Computer Graphics
رسوم الحاسوب

قسم علوم الحاسبات

References

1- "Principles of Interactive Computer Graphics", William M. Newman and Robert F. Sprooull, McGraw-Hill International Book Company, 1984.

2- "Computer Graphics with Pascal", Marc Berger, the Benjamin / Cummings Publishing Company, 1986.

3- "Computer Graphics",Zhigang Xiang and Roy A. Plastock, Schaum's outline Series, McGraw-Hill Company, 1992.

4- "Computer Graphics C Version", Donald Hearn and M. Pauline Baker, Prentice-Hall Company, 1997.

Computer Graphics

Introduction

Computer graphics can be defined as the creation and manipulation of graphic image by means of 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. Large amounts of data are rapidly converted into bar charts, pie charts, and graphs. Graphics displays have also improved our understanding of complex system such as molecular biology; reaffirming the saying that one picture is worth a thousand words.

Display Screens

Display screens are output devices that show programming instructions and data as they are being and information after it is processed, display screens are either CRT (Cathode-Ray-Tube) or flat-panel display.

CRT Displays: use vacuum tube like that in a TV set.

Flat-panel displays: are thinner, weightless, and consume less power than CRT displays but are not as clear. Principal flat-panel displays are liquid-crystal displays (LCD) and gas-plasma display.

The size of screen is measured diagonally from corner to corner in inches.

The Cathode Ray Tube (CRT) Display:

The CRT display screens consist of three components:

1- **Cathode Ray Tube (CRT).**
2- **Frame buffer.**
3- **Display controller.**

Cathode Ray Tube (CRT):

Consists of electron gun that contains a cathode that when heated emits a beam of negatively charged electrons towards a positively charged phosphor coated screen. The electron beam passes through the focusing and deflection system, which consist of an electrostatic or magnetic field. A color CRT has three electron guns, one for each of three primary colors: red, green, and blue.

The focusing system concentrates the beam so that by the time the electrons reach the screen, they have converted to small dot. The deflection system, which consists of two pairs of deflection plates (horizontal and vertical) directs the electron beam to any point on the screen.

When the electron beam strikes the screen, the phosphor emits a spot of visible light that intensity depends on the number of electrons on the beam. The duration of this light, called persistence, depends on the type of phosphor that coats the screen. In order to give the viewer the appearance of continuous flicker-free image, each dot on the screen must be intensified many times per second. This type of CRT is called a refresh CRT. Two types of refresh CRTs are available: raster-scan and random vector.

Frame Buffer:

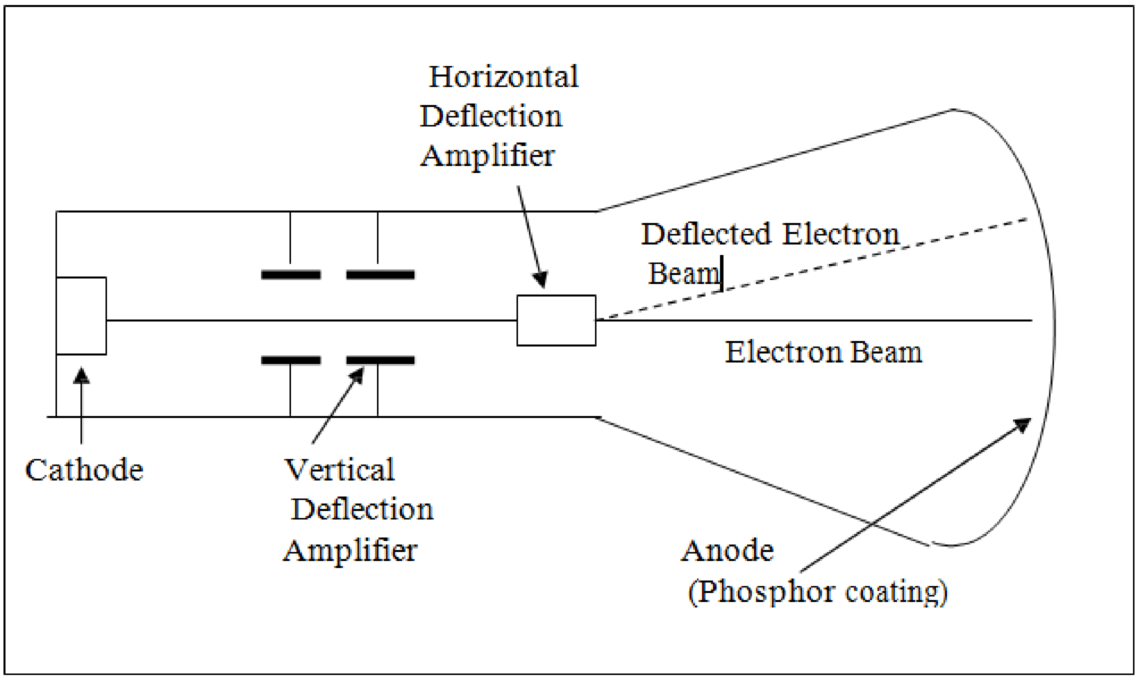
Each screen pixel corresponds to a particular entry in a two dimensional array residing in memory. This memory is called a frame buffer or a bit map. The number of rows in the frame buffer array equals the number of raster lines on the display screen. The number of columns in this array equals the number of pixels on each raster line.

The term pixel (picture element) is also used to describe the row and column location in the frame buffer arrays that corresponds to the screen location. A 512$\times$512 display screen requires 262,144 pixel memory location. Whenever we wish to display a pixel on the screen, a specific value is placed into the corresponding memory location in the frame buffer array. Each screen location pixel and corresponding memory location in the frame buffer is accessed by an (X,Y) integer coordinate pair. The x value refers to the columns, the y value to the row position.

Each pixel in the frame buffer array is composed of a number of bits. A black and white image that has only two intensity levels, on-off, has single bit plane frame buffer. In order to display a color or a black and white quality image with shades of gray, additional bit planes are needed.

Display Controller:

The hardware device that read the contents of the frame buffer into video buffer, which then converts the digital representation of a string of pixel values into analogue voltage signals that are sent serially to the video display screen (CRT).



The Flat Panel Display:

Compared to CRT displays, flat panel displays are much thinner, weightless, and consuming less power. Thus they are better for portable computers. Flat panel displays are made up of two plates of glass with a substance between them, which is activating in different ways. Flat panel displays are distinguished in two ways:

1- By the substance between the plates of glass.

2- By the arrangement of the transistors in the screens.

Two common types of technology used in flat panel display screens are:

a\ Liquid Crystal display (LCD):

It consists of a substance called liquid crystal, the molecules of which line up in a way that alter their optical properties. As a

result, light usually backlighting behind the screen is blocked or allowed through to create an image.

b\ Gas Plasma Display:

It is like a neon bulb, in which the display uses a gas that emits light in the presence of an electric current. That is, the technology uses neon gas and electrodes above and below the gas. When electric current passes between the electrodes, the gas glows. Although gas plasma technology has better resolution than LCD technology, it is more expensive and thus is not used as often as a LCD. On the other hand, LCDs are not practical for screens larger than 20 inches and so are not practical for TV size screen.

Video Display Card:

To display graphics, a display screen must have a video display adapter. A video display adapter also called a graphics adapter card, which is a circuit board that determines the resolution, number of colors, and how fast images appear on the display screen. Video display adapter come with their own memory chips, which determine how the card process the image and how many colors it can display. A video display adapter with 256 kb of memory will provide 16 colors; one with megabyte will support 16.7 million colors.

The video display adapter is often built into the motherboard although it may also be an expensive card that plugs into an expensive slot.

Monitor Type	Remarks
EGA (Enhanced Graphics Array)	Support 16 colors in 640-by-350 pixel resolution introduced by IBM in 1984.superseded CGA
VGA (Video Graphics Array)	Displays 16 colors in 640-by-480 pixel resolution and 256 colors at 320 by 200 pixels. This color bitmapped graphics display standard was introduced in 1987 for IBM PS/2 computers
SVGA (Super Graphics Array)	Supports 256 colors with 800 by 600 pixel resolution and a 1024 by 768 resolutions. This is a higher version of VGA.
XGA (Extended Graphics Array)	Displays up to 16,777,216 colors at resolutions up to 1024 by 768 pixels.

Screen Clarity:

The screen clarity depends on three qualities:

1- Resolution:

The clarity or sharpness of display screen is called resolution. The more pixels per square inch, and the better of resolution. Resolution is expressed in terms of the formula (horizontal pixels×vertical pixels). Each pixel can be assigned a color or particular shade of gray. A screen with 640×480 pixels multiplied together equals 307200 pixels. This screen will be less clear and sharp

than a screen with 800×600 (equals 480000) 0r 1024×768 (equals 786432) pixels.

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- 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.

Graphics versus text:

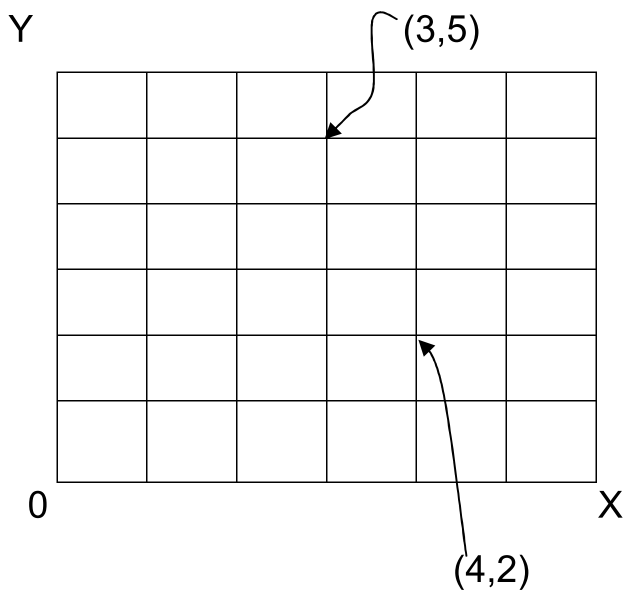
Compare with the graphic mode, the PC's text mode is easy to use. Displaying information on the screen is a simple as placing ASCII char in specific memory location. The text screen is divided into 80 column and 25 rows. The graphic mode requires a completely different orientation instead of character; you have pixels, the smallest picture element on your computer display. Today most screens can display text and graphics.

Elementary Figures

Plotting Points

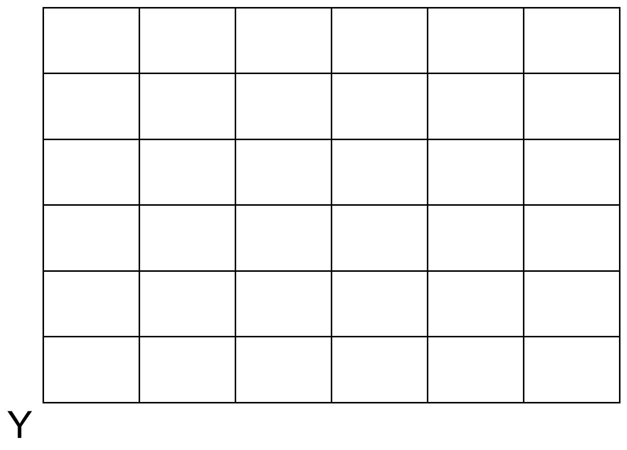
In order to draw a picture on a raster display, we must determine the corresponding points in the frame buffer that make up the picture. To perform this task we must write scan conversion point plotting algorithms.

Both the frame buffer and the display screen are given a two-dimensional coordinates system with the origin at the lower left corner.



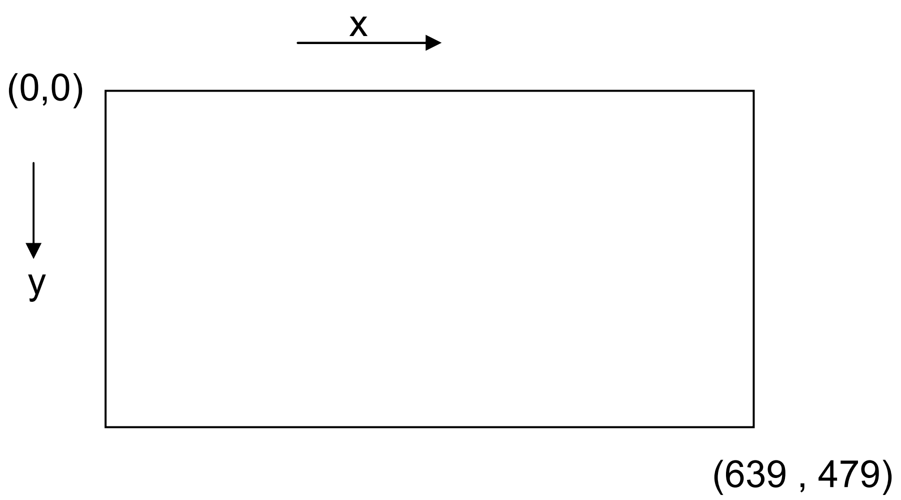
Each pixel is accessed by a non negative integer (x , y) coordinate pair. The (x) values start at the origin 0, and increase from left to right; the (y) values start at 0, and increase from bottom to top as shown in previous figure. In fact hardware engineers prefer to replace the origin at upper left corner since this corresponds to the scanning operation of the CRT display controller.

0 X



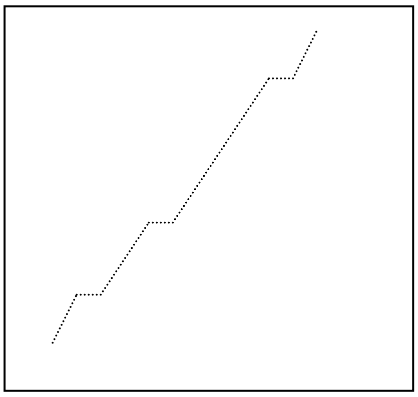
To draw a point on the display screen, a point plotting procedure is required. We assume the availability of the command: Putpixel (x , y , color).

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 below:

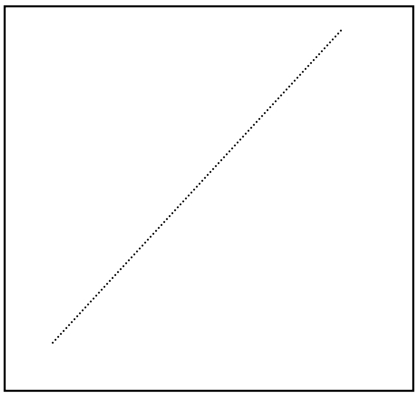


Line Drawing

Many computer pictures are composed of straight line segments. A line segment is displayed by turning on a string of adjacent 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.



(a) Low resolution



(b) High resolution

Above figure illustrates a line drawn on a raster display. Observe that the accuracy and quality of the displayed line depends on the resolution of the display device. High resolution displays (1024 $\times$ 1024) draw lines that look straight and continuous, and that start and end accurately. Lower resolution displays may draw lines with gaps. Line drawing routines in high performance graphics systems are implemented in hardware that rapidly generates the pixels comprising the line when it is given the two end points. Most low cost display systems still relay on slower software routines to accomplish line drawing. In either case, the line drawing routine should be accurate, fast, and easy to implement.

Horizontal and Vertical Lines:

The simplest lines to draw are horizontal and vertical lines.

Horizontal Lines:

The screen coordinates of the points on a horizontal line are obtained by keeping the value of (y) constant and repeatedly incrementing the (x) value by one unit as in algorithm (1).

```
Algorithm (1):
Input: $X_{start}$, $X_{end}$, $Y_{specified}$.
Output: Horizontal line.
{   for x= $X_{start}$ to $X_{end}$
        putpixel (x , $y_{specified}$ , color);
}
```

If $X_{start} > X_{end}$ then replace X_{end} by X_{start} and vice versa in for loop at algorithm(1).

<u>H.W</u>.: Write complete program in order to draw horizontal line?

Vertical Lines:

The screen coordinates of the points on a vertical line are obtained by keeping the value of (x) constant and repeatedly incrementing the (y) value by one unit as in algorithm (2).

```
Algorithm (2):
Input: $Y_{start}$,$Y_{end}$, $X_{specified}$.
Output: Vertical line.
{
  for y= $Y_{start}$ to $Y_{end}$
        putpixel ($x_{specified}$ , y , color);  }
```

If $Y_{start} > Y_{end}$ then replace Y_{end} by Y_{start} and vice versa in for loop at algorithm (2).

<u>H.W.</u>: Write complete program in order to draw vertical line?

Diagonal Lines:

To draw a diagonal line with a slope equal to (+1), we need only repeatedly increment by one unit both x and y values from the start to end pixels as shown in algorithm (3).

```
Algorithm (3):
Input: $X_{start}$, $Y_{start}$, $X_{end}$, $Y_{end}$.
          i=0
Output: Diagonal line.
{
   while ($x_{start}$ + i) $\le$ $X_{end}$ )
        {
           putpixel ($x_{start}$ + i, $y_{start}$ + i, color);
           i= i+1;
        }
}
```

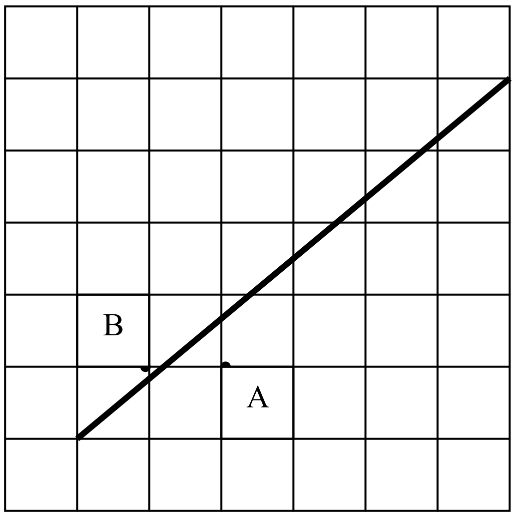
To draw a line with slop equals to (-1), replace (y_{start}+i) by (y_{start}-i) in algorithm (3).

<u>H.W.</u>: Write complete program in order to draw diagonal line using algorithm (3) with slope (+1) and slope (-1)?

Arbitrary Lines:

Drawing lines with arbitrary slope creates several problems, such as:

1- **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 below:**



Although it may not be possible to choose pixels that lie on the actual line, we want to turn on pixels lying closest to it. For example, in previous figure, the pixel at location B is a better choice than the one at location A.

2- **Determining the closest (best) pixels is not easy.**

Different algorithms calculate different pixels to make up the approximating line. The choice of algorithm depends on:

1- **The speed of line generation.**
2- **The appearance of the line.**

Therefore, to understand these criteria better let's look at several different line generating algorithms.

a- Direct method:

In this method, we learn how to 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:

$$Y = a \times X + b$$

Where (a) is the slope and (b) is a constant which represents the clipping from y-axis (y-intercept).

$$a=\frac{y_{end}-y_{start}}{x_{end}-x_{start}} \quad , \quad b=y_{start} - a \times x_{start}$$

Note: start may be 1, and end may be 2.

Direct method for drawing lines can be shown in algorithm (4).

```
Algorithm (4):
Input: $X_{start}, Y_{start}, X_{end}, Y_{end}$.
Output: Arbitrary line.
{
    $a=\frac{y_{end}-y_{start}}{x_{end}-x_{start}}$ ;
    $b= y_{start} - a \times x_{start}$ ;
    for x= $X_{start}$ to $X_{end}$
        {
            $Y=a \times x + b + 0.5$;
            putpixel (x , y , color);
        }
}
```

<u>H.W.</u>: Write complete program in order to draw arbitrary line using direct method?

b- DDA (Digital Differential Analyzer) Line Algorithm:

One technique for obtaining a straight line is to solve the differential equation for straight line.

$y = a \times x + b$ .... straight line equation

$$\therefore \frac{\partial y}{\partial x} = a = \text{constant} \qquad \text{or} \qquad \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \qquad \therefore \Delta y = \frac{y_2 - y_1}{x_2 - x_1} \Delta x$$

where: a=slope, $y_2=y_{end}$, $y_1=y_{start}$, $x_2=x_{end}$, $x_1=x_{start}$

But: $y_{i+1} = y_i + \Delta y$

Where: y_i is the initial value for a given step along the line.

y_{i+1} is the second value along the line (after y_i).

Therefore: $y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x$

Where: (x_1,y_1) and (x_2,y_2) are the end points of the required straight line.

In fact the last equation represents a recursion relation for successive values of y along the required line, which is called DDA.

A simple algorithm for DDA is clarified in algorithm (5), assuming that:

1. **Sign is a function returns (-1,0,+1) as it's argument is (<0,=0,>0).**
2. **Round is a function approximates float (real) numbers to nearest larger number.**

Algorithm (5):

```
Input: $x_1$, $y_1$, $x_2$, $y_2$.
        i=1
Output: Arbitrary line.
{
    If (abs($x_2$-$x_1$)$\geq$abs($y_2$-$y_1$))
         length= abs($x_2$-$x_1$);
    else
         length= abs($y_2$-$y_1$);
    $\Delta x = \frac{(x_2 - x_1)}{length}$;
    $\Delta y = \frac{(y_2 - y_1)}{length}$;
    x=$x_1$+0.5 $\times$ sign ($\Delta x$);
    y=$y_1$+0.5 $\times$ sign ($\Delta y$);
    while (i$\leq$length)
        {
           putpixel(round(x),round(y),color);
           x=x+ $\Delta x$;
           y=y+ $\Delta y$;
           i=i+1;
        }
}
```

H.W.: Write complete program in order to draw arbitrary line using DDA algorithm?

Example: Consider the line from (0,0) to (5,5). Use the simple DDA to draw this line?

Initial calculations:

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.

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

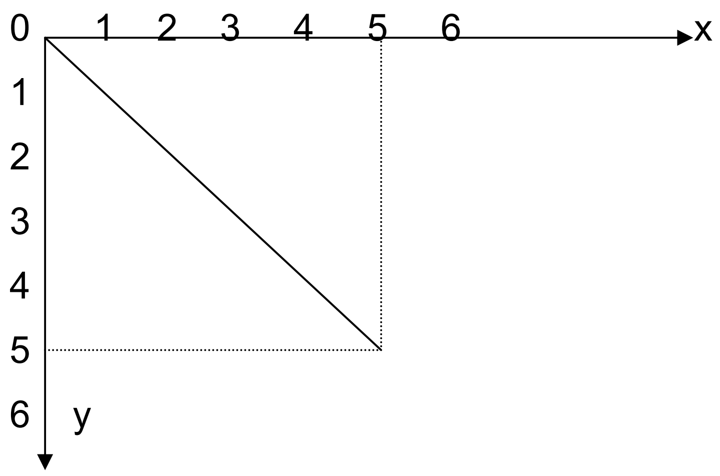
$\Delta y = \frac{(y_2 - y_1)}{length} = \frac{5}{5} = 1$

$x = x_1 + 0.5 \times$ sign (Δx)=0+0.5$\times$sign(1)=0.5

$y = y_1 + 0.5 \times$ sign (Δy)=0+0.5$\times$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		

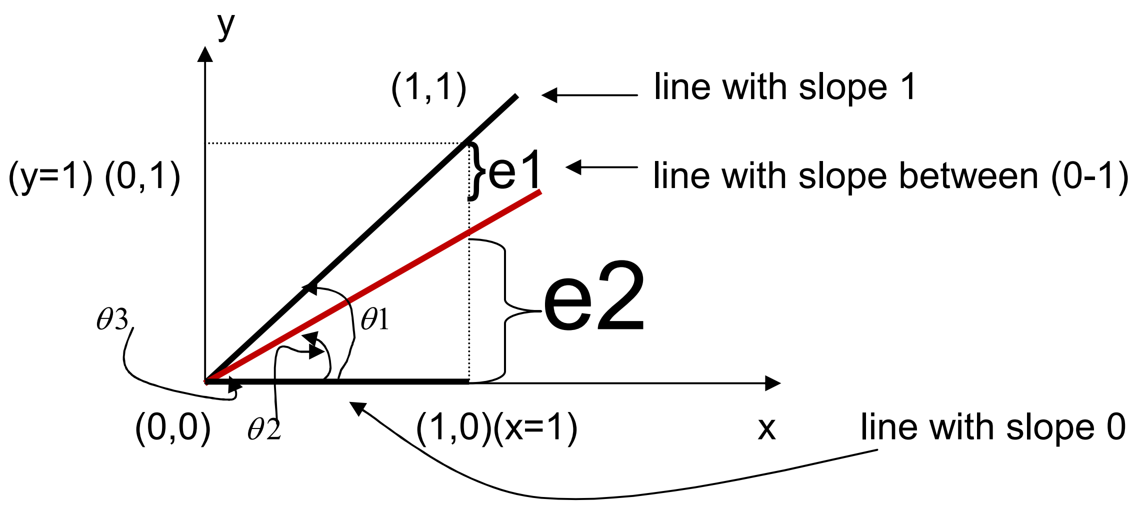


NOTE:

The DDA algorithm is faster than the direct use of the line equation since it calculates points on the line without any floating point multiplication. However, a floating point addition is still needed in determining each successive point. Furthermore, cumulative error due to limited precision in the floating point representation may cause calculated points to drift away from their true position when the line relatively long.

c- Bresenham's Line Algorithm:

This algorithm seeks to select the optimum screen locations to represent a straight line. To accomplish this algorithm take, as an example, a line in the first quadrant (i.e. a line with slope between 0 and 1).



slope= $\frac{y_2-y_1}{x_2-x_1}$ **=** $\frac{\Delta y}{\Delta x}$ **= tan(**θ**)**

e=e+ $\frac{\Delta y}{\Delta x}$**, where: e=error**

if we initialize the value (e = -$\frac{1}{2}$**)**

Then we have two cases:

1- **tan(**$\theta 2$**)** $\leq$ $\frac{\Delta y}{\Delta x}$**(slope)** $\leq$ **tan(**$\theta 1$**)**

tan(26°) $\leq$ $\frac{\Delta y}{\Delta x}$**(slope)** $\leq$ **tan(45°)**

$\frac{1}{2}$ $\leq$ $\frac{\Delta y}{\Delta x}$**(slope)** $\leq$ **1**

e=e+ $\frac{\Delta y}{\Delta x}$**, e= -**$\frac{1}{2}$ **+ (**$\frac{1}{2}\rightarrow$**1),** $\therefore$ **e** $\geq$ **0, plot (1,1)**

2- **tan(**$\theta 3$**)** $\leq$ $\frac{\Delta y}{\Delta x}$**(slope) < tan(**$\theta 2$**)**

tan(0°) $\leq$ $\frac{\Delta y}{\Delta x}$**(slope) < tan(26°)**

0 $\leq$ $\frac{\Delta y}{\Delta x}$**(slope) <** $\frac{1}{2}$

e=e+ $\frac{\Delta y}{\Delta x}$**, e= -**$\frac{1}{2}$ **+ (0**$\rightarrow\frac{1}{2}$**),** $\therefore$ **e < 0, plot (1,0)**

Conclusion:

1- **If the slope of the required line through (0,0) >** $\frac{1}{2}$**, 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) <** $\frac{1}{2}$ **, then the opposite is true.**

3- **For the slope=$\frac{1}{2}$, the algorithm chooses point (1,1) to plot.**

Bresenham's line algorithm is shown in algorithm (6).

```
Algorithm (6):
Input: $x_1$, $y_1$, $x_2$, $y_2$.
Output: Arbitrary line.
{   x= $x_1$; y= $y_1$; $\Delta x$= $x_2$-$x_1$; $\Delta y$= $y_2$-$y_1$;
    e=$\frac{\Delta y}{\Delta x}-\frac{1}{2}$;
    for i=1 to $\Delta x$
    {   putpixel(x,y,color);
        while(e$\geq$0)
        { y=y+1; e=e-1; }
        x=x+1;
        e=e+ $\frac{\Delta y}{\Delta x}$;    }
}
```

H.W.: Write complete program in order to draw arbitrary line using Bresenham's algorithm?

Example: Consider the line from (0,0) to (5,5). Use Bresenham's algorithm to draw this line?

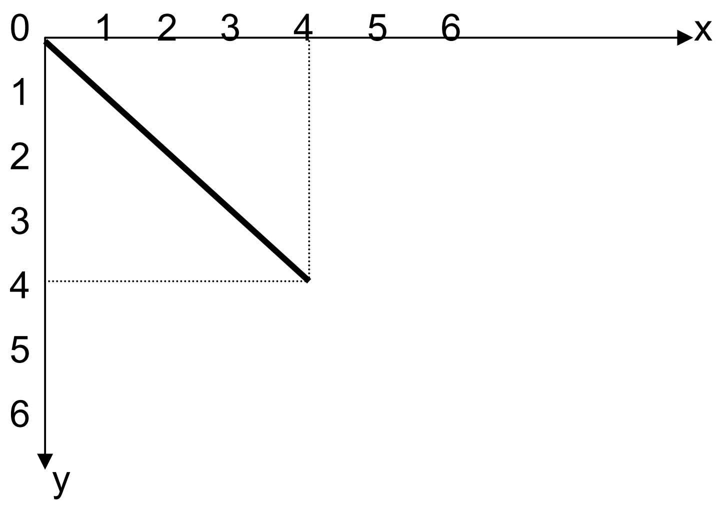
Initial calculations:

x=0, y=0. $\Delta x= x_2-x_1=5$, $\Delta y= y_2-y_1=5$, $\frac{\Delta y}{\Delta x}=1$.

$e=\frac{\Delta y}{\Delta x}-\frac{1}{2}=1-\frac{1}{2}=\frac{1}{2}$.

Incrementing through the main loop yields:

i	plot	e	x	y	notes
		0.5	0	0	
1	(0,0)				
		-0.5	0	1	e=e-1 y=y+1
		0.5	1	1	e=e+1 x=x+1
2	(1,1)				
		-0.5	1	2	e=e-1 y=y+1
		0.5	2	2	e=e+1 x=x+1
3	(2,2)				
		-0.5	2	3	e=e-1 y=y+1
		0.5	3	3	e=e+1 x=x+1
4	(3,3)				
		-0.5	3	4	e=e-1 y=y+1
		0.5	4	4	e=e+1 x=x+1
5	(4,4)				
		-0.5	4	5	e=e-1 y=y+1
		0.5	5	5	e=e+1 x=x+1



Note that the point (5,5) is not activated, and it may be activated by changing the for loop to (0$\rightarrow \Delta x$) or (1$\rightarrow \Delta x$+1).

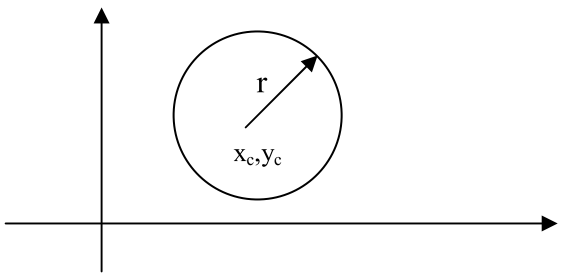
Note:

The command (line(x_1,y_1,x_2,y_2)) can draw a straight line between two points (x_1,y_1) and (x_2,y_2).

<u>H.W.</u>: Write complete program in order to draw a line with different colors using the commands (line(x_1,y_1,x_2,y_2)) and (setcolor(color))?

Circle Drawing

The circle is a special kind of curves. The circle is a closed curve with same starting and ending point. Circles are probably the most used curves in elementary graphics.



A circle is specified by the coordinates of its center (x_c,y_c) and its radius r as shown above. The most familiar equation of the circle is: $(x - x_c)^2 + (y - y_c)^2=r^2$, $y= y_c \pm \sqrt{r^2-(x-x_c)^2}$

If the center of the circle is at the origin (0,0) the above equation reduces to: $X^2+y^2=r^2$, $y=\sqrt{r^2-x^2}$ for $y\geq 0$.

Where: x,y $\rightarrow$are coordinates of the circle center point.

r $\rightarrow$ is the radius of the circle.

Circle may be drawn using the command putpixel() and the command circle(x,y,r). To draw a circle, increment the x values by one unit from (-r) to (+r) and use circle equation to solve y values at each step. This method of drawing a circle 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.

H.W.: Write complete program in order to draw a circle using the command putpixel() by substituting in circle equation?

H.W.: Write complete program in order to draw a circle using the command circle()?

H.W.: Write complete program in order to draw a group of circles, that have same center point, using the command circle()?

H.W.: Write complete program in order to draw a group of circles, that have different center points but with same radius, using the command circle()?

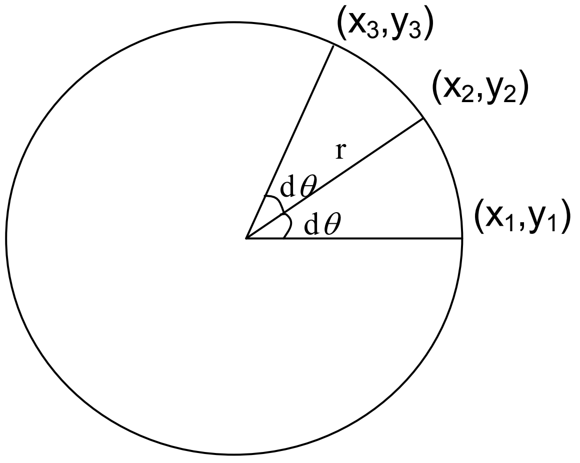
Several algorithms are proposed for drawing circles as clarified below.

Incremental Polar Circle Algorithm:

One method of eliminating the problem of plotting points evenly spaced around the circle is to use polar representation of a circle: $x = x_c + r\cos\theta$, $y = y_c + r\sin\theta$.

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

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



Equal increments of θ =dθ which is result in equal spacing (arc length) between successively plotted points as shown above.

One problem of using polar representation to draw a circle is that, on most computers, the repeated calculation of values for cosθ and sinθ consumes a significant amount of processing time. We can speed up the computation by using an incremental method that calculates the points on a circle from the coordinates of the previously calculated points. This technique requires only an initial calculation of the sin and cos. For clarity of explanation, the center of the circle is placed at the origin. Two consecutive points (x_1,y_1) and (x_2,y_2) on a circle are related by:

$x_1= r \cos\theta, \quad y_1 = r \sin\theta.$

$x_2= r \cos(\theta+d\theta), \quad y_2 = r \sin(\theta+d\theta).$

Where d$\theta \rightarrow$ is a fixed angular step size.

Using trigonometry, we get:

$x_2=r \cos\theta \cos d\theta - r \sin\theta \sin d\theta$

$y_2=r \sin\theta \cos d\theta + r \cos\theta \sin d\theta$

Substituting x_1 and y_1 at last two equations we get:

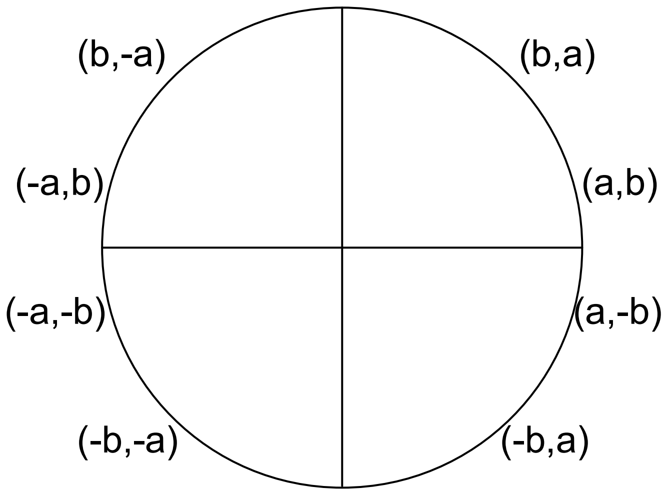
$x_2= x_1 \cos d\theta - y_1 \sin d\theta$

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

These equations are the incremental equations.

To generate a circle we start at $x_1=0$, $y_1=r$, and fixed angle increment dθ, we can compute all points on the circle by calculating cos dθ and sin dθ only once. Before using these equations to draw a circle, let's discuss symmetry, if a point (a,b)

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



To verify this, substitute all eight points into the circle equation. 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.

Incremental polar circle algorithm is shown in algorithm (7).

<u>Algorithm (7):</u>

Input: x_c, y_c, r.

Output: circle.

```
{  $d\theta=\frac{1}{r}$; ct=cos($d\theta$); st=sin($d\theta$);
     x=0; y=r;
     while ( y $\geq$ x)
     {
     putpixel(floor($x_c$+x),floor($y_c$+y),color);
     putpixel(floor($x_c$-x),floor($y_c$+y),color);
     putpixel(floor($x_c$+x),floor($y_c$-y),color);
     putpixel(floor($x_c$-x),floor($y_c$-y),color);
```

Continue of Algorithm (7):

```
putpixel(floor($x_c$+y),floor($y_c$+x),color)
;
putpixel(floor($x_c$-y),floor($y_c$+x),color);
putpixel(floor($x_c$+y),floor($y_c$-x),color);
putpixel(floor($x_c$-y),floor($y_c$-x),color);
$x_{temp}$=x;
x= x $\times$ ct – y $\times$ st;
y= y $\times$ ct + $x_{temp}$ $\times$st;
}
```

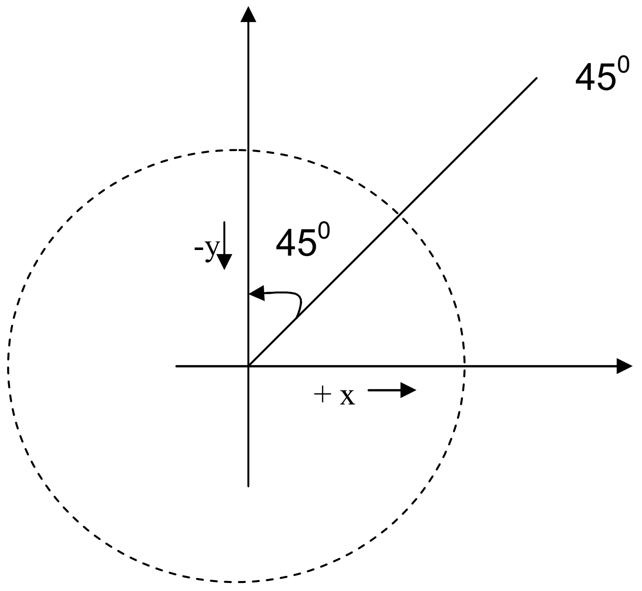
H.W.: Write complete program in order to draw a circle using Incremental polar circle algorithm?

Bresenham's Circle Algorithm:

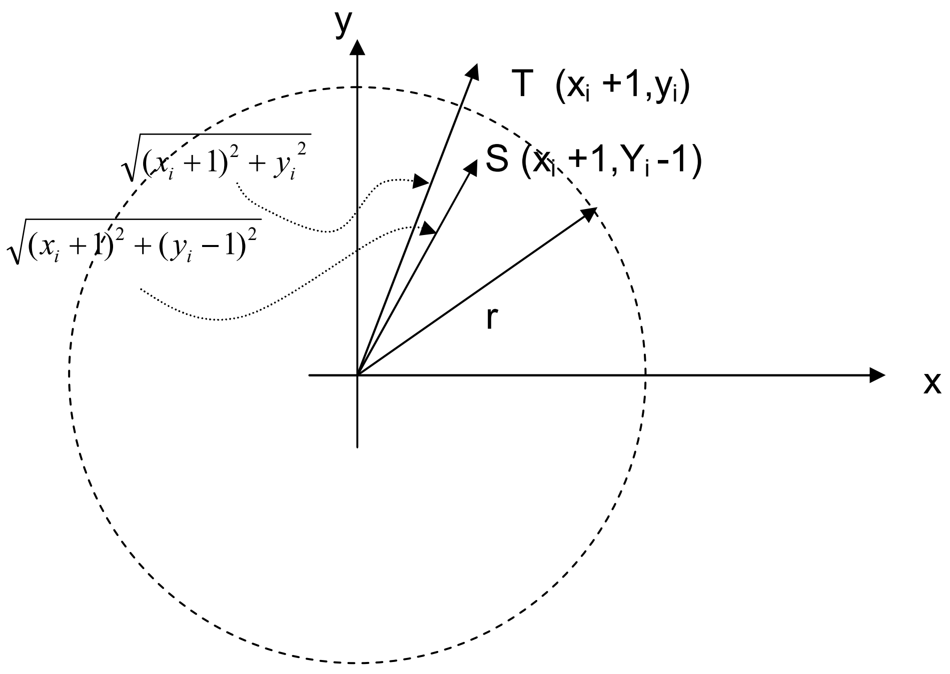
If a circle is to be plotted efficiently, the use of trigonometric and power functions must be avoided. And, as with generation of a straight line, it is desirable to perform the calculations necessary to find the scan converted points with only integer addition, subtraction, and multiplication. Bresenham's circle algorithm allows these goals to be met.

Scan converting a circle using Bresenham's algorithm works as follows. If the eight way symmetry of a circle is used to generate a circle, points will only have to be generated through a 45^0 angle. And if points are generated from 90^0 to 45^0, moves will be made only in the (+x) and (–y) directions.

90^0



The best approximation of the true will be described by those pixels in the raster that fall the least distance from the true circle. Examin the following figure:



Notice That, if points are generated from 90^0 to 45^0 each new point closest to the true circle can be found by taking either of two actions:

1- **Move in the x-direction one unit.**

Or

2- **Move in the x-direction one unit and move in the y-direction one unit.**

Therefore, a method of selecting between these two choices is all that necessary to find points closest to the true circle.

Assume that:

D(T)=The distance from the origin to pixel (T) squared minus the distance to the true circle squared.

D(S)= The distance from the origin to pixel (S) squared minus the distance to the true circle squared.

As the coordinates of (T) are $(x_i +1,y_i)$ and those of (S) are $(x_i+1,Y_i -1)$, following expressions can be developed:

$D(T)= (x_i +1)^2 + y_i^2 - r^2$

$D(S)= (x_i+1)^2 + (y_i -1)^2 - r^2$

Since D(T) will always be positive (T is outside the true circle) and D(S) will always be negative (S is inside the true circle), a decision variable (d_i) may be defined as follows:

$d_i= D(T)+D(S)$

Therefore:

$d_i= (x_i +1)^2 + y_i^2 - r^2+(x_i+1)^2 + (y_i -1)^2 - r^2$

$d_i= 2 (x_i +1)^2+ y_i^2+(y_i -1)^2 - 2 r^2$

when $(d_i < 0)$ we have $|D(T)| < |D(S)|$ and pixel T is chosen. When $(d_i \geq 0)$, we have $|D(T)| \geq |D(S)|$ and pixel S is selected.

We can also write decision variable d_{i+1} for the next step:

$d_{i+1} = 2(x_{i+1}+1)^2 + y^2_{i+1} + (y_{i+1}-1)^2 - 2r^2$

Hence:

$d_{i+1} - d_i = 2(x_{i+1}+1)^2 + y^2_{i+1} + (y_{i+1}-1)^2 - \cancel{2r^2} - 2(x_i+1)^2 - y_i^2 - (y_i-1)^2 + \cancel{2r^2}$

$d_{i+1} - d_i = 2(x_{i+1}+1)^2 + y^2_{i+1} + (y_{i+1}-1)^2 - 2(x_i+1)^2 - y_i^2 - (y_i-1)^2$

Since: $x_{i+1} = x_i + 1$ **then**

$d_{i+1} - d_i = \cancel{2x^2_{i+1}} + 4x_{i+1} + 2 + y^2_{i+1} + (y_{i+1}-1)^2 - \cancel{2(x_i+1)^2} - y_i^2 - (y_i-1)^2$

$d_{i+1} - d_i = \boxed{4x_i + 4} + 2 + y^2_{i+1} + (y_{i+1}-1)^2 - y_i^2 - (y_i-1)^2$

$d_{i+1} - d_i = 4x_i + 6 + \underline{y^2_{i+1} + y^2_{i+1}} - 2y_{i+1} \cancel{+1} \underline{- y_i^2 - y^2_i} + 2y_i \cancel{-1}$

$d_{i+1} = d_i + 4x_i + 2(y^2_{i+1} - y_i^2) - 2(y_{i+1} - y_i) + 6$

If T is the chosen pixel $(d_i < 0)$ **then:** $(y_{i+1} = y_i)$ **therefore:**

$d_{i+1} = d_i + 4x_i + 2(y^2_{i+1} - y_i^2) - 2(y_{i+1} - y_i) + 6 = d_i + 4x_i + 2(0) - 2(0) + 6$

$d_{i+1} = d_i + 4x_i + 6$

On the other hand, if S is the chosen pixel $(d_i \geq 0)$ **then :**$(y_{i+1} = y_i - 1)$

Therefore:

$d_{i+1} = d_i + 4x_i + 2(y^2_{i+1} - y_i^2) - 2(y_{i+1} - y_i) + 6$

$d_{i+1} = d_i + 4x_i + 2((y_i-1)^2 - y_i^2) - 2(\cancel{y_i} - 1 \cancel{- y_i}) + 6$

$d_{i+1} = d_i + 4x_i + 2(\cancel{(y_i^2} - 2y_i + 1) \cancel{- y_i^2}) + 2 + 6$

$d_{i+1} = d_i + 4x_i - 4y_i + 2 + 2 + 6$

$d_{i+1} = d_i + 4(x_i - y_i) + 10$

Hence we have:

$$d_{i+1} = \begin{cases} d_i + 4x_i + 6 & \text{if } (d_i < 0) \\ d_i + 4(x_i - y_i) + 10 & \text{if } (d_i \geq 0) \end{cases}$$

Finally, we set (0,r) to be the starting pixel coordinates and compute the base case value d_1**, we have:**

$d_1 = 2(x_i+1)^2 + y_i^2 + (y_i-1)^2 - 2r^2$

$d_1= 2\,(0+1)^2+ r^2+(r-1)^2 - 2\,r^2$

$d_1= 2+\cancel{r^2}+\cancel{r^2}-2r+1-\cancel{2\,r^2}$

$d_1= 3 - 2\,r$

We can now summarize the algorithm for generating all the pixel coordinates in the 90^0 to 45^0 that are needed when scan converting a circle of radius r as shown in algorithm (8):

<u>Algorithm (8):</u>

Input: r.

Output: circle.

```
{   x=0; y=r;d=3-2r;
    while ( x ≤ y)
    {  putpixel(x,y,color);
       if (d<0)
           d=d+4x+6;
       else
         {  d=d+4(x-y)+10; y--;}
       x++
    }
}
```

<u>H.W</u>.: Write complete program in order to draw a circle using Bresenham's circle algorithm?

<u>H.W</u>.: Trace and draw a circle using Bresenham's circle algorithm for circle of r=10?

Geometric Transformations

Two-Dimensional Transformations

A graphic system should allow the programer to define pictures that include a variety of transformations. For example, he should be able to magnify a picture so that detail appears more clearly, or reduce it so that more of the picture is visible.

Transformation: is a single mathematical entity and as such can be denoted by a single name or symbol (translation, rotation, or scaling). Each of these transformations is used to generate a new point ($\bar{x}$, $\bar{y}$) from the coordinates of a point (x,y) in the original picture description. If the original definition includes a line, it suffices to apply the transformation to the endpoints of the line and display the line between the two transformed endpoints.

1- Translation:

The form of translation transformation is:

$$\bar{x} = x + T_x \,, \quad \bar{y} = y + T_y$$

Translation transformation can be represented in a uniform way by a 3×3 matrix as shown below:

$$[\bar{x} \quad \bar{y} \quad 1] = [x \quad y \quad 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 1 \end{bmatrix}$$

<u>Example:</u>

Consider triangle defined by its three vertices (20,0), (60,0), (40,100) being translated 100 units to the right and 10 units up. What are the new vertices?

T_x=100, T_y=10

The new vertices are:

(20,0)$\rightarrow$(20+100,0+10) $\rightarrow$(120,10)

(60,0) $\rightarrow$(60+100,0+10)$\rightarrow$(160,10)

(40,100) $\rightarrow$(40+100,100+10) $\rightarrow$(140,110)

or:

$$[\bar{x} \quad \bar{y} \quad 1] = [20 \quad 0 \quad 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 100 & 10 & 1 \end{bmatrix} = [120 \quad 10 \quad 1]$$

$$[\bar{x} \quad \bar{y} \quad 1] = [60 \quad 0 \quad 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 100 & 10 & 1 \end{bmatrix} = [160 \quad 10 \quad 1]$$

$$[\bar{x} \quad \bar{y} \quad 1] = [40 \quad 100 \quad 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 100 & 10 & 1 \end{bmatrix} = [140 \quad 110 \quad 1]$$

H.W.: Draw the vertices of the above example before and after the translation?

2- Rotation:

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

$$\bar{x} = x\cos\theta + y\sin\theta \;, \quad \bar{y} = -x\sin\theta + y\cos\theta$$

Rotation transformation can be represented in a uniform way by a 3$\times$3 matrix as shown below:

$$[\bar{x} \quad \bar{y} \quad 1] = [x \quad y \quad 1]\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example:

Consider triangle defined by its three vertices (20,0), (60,0), (40,100) being rotated 45^0 clockwise about the origin. What are the new vertices?

The new vertices are:

(20,0) $\rightarrow$ $\bar{x}$ **=20cos45+0 sin45=14.14**

$\rightarrow$ $\bar{y}$ **=-20sin45+ 0 cos45=-14.14**

$\rightarrow$ **(14.14, -14.14)**

(60,0) $\rightarrow \bar{x}$ **=60cos45+0 sin45=42.43**

$\rightarrow$ $\bar{y}$ **=-60sin45+ 0 cos45=-42.43**

$\rightarrow$ **(42.43, -42.43)**

(40,100) $\rightarrow \bar{x}$ **=40cos45+100 sin45=98.99**

$\rightarrow$ $\bar{y}$ **=-40sin45+ 100 cos45=42.43**

$\rightarrow$ **(98.99, 42.43)**

or:

$$[\bar{x} \quad \bar{y} \quad 1] = [20 \quad 0 \quad 1]\begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [14.14 \quad -14.14 \quad 1]$$

$$[\bar{x} \quad \bar{y} \quad 1] = [60 \quad 0 \quad 1]\begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [42.43 \quad -42.43 \quad 1]$$

$$[\bar{x} \quad \bar{y} \quad 1] = [40 \quad 100 \quad 1]\begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [98.99 \quad 42.43 \quad 1]$$

H.W.: Draw the vertices of the above example before and after the rotation?

3- Scaling:

The form of the scaling transformation is:

$\bar{x}$ **=xS$_x$,** $\bar{y}$ **=yS$_y$**

The scaling transformation can be used for a variety of purposes. If the picture is to be enlarged to twice it's original size we might choose S$_x$=S$_y$=2. Notice that the elargement is relative to the origin of the coordinate system.

scaling transformation can be represented in a uniform way by a 3×3 matrix as shown below:

$$[\bar{x} \quad \bar{y} \quad 1] = [x \quad y \quad 1]\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example:

Consider triangle defined by it's three vertices (20,0), (60,0), (40,100) being twice enlarged.. What are the new vertices?

S_x=S_y=2

The new vertices are:

(20,0)→(20×2,0×2) →(40,0)

(60,0) →(60×2,0×2)→(120,0)

(40,100) →(40×2,100×2) →(80,200)

or:

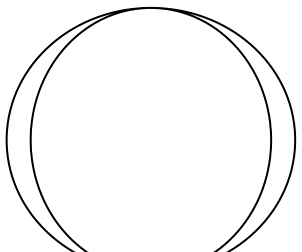
$$[\bar{x} \quad \bar{y} \quad 1] = [20 \quad 0 \quad 1]\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [40 \quad 0 \quad 1]$$

$$[\bar{x} \quad \bar{y} \quad 1] = [60 \quad 0 \quad 1]\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [120 \quad 0 \quad 1]$$

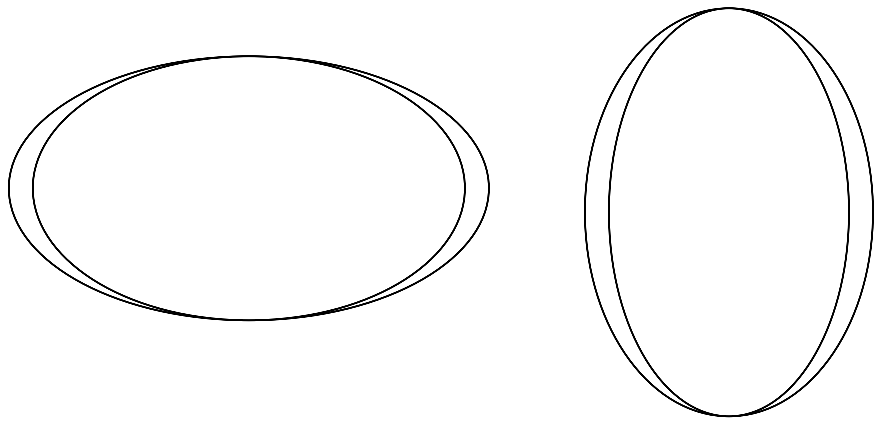
$$[\bar{x} \quad \bar{y} \quad 1] = [40 \quad 100 \quad 1]\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [80 \quad 200 \quad 1]$$

H.W.: Draw the vertices of the above example before and after the scaling?

Note: If S_x and S_y are not equal, they have the effect of distorting pictures by elongating or shrinking them along the directions parallel to the coordinate axes. For instance the foolowing figure:

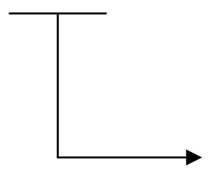


Can be distorted as shown in figures below:



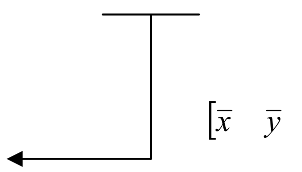
4- Reflection (Mirroring):

Reflection transformation can be considered as a special case of scaling, because the mirror image of an object can be generated using negative values of S_x or S_y. Mirror images of figure below:

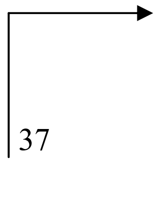


Can be generated as shown below:

$\bar{x}$ =x(-S_x) , $\bar{y}$ =yS_y



$$[\bar{x} \quad \bar{y} \quad 1] = [x \quad y \quad 1]\begin{bmatrix} -Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



$\bar{x}$=xS$_x$, $\bar{y}$=y(-S$_y$)

$$[\bar{x} \quad \bar{y} \quad 1] = [x \quad y \quad 1]\begin{bmatrix} Sx & 0 & 0 \\ 0 & -Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\bar{x}$=x(-S$_x$) , $\bar{y}$=y(-S$_y$)

$$[\bar{x} \quad \bar{y} \quad 1] = [x \quad y \quad 1]\begin{bmatrix} -Sx & 0 & 0 \\ 0 & -Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

<u>H.W.</u>: Write complete program in order to perform all types of transformations on a line between (0,1) and (5,6)?

<u>Example:</u>

The triangle with vertices ((20,0),(60,0),(40,100)) is rotated through 90^0, and then translated with T$_x$=-80,T$_y$=0. What is the new vertices of the resulted figure?

Rotation:

$\bar{x}$=x cosθ + y sinθ = x cos90 + y sin90= y

$\bar{y}$= - x sinθ + y cosθ = - sin90 + y sin90= -x

Translation:

$\bar{\bar{x}}$ = $\bar{x}$ + T$_x$ = y – 80

$\bar{\bar{y}}$ = $\bar{y}$ + T$_y$ = -x

then:

(20,0): $\bar{\bar{x}}$ = y – 80 =0-80=-80, $\bar{\bar{y}}$ = -x = -20 $\rightarrow$ (-80,-20)

(60,0): $\bar{\bar{x}}$ = y – 80 =0-80=-80, $\bar{\bar{y}}$ = -x = -60 $\rightarrow$ (-80,-60)

(40,100): $\overset{=}{x}$ **= y – 80 =100-80=20,** $\overset{=}{y}$ **= -x = -40** $\rightarrow$ **(20,-40)**

<u>H.W</u>.: Draw the vertices of the above example before and after the scaling?

<u>Example:</u>

Perform a (45^0) rotation of triangle A(0,0), B(1,1), C(5,2) about the origin using matrix representation?

Representation of triangle: $\begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$

Rotation matrix= R_{45} = $\begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ **=** $\begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Then:

$$\begin{bmatrix} \overline{A} & \overline{B} & \overline{C} \end{bmatrix} = R_{45} \times \begin{bmatrix} A & B & C \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{3\sqrt{2}}{2} \\ 0 & \sqrt{2} & \frac{7\sqrt{2}}{2} \\ 1 & 1 & 1 \end{bmatrix}$$

Thus: $\overline{A} = (0,0),\ \overline{B} = (0,\sqrt{2}), \overline{C} = (\frac{3\sqrt{2}}{2}, \frac{7\sqrt{2}}{2})$

<u>H.W</u>.: Draw the vertices of the above example before and after the scaling?

Display File Segmentation

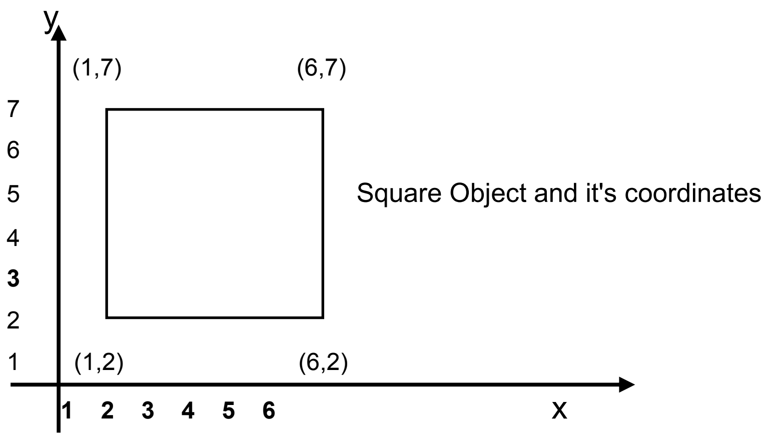
(Writing and Reading Graphics Data Files)

Dynamically changing displayed pictures are attractive because they are so unlike the static pictures we draw on paper. But, it is not possible to change the dislayed picture fast enough to produce a smooth transition from one picture to the next. The dynamic graphics demands speed of regenration of successive pictures. Before achieving this speed, we should consider what else is needed. Applications of computer graphics sometimes require that the complete picture be redrawn at each change, more frequently only a small part of the picture changes and the rest remains unchanged. The second requirement for dynamic computer graphics is the ability to make selective modifications to the picture, i.e., to add new parts, move them around, and delete them without disturbing the rest.

Three basic functions are required for dynamic graphics: (addition, replacement, and deletion of information).

Display file:

An image is described by a sequence of primitive graphics commands. The collection of graphics commands that represent the image is called a display file, as shown in figure below:



In order to draw the square in previous figure using line procedures:

Command		
Line	**(1,2)**	**(1,7)**
Line	**(1,7)**	**(6,7)**
Line	**(6,7)**	**(6,2)**
Line	**(6,2)**	**(1,2)**

Type(A)

Command	
Moveto	**(1,2)**
Linerel	**(0,5)**
Linerel	**(5,0)**
Linerel	**(0,-5)**
Linerel	**(-5,0)**

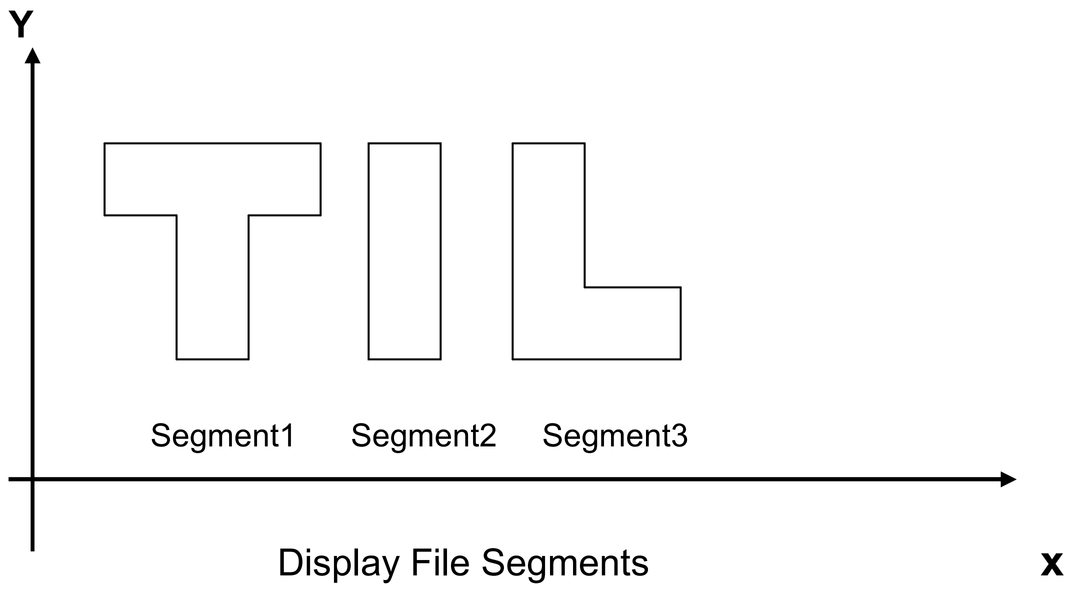
Type(B)

Command	
Moveto	**(1,2)**
Linerel	**(1,7)**
Linerel	**(6,7)**
Linerel	**(6,2)**
Linerel	**(1,2)**

Type(C)

Segmented Display File:

Since the display file consists of the graphics commands necessary to construct an image, dividing the display file into distinct logical units, or segments, partitions the image into distinct parts. Each display file segment consists of a sequence of graphics commands treated as one unit. The portion of the image represented by a segment can be translated, rotated, scaled or transformed in any other manner without effecting any other segment as shown below:



Display File Segments

The segment is a unit of the display file; thus the use of segments by the programmer requires also to recogonize the existance of the display file as a stored representation of the displayed image.

Functions for the Segmenting the Display File:

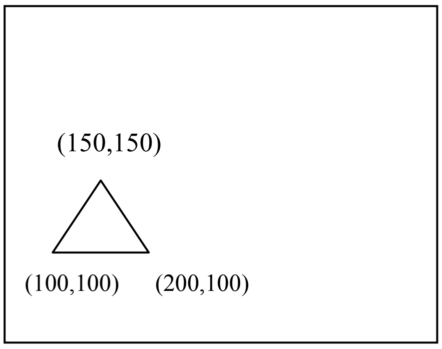
In sequentail disk files, we open a file before we add data to it, and we close the file when have added the last item and the file is complete. To change the contents of a file, we open it

again, add the new data to replace the old, and close the new file. To get a rid of a file, we delete it.

The very same operations are ideal for manipulating display file segments. To create a new segment, we open it and then call graphic primitives to add to the segment the lines and and the text to be displayed; then we close the segment. The same sequence of operations applied to an existing segment will cause that segment to be replaced by a new one. To remove a segment from the display file, we delete it. Thus we need only three basic functions:

1- **Open Segment (n): open a display file segment named n.**
2- **Close Segment (n): close the opened segment.**
3- **Delete Segment (n): remove from the display file the segment named n.**

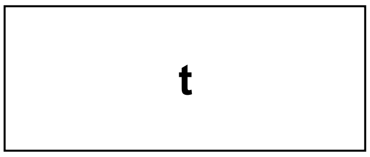
To illustrate the use of these functions, suppose we wish to display a triangle as shown in figure below:



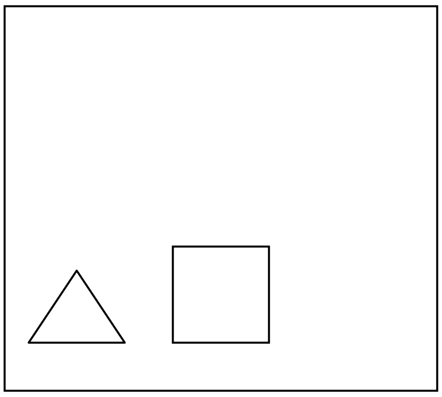
We may use the following statements:

```
initGraphics;   0   0   400    400     0
setviewport(left,top,right,bottom,clip);
opensegment(t);
moveto(100,100);
linerel(150,150);
linerel(200,100);
linerel(100,100);
closesegment;
```

Now the display file contains the single segment (t), if it is originally empty, as shown in following figure:



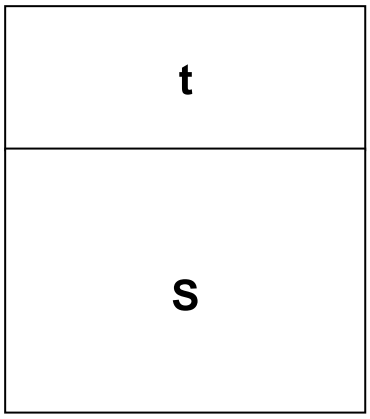
We can add a square as shown in figure below:



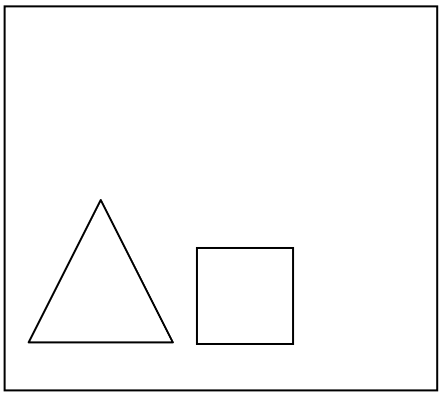
The following statements are used for adding square:

```
opensegment(s);
moveto(300,100);
linerel(300,200);
linerel(400,200);
linerel(400,100);
linerel(300,100);
closesegment;
```

Now the display file contains the two segments (s) and (t) as shown in figure below:

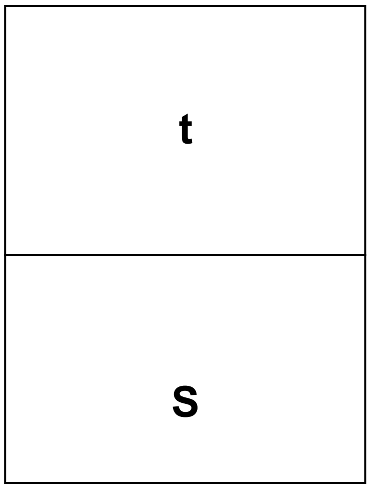


We can redefine the triangle as follows, to achieve the following figure:

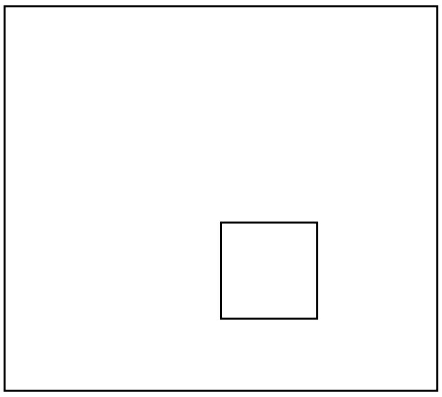


```
opensegment(t);
moveto(100,100);
linerel(150,300);
linerel(200,100);
linerel(100,100);
closesegment;
```

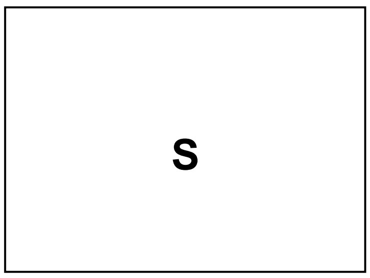
The original segment (t) has been replaced by a new one, as shown in following figure:



Finally we can erase the triangle using (deletesegment(t)) as shown in figure below:



The display file now contains only the single segment (s), as shown in figure below:

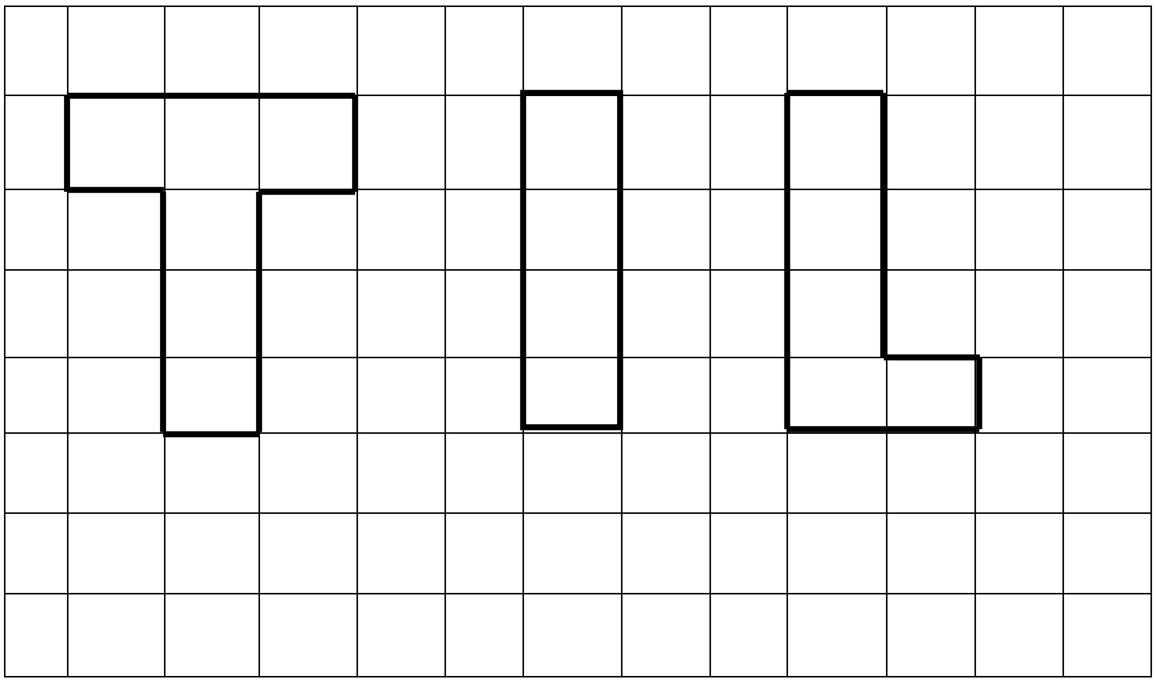


Segment Table:

The display file is usually composed of several segments. In order to reference the correct segment, a unique name is assigend to each. Once we have the name of the segment, we must find the segment's location in the display file. We also need to know the number of display files entries for this segment, otherwise we would not know where a particular segment ends. The location and the length of a segment are two additional attributes of a segment. It is convenient to store the names of the segments and their corresponding attributes in a data structure called a segment table. We can reference and manipulate a segment by locating its entry in the segment table.

Example:

Construct display file and segment table for the image below:



Display File

	Command	Op1	Op2
1	Moveto	2	3
2	Linerel	3	3
3	Linerel	3	6
4	Linerel	4	6
5	Linerel	4	7
6	Linerel	1	7
7	Linerel	1	6
8	Linerel	2	6
9	Linerel	2	3
10	Moveto	6	3
11	Linerel	7	3
12	Linerel	7	7
13	Linerel	6	7
14	Linerel	6	3
15	Moveto	9	3
16	Linerel	11	3
17	Linerel	11	4
18	Linerel	10	4
19	Linerel	10	7
20	Linerel	9	7
21	Linerel	9	3

Segment Table

Sn Segment Name	Ss Segment Start	SL Segment Length	Visible	Trans-X	Trans-Y	Sx	Sy	R
1	1	9	True	0	0	1	1	0
2	10	5	True	0	0	1	1	0
3	15	7	True	0	0	1	1	0

Some Drawings related to Circle

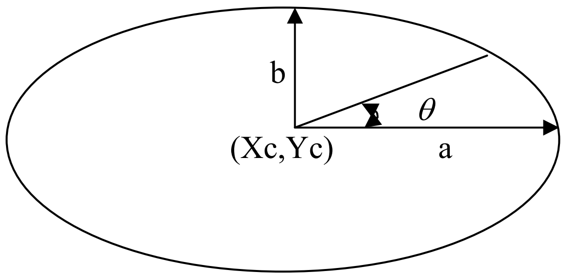
1- Ellipse:

An ellipse is a variation of a circle. Stretching a circle in one direction produces an ellipse. The polar equations for this type of ellipse centered at (Xc,Yc) are:

$$X=Xc + a\ \cos\theta \quad \ldots\ldots\ldots..(1)$$

$$Y=Yc + b\sin\theta \quad \ldots\ldots\ldots\ldots(2)$$

Where the angle θ assumes values between 0 and 2π radians as shown in figure:



The values of a and b affect the shape of the ellipse. If (b>a), the ellipse is longer in the y direction. If (b<a) the ellipse is longer in the x direction.

The ellipse can be drawn using four points symmetry: if (c,d) lies on the ellipse, so do the points ((-c,d), (c,-d), and (-c,-d)).

From equations (1 and 2), the incremental equations for an ellipse are derived as follows:

$X1= a\ \cos\theta\ ;\ Y1= b\sin\theta$

$X2=a\cos(\theta+d\theta)=a\,[\cos\theta\ .\ \cos d\theta\ - \sin\theta\ .\ \sin d\theta]$

$\quad = a\cos\theta\ .\ \cos d\theta\ - a\sin\theta\ .\ \sin d\theta$

$\quad =X1\cos d\theta\ - a\ \frac{b}{b}\ \sin\theta\ .\ \sin d\theta=$

$\therefore\ X2=X1\cos d\theta\ - \frac{a}{b}\ Y1\sin d\theta \quad \ldots\ldots.(3)$

Y2= b sin(θ+dθ)=b [sinθ. cos dθ + cosθ. sin dθ]

= b sinθ. cos dθ + b cosθ. sin dθ

= Y1 cos dθ + b cosθ. sin dθ

= Y1 cos dθ + b $\frac{a}{a}$ cosθ. sin dθ

$\therefore$Y2 = Y1 cos dθ + $\frac{b}{a}$ X1. sin dθ ……….(4)

Equations (3 and 4) are the incremental equations for ellipse.

<u>Note:</u> To draw an ellipse using (C or C++) you can use the function: ellipse(XE,YE,Stangle,Endangle,Xrad,Yrad)

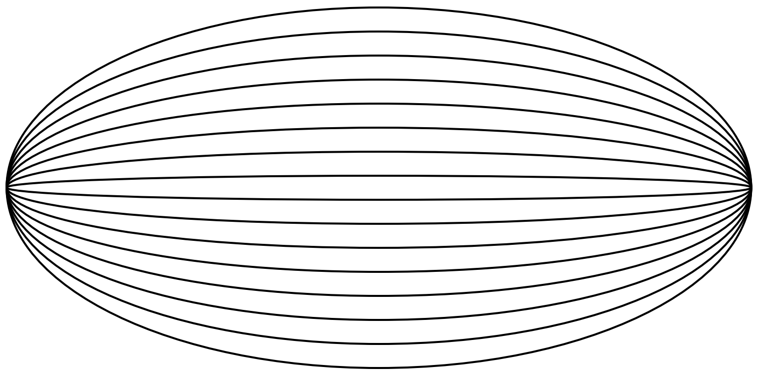
Where XE,YE: are coordinates of the ellipse center.

Stangle: is starting angle.

Endangle: is ending angle.

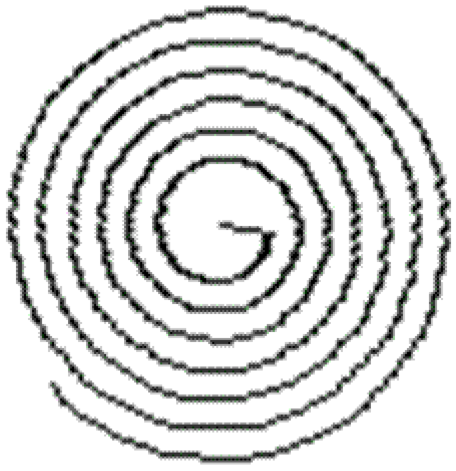
Xrad,Yrad: are the two radii of the ellipse.

<u>H.W.:</u> Write complete (C or C++) program in order to draw the following figure using the function ellipse()?



2- Spiral:

A spiral is another variation of a circle. Spirals can be plotted by gradually increasing the radius when plotting a circle. The spiral shown in following figure can be produced using parametric equations of the circle.



It does this by incrementing the radius while incrementing the polar angle. The initial and final values of the radius are arbitrary and affect the size and the number of loops in the spiral. The algorithm of the spiral using polar equations can be shown in the algorithm (10).

Algorithm (10):

```
Input: Xc,Yc, initial radius, final radius.
Output: spiral.
  {
      radius=initial radius; theta=0;
      Xinit=initial radius ×cos (theta);
      Yinit=initail radius ×sin(theta);
      moveto(ceil(Xc+Xinit),ceil(Yc+Yinit));
      while(initial radius<final radius)
      {
        theta=theta+0.1;
        initial radias=initial radius+0.1;
        X=initial radius × cos(theta);
        Y=initial radius × sin(theta);
        Putpixel(ceil(Xc+X),ceil(Yc+Y),color);
      }
  }
```

H.W.: Write complete (C or C++) program in order to draw a spiral using polar Algorithm?

3- Arcs:

The arc is part of a circle. But, the arc has starting point differs from its ending point. (C or C++) compiler has three functions to draw the arcs:

a- Arc(Xa,Ya,Stangle,Endangle,Radius)

Where Xa, Ya: are coordinates of the arc center.

Stangle: starting angle for the arc.

Endangle: ending angle for the arc.

Radius: radius of the arc.

b- Pieslice(Xp,Yp,Stangle,Endangle,Radius)

Where Xp,Yp: are coordinates of arc center.

Stangle: starting angle for the arc.

Endangle: ending angle for the arc.

Radius: radius of the arc.

The function piesliece() draws an arc, with ability to fill the space inside the arc by color and pattern. This function is commonly used with business graphics in order to draw pie chart.

c- Sector (X,Y,M,N,a,b)

Where X,Y: are coordinates of arc center.

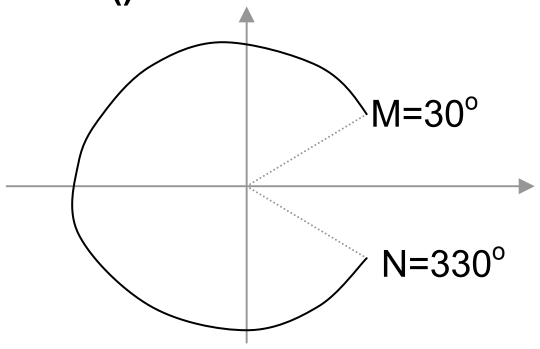
M: starting angle for the arc.

N: ending angle for the arc.

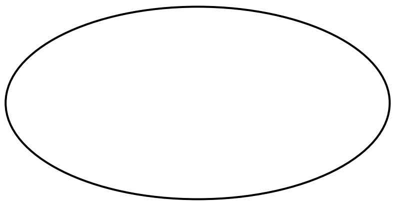
A,b: are radii of the arc.

The arc that is drawn by the function sector() contains two radii, which is similar to ellipse. Therefore, the function draws an arc which is a part of ellipse.

<u>H.W.:</u> Write complete (C or C++) program in order to draw the following figure using the function arc()?Repeat with function pieslice()?



<u>H.W.:</u> Write complete (C or C++) program in order to draw the following figure using the function sector()?



Some Drawings related to Line

Polygons:

Polygons are figures consist of certain number of straight lines, as triangles, rectangles, parallelograms, pentagons, and..etc.. In (c or c++) compiler the function drawpoly(number,addresslist) is used to draw polygons.

Where number: is the number of points in a list.

Addresslist: are the addresses of points in a list. This list contains the starting and ending points (for every line at the polygon).

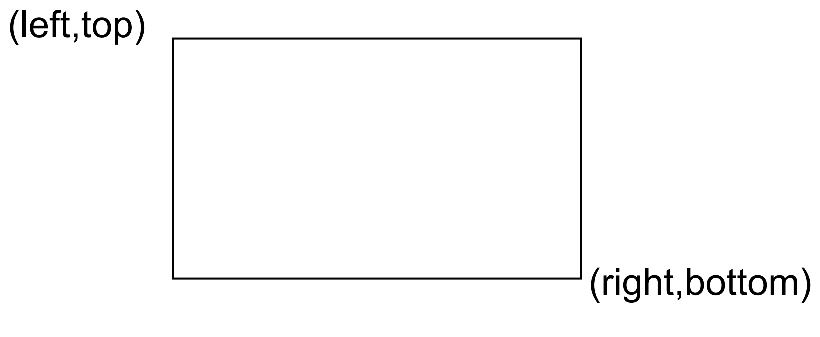
However, The command rectangle (left,top,right,bottom) is used to draw the rectangle. This command has four variables:

Left: specify the horizontal coordinate (x) for upper point.

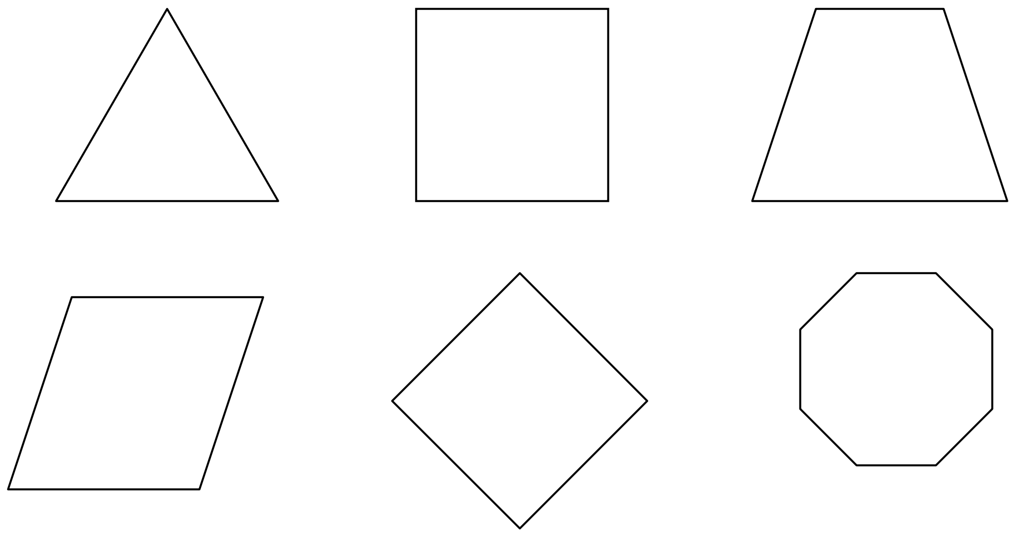
TOP: specify the vertical coordinate (y) for upper point.

Right: specify: the horizontal coordinate (x) for lower point.

Bottom: specify the vertical coordinate (y) for lower point.



<u>H.W.:</u> Write complete (C or C++) program in order to draw the following figures using the function drawpoly()?



Clipping and Windowing

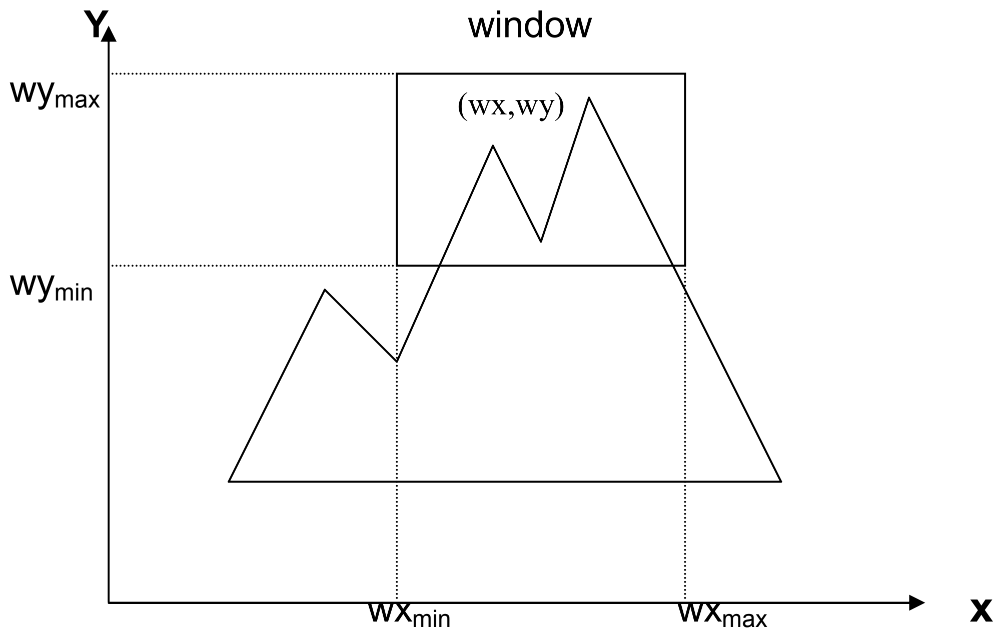
Many graphics application programs give the user the impression of looking through a window at a very large picture. To display an enlarged portion of a picture we must
not only apply the appropriate scaling and translation but identify the visible parts of the picture for inclusion in the displayed image. The correct way to select visible information for display is to use clipping, a process which divides each element of the picture into its visible and invisible portions, allowing the invisible portion to be discarded. Clipping can be applied to a variety of different types of picture elements: vectors, curves of various kinds, and even polygons. The
basis for these clipping opertions is a simple pair of inequalities that determine whether a point (x,y) is visible or not:

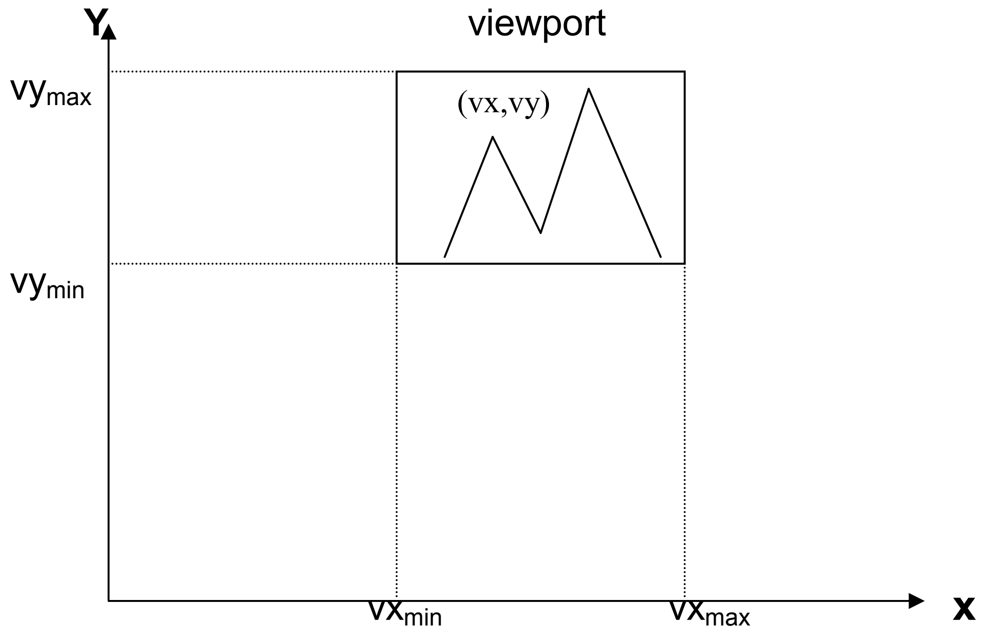
$$x_{left} \leq x \leq x_{right} \;,\; y_{bottom} \leq y \leq y_{top}$$

Where x_{left}, x_{right}, y_{bottom}, y_{top} are the positions of the edges of the screen. These inequalities provide us with a very simple method of clipping pictures on a point by point basis; we substitute the coordinates of each point for x and y and if the point fails to satisfy either inequality; it is invisible. It would be quite inappropriate to clip pictures by converting all picture elements into points and using these inequalities; the clipping process would take far too long and would leave the picture in a form no longer suitable for a line drawing display. We must attempt to clip larger elements of the picture. This involves developing more powerful clipping algorithms that can be determine the visible and invisible portions of such picture elements.

Window to Viewport Mapping:

A window is specifiied by four world coordinates: wx_{max}, wx_{min}, wy_{max}, and wy_{min} as shown in figure below:





Similarly, a viewport is described by four normalized device coordinates: vx_{max}, vx_{min}, vy_{max}, and vy_{min}. 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: $\frac{wx - wx_{\min}}{wx_{\max} - wx_{\min}} = \frac{vx - vx_{\min}}{vx_{\max} - vx_{\min}}$ **and** $\frac{wy - wy_{\min}}{wy_{\max} - wy_{\min}} = \frac{vy - vy_{\min}}{vy_{\max} - vy_{\min}}$

Thus:

$$vx = \frac{wx - wx_{\min}}{wx_{\max} - wx_{\min}}(vx_{\max} - vx_{\min}) + vx_{\min}$$

$$vy = \frac{wy - wy_{\min}}{wy_{\max} - wy_{\min}}(vy_{\max} - vy_{\min}) + vy_{\min}$$

Note that geometric distortion occur (e.g. squares in the window become rectangles in the viewport) whenever the two scaling constant differ.

In (C and C++) there is a function to draw in a part of screen which is: setviewport(left,top,right,bottom,clip)

Where left, top, right, bottom are coordinates of the viewport.

Clip is the possibility of clipping:

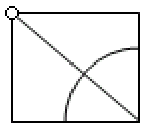
Clip=0$\rightarrow$no clipping.

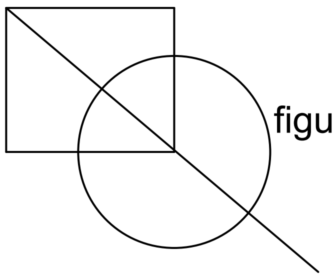
clip$\neq$0$\rightarrow$there is clipping (only the drawing inside the viewport is present).

Clipping

Is the possibility to draw part of drawing on the screen. This part is inside the area viewport, whereas the rest of the drawing which is outside viewport is clipped (is not appear on the screen). Note that clipping takes a part of time when we use it with any program, because it needs a group of tests in order to specify what part to display and what part to delete.

H.W. **Use setviewport function to draw figure(1) from figure(2)?**

Figure(1) 

 figure(2)

Line Clipping:

Lines that do not intersect the clipping window are either completely inside the window or completely outside the window. On the other hand a line that intersects the clipping window is divided by the intersection point (s) into segments that are either inside or outside the window. The following algorithm provide efficient way to dicide the relationship between an arbitrary line and the clipping window to find intersection point (s).

The Cohen-Sutherland Algorithm for line clipping:

In this algorithm we divide the line clipping into two phases:

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

All lines fall into one of the following clipping categories:

1. **Visible: both end points of the line lie within the window.**
2. **Not visible: the line lies outside the window. This will occur if the line from (x1,y1) to (x2,y2) satisfies any one of the following four inequalities:**

$$x1, x2 > x_{\max}$$
$$x1, x2 < x_{\min}$$
$$y1, y2 > y_{\max}$$
$$y1, y2 < y_{\min}$$

3- Clipping candidate: the line is in neither 1 or 2.

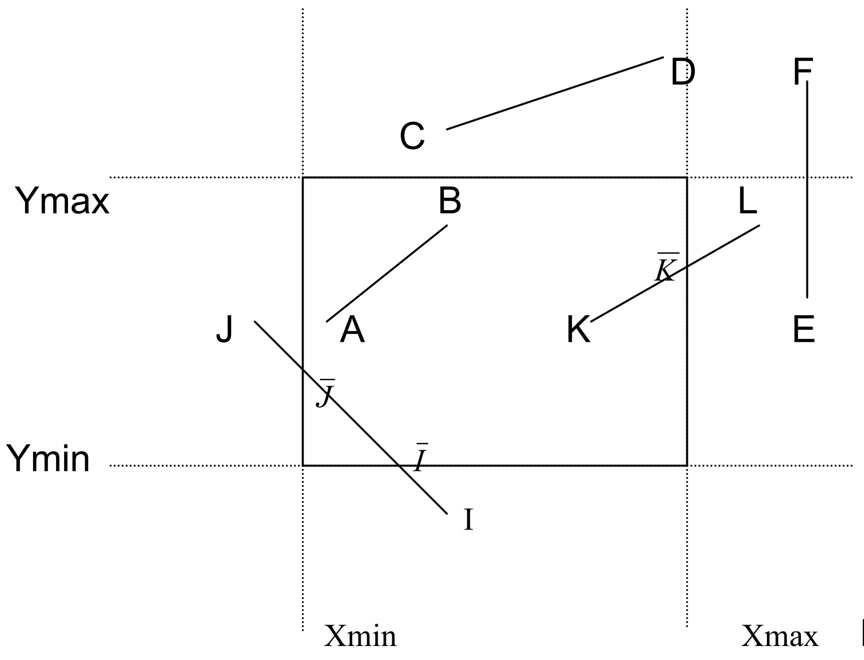
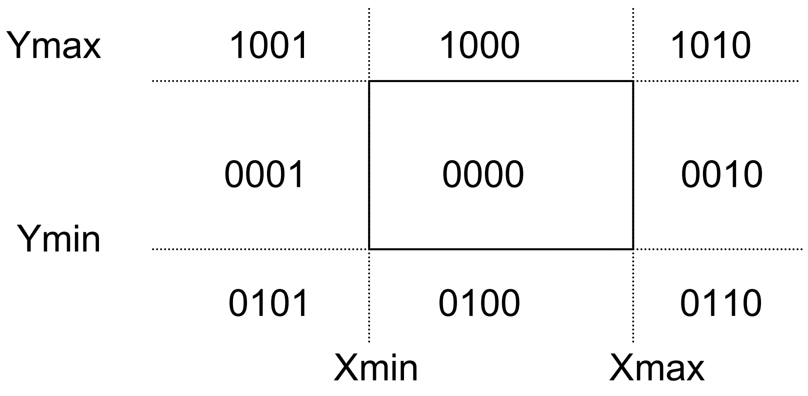


Figure (1)

In figure (1), line AB is in category 1 (visible); lines CD and EF are in category 2 (not visible); and lines IJ and KL are in category 3 (clipping candidate).

The algorithm employs an efficient procedure for finding the category of a line. It proceeds in two steps:

1. **Assign a (4 bit) region code to each endpoint of the line. The code is determined according to which of the following nine regions of the plane the end point lies in:**



2- The line is visible if both region codes are (0000), and not visible if the bitwise logical AND of the codes is not (0000), and a candidate for clipping if the bitwise logical AND of the region is (0000).

For a line in category 3 we proceed to find the intersection point of the line with one of the boundaries of the clipping window, or to be exact, with the infinite extension of one of the boundaries.

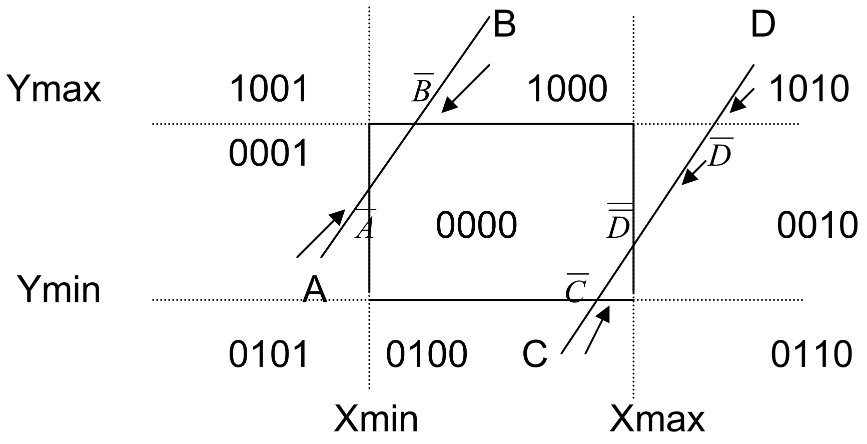


Figure (2)

We choose an endpoint of the line say(x1,y1), that is outside the window, i.e., whose region code is not (0000). We then select an extended boundary line by observing that those boundary lines that are candidates for intersection are the ones for which the

chosen endpoint must be (pushed across) so as to change a (1) to a (0). This means:

If bit 1 is 1: intersect with line Y=Ymax.

If bit 2 is 1: intersect with line Y=Ymin.

If bit 3 is 1: intersect with line X=Xmax.

If bit 4 is 1: intersect with line X=Xmin.

Consider line CD in figure (2). If endpoint C is chosen, then the bottom boundary line Y=Ymin is selected for computing intersection. On the other hand, if endpoint D is chosen, then either the top boundary line Y=Ymax or the right boundary line X=Xmax is used. The coordinates of the intersection point are:

$X_i=X_{min}$ **or** X_{max}	**if the boundary**
$Y_i=Y_1+m(X_i-X_1)$	**line is vertical**

Or:

$X_i = X_1 + \frac{Y_i - Y_1}{m}$	**if the boundary**
$Y_i=Y_{min}$ **or** Y_{max}	**line is horizontal**

Where ($m = \frac{y_2 - y_1}{x_2 - x_1}$ **) is the slope of the line. Now we replace endpoint (x1,y1) with the intersection point (xi,yi), effectively eliminating the portion of the original line that is on the outside of the selected window boundary. The new endpoint is then assigned an updated region code and the clipped line re-categorized and handled in the same way. This iterative process terminates when we finally reach a clipped line that belongs to either category 1 (visible) or category 2 (not visible).**

<u>Example</u>: Let R be the rectangular window which lower left corner is at L(-3,1) and upper right corner is at R(2,6). Use Cohen-Sutherland algorithm to clip the line segments at figure (3)?

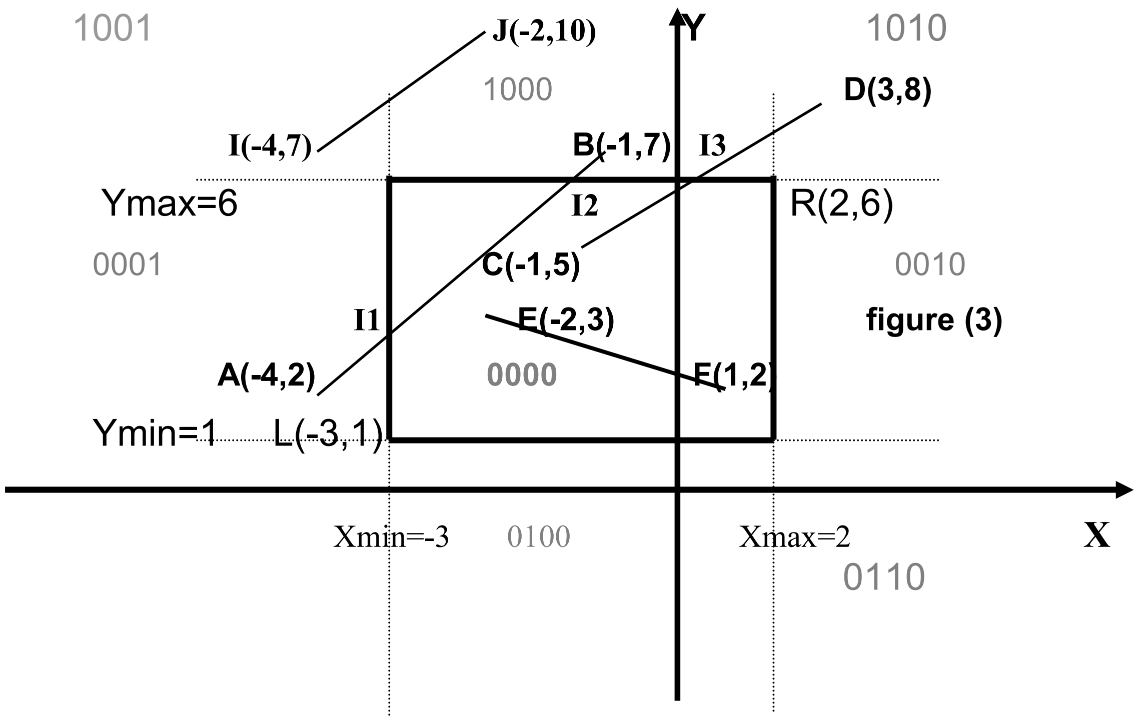


figure (3)

We place the line segments in their appropriate categories by testing the region codes:

Category 1 (visible): $\overline{EF}$ **since the region code for both endpoints is 0000.**

Category 2 (not visible): $\overline{IJ}$ **since 1001 AND 1000= 1000 (which is not 0000.**

Category 3 (candidates for clipping): $\overline{AB}$ **since 0001 AND 1000= 0000, and** $\overline{CD}$ **since 0100 AND 0010= 0000.**

Clipping $\overline{AB}$ **:**

The code for A is 0001. To push the 1 to 0, we clip against the boundary line Xmin=-3. Then:

Xi=Xmin=-3.

Yi=Y1+m(Xi-X1), m=$\frac{y_2 - y_1}{x_2 - x_1} = \frac{7-2}{-1-(-4)} = \frac{5}{3} = 1.6667$**.**

Yi=2+(1.6667)(-3-(-4))=2+1.6667=3.6667.

The resulting intersection point is I1 (-3,3.6667). We clip (do not display) $\overline{AI_1}$ **and work on** $\overline{I_1B}$ **.**

The code for I1 is 0000. The clipping category for $\overline{I_1B}$ is 3 since 0000 AND 1000= 0000.

Now B is outside the window (i.e. its code is 1000), so we push the 1 to 0 by clipping against the line Ymax=6. Then:

Yi=Ymin or Ymax= Ymax=6.

$$x_i = x_1 + \frac{y_i - y_1}{m}, \quad m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{7 - 3.6667}{-1 - (-3)} = 1.6667 .$$

$$x_i = -3 + \frac{6 - 3.6667}{1.6667} = -1.6$$

The resulting intersection point is I_2 (-1.6,6). Thus $\overline{I_2B}$ is clipped. The code for I_2 is 0000. The remaining segment $\overline{I_1I_2}$ is display since both end points lie in the window (i.e. their codes are 0000).

Clipping $\overline{CD}$:

We start with D since it is outside the window. Its code is 1010. We push the first 1 to 0 by clipping against the line Ymax=6. Then:

Yi=Ymin or Ymax= Ymax=6.

$$x_i = x_1 + \frac{y_i - y_1}{m}, \quad m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{8 - 5}{3 - (-1)} = 0.75 .$$

$$x_i = -1 + \frac{6 - 5}{0.75} = 0.333$$

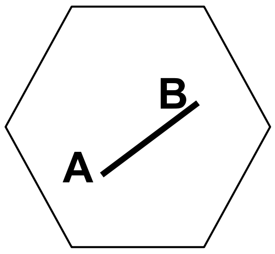
The resulting intersection point I_3 is (0.333,6). Its code is 0000. Thus $\overline{I_3D}$ is clipped and the remaining segment $\overline{CI_3}$ has both end points coded 0000 and so it is displayed.

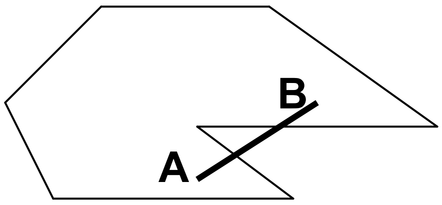
Polygon Clipping

In this clipping we consider the case of using a polygonal clipping window to clip a polygon.

Convex Polygonal Clipping Window:

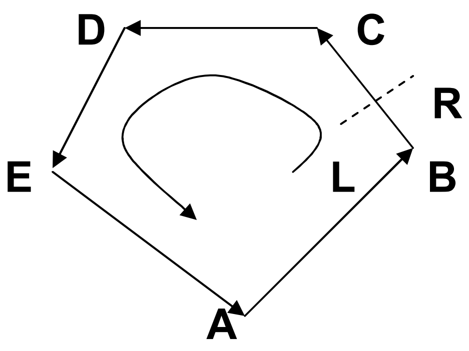
A polygon is called convex if the line joining any two interior points of the polygon lies completely inside the polygon as shown in following figure:

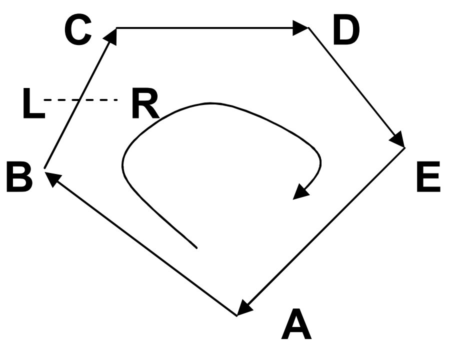




Convex Polygon **Concave Polygon**

A non convex polygon is said to be concave. By convention, a polygon with vertices $P_1, \ldots, P_n$ and edges $P_{i-1} P_i$ and $P_n P_1$ is said to be positively oriented if a tour of the vertices in the given order produces a counter clockwise circuit, as shown in the following figure:





Positive Orientation **Negative Orientation**

Equivalently, the left hand of a person standing along any direction edge $\overrightarrow{P_{i-1}P_i}$ or $\overrightarrow{P_nP_1}$ would be pointing inside the polygon.

Let A(x1,y1) and B(x2,y2) be the endpoints of a directed line segment. A point P(x,y) will be to the left of the line segment if the expression: $C=(X_2-X_1)(Y-Y_1)-(Y_2-Y_1)(X-X_1)$ is positive.

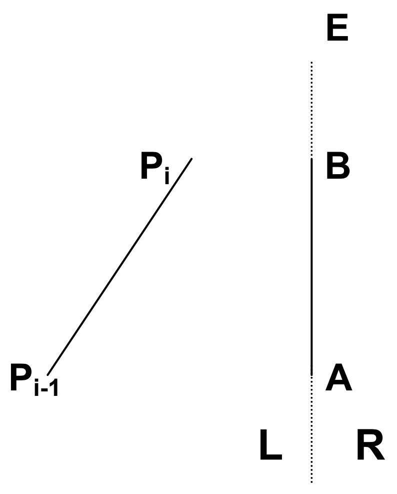
We say that the point is to the right of the line segment if this quantity is negative. If a point (P) is to the left of every edge of a positively oriented, convex polygon, it is inside the polygon. If it is to the right of every edge of the polygon, it is outside the polygon. This observation forms the basis for clipping any polygon, convex or concave, against a convex polygonal clipping window.

The Sutherland-Hodgman Algorithm for clipping Polygons:

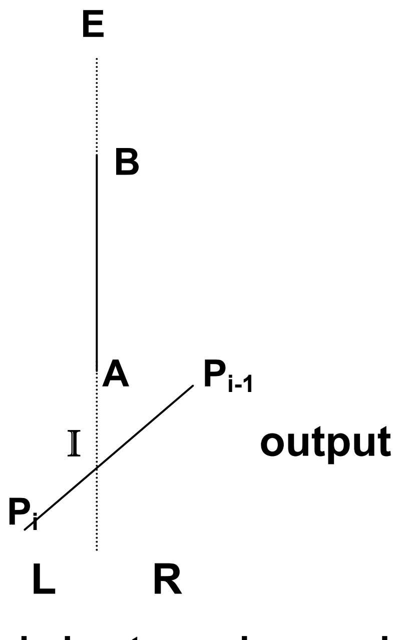
Let P_1,….., P_n be the vertex list of the polygon to be clipped. Let edge E, determined by endpoints A and B, be any edge of the positively oriented, convex clipping polygon. We clip each edge of the polygon in turn against the edge E of the clipping polygon, forming a new polygon whose vertices are determined as follows:

Consider the edge $\overrightarrow{P_{i-1}P_i}$:

1- If both P_{i-1} and P_i are to the left of the edge, vertex P_i is placed on the vertex output list of the clipped polygon.

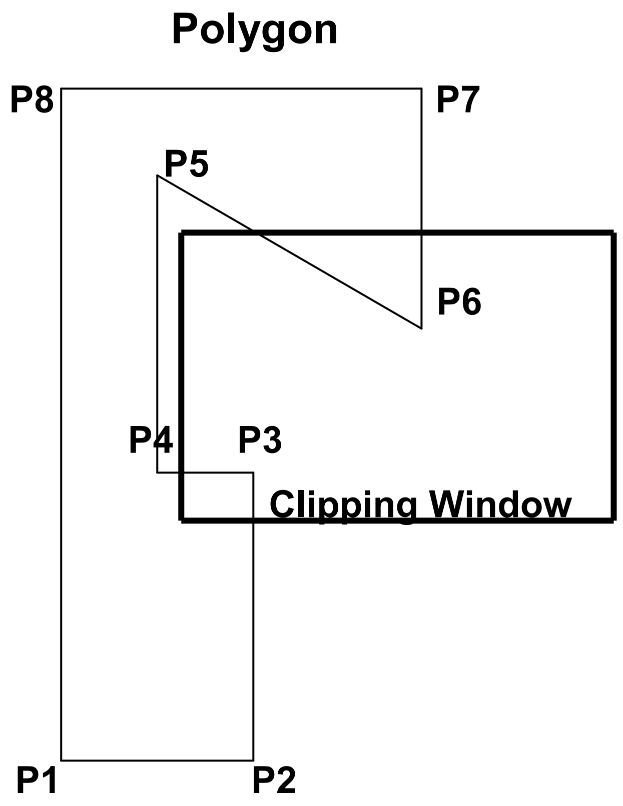


2- If P_{i-1} is to the right and P_i is to the left of edge E, the intersection point $\mathbb{I}$ of the line segment $\overrightarrow{P_{i-1}P_i}$ with the extended edge E is calculated. Both $\mathbb{I}$ and P_i are placed on the vertex output list.

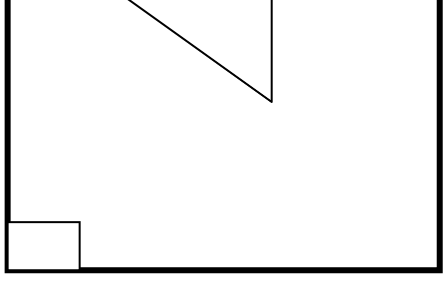


The Algorithm proceeds in stages by passing each clipped polygon to the next edge of the window and clipping.

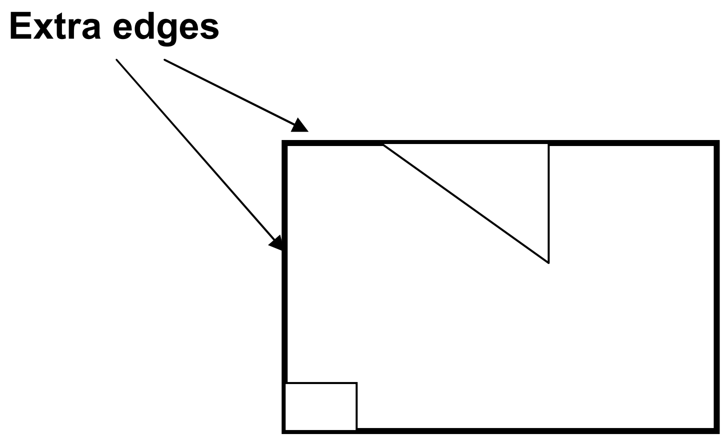
Special attention is necessary in using the Sutherland-Hodgman algorithm in order to avoid unwanted effects. Consider the example in following figure:



The correct result should consist of two disconnected parts, a square in lower left corner of the clipping window and a triangle at the top.

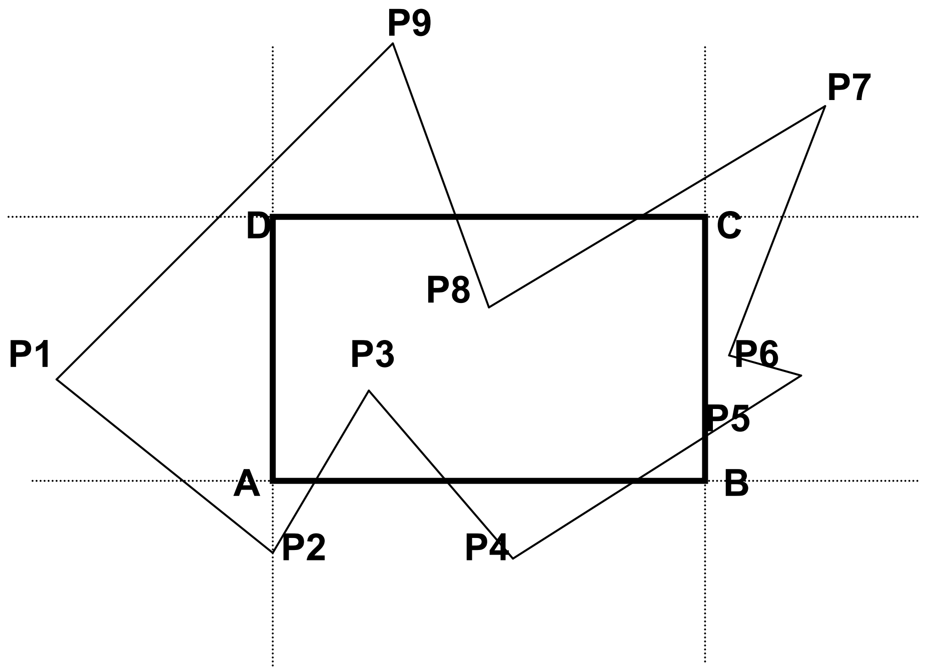


However, the algorithm produces a list of vertices that forms a figure with the two parts connected by extra edges.

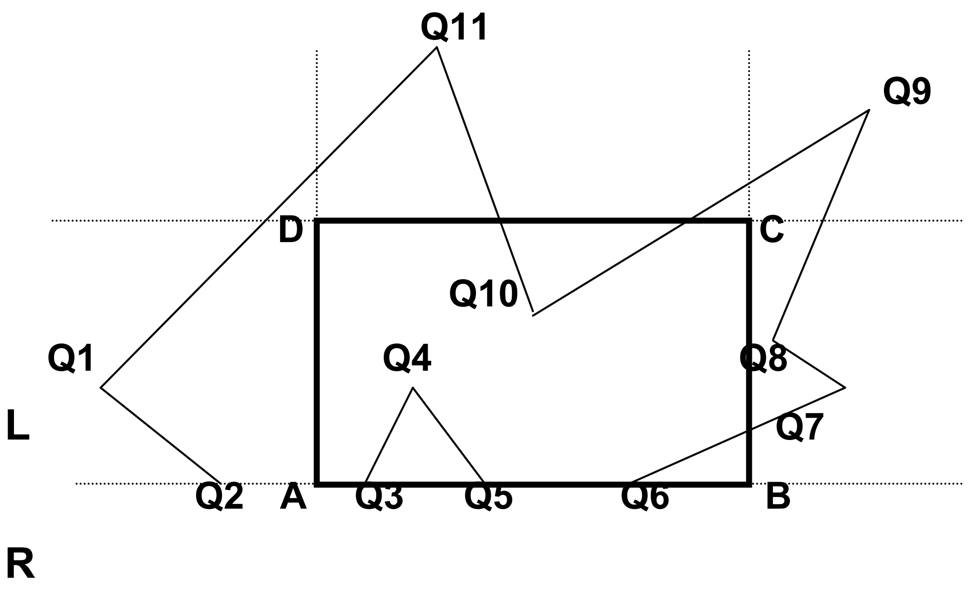


The fact that these edges are drawn twice in opposite direction can be used to devise a post processing step to eliminate them.

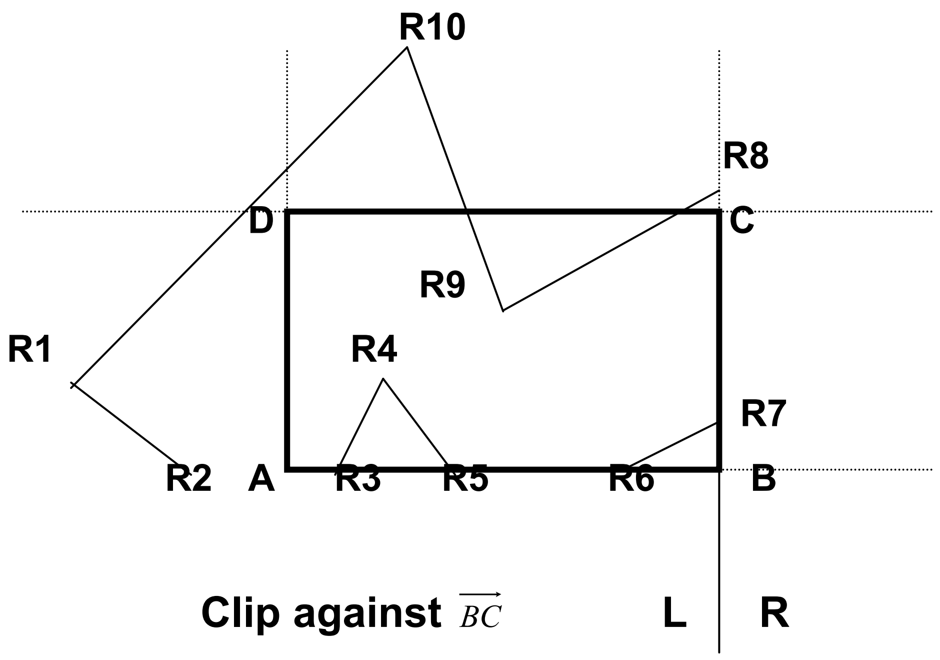
Example: Clip the polygon P1,……..,P9 in the following figure against the window ABCD using the Sutherland-Hodgman algorithm?



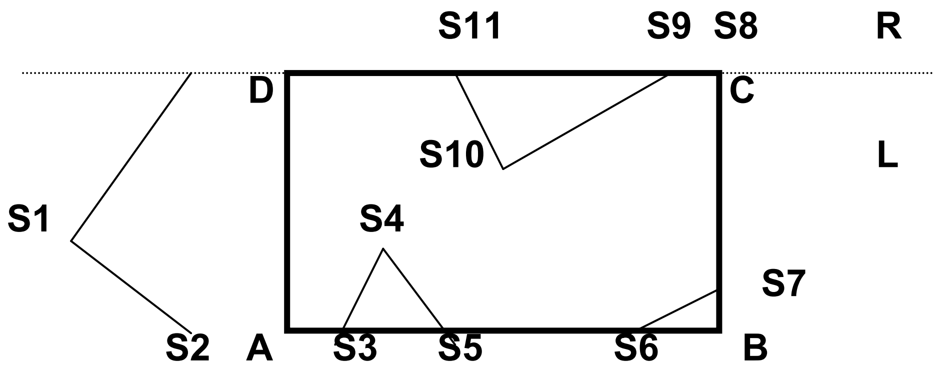
At each stage the new output polygon, whose vertices are determined by applying the Sutherland Hodgman algorithm, is passed on to the next clipping edge of the window ABCD. The results are illustrated in figures:



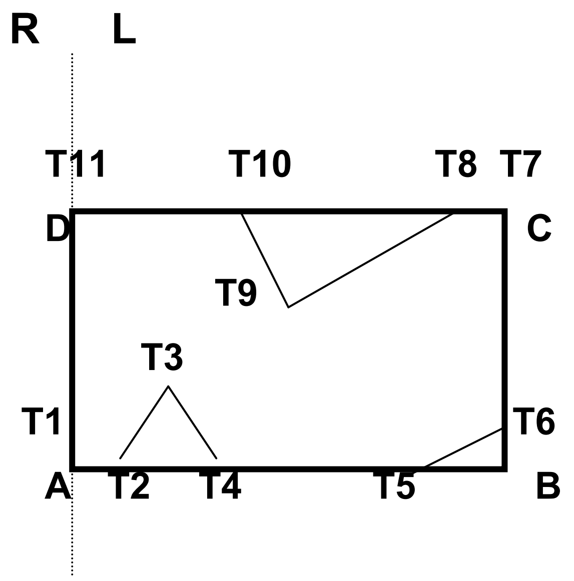
Clip against $\overrightarrow{AB}$



Clip against $\overrightarrow{BC}$



Clip against $\overrightarrow{CD}$



Clip against $\overrightarrow{DA}$

Three-Dimensional (3-D) Graphics Coordinates systems

Our world is composed of the 3-dimensional images. Objects not only have height and width but also depth. Displaying 3-D objects on a 2-D display screen seem to be a hard task. If height and width are represented by (x,y) coordinates, how the third dimensional (depth) be displayed? The techniques used in computer graphics to display this 3-D world are based on the same principles an artist or a photographer employs in producing a realistic image on paper or film, the difference is that the computer uses a mathematical model instead of lens to create the image.

Coordinates Systems:

There are three types of coordinates systems to locate any point in the space: Cartesian, Cylindrical, Spherical coordinates.

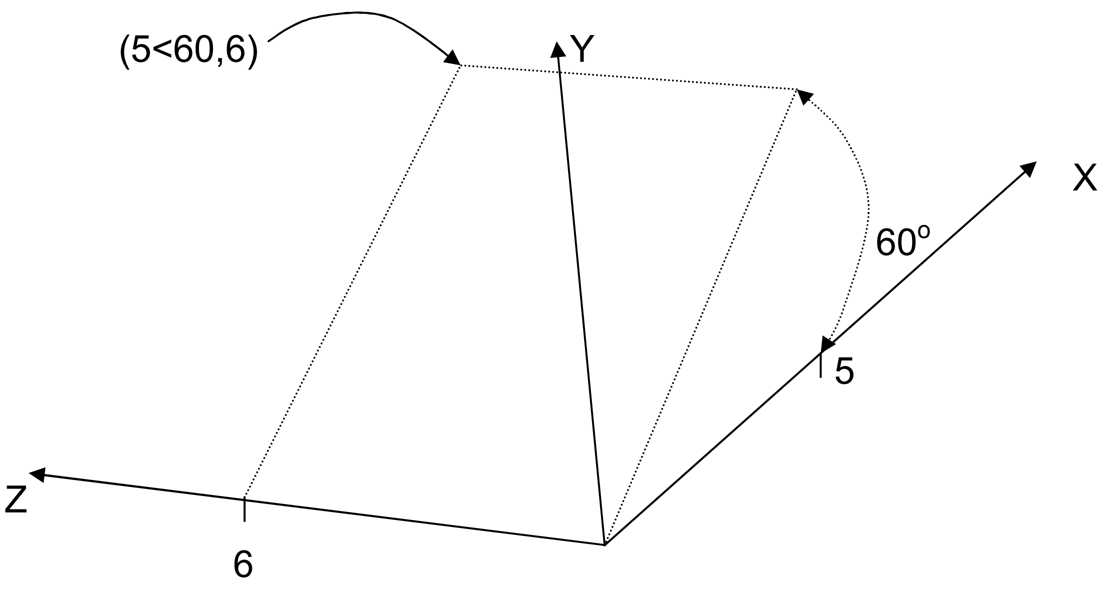
1- Cartesian Coordinates System:

This coordinates have three axes: x, y, and z. When you enter coordinates values, you indicate a point's distance (in units) and its direction (+ or -) along the x, y, and z axes relative to the coordinates system origin (0,0,0) or relative to the previous point.

2- Cylindrical Coordinates System:

2-D polar coordinates system uses a distance and an angle to locate a point. When you enter polar coordinates values, you indicate a point's distance from the origin or from the previous

point and its angle along the (x,y) plane of the current coordinates system. Cylindrical coordinates entry is similar to 2-D polar coordinates entry, but with an additional distance from the polar coordinates perpendicular to the (x,y) plane. You locate a point by specifying its distance along relative to the x-axis and its z value perpendicular to the (x,y) plane as shown in following figure.

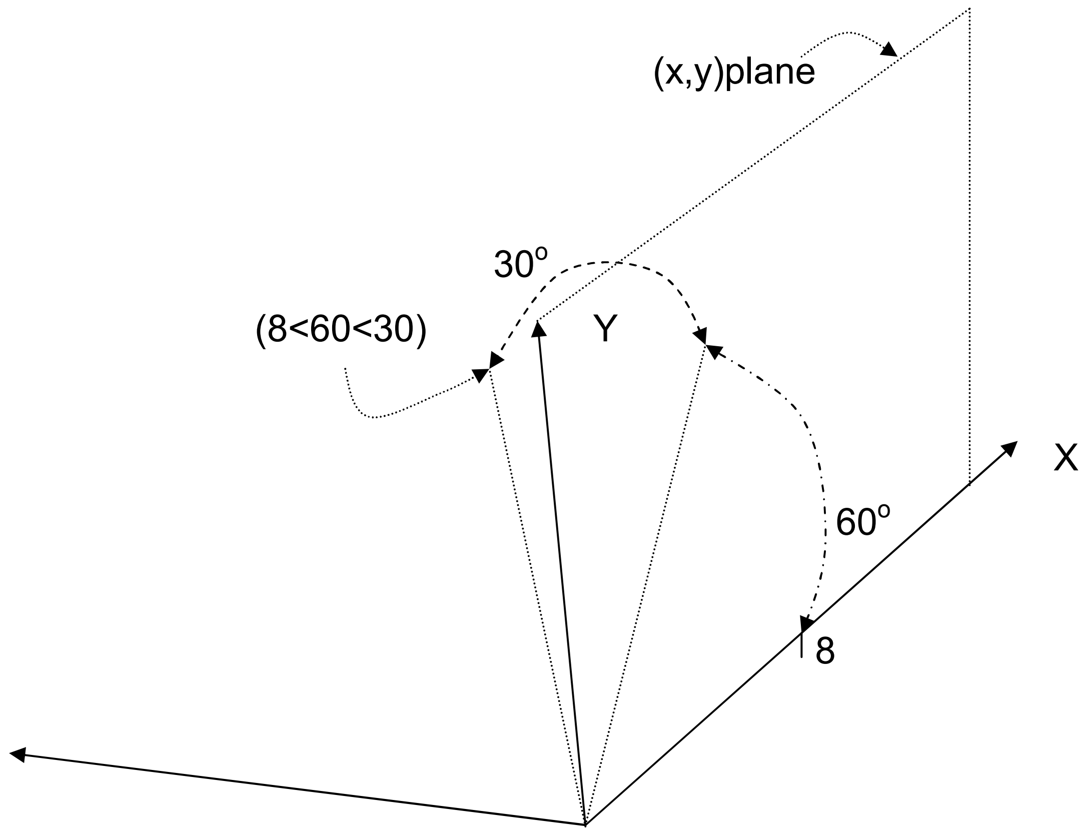


The point coordinate (5<60,6) or (r, θ, z) indicates a point 5 units from the origin, 60 degrees from the x-axis in the (X,Y) plane, and 6 units along the z-axis. The coordinate (8<30,1) indicates a point 8 units from the origin in the (X,Y) plane, 30 degrees from the x-axis in the (X,Y) plane and 1 unit along the z-axis.

3- Spherical Coordinates System:

Spherical coordinate entry in 3-D is also similar to polar coordinate entry in 2-D. You can locate a point by specifying its distance from the origin, its angle from x-axis in the (x,y) plane,

and its angle from the (x,y) plane. The coordinate (8<60<30) indicates a point 8 units from the origin, 60 degrees from the x-axis in the (x,y) plane, and 30 degrees up from the (x,y) plane as shown in figure below.

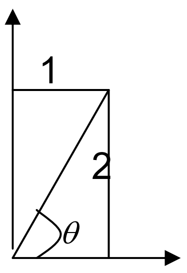


Example: Convert the Cartesian coordinates of the point (1,2,3) to cylindrical coordinates?

tan(θ)=$\frac{2}{1}$ $\rightarrow$ θ=63.4

r= $\sqrt{1^2+2^2}$ $\rightarrow$ r=2.24 , z=3

Cylindrical coordinates = (2.24<63.4,3)



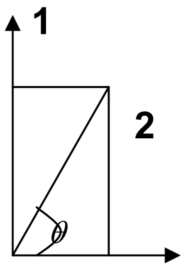
Example: Convert the Cartesian coordinates of the point (1,2,3) to spherical coordinates?

tan(θ)=$\frac{2}{1}$ $\rightarrow$ θ=63.4 (from x-axis)

tan(θ)=$\frac{3}{\sqrt{1^2+2^2}}$ $\rightarrow$ θ=53.3(from xy-plane)

r=$\sqrt{1^2+2^2+3^2}$ $\rightarrow$r=3.74

Spherical coordinates = (3.74<63.4<53.3).



Vectors:

Vector is the difference between two points which the vector passes through these points as shown in the following example.

Example: Find the vector which pass through the point p1(1,1,1) and the point p2(1,2,3)?

V(1-1,2-1,3-1)=(0,1,2)

- Let V1=(x1,y1,z1) and V2=(x2,y2,z2) be two three dimensional vectors. The dot product of these two vectors is defined as:

V1.V2=x1x2+y1y2+z1z2

The dot product of two vectors is not a vectors but a real number (scalar).

Example:

If V1=(1,2,3) and V2=(-2,1,-4) then:

V1.V2=1(-2)+2(1)+3(-4)=-12

- The cosine angle θ between two vectors V1 and V2 is defined as:

$$Cos\theta = \frac{V1.V2}{|V1| \times |V2|}$$

Where $|V1|$ is the length of the vector V1 and $|V2|$ is the length of the vector V2, and $|V|$ = $\sqrt{x^2+y^2+z^2}$.

Example:

Find the angle between the vector V1(1,0,0) and the vector V2(1,1,0)?

$|V|$ **=** $\sqrt{x^2+y^2+z^2}$ **,** $|V1|=\sqrt{1^2+0^2+0^2}$ **=1,** $|V2|=\sqrt{1^2+1^2+0^2}$ **=1.414.**

$Cos\theta = \frac{V1.V2}{|V1|\times|V2|}$ **=** $\frac{(1\times1)+(0\times1)+(0\times0)}{1\times1.414}=0.7072$**,** $\therefore \theta = 45^\circ$ **.**

- **The cross product of the two vectors V1(x1,y1,z1) and V2(x2,y2,z2) is another vector:**

$$V1\times V2=(y1z2-z1y2, z1x2-x1z2, x1y2-y1x2)$$

- **Normalization (unit vector): the normalization of vector V1(x1,y1,z1) is (**$\frac{x1}{|V1|},\frac{y1}{|V1|},\frac{z1}{|V1|}$**).**

Example: Normalize the vector V1(1,2,3)?

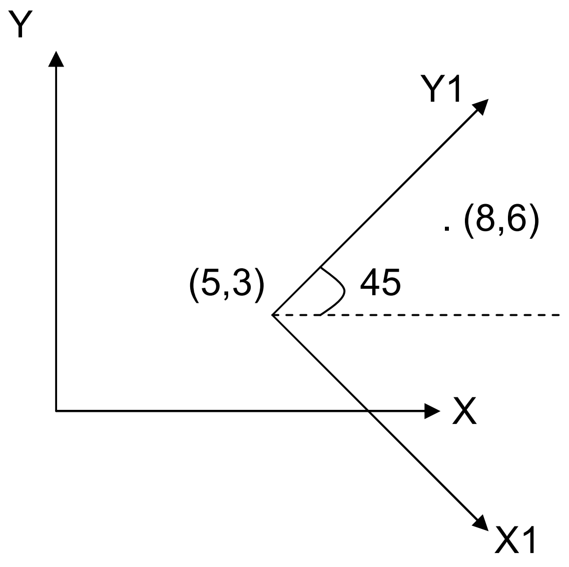
$|V|$ **=** $\sqrt{x^2+y^2+z^2}$ **,** $|V1|=\sqrt{1^2+2^2+3^2}$ **=3.742.**

The normalized vector V1(x1,y1,z1) = ($\frac{x1}{|V1|},\frac{y1}{|V1|},\frac{z1}{|V1|}$**)=**

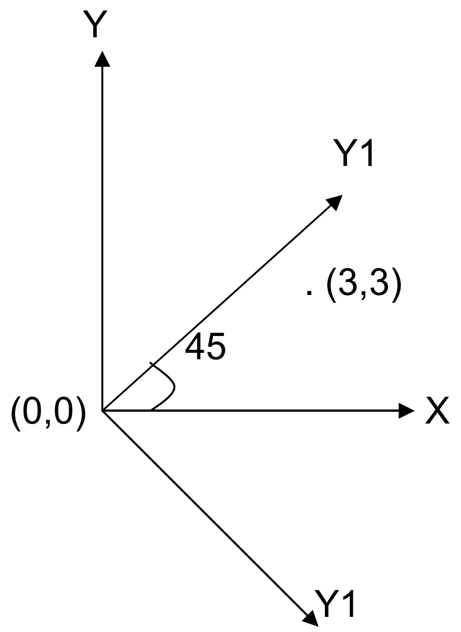
($\frac{1}{3.742},\frac{2}{3.742},\frac{3}{3.742}$**)= (0.267,0.534,0.801).**

Transforming Coordinates Systems

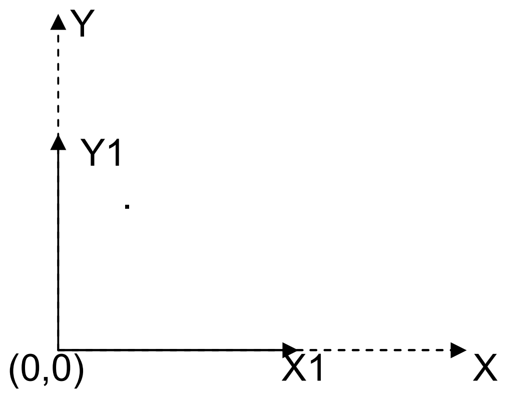
Object that is defined in one coordinates system may have to be expressed in terms of another coordinates system. In general, if an object is defined in terms of coordinates system1, it can be defined in terms of another coordinates system2 by transforming the axis of coordinates system2 into the axis of coordinates system1. This transformation, when applied to the representation of the object in coordinates system2, gives the object's representation in coordinates system2. For example:



a- Original figure



b-First step (translation)



c- second step (rotation)

Where the second coordinates system, denoted by (x1,y1) is at clockwise angle 45^{o} with the original coordinates system (x,y).

The steps of transforming coordinates system for the point (8,6) at figure (a) above in 2-D are:

First step (shown at figure (b) above): is 2-D translation of second coordinates system (x1,y1) to first coordinates system (x,y) by:

$$\textbf{Translate (-5,-3)=}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & -3 & 1 \end{bmatrix}$$

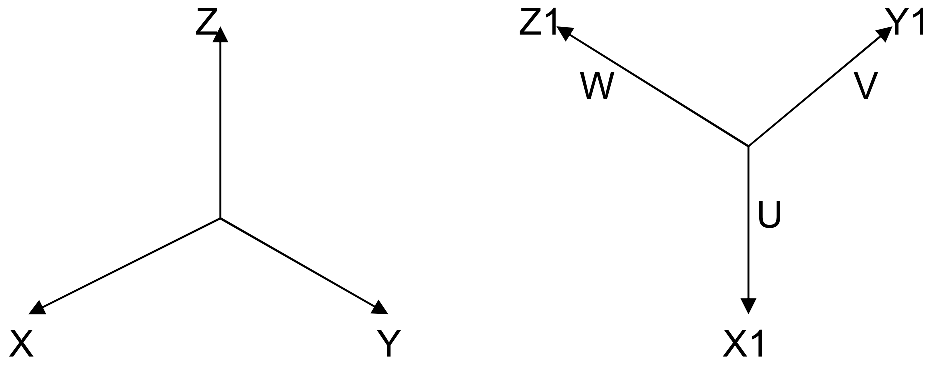
Second step (shown at figure (c) above): is 2-D rotation of second coordinates system (X1,Y1) counterclockwise 45^o by:

$$\textbf{Rotate(45)=}\begin{bmatrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The above example can be extended to 3-D. Let a coordinates system (x1,y1,z1) be defined in terms of the standard (x,y, and z) coordinates by the unit vector:

u=(ux,uy,uz), v=(vx,vy,vz), w=(wx,wy,wz)

also assume the origin of the (x1,y1,z1) coordinates system is at (a,b,c) with respect to the (x,y,z) coordinates system as shown in figure below:



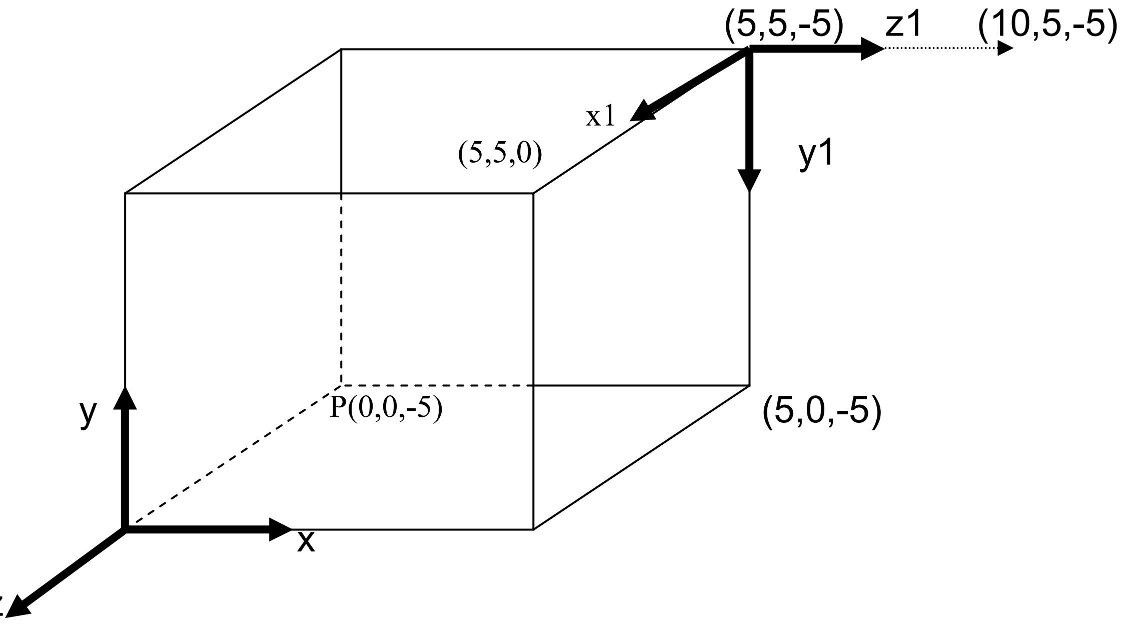
The transformation sends the (x1,y1,z1) coordinates system to the (x,y,z) system (with u going to x, v going to y, and w going to z) by using the product of translation followed by a rotation:

$$\mathbf{Tr\times R=}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\-a&-b&-c&1\end{bmatrix}\begin{bmatrix}ux&uy&uz&0\\vx&vy&vz&0\\wx&wy&wz&0\\0&0&0&1\end{bmatrix}$$

A point P defined in the (x,y,z) coordinates system has representation P1 in (x1,y1,z1) coordinates system given by:

$$\mathbf{P1=P\times Tr\times R}$$

Example: The coordinates of point P in terms of (x,y,z) is (0,0,-5), find the coordinates of the same point in terms of (x1,y1,z1) using transforming coordinates system with center at (5,5,-5) as shown in figure below:



	P1	P2	Vector (P2-P1)	Normalized

V_{x1}	(5,5,-5)	(5,5,0)	(0,0,5)	(0,0,1)
V_{y1}	(5,5,-5)	(5,0,-5)	(0,-5,0)	(0,-1,0)
V_{z1}	(5,5,-5)	(10,5,-5)	(5,0,0)	(1,0,0)

P in terms of (x1,y1,z1)=P1=

$$=P \times Tr \times R=\begin{bmatrix}0 & 0 & -5 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ -5 & -5 & 5 & 1\end{bmatrix}\begin{bmatrix}0 & 0 & 1 & 0\\ 0 & -1 & 0 & 0\\ 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}0 & 5 & -5 & 1\end{bmatrix}$$

Coordinates of (P1) is (0,5,-5).

Geometric Transformation

(3-D) Three Dimensional Transformations

A 3-D geometric transformations are extensions of the two dimensional transformations techniques as shown below:

1- 3-D Translation:

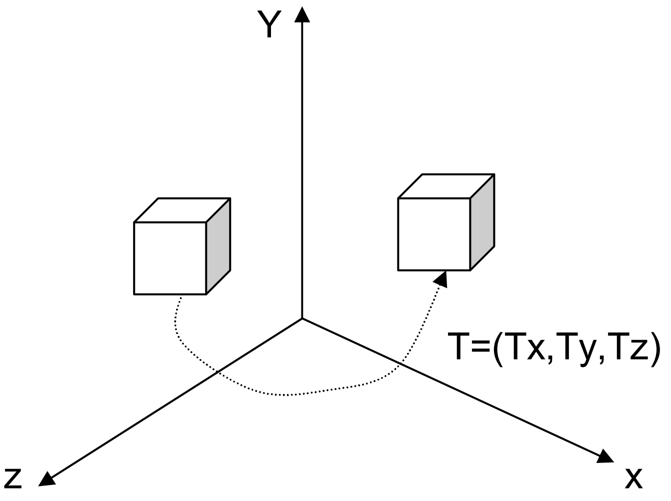
A 3-D object can be translated from one place to another by translating every point of the 3-D object to the new place as follows:

$$\bar{x}=x+T_x, \quad \bar{y}=y+T_y, \quad \bar{z}=z+T_z$$

3-D Translation transformation can be represented in a uniform way by a 4×4 matrix as shown below:

$$[\bar{x} \quad \bar{y} \quad \bar{z} \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{bmatrix}$$

As shown in following figure:



2- 3-D Scaling:

A 3-D object can be scaled as follows:

$$\bar{x} = xS_x , \quad \bar{y} = yS_y , \quad \bar{z} = zS_z$$

Where S_x, S_y, and S_z are scaling factors.

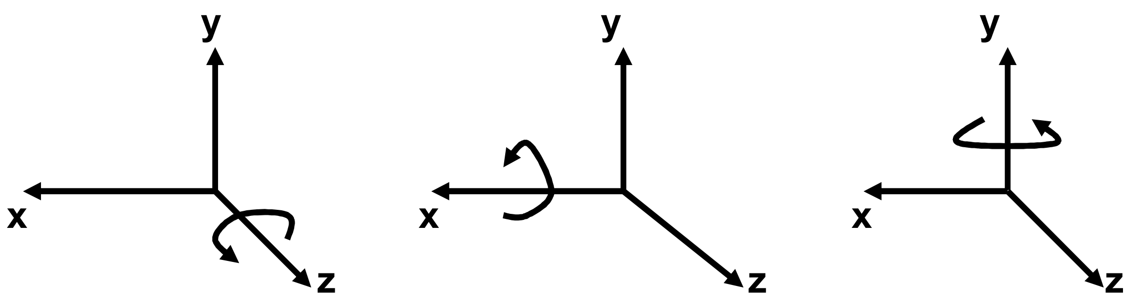
3-D scaling transformation can be represented in a uniform way by a 4×4 matrix as shown below:

$$[\bar{x} \quad \bar{y} \quad \bar{z} \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3- 3-D Rotation:

In order to rotate an object in 3-D world, the rotation axis and rotation angle must be specified firstly. Any of the three axes (x,y,z) may be the rotation axis. Rotation of a point (x,y,z)

through a counterclockwise angle θ about the positive part of the axis, can be shown in following figures:



Rotation about z-axis:

Rotation transformation equations:

$\bar{x}$ **=x cosθ+ y sinθ ,** $\bar{y}$**= -x sinθ+ y cosθ ,** $\bar{z}$**=z**

Rotation transformation can be done using a 4×4 matrix::

$$[\bar{x} \quad \bar{y} \quad \bar{z} \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about x-axis:

Rotation transformation equations:

$\bar{y}$**=y cosθ+ z sinθ ,** $\bar{z}$**= -y sinθ+ z cosθ ,** $\bar{x}$**=x**

Rotation transformation can be done using a 4×4 matrix::

$$[\bar{x} \quad \bar{y} \quad \bar{z} \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about y-axis:

Rotation transformation equations:

$\bar{z}$**= - x sinθ + z cosθ,** $\bar{x}$**= x cosθ + z sinθ,** $\bar{y}$**=y**

Rotation transformation can be done using a 4×4 matrix::

$$[\bar{x} \quad \bar{y} \quad \bar{z} \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

H.W.:

Write complete (C or C++) program in order to perform all types of 3-D transformations on a cubic object (Use suitable vertices for the cubic object)?

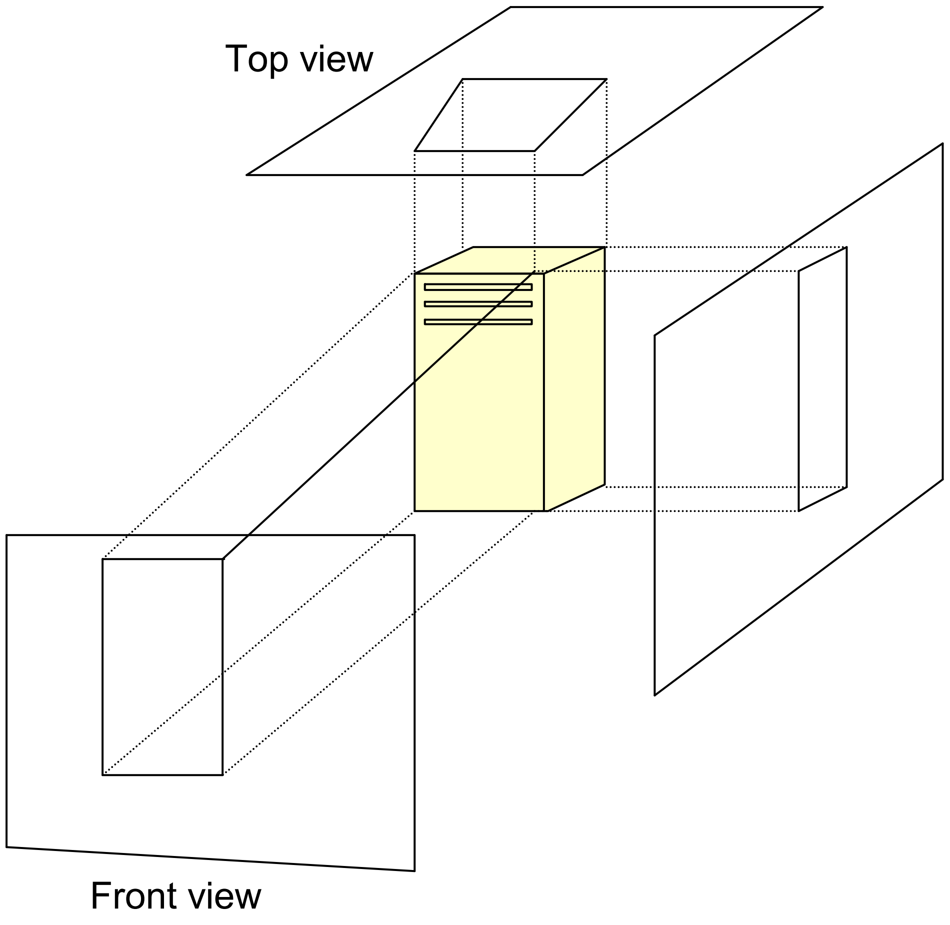
Projections

It is easy to see a group of points in 2-D world and convert them onto computer screen. This is can be done by putting these points in a viewport with clipping processing for enhancement.

In 3-D world the subject is different. In order to see 3-D object, there are different views such as: top view, side view, or front view. In addition, 3-D object must be processed with projections techniques in order to see it as a flat object on a 2-D screen. There are two methods for projecting 3-D object on 2-D plane:

1- Parallel Projection:

Parallel projection represents the 3-D object relatively; therefore this type is used with quick drawing (draft drawing). Parallel projection can give different views from the 3-D object from different sides as shown in figure below:



Note: parallel projection cannot give a realistic representation for 3-D object.

Parallel Orthographic Projection

The simplest of the parallel projections is the orthographic projection, commonly used for engineering drawings. They accurately show correct size and shape of a single plane face of an object. Orthographic projections are projections onto one of the coordinate planes x=0, y=0, or z=0. The matrix for projection onto the z=0 plane is:

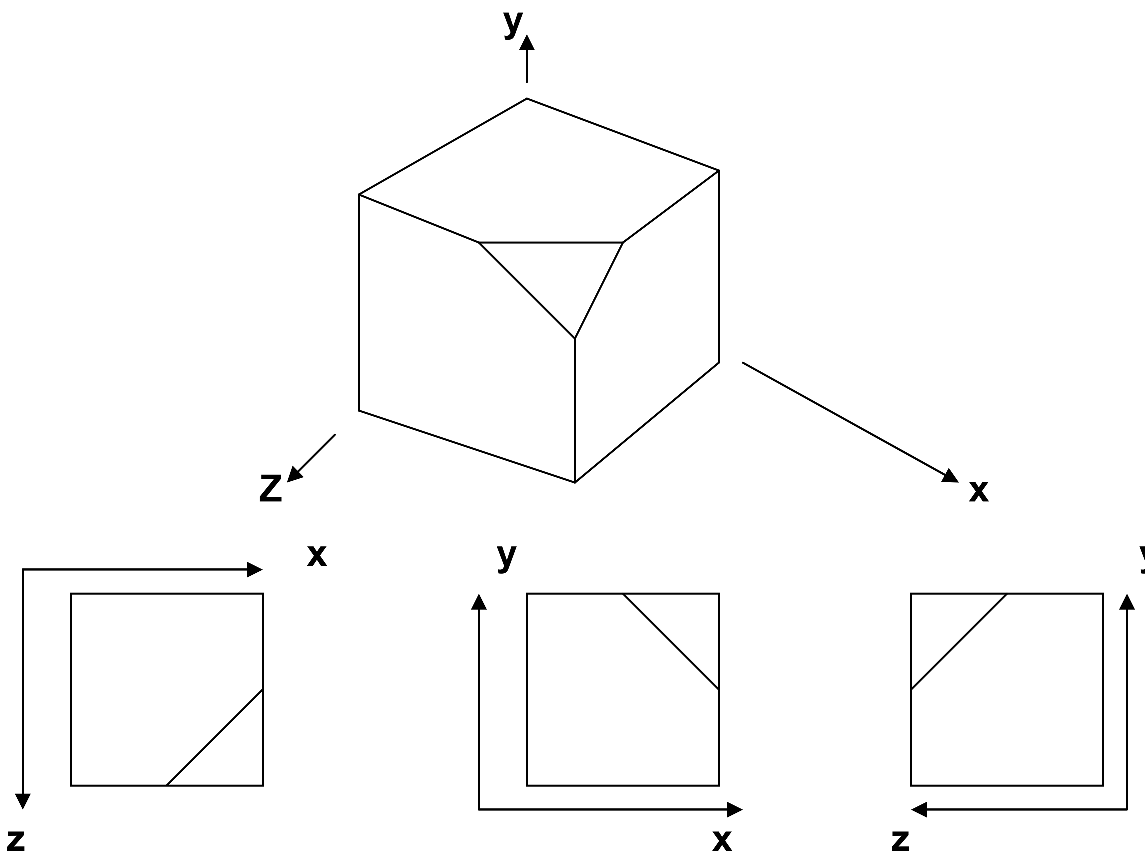
$$[Pz] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly, the matrices for projection onto the x=0 and y=0 planes are:

$$[Px] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[Py] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

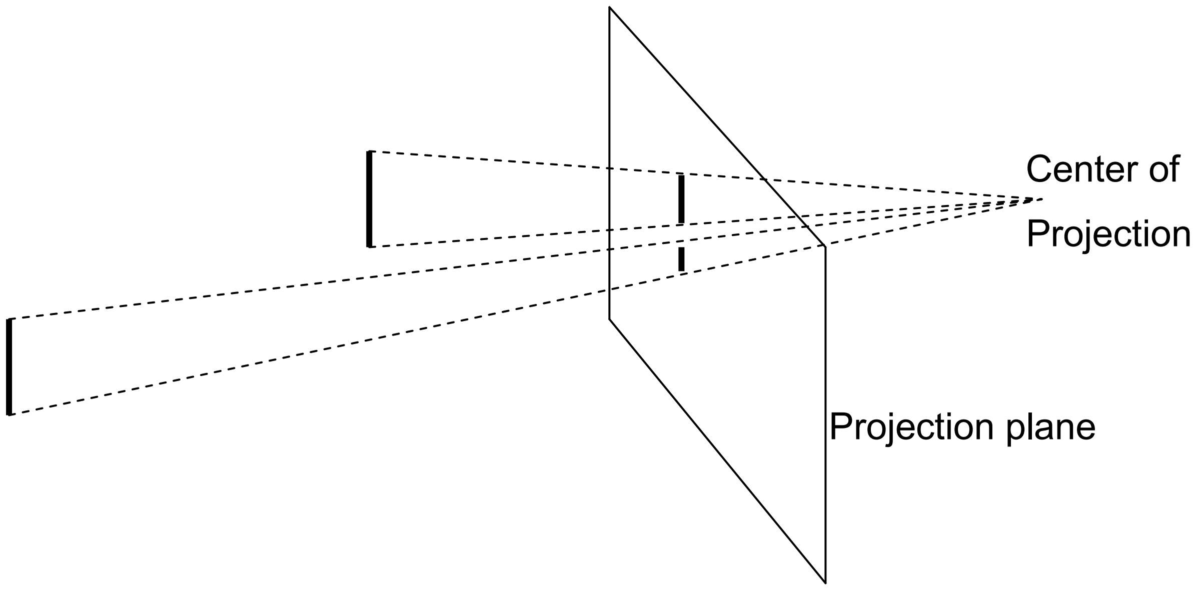
Orthographic projections of the object in figure below:



A single orthographic projection does not provide sufficient information to visually and practically reconstruct the shape of an object. Consequently, multiple orthographic projections are necessary.

2- Perspective Projection:

Perspective projection gives a realistic representation for 3-D object, but without real dimensions values.



In above figure, there are two equal length lines, but with different perspective projections. The closer line to the projection plane is the larger projection length.

To calculate the perspective projection lengths, the following equations are used:

$$Xp = X\left(\frac{d}{d+Z}\right)$$

$$Yp = Y\left(\frac{d}{d+Z}\right)$$

$$Zp = 0$$

Where x,y,and z: represent coordinates of a point in 3-D world P(x,y,z).

Xp,Yp, and Zp: represent coordinates of a point in projection plane (Xp,Yp,0).

d: the distance between 3-D coordinates of a point, and projection plane.

H.W.:

Find Perspective Projection of cubic object :((0,0,0), (5,0,0), (5,5,0), (0,5,0), (0,5,5), (0,0,5), (5,0,5), (5,5,5)). Note that d=5 units?

Three-Dimensional (3-D)
Graphics Coordinates systems

Our world is composed of the 3-dimensional images. Objects not only have height and width but also depth. Displaying 3-D objects on a 2-D display screen seem to be a hard task. If height and width are represented by (x,y) coordinates, how the third dimensional (depth) be displayed? The techniques used in computer graphics to display this 3-D world are based on the same principles an artist or a photographer employs in producing a realistic image on paper or film, the difference is that the computer uses a mathematical model instead of lens to create the image.

Coordinates Systems:

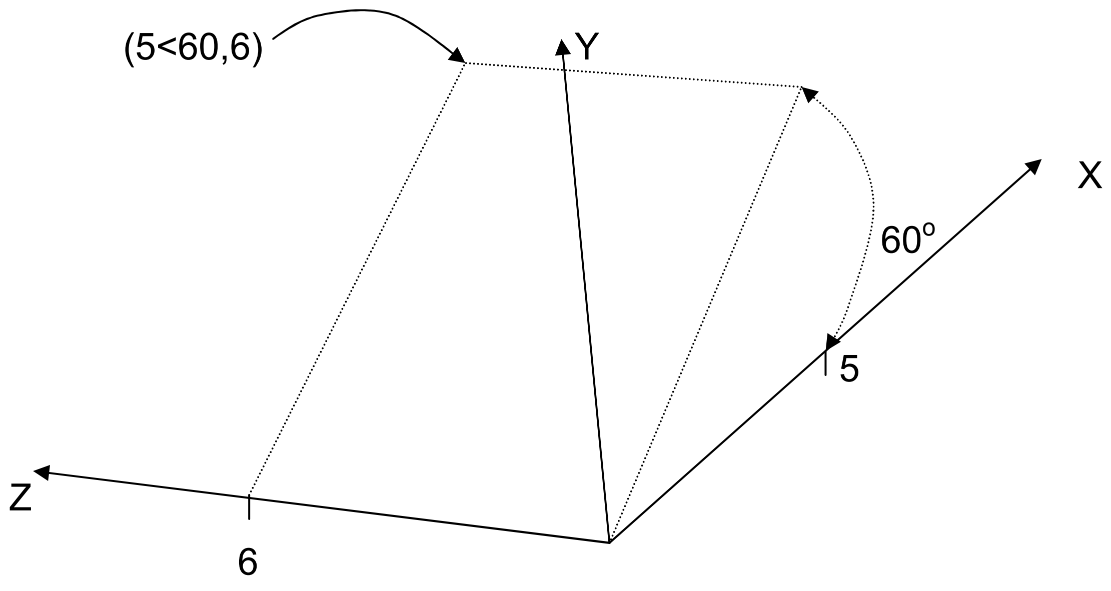
There are three types of coordinates systems to locate any point in the space: Cartesian, Cylindrical, Spherical coordinates.

2- Cartesian Coordinates System:

This coordinates have three axes: x, y, and z. When you enter coordinates values, you indicate a point's distance (in units) and its direction (+ or -) along the x, y, and z axes relative to the coordinates system origin (0,0,0) or relative to the previous point.

2- Cylindrical Coordinates System:

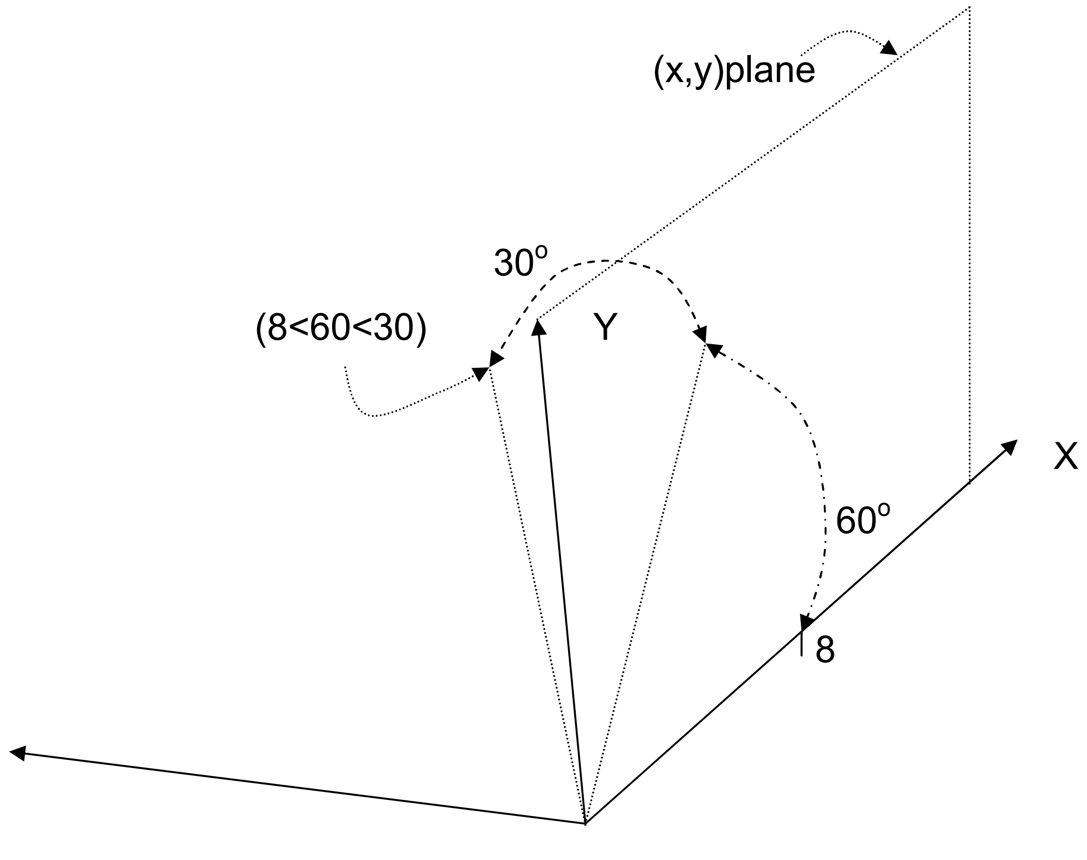
2-D polar coordinates system uses a distance and an angle to locate a point. When you enter polar coordinates values, you indicate a point's distance from the origin or from the previous point and its angle along the (x,y) plane of the current coordinates system. Cylindrical coordinates entry is similar to 2-D polar coordinates entry, but with an additional distance from the polar coordinates perpendicular to the (x,y) plane. You locate a point by specifying its distance along relative to the x-axis and its z value perpendicular to the (x,y) plane as shown in following figure.



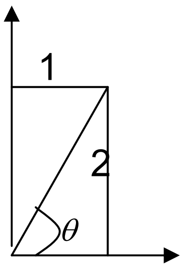
The point coordinate (5<60,6) or (r, θ, z) indicates a point 5 units from the origin, 60 degrees from the x-axis in the (X,Y) plane, and 6 units along the z-axis. The coordinate (8<30,1) indicates a point 8 units from the origin in the (X,Y) plane, 30 degrees from the x-axis in the (X,Y) plane and 1 unit along the z-axis.

3- Spherical Coordinates System:

Spherical coordinate entry in 3-D is also similar to polar coordinate entry in 2-D. You can locate a point by specifying its distance from the origin, its angle from x-axis in the (x,y) plane, and its angle from the (x,y) plane. The coordinate (8<60<30) indicates a point 8 units from the origin, 60 degrees from the x-axis in the (x,y) plane, and 30 degrees up from the (x,y) plane as shown in figure below.



Example: **Convert the Cartesian coordinates of the point (1,2,3) to cylindrical coordinates?**

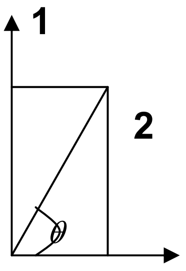


tan(θ)=$\frac{2}{1}$ $\rightarrow$ θ**=63.4**

r= $\sqrt{1^2+2^2}$ $\rightarrow$ **r=2.24 , z=3**

Cylindrical coordinates = (2.24<63.4,3)

Example: **Convert the Cartesian coordinates of the point (1,2,3) to spherical coordinates?**



tan(θ)=$\frac{2}{1}$ $\rightarrow$ θ**=63.4 (from x-axis)**

tan(θ)=$\frac{3}{\sqrt{1^2+2^2}}$ $\rightarrow$ θ**=53.3(from xy-plane)**

r=$\sqrt{1^2+2^2+3^2}$ $\rightarrow$**r=3.74**

Spherical coordinates = (3.74<63.4<53.3).

Vectors:

Vector is the difference between two points which the vector passes through these points as shown in the following example.

Example: Find the vector which pass through the point p1(1,1,1) and the point p2(1,2,3)?

V(1-1,2-1,3-1)=(0,1,2)

- Let V1=(x1,y1,z1) and V2=(x2,y2,z2) be two three dimensional vectors. The dot product of these two vectors is defined as:

V1.V2=x1x2+y1y2+z1z2

The dot product of two vectors is not a vectors but a real number (scalar).

Example:

If V1=(1,2,3) and V2=(-2,1,-4) then:

V1.V2=1(-2)+2(1)+3(-4)=-12

- The cosine angle θ between two vectors V1 and V2 is defined as:

$$Cos\theta = \frac{V1.V2}{|V1| \times |V2|}$$

Where $|V1|$ is the length of the vector V1 and $|V2|$ is the length of the vector V2, and $|V|$ = $\sqrt{x^2 + y^2 + z^2}$.

Example:

Find the angle between the vector V1(1,0,0) and the vector V2(1,1,0)?

$|V|$ **=** $\sqrt{x^2 + y^2 + z^2}$ **,** $|V1| = \sqrt{1^2 + 0^2 + 0^2}$ **=1,** $|V2| = \sqrt{1^2 + 1^2 + 0^2}$ **=1.414.**

$Cos\theta = \frac{V1.V2}{|V1| \times |V2|}$ **=** $\frac{(1\times1)+(0\times1)+(0\times0)}{1\times1.414} = 0.7072$**,** $\therefore \theta = 45^\circ$ **.**

- The cross product of the two vectors V1(x1,y1,z1) and V2(x2,y2,z2) is another vector:

$$V1 \times V2 = (y1z2 - z1y2, z1x2 - x1z2, x1y2 - y1x2)$$

- Normalization (unit vector): the normalization of vector V1(x1,y1,z1) is ($\frac{x1}{|V1|}, \frac{y1}{|V1|}, \frac{z1}{|V1|}$**).**

Example: Normalize the vector V1(1,2,3)?

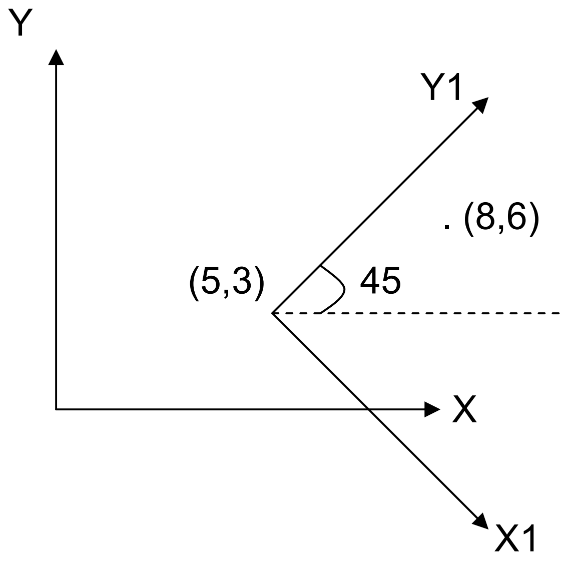
$|V|$ = $\sqrt{x^2+y^2+z^2}$, $|V1| = \sqrt{1^2+2^2+3^2}$ **=3.742.**

The normalized vector V1(x1,y1,z1) = $(\frac{x1}{|V1|}, \frac{y1}{|V1|}, \frac{z1}{|V1|})$**=**

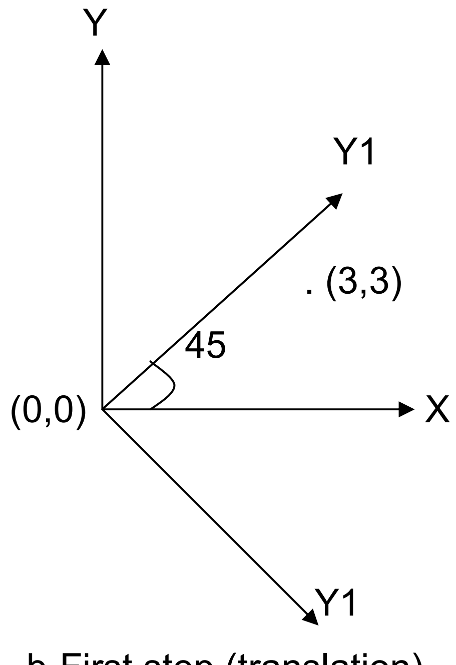
$(\frac{1}{3.742}, \frac{2}{3.742}, \frac{3}{3.742})$**= (0.267,0.534,0.801).**

Transforming Coordinates Systems

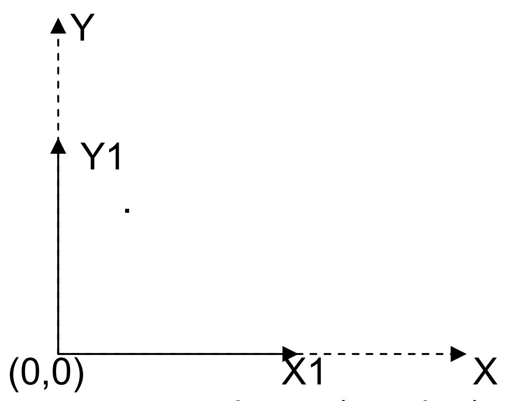
Object that is defined in one coordinates system may have to be expressed in terms of another coordinates system. In general, if an object is defined in terms of coordinates system1, it can be defined in terms of another coordinates system2 by transforming the axis of coordinates system2 into the axis of coordinates system1. This transformation, when applied to the representation of the object in coordinates system2, gives the object's representation in coordinates system2. For example:



a- Original figure



b-First step (translation)



c- second step (rotation)

Where the second coordinates system, denoted by (x1,y1) is at clockwise angle 45^{o} with the original coordinates system (x,y). The steps of transforming coordinates system for the point (8,6) at figure (a) above in 2-D are:

First step (shown at figure (b) above): is 2-D translation of second coordinates system (x1,y1) to first coordinates system (x,y) by:

$$\textbf{Translate (-5,-3)=}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -5 & -3 & 1 \end{bmatrix}$$

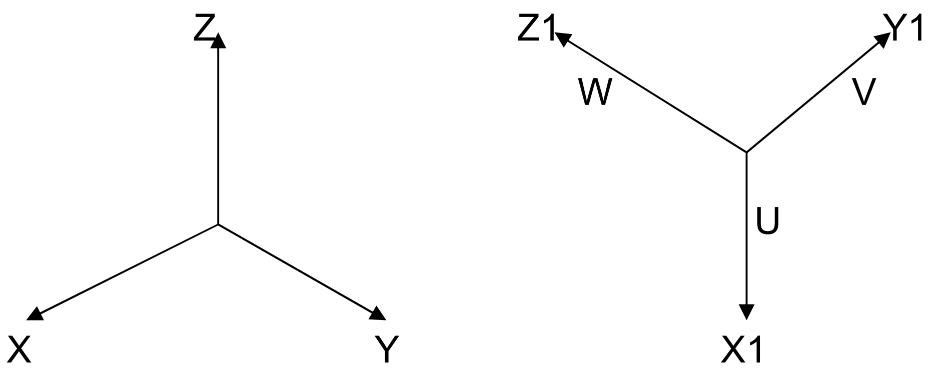
Second step (shown at figure (c) above): is 2-D rotation of second coordinates system (X1,Y1) counterclockwise 45^{o} by:

$$\textbf{Rotate(45)=}\begin{bmatrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The above example can be extended to 3-D. Let a coordinates system (x1,y1,z1) be defined in terms of the standard (x,y, and z) coordinates by the unit vector:

u=(ux,uy,uz), v=(vx,vy,vz), w=(wx,wy,wz)

also assume the origin of the (x1,y1,z1) coordinates system is at (a,b,c) with respect to the (x,y,z) coordinates system as shown in figure below:



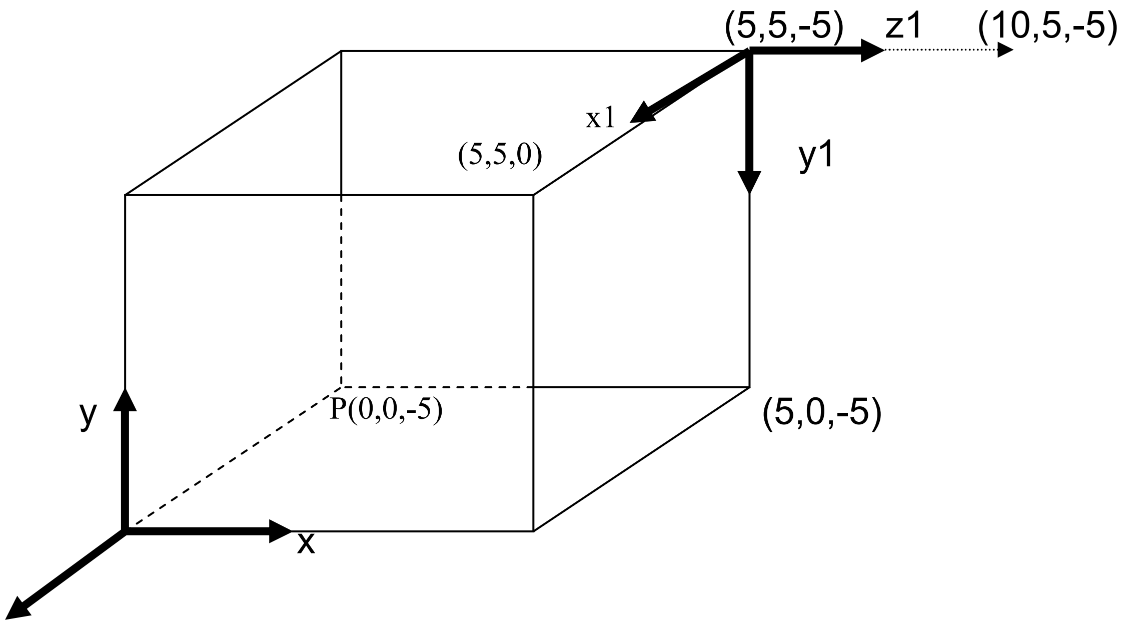
The transformation sends the (x1,y1,z1) coordinates system to the (x,y,z) system (with u going to x, v going to y, and w going to z) by using the product of translation followed by a rotation:

$$\textbf{Tr}\times\textbf{R}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{bmatrix}\begin{bmatrix} ux & uy & uz & 0 \\ vx & vy & vz & 0 \\ wx & wy & wz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A point P defined in the (x,y,z) coordinates system has representation P1 in (x1,y1,z1) coordinates system given by:

$$\textbf{P1=P}\times\textbf{Tr}\times\textbf{R}$$

Example: The coordinates of point P in terms of (x,y,z) is (0,0,-5), find the coordinates of the same point in terms of (x1,y1,z1) using transforming coordinates system with center at (5,5,-5) as shown in figure below:



	P1	**P2**	**Vector (P2-P1)**	**Normalized**
V_{x1}	**(5,5,-5)**	**(5,5,0)**	**(0,0,5)**	**(0,0,1)**
V_{y1}	**(5,5,-5)**	**(5,0,-5)**	**(0,-5,0)**	**(0,-1,0)**
V_{z1}	**(5,5,-5)**	**(10,5,-5)**	**(5,0,0)**	**(1,0,0)**

P in terms of (x1,y1,z1)=P1=

$$=P \times Tr \times R = \begin{bmatrix} 0 & 0 & -5 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & -5 & 5 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 5 & -5 & 1 \end{bmatrix}$$

Coordinates of (P1) is (0,5,-5).

Geometric Transformation

(3-D) Three Dimensional Transformations

A 3-D geometric transformations are extensions of the two dimensional transformations techniques as shown below:

1- 3-D Translation:

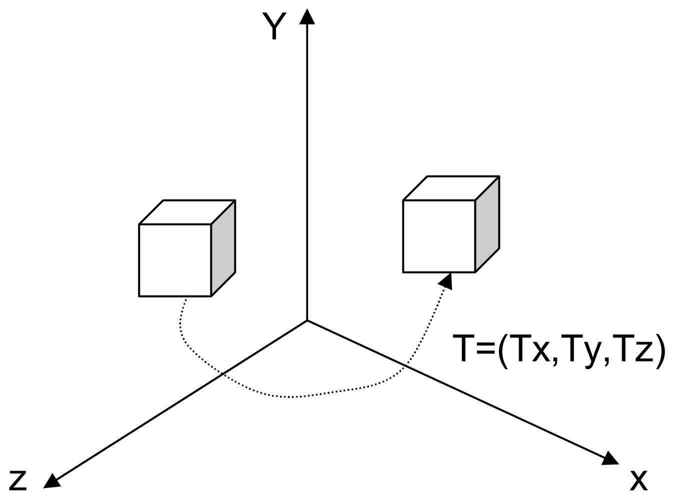
A 3-D object can be translated from one place to another by translating every point of the 3-D object to the new place as follows:

$$\bar{x} = x + T_x , \quad \bar{y} = y + T_y , \quad \bar{z} = z + T_z$$

3-D Translation transformation can be represented in a uniform way by a 4×4 matrix as shown below:

$$[\bar{x} \quad \bar{y} \quad \bar{z} \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{bmatrix}$$

As shown in following figure:



2- 3-D Scaling:

A 3-D object can be scaled as follows:

$$\bar{x} = xS_x, \quad \bar{y} = yS_y, \quad \bar{z} = zS_z$$

Where S_x, S_y, and S_z are scaling factors.

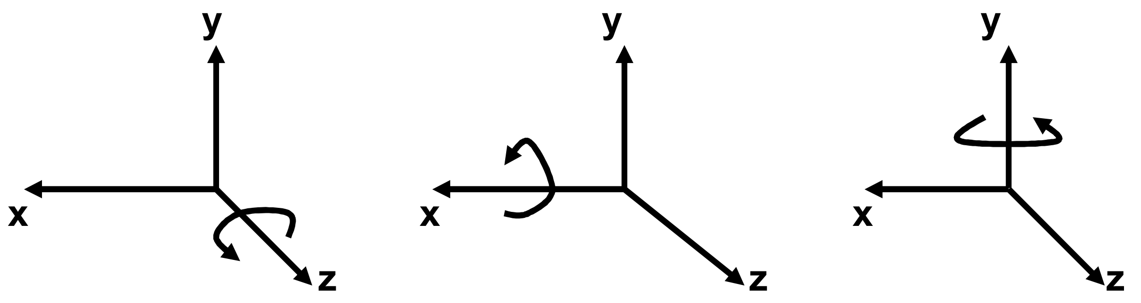
3-D scaling transformation can be represented in a uniform way by a 4×4 matrix as shown below:

$$[\bar{x} \quad \bar{y} \quad \bar{z} \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3- 3-D Rotation:

In order to rotate an object in 3-D world, the rotation axis and rotation angle must be specified firstly. Any of the three axes (x,y,z) may be the rotation axis. Rotation of a point (x,y,z)

through a counterclockwise angle θ about the positive part of the axis, can be shown in following figures:



Rotation about z-axis:

Rotation transformation equations:

$\bar{x}$ **=x cosθ+ y sinθ ,** $\bar{y}$**= -x sinθ+ y cosθ ,** $\bar{z}$**=z**

Rotation transformation can be done using a 4×4 matrix::

$$[\bar{x} \quad \bar{y} \quad \bar{z} \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about x-axis:

Rotation transformation equations:

$\bar{y}$**=y cosθ+ z sinθ ,** $\bar{z}$**= -y sinθ+ z cosθ ,** $\bar{x}$**=x**

Rotation transformation can be done using a 4×4 matrix::

$$[\bar{x} \quad \bar{y} \quad \bar{z} \quad 1] = [x \quad y \quad z \quad 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about y-axis:

Rotation transformation equations:

$\bar{z}$**= - x sinθ + z cosθ,** $\bar{x}$**= x cosθ + z sinθ,** $\bar{y}$**=y**

Rotation transformation can be done using a 4×4 matrix::

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

H.W.:

Write complete (C or C++) program in order to perform all types of 3-D transformations on a cubic object (Use suitable vertices for the cubic object)?

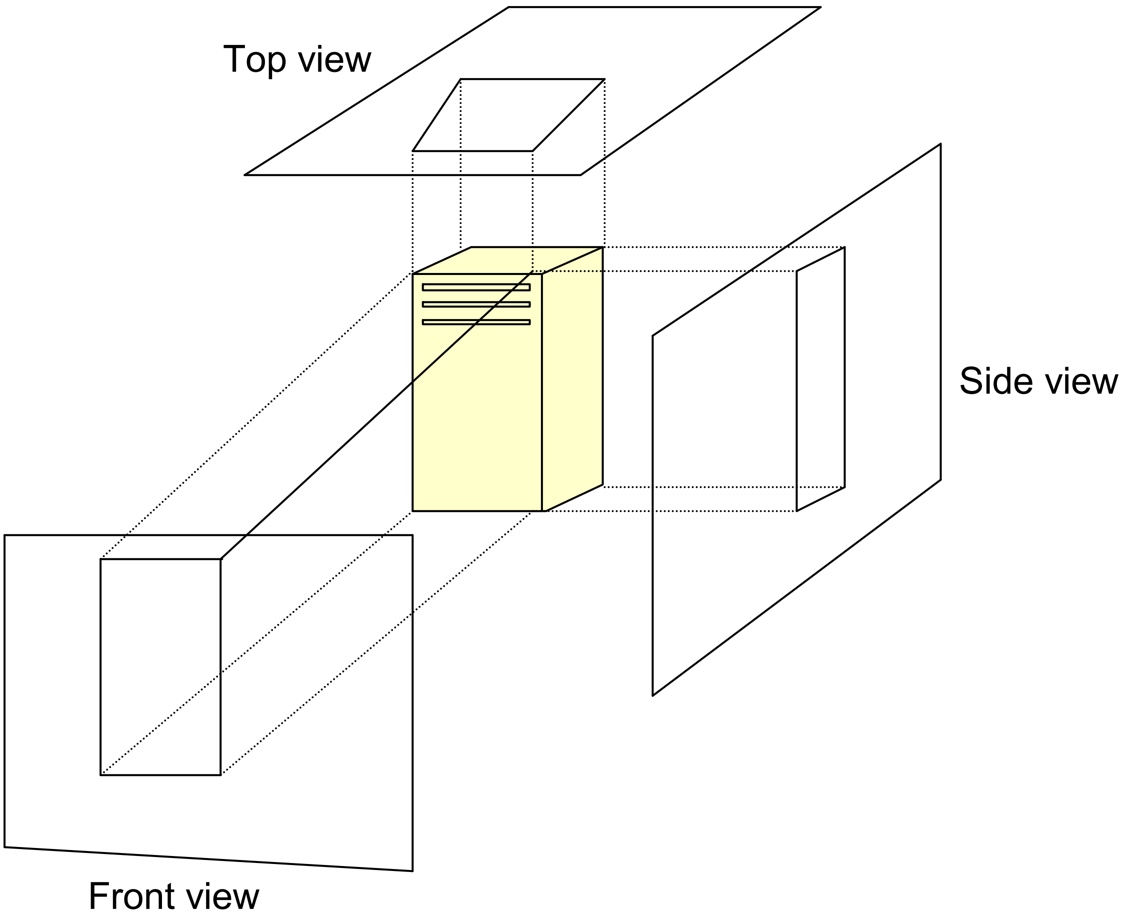
Projections

It is easy to see a group of points in 2-D world and convert them onto computer screen. This is can be done by putting these points in a viewport with clipping processing for enhancement.

In 3-D world the subject is different. In order to see 3-D object, there are different views such as: top view, side view, or front view. In addition, 3-D object must be processed with projections techniques in order to see it as a flat object on a 2-D screen. There are two methods for projecting 3-D object on 2-D plane:

1- Parallel Projection:

Parallel projection represents the 3-D object relatively; therefore this type is used with quick drawing (draft drawing). Parallel projection can give different views from the 3-D object from different sides as shown in figure below:



Note: parallel projection cannot give a realistic representation for 3-D object.

Parallel Orthographic Projection

The simplest of the parallel projections is the orthographic projection, commonly used for engineering drawings. They accurately show correct size and shape of a single plane face of an object. Orthographic projections are projections onto one of the coordinate planes x=0, y=0, or z=0. The matrix for projection onto the z=0 plane is:

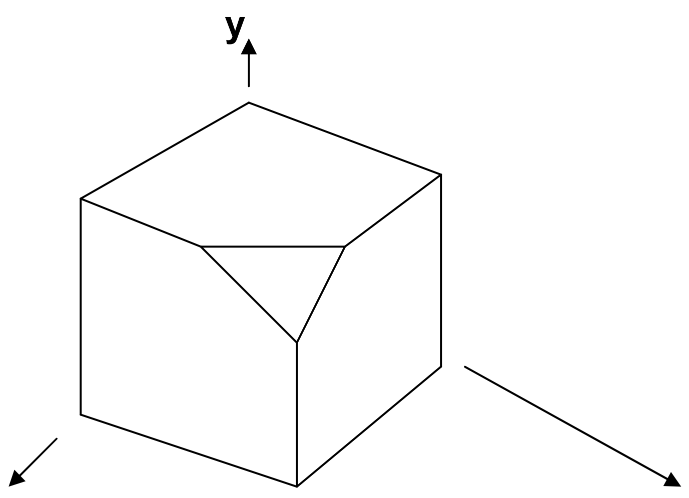
$$[Pz] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

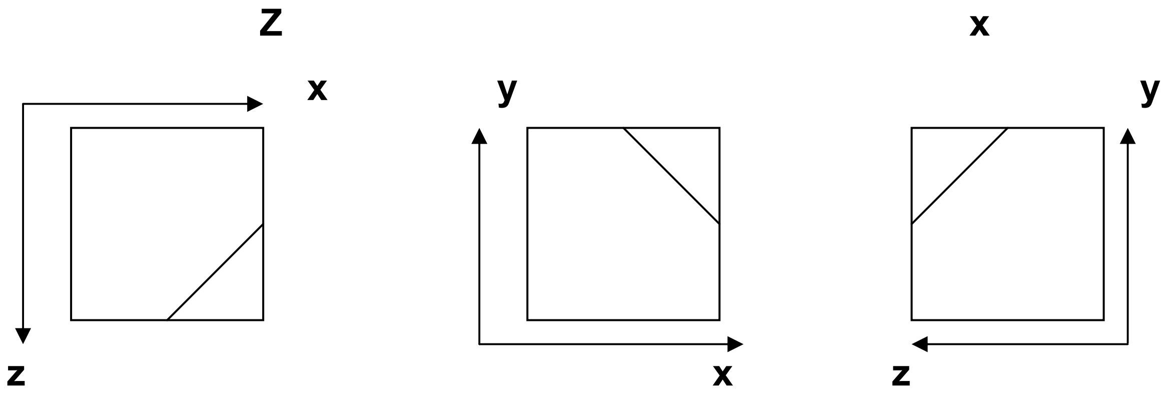
Similarly, the matrices for projection onto the x=0 and y=0 planes are:

$$[Px] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[Py] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Orthographic projections of the object in figure below:

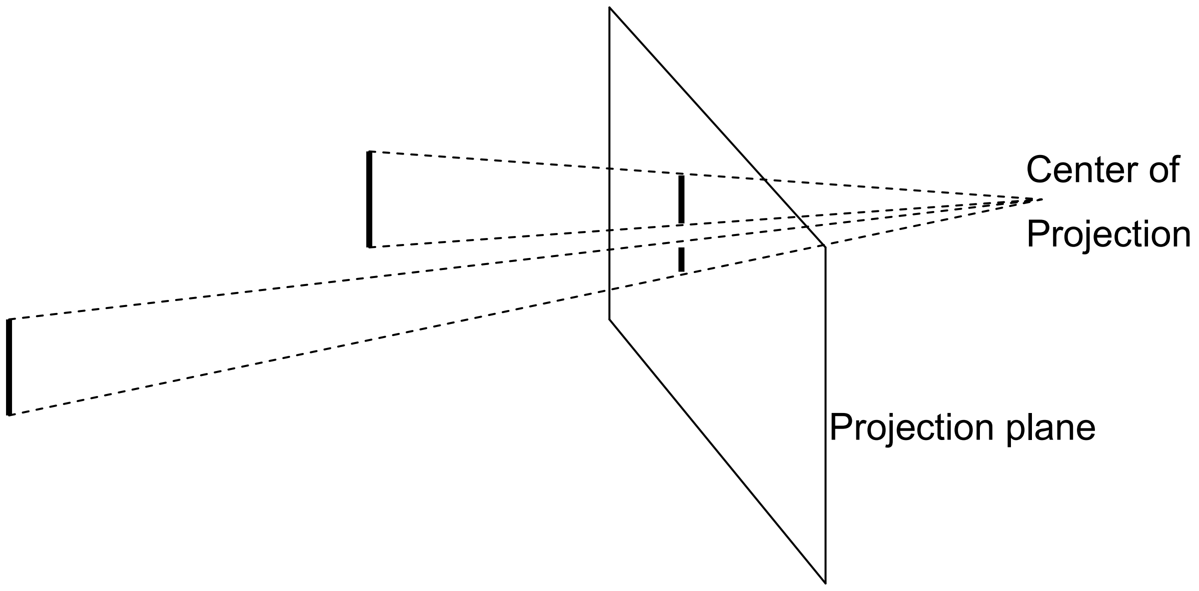




A single orthographic projection does not provide sufficient information to visually and practically reconstruct the shape of an object. Consequently, multiple orthographic projections are necessary.

2- Perspective Projection:

Perspective projection gives a realistic representation for 3-D object, but without real dimensions values.



In above figure, there are two equal length lines, but with different perspective projections. The closer line to the projection plane is the larger projection length.

To calculate the perspective projection lengths, the following equations are used:

$$Xp = X(\frac{d}{d+Z})$$

$$Yp = Y(\frac{d}{d+Z})$$

$$Zp = 0$$

Where x,y,and z: represent coordinates of a point in 3-D world P(x,y,z).

Xp,Yp, and Zp: represent coordinates of a point in projection plane (Xp,Yp,0).

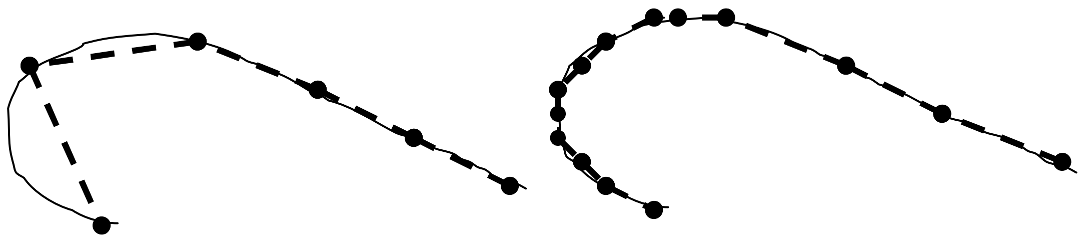
d: the distance between 3-D coordinates of a point, and projection plane.

<u>H.W.:</u>

Find Perspective Projection of cubic object :((0,0,0), (5,0,0), (5,5,0), (0,5,0), (0,5,5), (0,0,5), (5,0,5), (5,5,5)). Note that d=5 units?

Curves

A curve may be represented as a collection of points. Provided the points are properly spaced, connection of the points by short straight line segments yields an adequate visual representation of the curve. The following figure shows two alternate point representations of the same plane curve.


(a) (b)

Points along the curve in above figure (a) are equally spaced along the curve length. Notice that connection of the points by short straight line segments yields a poor representation of the curve. The representation is especially poor where the radius of curvature is small. Increasing the point density in these regions, as shown in figure above (b) improves the representation.

Non Parametric Curves:

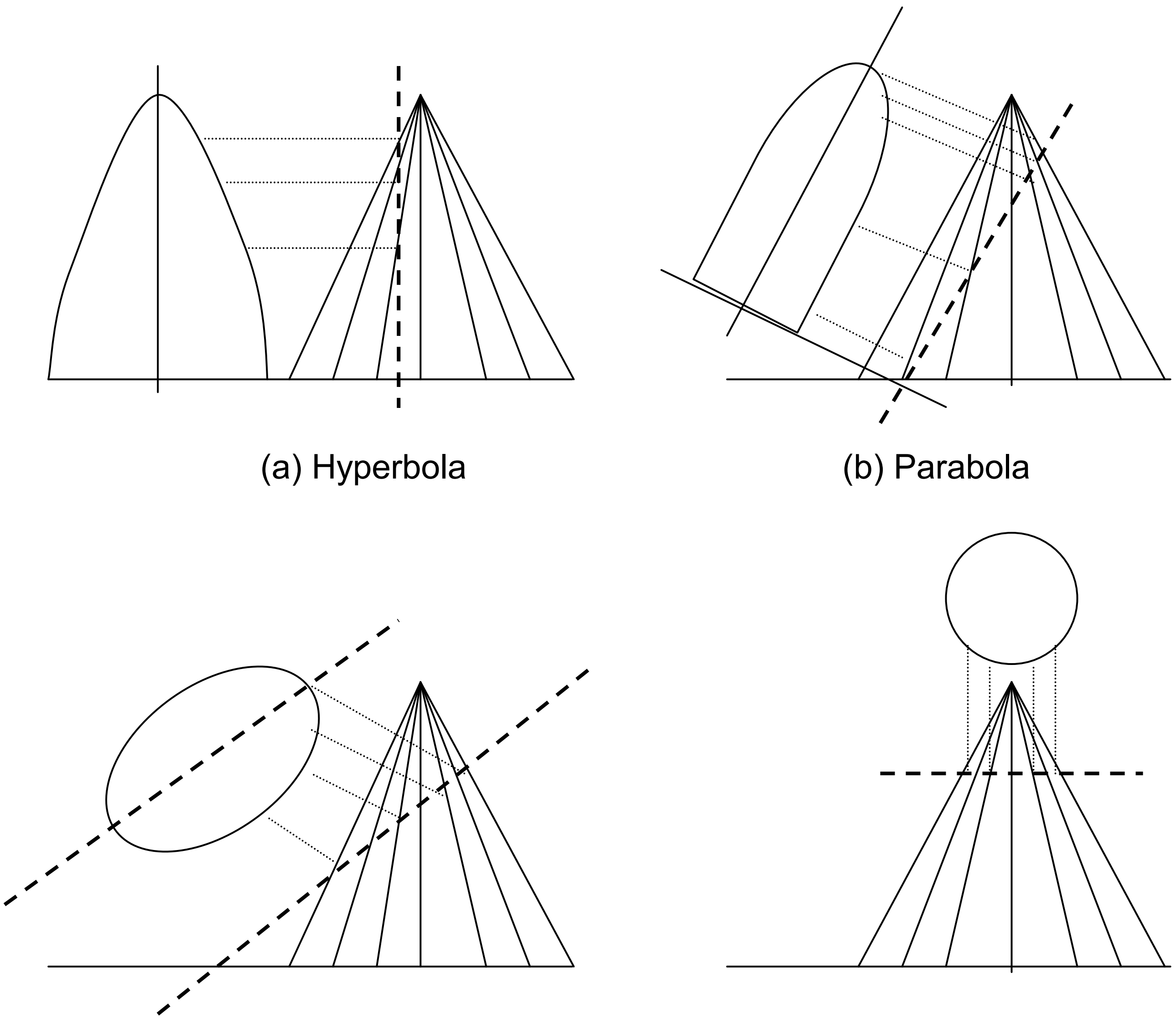
Mathematically, either a parametric or a nonparametric form is used to represent a curve. A nonparametric representation is either explicit or implicit. For a plane curve, an explicit, nonparametric form is given by: $y = f(x)$. An example is the equation of a straight line, y=mx+b. In this form, for each x-value only one y-value is obtained. Consequently, closed or multiple value curves, e.g., a circle, cannot be represented explicitly.

Implicit representation of the form: $f(x,y)=0$, do not have this limitation.

A general second-degree implicit equation written as:

$$ax^2+2bxy+cy^2+2dx+2ey+f=0$$

Provides a variety of two dimensional curve forms called conic sections. The three types of conic sections are the parabola, the hyperbola and the ellipse. As shown in following figure:



(a) Hyperbola (b) Parabola

(c) Ellipse (d) Circle

Circle is a special case of an ellipse. By defining the constants a,b,c,d,e, and f , several different types of conic sections are

produced. A simpler curve is defined by first setting a=1.0, b=0, and c=1.0, then the form of the curve is :

$$x^2 + y^2 + 2dx + 2ey + f = 0$$

A straight line is obtained by setting a=b=c=0. The equation is then: $dx + ey + f = 0$**,** **or** $y = -(\frac{d}{e})x - \frac{f}{e} = mx + b$

Where, as usual, m is the slope of the line and b is its y intercept.

Parametric Curves:

In parametric form each coordinate of a point on a curve is represented as a function of a single parameter. The position vector of a point on the curve is fixed by the value of the parameter, the Cartesian coordinates of a point on the curve are:

x=x(t) , y=y(t)

The position vector of a point on the curve is then:

P(t)=[x(t) y(t)]

The parametric form is suitable for representing closed and multiple valued curves. The curve end points and length are fixed by the parameter range. Often it is convenient to normalize the parameter range for the curve of interest to 0 $\le$ t $\le$1.

The simplest parametric curve representation is for a straight line. For two position vectors P1 and P2, a parametric representation of the straight line segment between them is:

P(t)=P1+(P2-P1)t   0 $\le$ t $\le$1

Since P(t) is a position vector, each of the components of P(t) has a parametric representation x(t) and y(t) between P1 and P2, i.e.,

$$x(t)=x1+(x2-x1)t \qquad 0 \le t \le 1$$
$$y(t)=y1+(y2-y1)t$$

Example:

For the position vector P1[1 2] and P2[4 3] determine the parametric representation of the line segment between them. Also determine the slope and tangent vector of the line segment?

A parametric representation is:

$$P(t)=P1+(P2-P1)t=[\,1\;\;2\,] + ([4\;\;3] - [1\;\;2])\,t \qquad 0 \le t \le 1$$
$$P(t)= [1\;\;2]+[3\;\;1]\,t \qquad 0 \le t \le 1$$

Parametric representation of the x and y components are:

$$x(t)=x1+(x2-x1)t = 1+3t$$
$$y(t)=y1+(y2-y1)t=2+t$$

The tangent vector is obtained by differentiating P(t). Specifically:

$$P'(t) = [x'(t)y'(t)] = [\,3\;\;1\,]$$

$$\text{or} \qquad \vec{v}_t = 3i + j$$

Where $\vec{v}_t$ is the tangent vector and I,j are unit vectors in the x,y directions, respectively.

The slope of the line segment is:

$$\frac{dy}{dx} = \frac{dy/dt}{dx/dt} = \frac{y'(t)}{x'(t)} = \frac{1}{3}$$

Representation of Space Curves:

Three-dimensional space curves are represented non parametrically or parametrically. An explicit non parametric representation is:

$$X=x \quad ; \quad y=f(x) \quad ; \quad z=g(x)$$

Alternatively, a nonparametric implicit representation of the curve as the intersection of two surfaces is given by:

$$f(x,y,z)=0 \quad ; \quad g(x,y,z)=0$$

Example:

Determine the curve described by the intersection of the two second degree surfaces by:

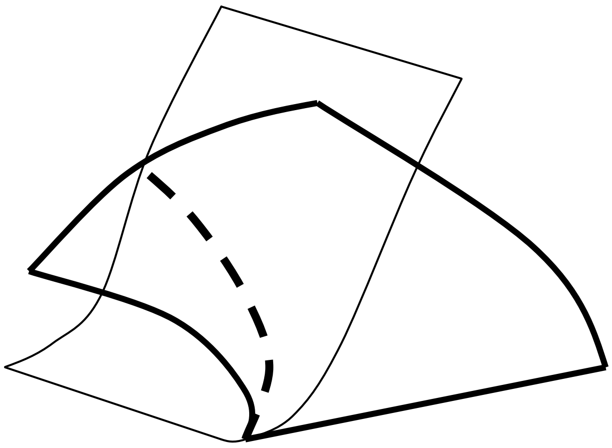
$$f(x,y,z)=y-z^2=0$$

$$g(x,y,z)=zx-y^2=0$$

Provided that $z\neq 0$, x and y can be expressed in terms of z to obtain the explicit form of the intersection curve

$$Y=z^2 \quad ; \quad x=\frac{y^2}{z}=z^3$$

Notice that the intersection of two second degree surfaces yields a third degree curve. The surfaces and intersection curve are shown in figure below:



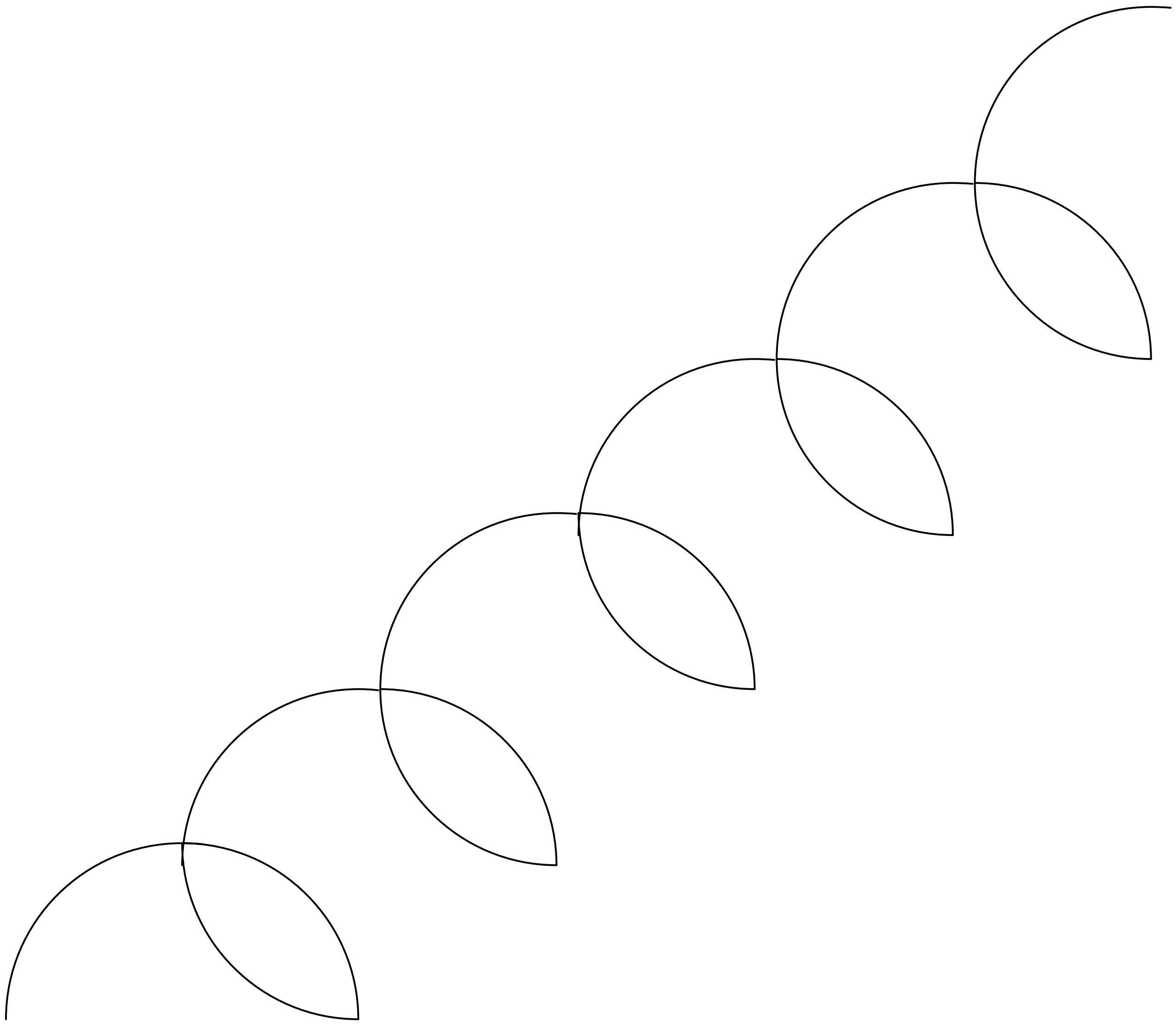
In general, parametric space curve is expressed as:

$$x=t \; ; \; y=f(t) \; ; \; z=g(t)$$

Where the parameter t varies over a given range ($t1 \le t \le t2$). One of the useful parametric space curves is the circular helix as shown in figure below. The parametric equations are given by:

x=r cos(t); y=r sin(t); z=bt

For r and b $\neq 0$ and (-$\infty \langle t \langle \infty$).



Cubic Splines:

Cubic polynomials: $f(u) = Ax^3 + Bx^2 + Cx + D$

Cubic is minimum degree that allows both position and first derivatives at endpoints to be independently controlled.

Higher degrees are possible too (and allow better control over higher orders of continuity). But higher order polynomials are also harder to control; they tend to want to oscillate.

We typically use cubic polynomials in computer graphics.

Specifying position and slope of endpoints:

This one is called a Hermite Curve.

left endpoint (u=0):

position = p0

slope = s0

right endpoint (u=1):

position = p1

slope = s1

Therefore:

$$\left.\begin{aligned} A(0)^3 + B(0)^2 + C(0) + D &= p0 \\ A(1)^3 + B(1)^2 + C(1) + D &= p1 \end{aligned}\right\} \leftarrow \text{Equation}$$

$$\left.\begin{aligned} 3A(0)^2 + 2B(0) + C &= s0 \\ 3A(1)^2 + 2B(1) + C &= s1 \end{aligned}\right\} \leftarrow \text{Derivative of equation}$$

It is possible to use a different basis for specifying the polynomial.

Basis #1: A, B, C, D

Basis #2: p0, p1, s0, s1

Tangent line specifies slope of the curve.

Approximating control points:

By specifying four positions, we have what are called "interpolating" control points.

What we're going to do is to use a basis that consists of four different cubic curves. The weight for each one is specified by one of the control points.

The four curves are the Berstein polynomials:

$b0(u) = (1-u)^3$

$b1(u) = 3u(1-u)^2$

$b2(u) = 3u^2(1-u)$

$b3(u) = u^3$

Note that $b0(u) + b1(u) + b2(u) + b3(u) = 1$

A curve constructed using the Bernstein polynomials is called a Bezier curve.

Uniform B-splines:

Four control vertices, just like before. But basis functions are somewhat different. We are going to use a single 4-unit-wide cubic curve, but then shift it for each control point.

The four shifted versions of the curve (truncated to [0,1] range) are:

$B0(u) = 1/6\ u^3$

$B1(u) = 1/6\ (-3u^3 + 3u^2 + 3u + 1)$

$B2(u) = 1/6\ (3u^3 - 6u^2 + 4)$

$B3(u) = 1/6\ (1-u)^3$

defined for u=[0,1]

Animation

Animation is based on a principle of human vision. If you view related still images in quick succession, you perceive them as continuous motion. Each individual image is referred to as a frame.

The main difficulty in creating animation has been the effort required by the animator to produce large number of frames. Creating images by hand is a big job. That's where the technique of keyframing comes in. Most of the frames in animation are routine and incremental changes from the previous frame directed toward some goal. Traditional animation studios realized they could increase the productivity of their artists by having them draw only the important frames, called keyframes. The in-between frames were called tweens. Once all of the keyframes and tweens were drawn, the images had to be inked or rendered to produce the final images. Even today, production of traditional animation usually requires hundreds of artists to generate the thousands of images needed.

Design of Animation Sequences:

In general, an animation sequence is designed with the following steps:

1- **Storyboard layout.**

2- **Object definitions.**

3- **Key frame specifications.**

4- **Generation of in between frames.**

For frame by frame animation, each frame of the scene is separately generated and stored. Later, the frames can be recorded on film or they can be consecutively displayed in real time playback mode.
Storyboard: is an outline of the action. It defines the motion sequence as a set of basic events that are take place.
Object definition: can be defined in terms of basic shapes, such as polygons or splines. In addition, the associated movements for each object are specified along with shape.
Key frame: is detailed drawing of the scene at a certain time in the animation sequence. Within each key frame, each object is positioned according to the time for that frame.
In betweens: are the intermediate frames between the key frames. The number of in betweens needed is determined by the media to be used to display the animation. Film requires 24 frames per second, and graphics terminals are refreshed at the rate of the 30 to 60 frames per second. Typically, Time intervals for the motion are setup so that there are from three to five in betweens for each pair of key frames. Depending on the speed specified for the motion, some key frames can be duplicated. For 1-minute film sequence with no duplication, we would need 1440 frames.

Key frame systems:

We generate each set of in betweens from the specification of two (or more) key frames. Motion paths can be given with a kinematic (it is branch of mechanics that describes the motions of bodies without considering the forces required to produce and maintain the motion) as a set of spline curves, or the motion

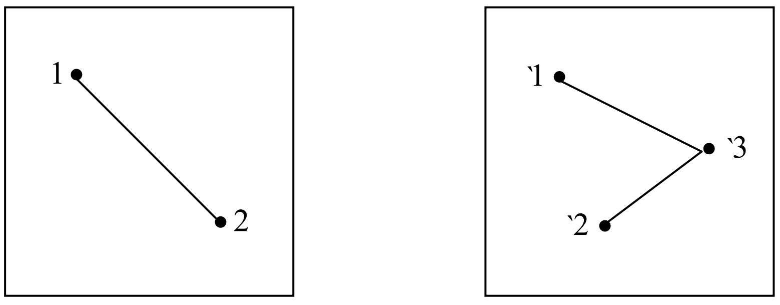
can be physically based by specifying the forces on objects to be animated (dynamics or kinetics).

For example scenes, we can separate the frames into individual components or objects. Given the animation paths, we can interpolate the positions of individual objects between any two times. With complex object transformations, the shapes of objects may change over time. If for example the shapes are described with polygon meshes, then the number of edges per polygon can change from one frame to the next. Thus, the total number of line segments can be different in different frames.

Morphing:

Transformation of object shapes from one form to another is called morphing. Morphing methods can be applied to any motion or transition involving a change in shape.

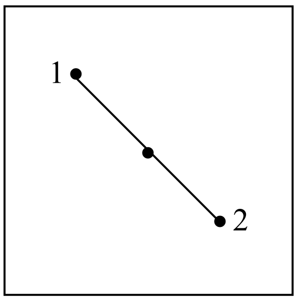
Given two key frames for an object transformation, we first adjust the object specification in one of the frames so that the number of polygon edges (or the number of vertices) is the same for the two frames. This preprocessing step is illustrated in following figure.

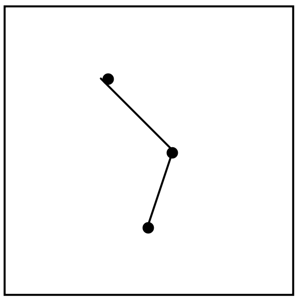


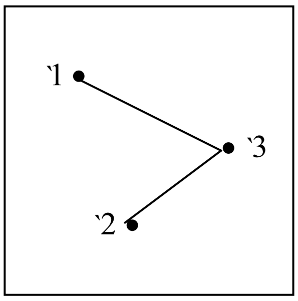
Key frame K Key frame K+1

A straight line segment in key frame K is transformed into two line segments in key frame K+1. Since key frame K has an extra vertex, we add a vertex between vertices 1 and 2 in key frame K

to balance the number of vertices (and edges) in the two key frames. To generate the in betweens, we transition the added vertex in key frame K into vertex `3 along the straight line path shown in following figure.

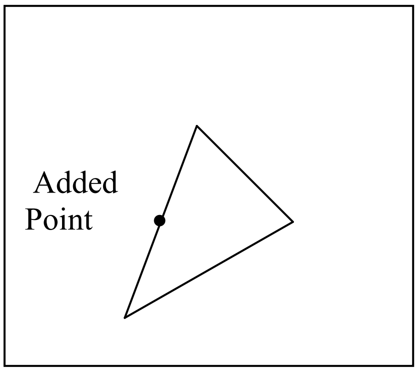


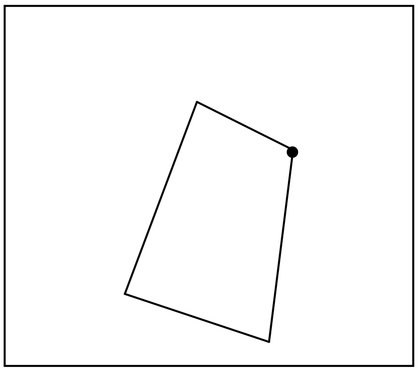




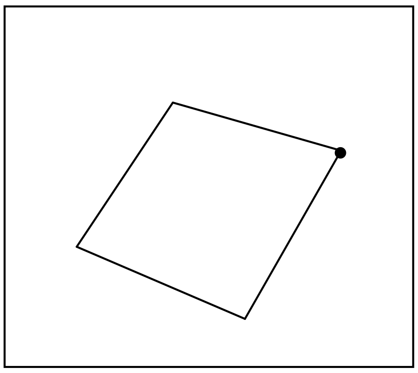
key frame K halfway key frame key frame K+1

An example of a triangle linearly expanding into a quadrilateral is given in following figure.





key frame K halfway key frame



key frame K+1

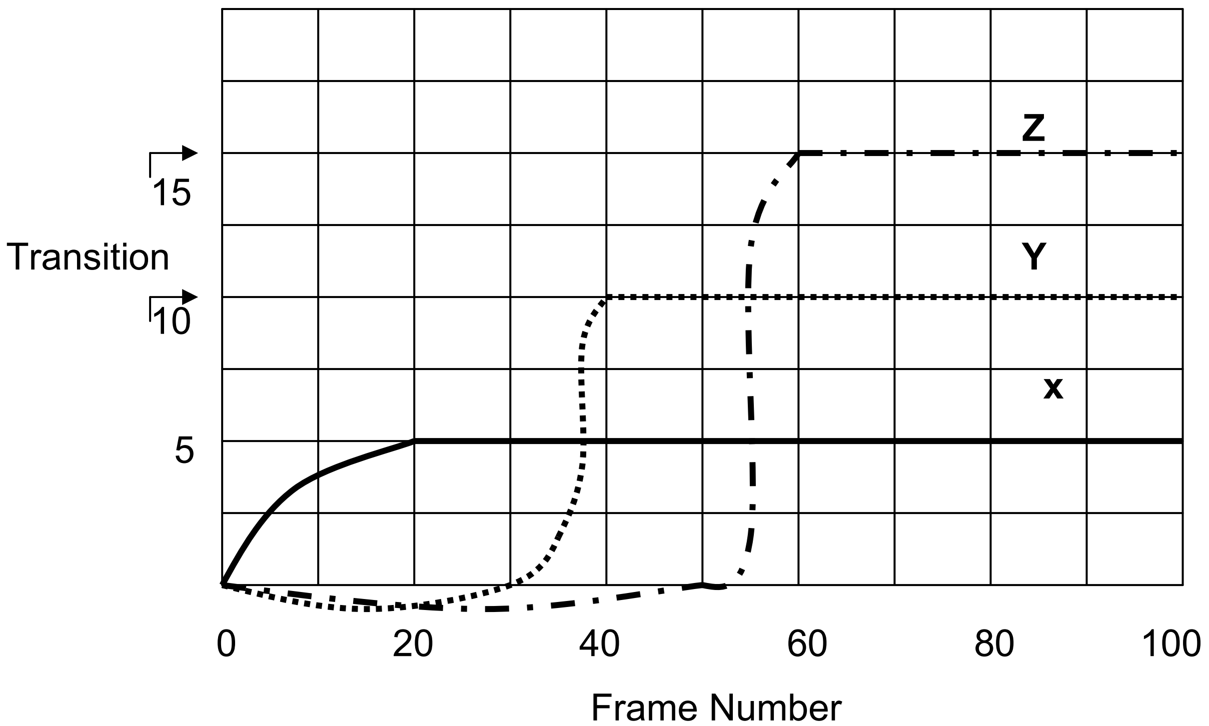
Key Framing Animation:

This is the animation used in traditional animation. The animator starts by specifying the positions and orientations at various key points in time. In betweeners (usually done by computer) generate the image in between the key frames. Splines are used to interpolate the positions on objects between key frames.

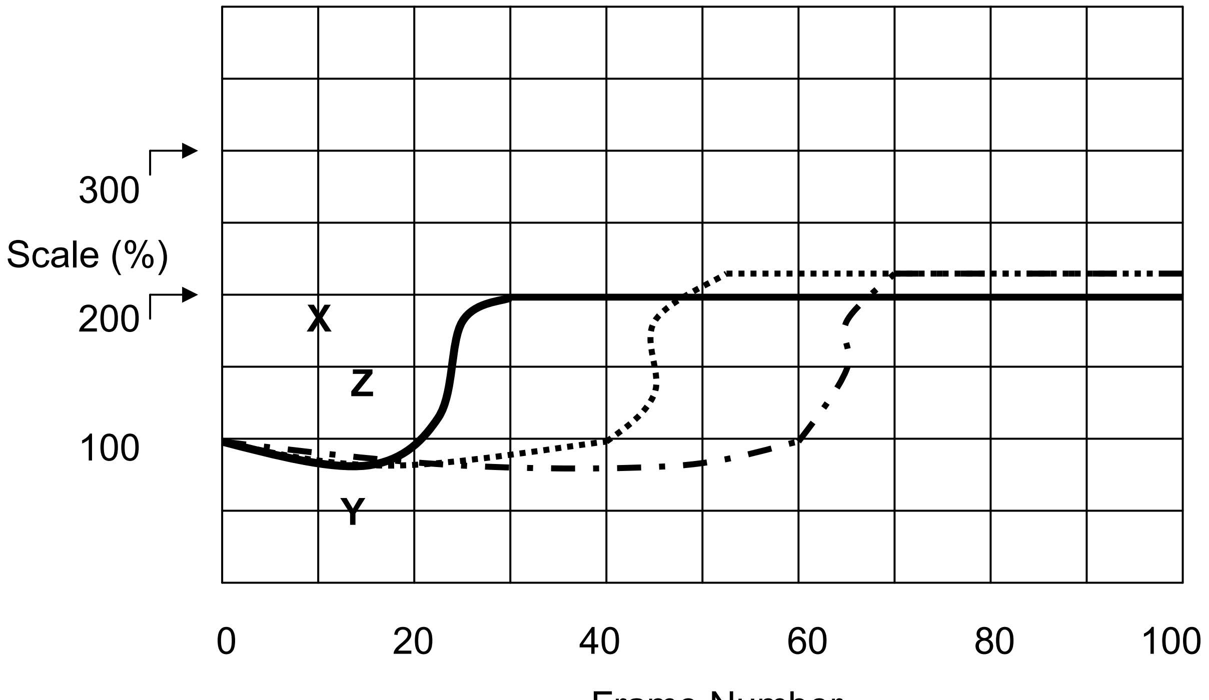
Representation of Key frame Table in Spline Curves:

We can represent a key frame table shown in following table as shown in following figures.

Frame no.	Transformation	Direction	Amount
0-20	Move	X	5
21-30	Scale	X	200%
31-40	Move	Y	10
41-50	Scale	Y	200%
51-60	Move	Z	15
61-70	Scale	Z	200%



move animation



Scale animation

Key Frame Representation