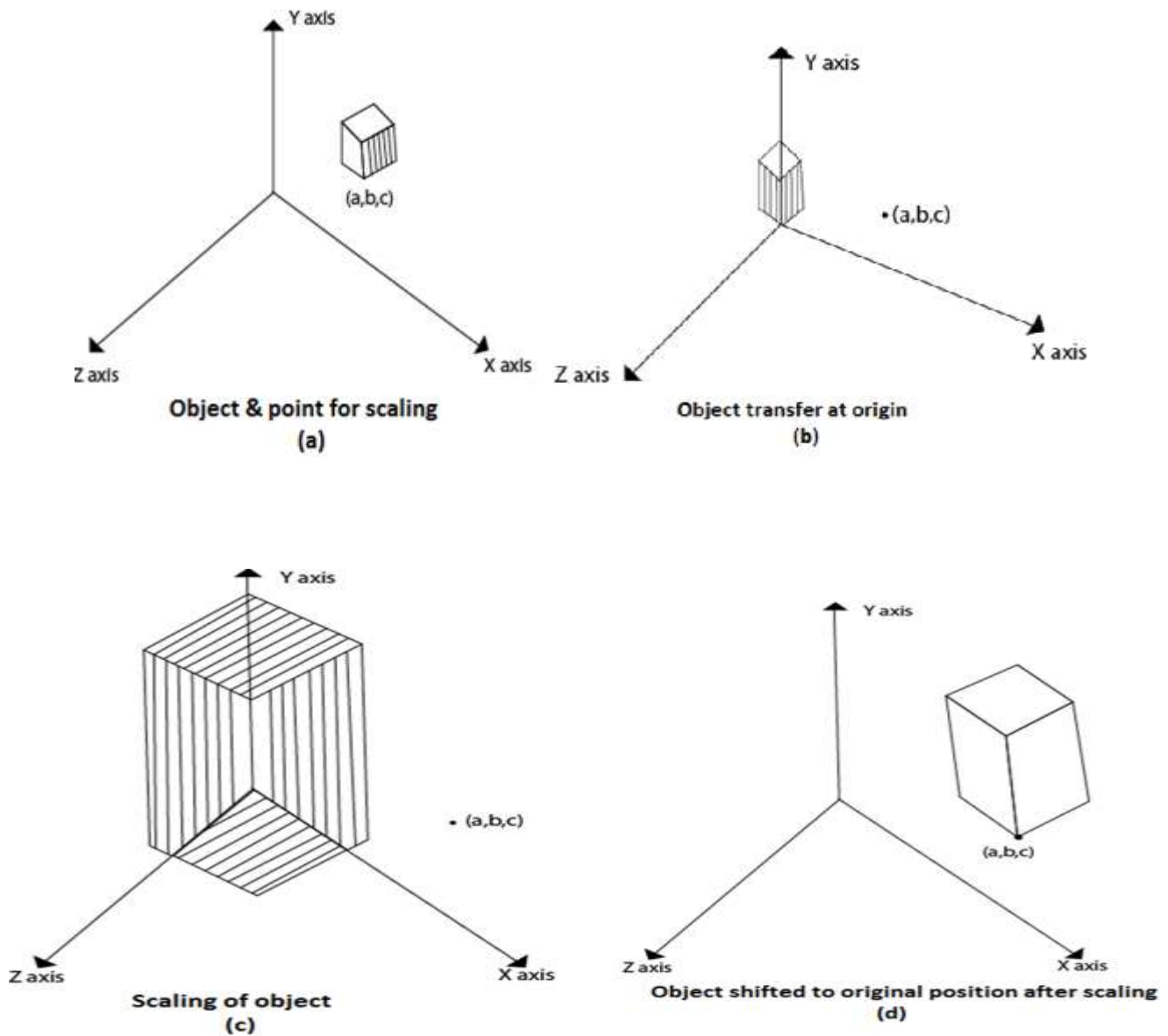


Figure 55: Scale an object from fixed point

2.3 Rotation

It is moving of an object about an angle. Movement can be anticlockwise or clockwise. 3D rotation is complex as compared to the 2D rotation. For 2D we describe the angle of rotation, but for a 3D angle of rotation and axis of rotation are required. The axis can be either x or y or z , figure 56 and figure 57 explain the rotation about various axes.

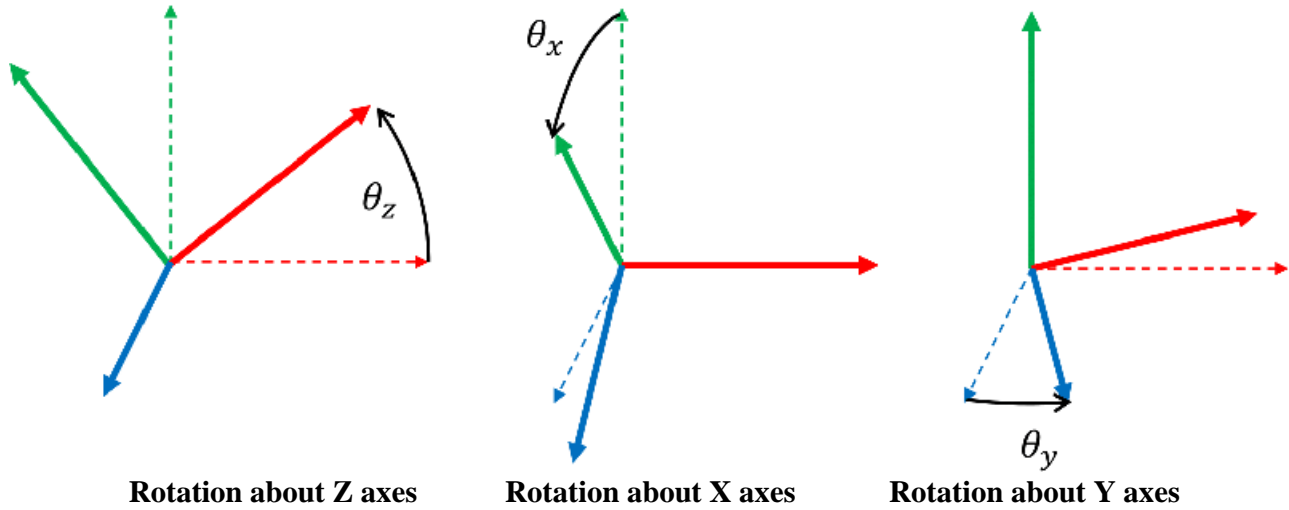


Figure 56: Axis-aligned rotations

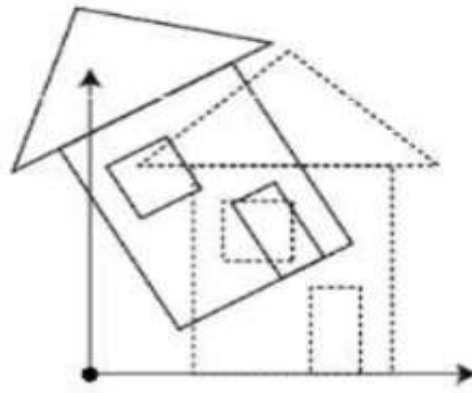


Figure 57: 3D Rotation

The *matrices* that are used to achieve *rotation* can be shown as:

a) **Rotation about X – axis:**

$$R_x(\varnothing) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varnothing) & -\sin(\varnothing) & 0 \\ 0 & \sin(\varnothing) & \cos(\varnothing) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

b) Rotation about Y – axis:

$$R_y(\varnothing) = \begin{bmatrix} \cos(\varnothing) & 0 & \sin(\varnothing) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\varnothing) & 0 & \cos(\varnothing) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

c) Rotation about Z – axis:

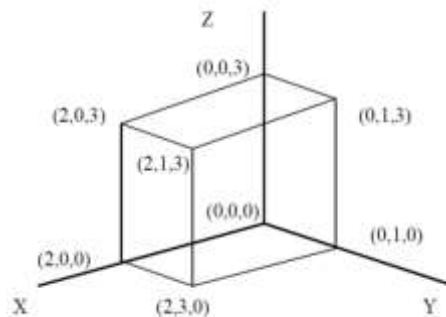
$$R_z(\varnothing) = \begin{bmatrix} \cos(\varnothing) & -\sin(\varnothing) & 0 & 0 \\ \sin(\varnothing) & \cos(\varnothing) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example

Draw the figure (0,0,0) , (0,1,0) ,(0,1,3) , (0,0,3) , (2,0,0) , (2,3,0), (2,0,3) , (2,1,3), and find:

- a) Translate it to the point (0,3,0)
- b) Scaling 4 times its size.
- c) Rotate its (90°) about the Z – axis.

Note: $\sin(90) = 1$, $\cos(90) = 0$



Solution

a-Translate it to the point (0,3,0)

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 3 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 2 & 0 & 3 & 1 \\ 2 & 1 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 3 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 0 & 1 \\ 0 & 4 & 0 & 1 \\ 0 & 4 & 3 & 1 \\ 0 & 3 & 3 & 1 \\ 2 & 3 & 0 & 1 \\ 2 & 6 & 0 & 1 \\ 2 & 3 & 3 & 1 \\ 2 & 4 & 3 & 1 \end{bmatrix}$$

b- Scaling 4 times its size

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 3 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 2 & 0 & 3 & 1 \\ 2 & 1 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 4 & 0 & 1 \\ 0 & 4 & 12 & 1 \\ 0 & 0 & 12 & 1 \\ 8 & 0 & 0 & 1 \\ 8 & 12 & 0 & 1 \\ 8 & 0 & 12 & 1 \\ 8 & 4 & 12 & 1 \end{bmatrix}$$

c- Rotate its (90°) about the Z – axis

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 3 & 1 \\ 0 & 0 & 3 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 2 & 0 & 3 & 1 \\ 2 & 1 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} \cos(90) & -\sin(90) & 0 & 0 \\ \sin(90) & \cos(90) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 3 & 1 \\ 0 & -2 & 0 & 1 \\ 3 & -2 & 0 & 1 \\ 0 & -2 & 3 & 1 \\ 1 & -2 & 3 & 1 \end{bmatrix}$$

Part Seven

Three – Dimensional Models

7.1 What is 3D Modeling?

The term “3D modeling” refers to the process of creating a three-dimensional representation of an object using specialized software. This representation, called a 3D model, can convey an object’s size, shape and texture. The process works with points, lines, and polygons to create the 3D shapes within the software.

7.2 Usage of 3D Modeling

3D models can be used within a variety of fields. Some of the most common applications are:

- Planning buildings using architectural visualization.
- 3D printing
- Animation
- Designing 3D characters for animated films and video games.
- Building up of a product design.
- Creating digitized 3D garments for the fashion industry.
- Publishing: The possibilities for book covers and images designed with 3D modeling are endless.
- Gaming industry: 3D modeling is used to create game characters and scenes.
- Medicine: 3D modeling is used to design prosthetics, parts to repair damaged organs, even dentists use 3D modeling.

7.3 3D Models Creating Method

There are various techniques for creating a 3D model. Regardless of what industry you are using 3D modeling for, there are four main methods to choose from:

1. Primitive Modeling
2. Polygonal Modeling
3. Rational B-Spline Modeling
4. Non-Uniform Rational Basis Spline (NURBS)
5. CAD Software
 - a- Solid Modeling
 - b- Wireframe Modeling
 - c- Surface Modeling

7.3.1 Primitive Modeling

This type of 3D modeling mostly uses spheres, cubes, and other variations of these two shapes to put together the desired shapes. It is called *primitive* because it is a very rudimentary form of 3D modeling, mainly created through the combination of different pre-existing shapes.

This kind of modeling generally utilizes basic Boolean processors to get the right shapes and outlines. Boolean operators are some of the most common ways of generating three-dimensional surfaces and shapes. Designers can combine two different shapes or subtract one shape from another to create a new object.

7.3.2 Polygonal Modeling

This type of 3D modeling is done by working with X, Y, and Z coordinates to define different shapes and surfaces and then combining the different surfaces into one giant shape or model. Figures 58,59.

When designers use the polygonal modeling technique, they typically begin by creating a wire mesh in the desired shape – this requires a good working knowledge of the polygonal mesh theory, which means that this type of modeling might be too complex for beginners.

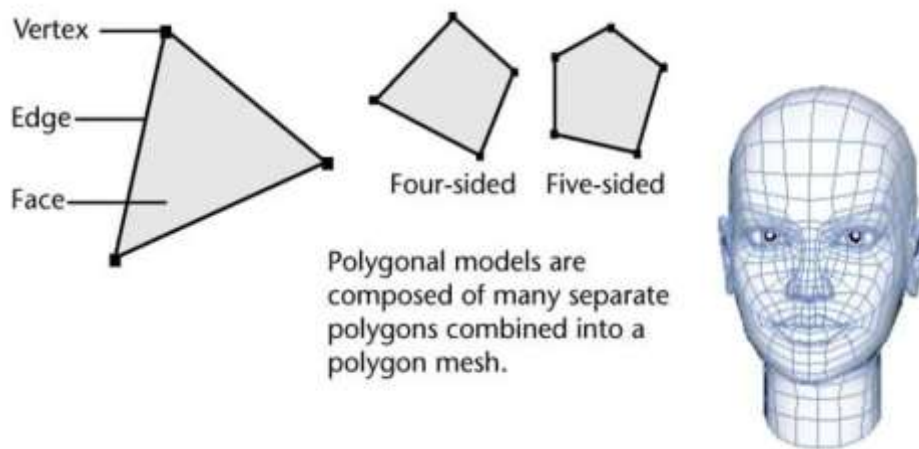


Figure 58: Polygons

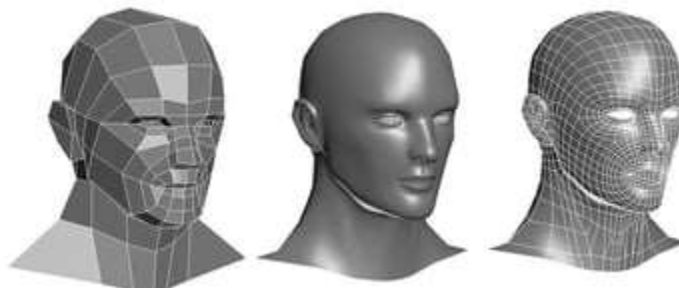


Figure 59: Polygons of a model are shaped using various tools to form a final model

7.3.3 Rational B-Spline Modeling

Rational B-Spline Modeling is the most common type of 3D modeling and also bases its technology on the combining and adjusting of geometric forms. Users begin by creating the dimensions of each shape, and these polygons can then be twisted and curved to get to achieve the desired 3D design.

7.3.4 Non-Uniform Rational Basis Spline (NURBS)

Are mathematical representations of 3D geometry that can accurately describe any shape from a simple 2D line, circle, arc, or curve to the most complex 3D organic free-form surface or solid. Because of their flexibility and accuracy, NURBS models can be used in any process, from illustration and animation to manufacturing.

7.3.5 CAD Software

CAD (Computer Aided Design) software is an important invention with regards to 3D modeling. It helps in the visualization of the desired objects, designs, and models in virtual reality. Some of the important tasks that can be accomplished with the use of CAD software are 3D printing, 3D sculpting, 3D rendering, and 3D modeling. The calculations involved in the making of 3D objects are undertaken by CAD software. The job of the designer is to define the shape and size of the 3D objects he/she wishes to create.

There are three major types of 3D modeling that fall under the rubric of CAD software: solid modeling, wireframe modeling, and surface modeling.

A- Solid Modeling

Solid Modeling is the computer modeling of 3D solid objects. The objective of Solid Modeling is to ensure that every surface is geometrically correct. It is considered the most complex aspect to master in CAD because it requires the CAD software to simulate the object from within and outside. This is critical as it lets designers provide cutaways of the design, such as an engine and its components. Figure 60.

In short, solid modeling allows the design, creation, visualization and animation of digital 3D models. The designer is able to see how the design looks and works from the very beginning.

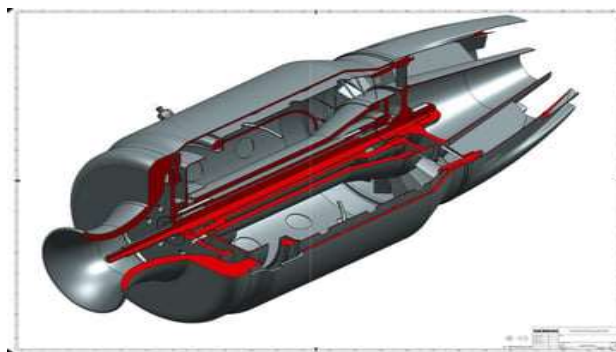


Figure 60: Solid Model

B- Wireframe modeling

Wireframe modeling represents shapes as a network of vertices. Each geometrical face is made out of minimum three vertices and every vertex can be important for at least one appearance. The size and state of things are altered by changing the position of every vertex.

Numerous wireframe demonstrating devices use triangles as their fundamental components, and the more triangles use, the real it looks. This is shown by “polygon check”, the absolute number of triangles (or other planar shapes) contained inside the wireframe of a model.

Wireframe modeling is based on generating a 3D model by “bending a wire” and following the edges of an object. These 3D models consist entirely of points, arcs, circles, curves, and lines.

A wireframe object is not recognized as solid. Instead, the boundary of the object is recorded as points and their connections. Figure 61.

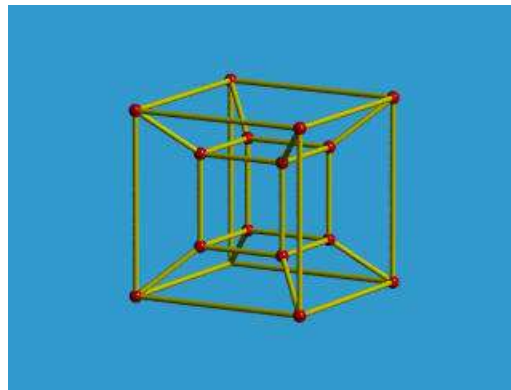


Figure 61: Wireframe of Blocks and Cylinder

C- Surface Modeling

Surface modeling is a mathematical method usually provided in computer-aided design applications for displaying solid-appearing objects. Surface modeling makes it possible for users to look at the specific object at specific angles with solid surfaces. It focuses on the external aspect of a 3D model, allowing you to view the 3D model from different angles. Figure 62. It's mostly used in architectural illustrations and animations in video games.

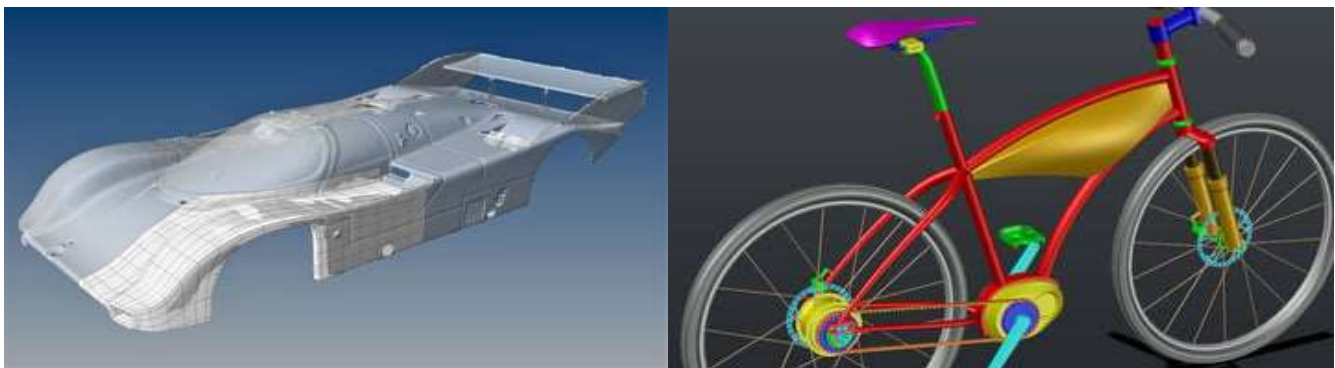


Figure 62: Shapes and Exterior Curves are First Defined in Surface Modeling

The Process of 3D Modeling

UNDERSTANDING THE PROCESS OF CREATING 3D Models

Creating 3D models may be a little stressful, but the process doesn't have to be overly complex. While this process may vary depending on the organization you work with, the below outlines the steps Concentrek takes to create high-quality and engaging 3D visuals.


STEP ONE: FILE REVIEW

Ensure the following questions are considered for all CAD files provided:

- ARE THE FILES UP-TO-DATE?**
The last thing you want is to produce a 3D model of an older version of a product; an engineer can ensure that the latest CAD files with the most updated product specs are being utilized.
- HAVE ALL TEAMS SUBMITTED APPROVAL?**
While working with engineering is important during this step, getting approval from product management and marketing can help square away any loose ends.


STEP TWO: DETERMINE RENDER STYLE

Depending on the goals and usage of the final renders, 3D models can be created based on two different approaches:



PHOTOREALISTIC

The goal of this approach is to make a 3D model that looks exactly like the product does in real life—just like a photo. This approach leaves little room for additional product interpretation.




STYLIZED


With this approach, the product being referenced needs to be enhanced for marketing purposes, so the final render may not look exactly like the product.


Note that whichever approach you choose will impact the future development of the 3D model in the coming steps.

STEP THREE: PROVIDE VISUAL REFERENCES

If whoever is developing the 3D model of your product needs additional visual references to ensure accuracy, we recommend the following:

- 


Set up an in-person meeting to further describe the product or have the product available for a more hands-on experience.
- 


Provide additional high-resolution photos or videos of the product with good lighting at various angles.
- 


Send a materials bill of material that includes what the product is made of to limit any confusion on if a certain piece is metal or plastic, for example.

STEP FOUR: DEVELOP MODELS

There are several steps taken during the development process—whether it's still image renders or a rendered video:

- 

Develop materials and textures to be applied to the model.
- 


Animate or add any special effects to specific parts if the 3D model will be used in a video.
- 

Create a virtual studio with appropriate lighting for the desired look.

STEP FIVE: RENDERING

The final step is to render any images or videos.

When rendering is complete, be sure to circle back to engineering, product management and marketing to ensure the final 3D model is accurate prior to sharing it with customers or utilizing it in materials.



Work with the Right Team

Creating 3D models is a complex process that requires a skilled team to complete. Fortunately, Concentrek is experienced with creating engaging 3D visuals.

REACH OUT TO US TODAY TO START THE PROCESS OF CREATING YOUR 3D MODELS.