Academic Year: 2023-24

SUBJECT: COMPUTER GRAPHICS LAB (CODE: 218557)

List of Assignments

SN	TITLE
1	Install and explore the OpenGL
2	Implement DDA and Bresenham line drawing algorithm to draw: i) Simple Line ii) Dotted Line iii) Dashed Line iv) Solid line using mouse interface Divide the screen in four quadrants with center as (0, 0). The line should work for all the slopes positive as well as negative.
3	Implement Bresenham circle drawing algorithm to draw any object. The object should be displayed in all the quadrants with respect to center and radius
4	Implement the following polygon filling methods: i) Flood fill / Seed fill ii) Boundary fill using mouse click, keyboard interface and menu driven programming
5	Implement Cohen Sutherland polygon clipping method to clip the polygon with respect the viewport and window. Use mouse click, keyboard interface
6	Implement following 2D transformations on the object with respect to axis : i) Scaling ii) Rotation about arbitrary point iii) Reflection
7	Generate fractal patterns using i) Bezier ii) Koch Curve
8	Implement animation principles for any object

Assignment No: 01

Title: Install and explore the OpenGL

Problem Statement: Install and explore the OpenGL.

Objectives: Understand how to install OpenGL in ubuntu.

Outcomes: 1. Students will be able to learn Install OpenGL.
2. Students will be able to understand different OpenGL functions

Theory:

The OpenGL specification specifies exactly what the result/output of each function should be and how it should perform. It is then up to the developers *implementing* this specification to come up with a solution of how this function should operate. Since the OpenGL specification does not give us implementation details, the actual developed versions of OpenGL are allowed to have different implementations, as long as their results comply with the specification

Steps to Follow:

1) Open a terminal and enter the following commands to install the necessary libraries for OpenGL development:

following commands:

$ sudo apt-get update

$ sudo apt-get install freeglut3

$ sudo apt-get install freeglut3-dev

$ sudo apt-get install binutils-gold

$ sudo apt-get install g++ cmake

$ sudo apt-get install libglew-dev

$ sudo apt-get install g++

$ sudo apt-get install mesa-common-dev

$ sudo apt-get install build-essential

$ sudo apt-get install libglew1.5-dev libglm-dev

2 **Get information about the OpenGL and GLX implementations running or not?**

glxinfo | grep -i OpenGL

Your will get following information: (It Means OpenGL Installed)

OpenGL vendor string: NVIDIA Corporation

OpenGL renderer string: GeForce 8800

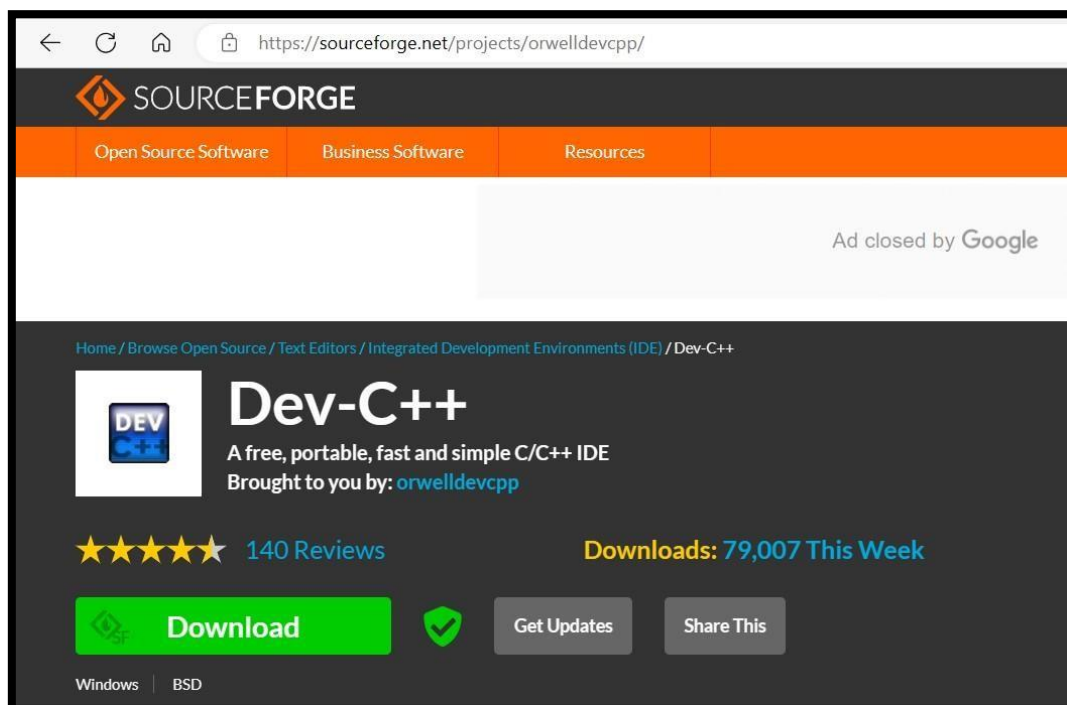GT/PCIe/SSE2OpenGL version string: 2.1.2

NVIDIA 310.44

OpenGL shading language version string: 1.20 NVIDIA via

Cg compilerOpenGL extensions:
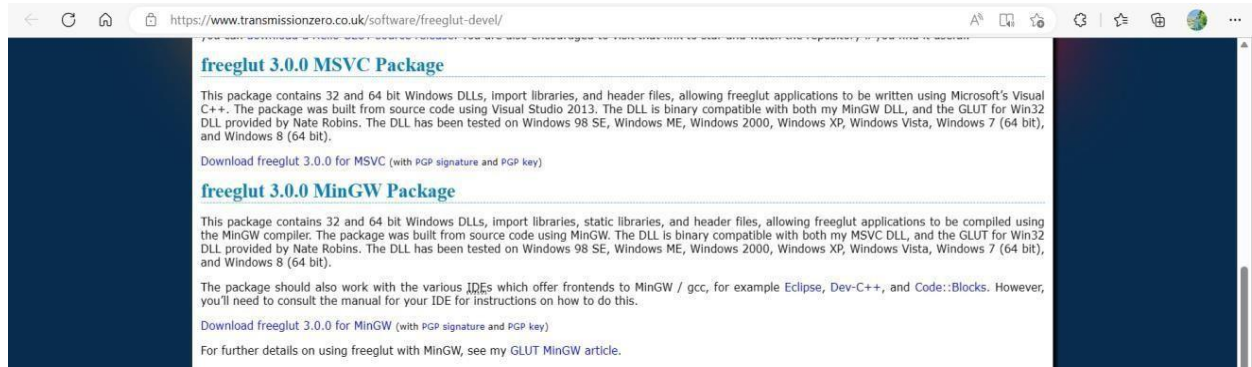
- **Install and explore the OpenGL =**
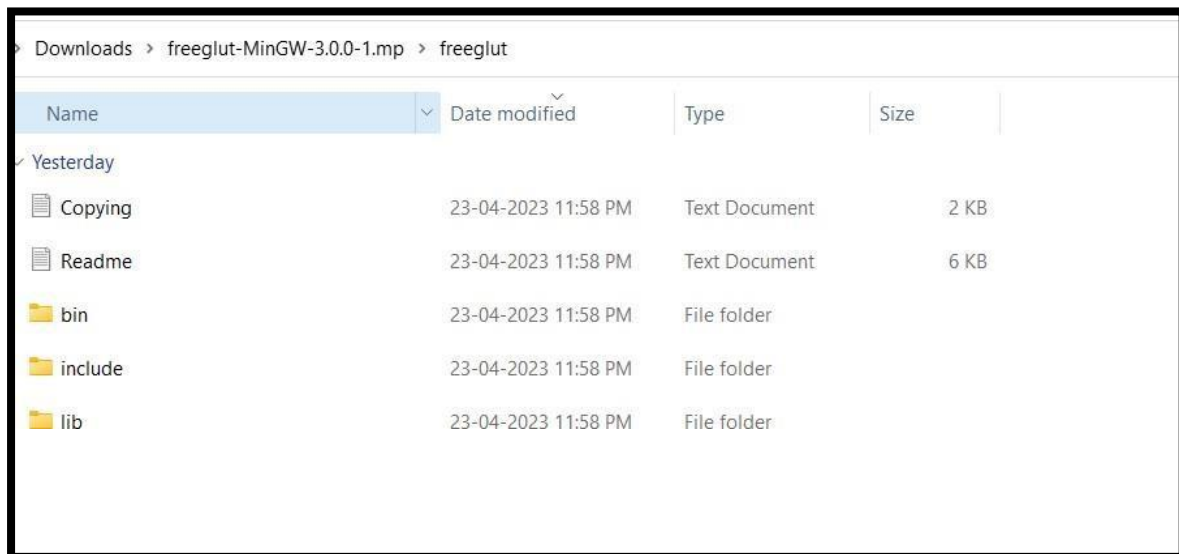
DOWNLOAD DEV C++ APPLICATION AND INSTALL IT.
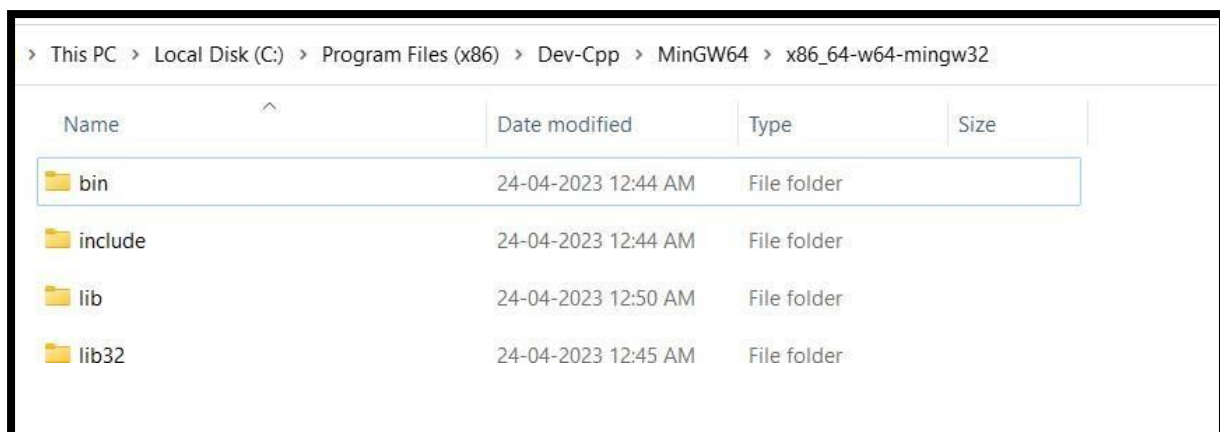
INSTALL OPENGL FREEGLUT3.0.0 FILE AND EXTRACT IT.

OPEN FREE GLUT INSTALLED FILE DESTINTION -

ALSO OPEN FOLLOWING PATH-
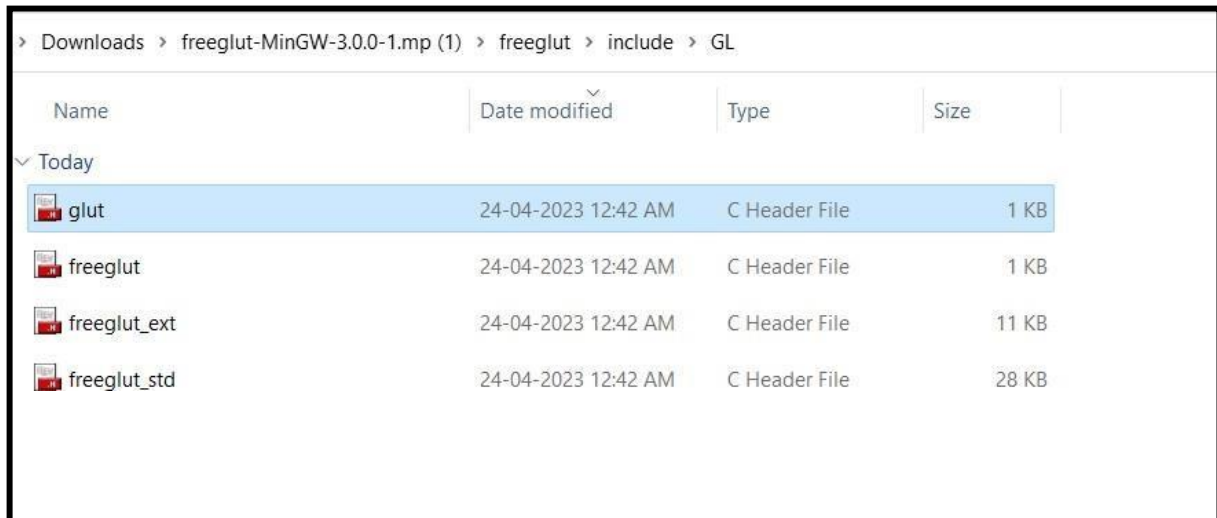
IN freeglut OPEN INCLUDE FOLDER COPY ALL FILES AND PASTE IN ABOVE OPEN
LOCATION ININCLUDE FOLDER

Copy above files and paste in following location
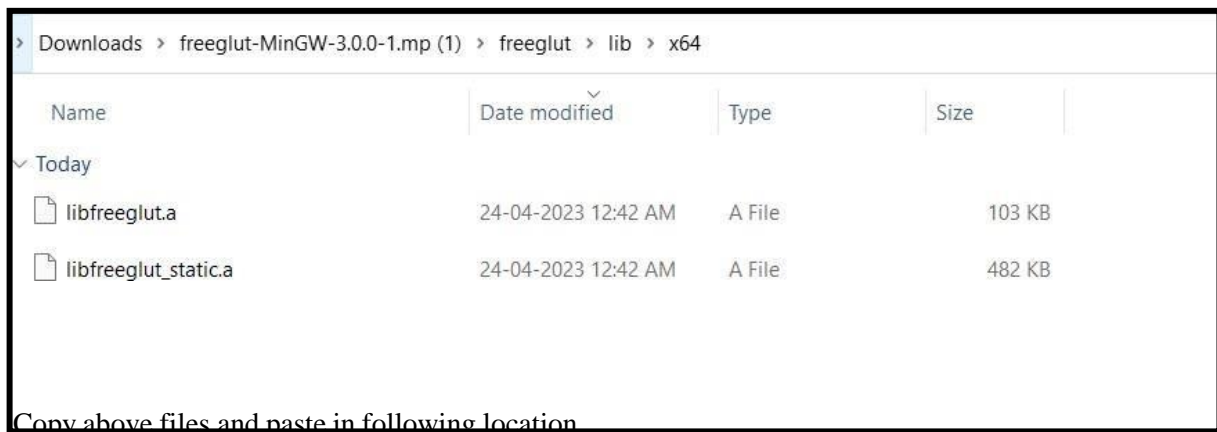
C:\Program Files (x86)\Dev-Cpp\MinGW64\x86_64-w64-mingw32\include

IN freeglut OPEN LIB FOLDER COPY ALL FILES

Copy above files and paste in following location

C:\Program   Files   (x86)\Dev-Cpp\MinGW64\x86_64-w64-mingw32\lib

IN freeglut OPEN BIN > X64 FOLDER COPY ALL FILES

Copy above files and paste in following location

C:\Windows\System32
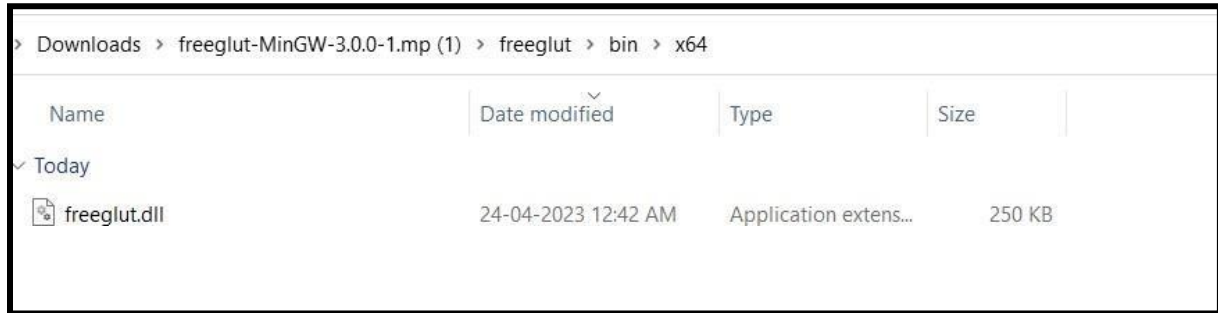
NOW INSTALLATION PROCESS IS COMPLETE

WE ARE READY TO RUN COMPUTER GRAPHICS PROGRAMS

STEP 7-

NOW OPEN DEV C++ AND CREATE NEW PROJECT

OPEN PROJECT OPTION AND COPY FOLLOWING COMMANDS IN LINKER OPTION
CLICK ON OK .

NOW YOU ARE READY TO WRITE PROGRAM AND RUN IT

HERE WE SUCCESSFULLY RUN THE PROGRAM

Conclusion:

Assignment No: 02

Title: DDA and Bresenham line drawing algorithm

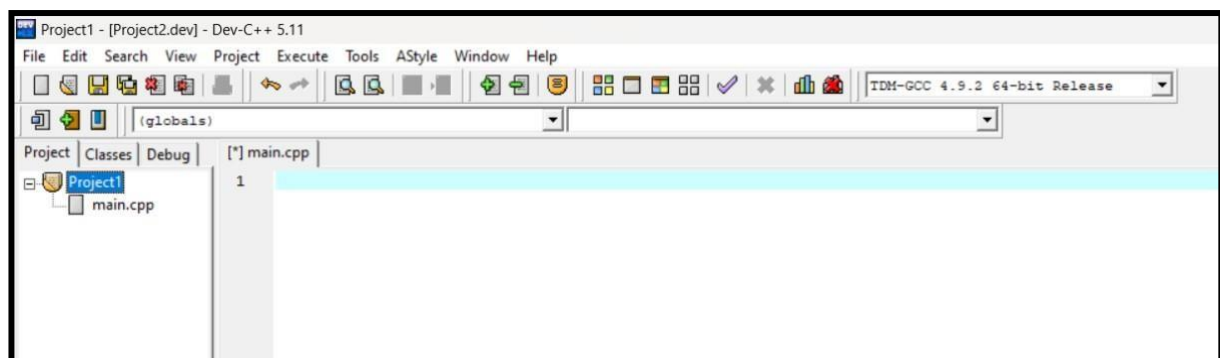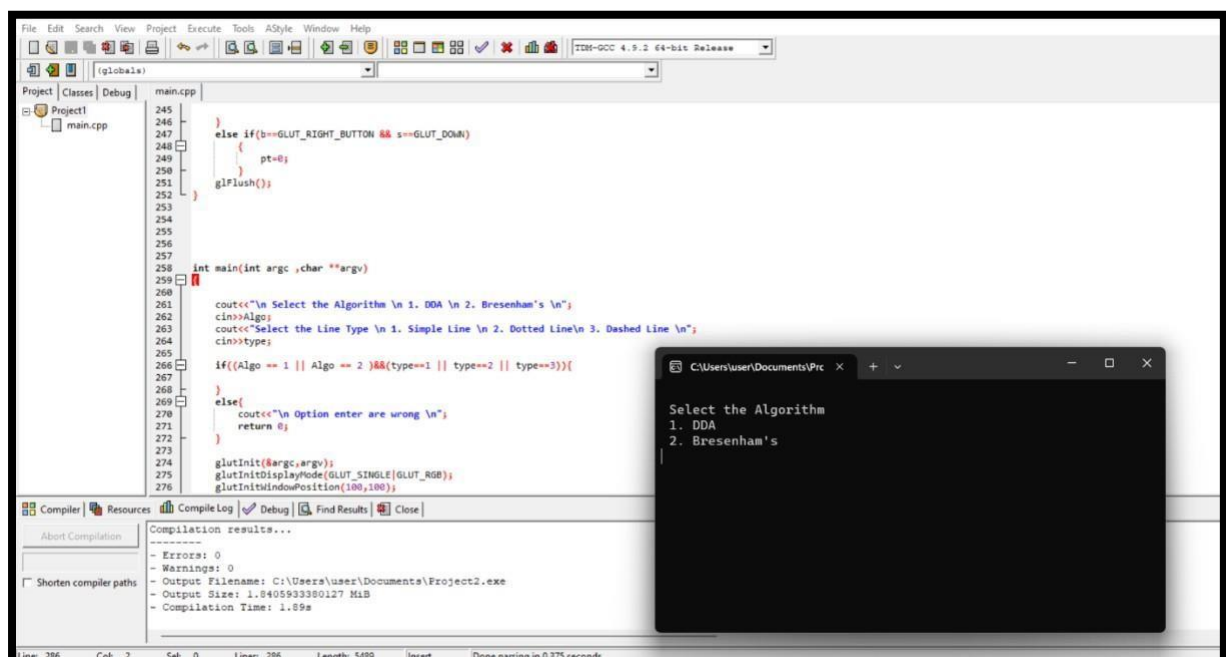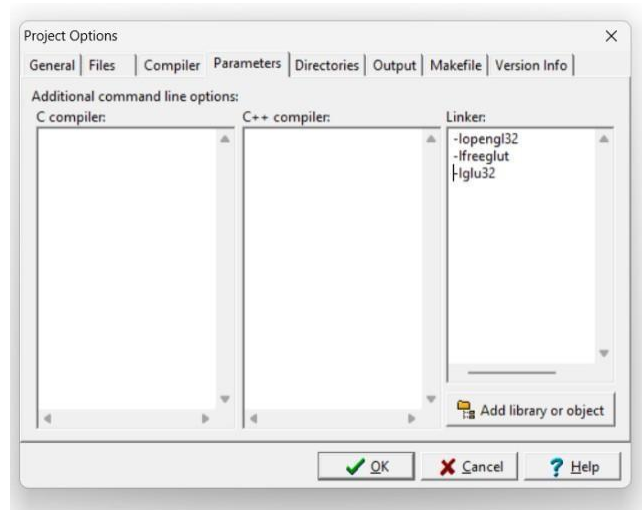**Problem Statement:** Implement DDA and Bresenham line drawing algorithm to draw: i) Simple Line ii) Dotted Line iii) Dashed Line iv) Solid line; using mouse interface Divide the screen in four quadrants with center as (0, 0). The line should work for all the slopes positive as well as negative.

Objectives: 1. Implementation of DDA and Bresenham's line drawing algorithm.
           2. To draw: i) Simple Line ii) Dotted Line iii) Dashed Line iv) Solid line.

Outcomes: 1. Students will be able to draw a line using DDA and Bresenham's algorithm.
           2. Students will be able to understand object-oriented programming concepts like inheritanceand function overloading.

Theory:
Introduction:

A line drawing algorithm is a graphical algorithm for approximating a line segment on discrete graphical media. On discrete media, such as pixel-based displays and printers, line drawing requires such an approximation.

Line Equation
The Cartesian slop-intercept equation for a straight line is

$y = mx + b$ ...... (1)
with
m->slope
b- >y intercept

The 2 end points of a line segment are specified at a position (x1, y1)
Determine the values for the slope m and y intercept b with the following calculation. Here, slopem:
$m = (y2 - y1) / (x2 - x1)$
$m = Dy / Dx$ .... (2)

y intercept b
$b = y1 - mx1$ .... (3)

- Algorithms for displaying straight line based on this equation
- y interval Dy from the equation

$m = Dy / Dx$
$Dy = m . Dx$ ...... (4)

Similarly, x interval Dx from the

equation $m = Dy / Dx$
$Dx = Dy / m$ ...... (5)

Line drawing algorithms:
• Digital Differential Analyzer
• Bresenham's Line Analyzer

Digital Differential Analyzer:

- The digital differential analyzer (DDA) is a scan conversion line algorithm based on calculation either Dy or Dx.
- The line at unit intervals is one coordinate and determine corresponding integer values nearest line for the other coordinate.
- Consider first a line with positive slope.

Step: 1
- If the slope is less than or equal to 1, the unit x intervals Dx=1 and compute each successive y values.

$Dx=1$
$m = Dy / Dx$
$m = (y2-y1) / 1$
$m = (yk+1 – yk) / 1$
$yk+1 = yk + m$ ……… (6)

- subscript k takes integer values starting from 1, for the first point and increment by 1 untilthe final end point is reached.
- m->any real numbers between 0 and 1
- Calculate y values must be rounded to the nearest integer

Step: 2
If the slope is greater than 1, the roles of x any y at the unit y intervals Dy=1 and compute each successive y values.

$Dy=1$
$m= Dy / Dx$
$m= 1/(x2-x1)$
$m = 1 / (xk+1 – xk)$
$xk+1 = xk + (1 / m)$ …… (7)

- Equation 6 and Equation 7 that the lines are to be processed from left end point to the rightend point.

Step: 3

If the processing is reversed, the starting point at the right
$Dx=-1$
$m= Dy / Dx$
$m = (y2 – y1) / -1$
$yk+1 = yk – m$.......... (8)

Iintervals Dy=1 and compute each successive y values.

Step: 4

Here, $Dy=-1$
$m= Dy / Dx$
$m = -1 / (x2 – x1)$
$m = -1 / (xk+1 – xk)$
$xk+1 = xk + (1 / m)$.............. (9)
Equation 6 and Equation 9 used to calculate pixel position along a line with –ve slope.

Bresenham's Line Analyzer:

Theory: -

- The basic principle of Bresenham's line algorithm is to <u>select</u> the optimum raster locations to represent a straight line.
- To accomplish this, the algorithm always increments either x or y by one unit depending on the slope of line.
- The increment in the other variable is determined by <u>examining</u> the distance between the actual line location & the nearest pixel. This distance is called **decision variable** or **error** or **decision parameter.**

Algorithm:

1. DDA line drawing Algorithm: -

1. Start.
2. Declare variables x, y, x1, y1, x2, y2, k, dx,dy,s,xi,yi
3. Initialize the graphic mode.
4. Input the two-line end-points and store the left end-points in (x1, y1).
5. Load (x1, y1) into the frame buffer;thatis,plot the first point.put x=x1,y=y1.6.Calculate dx=x2-x1 and dy=y2-y1.
7. If abs(dx) > abs(dy), do s=abs(dx).
8. Otherwise s= abs(dy).
9. Then xi=dx/s and yi=dy/s.
10. Start from k=0 and continuing till k<s,the points will be
    - i. x=x+xi.
    - ii. y=y+yi.
11. Place pixels using putpixel at points (x,y) in specified colour.
12. Close Graph.
13. Stop.

2. Bresenham's line drawing Algorithm: -

1. Start.
2. Declare variables x, y, x1, y1, x2, y2, p,dx,dy .
3. Initialize the graphic mode.
4. Input the two-line end-points and store the left end-points in (x1, y1).
5. Load (x1, y1) into the frame buffer; that is, plot the first point put x=x1, y=y1.
6. Calculate dx=x2-x1 and dy=y2-y1, and obtain the initial value of decision parameter p as:p = (2dy-dx).
7. Starting from first point (x,y) perform the following test:
8. Repeat step 9 while(x<=x2).
9. If p<0, next point is (x+1, y) and p=(p+2dy).
10. Otherwise, the next point to plot is (x+1, y+1) and p=(p+2dy-2dx).
11. Place pixels using put pixel at points (x,y) in specified color.
12. Close Graph.
13. Stop.

Conclusion:.

Assignment No: 03

Title: Bresenham circle drawing algorithm

Problem Statement: Implement Bresenham circle drawing algorithm to draw any object. The object shouldbe displayed in all the quadrants with respect to center and radius

Objectives: 1. Implementation of Bresenham's circle drawing algorithm.
 2. Implementation of object-oriented programming concepts like inheritance.

Outcomes: 1. Students will be able to draw a circle using Bresenham's circle drawing algorithm.
 2. Students will be able to understand object-oriented programming concepts like inheritance.

Theory:
Circle:

A circle is a set of points that are placed at a given distance 'r' from center (x_c, yc).
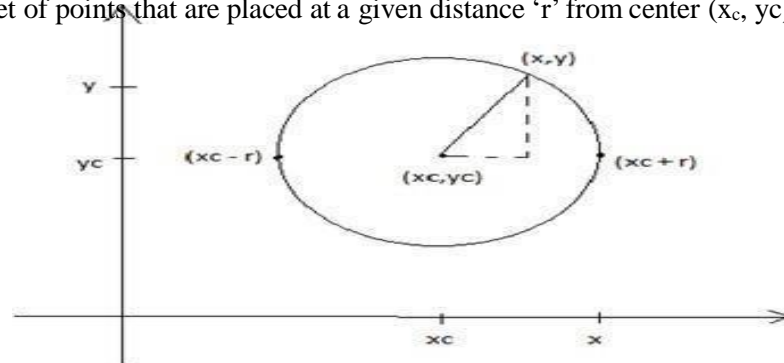
Fig. : Circle

- From the fig., the distance relationship is expressed by Pythagoras theorem as,

$(x_c, yc)^2 + (y-y_c)^2 = r^2$

8-way symmetry of circle:
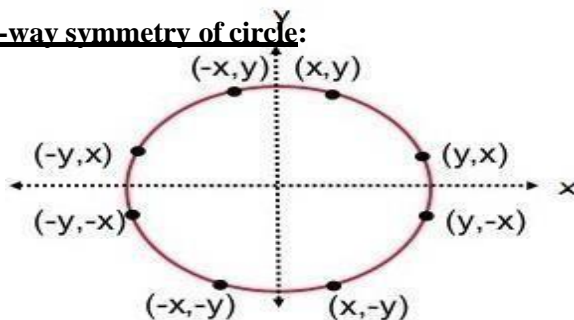
Fig.: 8-way Symmetry of circle

- We can divide the circle in 4 quadrants or ion 8 octants also. Here we find only one octant's co-ordinates and then we will just replicate that one octant's co-ordinates to rest seven octants.
- Suppose we have calculated a point (x,y) then from this (x,y) we will plot rest 7 points as,

(x,y) (x,-y) (-x, y) (-x,-y) (y,x) (y,-x) (-y,x) (-y,-x)

Circle Generation Algorithms:

   (a) Digital Differential Analyzer (DDA) circle drawing algorithm.
   (b) Bresenham's circle drawing algorithm.

Bresenham`s generation Algorithm:

     To begin, note that only one octant of the circle need be generated. The other parts are obtained by successive reflections. As shown in fig. If the first octant (0 to 45 deg. ccw) is generated, the second octant is obtained by reflection through the line y=x to yield the first quadrant. The result in the first quadrant is reflected through the line x=0 to obtain those in the first quadrant.

     The results in the first quadrant are reflected through the line $x = 0$ to obtain those in the second quadrant. The combined results in the upper semicircle are reflected through the line y= 0 to complete the circle. Figure gives the appropriate two-dimensional reflection matrices. To derive Bresenham's circle generation algorithm, consider the first quadrant of an origin centered circle. Notice that if the algorithm begins at x = 0, y = R, then for clockwise generation of the circle y is a monotonically decreasing function of x in the first quadrant.

     Similarly, if the algorithm begins at y = 0, x = R, then for counterclockwise generation of the circle x is a monotonically decreasing function of y. Here, clockwise generation starting at x= 0, y = R is chosen. The center of the circle and the starting point are both assumed located precisely at pixel elements.

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

From the following illustration, you can see that we have put the pixel at (X, Y) location and now need to decide where to put the next pixel − at N (X+1, Y) or at S (X+1, Y-1).
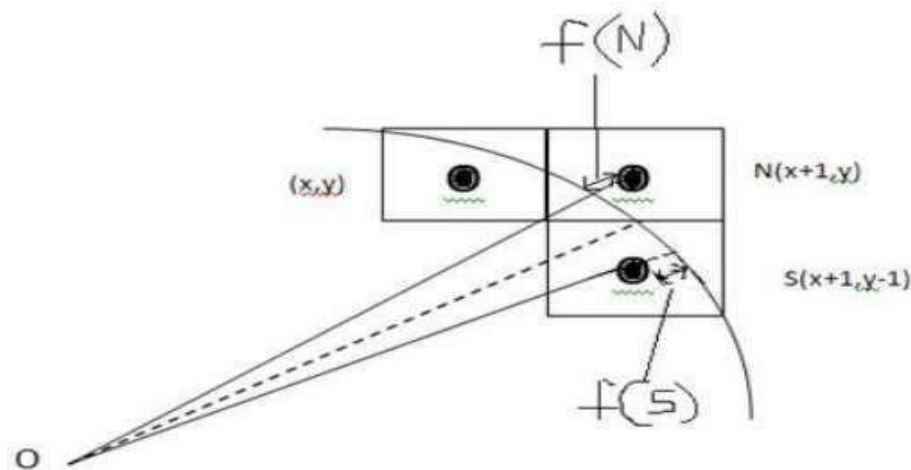
This can be decided by the decision parameter **d**.

- If d <= 0, then N(X+1, Y) is to be chosen as next pixel.
- If d > 0, then S(X+1, Y-1) is to be chosen as the next pixel.

Derivation:

Let us consider the circle at origin (0,0) i.e. $x^2 + y^2 = r^2$ and (xk, yk) be initial points
$F(x,y) = x^2 + y^2 - r^2$
$F(N) = (x_{k+1})^2 + y^2_k - r^2$ F(S)
$= (x_{k+1})^2 + (y_{k-1})^2 - r^2$
Decision parameter
$(Pk) = F(N) + F(S) P_k = 2(xk_{+1})^2 + y^2 + (y_{k-1})^2 - r^2$

Initial decision parameter $(P_0) = 2(1+0)^2 + r^2 + (r-1)^2 - 2r^2$

Po = 3-2r

$P_{k+1} = 2(x_k + 2)^2 + y^2_{k+1} + (y_{k+1} -1)^2 - 2r^2$

$P_{k+1} - P_k = 2[(x_k+2)^2 - (x_{k+1})^2] + (y^2_{k+1} - y^2_k) + (y_{k+1} -1)^2 - (y_{k-1})^2$

$P_{k+1} = p_k + 2(2x_k+3) + [(y_{k+1} + y_k) (y_{k+1} - y^1_{k+1})]_k + [^k(y_{k+1} + y_k - 2) (y_{k+1} - y_k)]$ ----------------------- (A)

Case 1:

If P is negative

$x_{k+1} - x_k = 1$

$y_{k+1} - y_k = 0$

Put in (A)

$P_{k+1} = P_k + 2(2x_k + 3)$

$P_{k+1} = Pk + 4x_k + 6$

Case 2:

If P is positive

$x_{k+1} - x_k = 1$

$y_{k+1} - y_k = -1$

Put in (A)

$P_{k+1} = P_k + 4x_k+6 + (y_{k+1} +y_k) (-1) + (y_{k+1} +y_k -2) (-1)$

$P_{k+1} = P_k + 4x_k - 4y_k + 10$

Algorithm:

1. Start.
2. Declare variables x, y, p
3. Initialise the graphic mode.
4. Input the radius of the circle r.
5. Load x-0, y=r, initial decision parameter p=1-r.so the first point is (0, r).
6. Repeat Step 7 while (x<y) and increment x-value simultaneously.
7. If (p>0), do p=p+2*(x-y) +1.
8. Otherwise, p=p+2*x+1 and y are decremented simultaneously.
9. Then calculate the value of the function circlepoints() with parameters (x,y).
10. Place pixels using putpixel at respective coordinate locations (x,y) in specified color in circlepoints() function.
11. Close Graph.
12. Stop.

Conclusion

Assignment No: 04

Title: Polygon filling methods: i) Flood fill / Seed fill ii) Boundary fill

Problem Statement: Implement the following polygon filling methods: i) Flood fill / Seed fill ii) Boundary fill;using mouse click, keyboard interface and menu driven programming

Objectives: 1. To learn and understand polygon filling.
2. To learn and understand scan line fill algorithm.

Outcomes: 1. Students will be able to understand the various polygon filling methods.
2. Students will be able to fill the given polygon using scan line fill algorithm.

Theory:

The basic scan-line algorithm is as follows:
- Find the intersections of the scan line with all edges of the polygon
- Sort the intersections by increasing x coordinate
- Fill in all pixels between pairs of intersections that lie interior to the polygon

Process involved:

The scan-line polygon-filling algorithm involves

• the horizontal scanning of the polygon from its lowermost to its topmost vertex,

• identifying which edges intersect the scan-line,

• and finally drawing the interior horizontal lines with the specified fill color process.
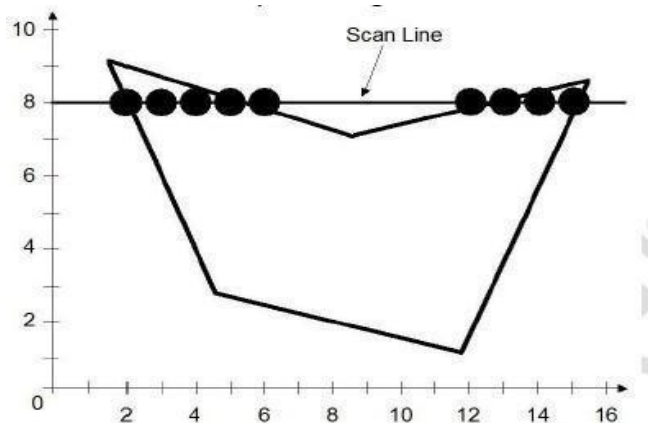
For e.g. Intersect scanline with polygon edges

− Fill between pairs of intersections

− Basic algorithm:

For y = ymin to ymax

1) intersect scanline y with each edge

2) sort intersections by increasing x i.e. [p0, p1, p2, p3]

3) fill pairwise (p0 −> p1, p2−> p3, ...)

What happens at edge end-point?
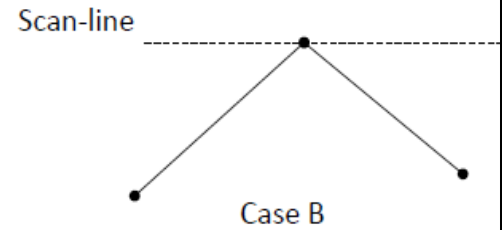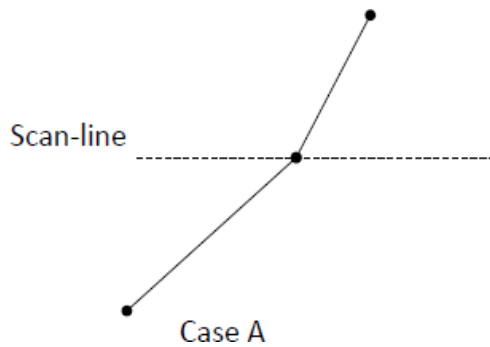
- Edge endpoint is duplicated.

  In other words, when a scan line intersects an edge endpoint, it intersects two edges.

- Two cases:

  Case A: edges are monotonically increasing or decreasing Case
  B: edges reverse direction at endpoint

- In Case A, we should consider this as only ONE edge intersection

- In Case B, we should consider this as TWO edge intersections

Computational structures
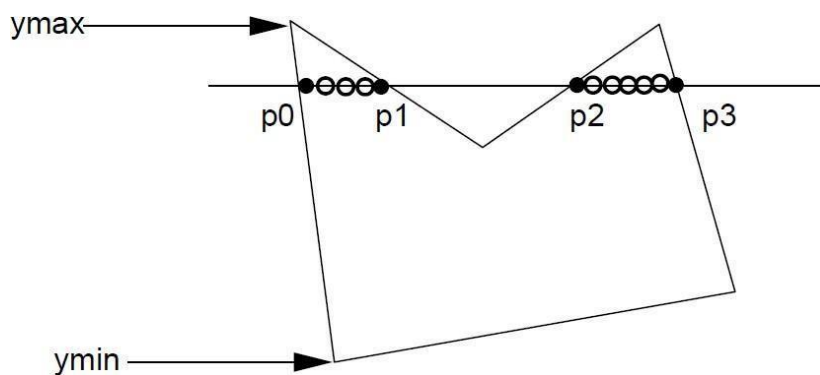
- Edge table

  - One table entry for each scan line which contains at least one "lower" vertexof a polygon edge
  - Properties of each edge are stored in a "bucket"
  - Table entry consists of an array of buckets

- Each bucket contains edges whose minimum y coordinate (ymin) starts at the bucket's line
- Each entry is a record containing:
  Ymax   xmin   increment
- The array of buckets sorted according to ymin

- Records in a bucket sorted according to xmin

Algorithm:

Steps:

1. the horizontal scanning of the polygon from its lowermost to its topmost vertex

2. identify the edge intersections of scan line with polygon

3. Build the edge table

a. Each entry in the table for a particular scan line contains the maximum y value for that edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge.

4. Determine whether any edges need to be splitted or not. If there is need to split, split the edges.
5. Add new edges and build modified edge table.

6. Build Active edge table for each scan line and fill the polygon based on intersection of scanline with polygon edges.

Conclusion:

Assignment No: 05

Title: Cohen Sutherland polygon clipping method

Problem Statement: Implement Cohen Sutherland polygon clipping method to clip the polygon with respectthe viewport and window. Use mouse click, keyboard interface.

Objectives: 1. To learn and understand the concept of clipping.
       2. To learn and understand line clipping algorithm.

Outcomes: 1. Students will be able to understand the concept of clipping.
         2. Students will be able to understand Cohen-Sutherland line clipping algorithm.

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

Line Clipping

In line clipping, we will cut the portion of line which is outside of window and keep only the portion that is inside the window.

Cohen-Sutherland Line Clipping:

This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is ($XWmin$,$YWmin$) and the maximum coordinate for the clipping region is ($XWmax$,$YWmax$)

.
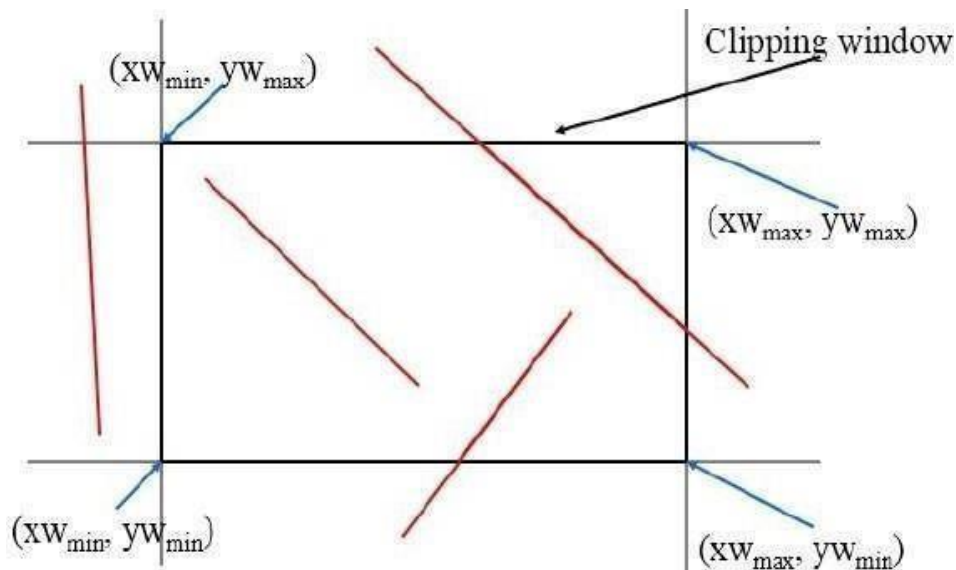
We will use 4-bits to divide the entire region. These 4 bits represent the Top, Bottom, Right, and Left of the region as shown in the following figure. Here, the **TOP** and **LEFT** bit is set to1 because it is the **TOP-LEFT** corner.
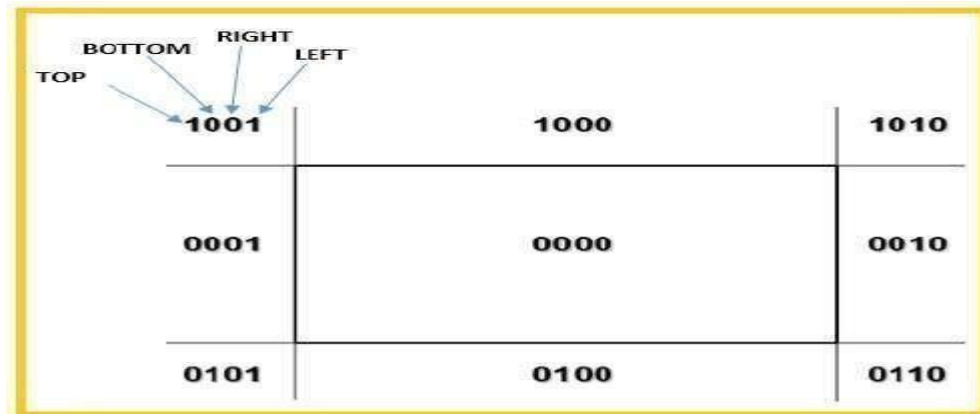
Fig: Region Codes

There are 3 possibilities for the line -
- Line can be completely inside the window (This line should be accepted).
- Line can be completely outside of the window (This line will be completely removed fromthe region).
- Line can be partially inside the window (We will find intersection point and draw only thatportion of line that is inside region).

Algorithm:

Step 1 − Assign a region code for each endpoint.

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

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

Step 3.1 − Ifthe result is not **0000,** then reject the line.
Step 3.2 − Else you need clipping.

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

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

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

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

Step 4 − Repeat step 1 for other lines.

Conclusion:

Assignment No: 06

Title: 2D transformation

Problem Statement: Implement following 2D transformations on the object with respect to axis:
       i) Scaling
       ii) Rotation about arbitrary point
       iii) Reflection

Objectives: 1. To learn and understand transformations.
              2. To learn and understand mathematical calculations related to various types of transformations like scaling, translation and rotation.

Outcomes: 1. Students will be able to understand transformations.
              2. Students will be able to do mathematical calculations related to various types of transformations like scaling, translation and rotation.
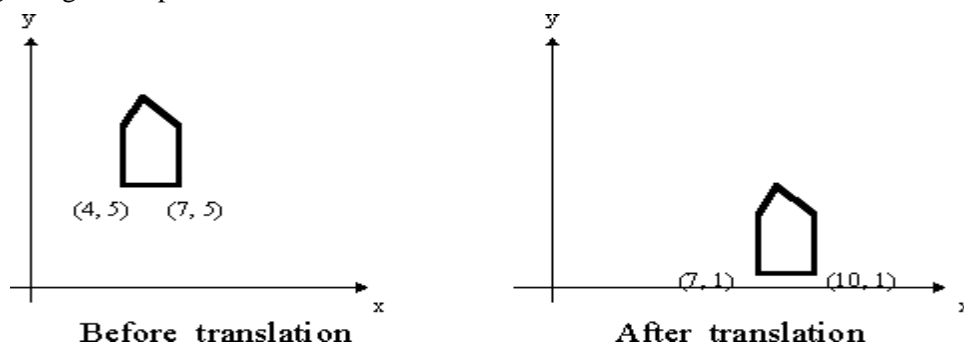
Theory:

We have to perform 2D transformations on 2D objects. Here we perform transformations on a line segment.

The 2D transformations are:

1. Translation
2. Scaling
3. Rotation

1. Translation: Translation is defined as moving the object from one position to another positionalong straight line path.

Before translation   After translation

We can move the objects based on translation distances along x and y axis. tx denotes translation distance
along x-axis and ty denotes translation distance along y axis.

Translation Distance: It is nothing but by how much units we should shift the object from one location to another along x, y-axis.

Consider (x,y) are old coordinates of a point. Then the new coordinates of that same point(x',y') can be obtained as follows:
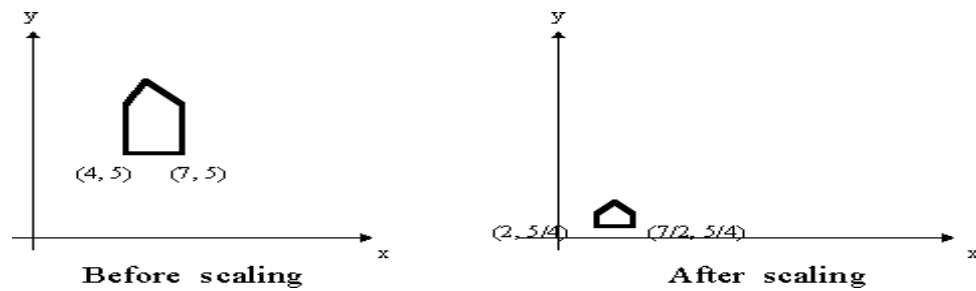
X'=x+tx

Y'=y+ty

We denote translation transformation as P. we express above equations in matrix form as:

$$P' = P + T$$

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

x,y---old coordinates
x',y'—new coordinates after translationtx,ty—
translation distances, T is

Scaling: scaling refers to changing the size of the object either by increasing or decreasing. We will increase or decrease the size of the object based on scaling factors along x and y- axis.

If (x, y) are old coordinates of object, then new coordinates of object after applying scaling transformation are obtained as:

x'=x*sx

y'=y*sy.

sx and sy are scaling factors along x-axis and y-axis. we express the above equations inmatrix form as:

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

Scaling Matrix

2. Rotation: A rotation repositions all points in an object along a circular path in the plane centered at the pivot point. We rotate an object by an angle theta. New coordinates after rotation depend on *both* *x* and *y*
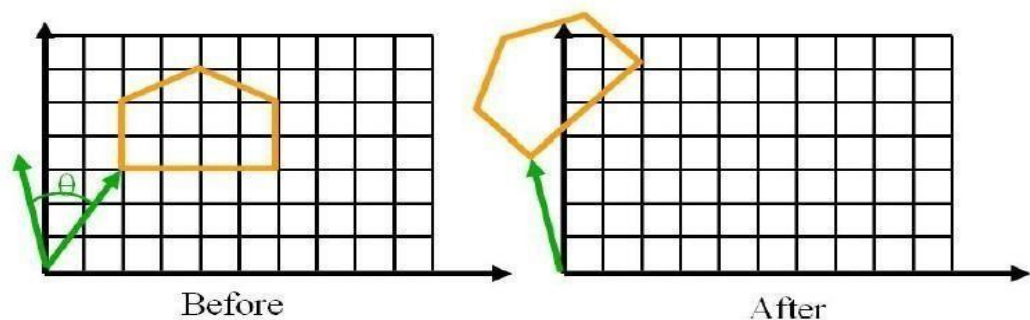
- $x' = x\cos\theta - y\sin\theta$
- $y' = x\sin\theta + y\cos\theta$

or in matrix form:

P' = R • P,

R-rotation matrix.

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

Conclusion:

Assignment No: 07

Title: Fractal patterns

Problem Statement: Generate fractal patterns using i) Bezier ii) Koch

Objectives: 1. To learn and understand the concept of curves.
2. To learn and understand curve generation techniques.

Outcomes: 1. Students will be able to understand the concept of curves.
2. Students will be able to understand various curve generation techniques.

Theory:

In computer graphics, we often need to draw different types of objects onto the screen. Objects are not flat all the time and we need to draw curves many times to draw an object.

Types of Curves

A curve is an infinitely large set of points. Each point has two neighbors except endpoints. Curves can be broadly classified into three categories − **explicit, implicit,** and **parametric curves**.

Implicit Curves

Implicit curve representations define the set of points on a curve by employing a procedure that can test to see if a point in on the curve. Usually, an implicit curve is defined by an implicit function of the form –

$f(x, y) = 0$

It can represent multivalued curves (multiple y values for an x value). A common example is the circle, whose implicit representation is

$$x^2 + y^2 - R^2 = 0$$

Explicit Curves

A mathematical function $y = f(x)$ can be plotted as a curve. Such a function is the explicit representation of the curve. The explicit representation is not general, since it cannot represent vertical lines and is also single-valued. For each value of x, only a single value of y is normally computed by the function.

Parametric Curves

Curves having parametric form are called parametric curves. The explicit and implicit curve representations can be used only when the function is known. In practice the parametric curves are used. A two-dimensional parametric curve has the following form −

$P(t) = f(t), g(t)$ or $P(t) = x(t), y(t)$

The functions f and g become the (x, y) coordinates of any point on the curve, and the points are obtained when the parameter t is varied over a certain interval [a, b], normally [0, 1].
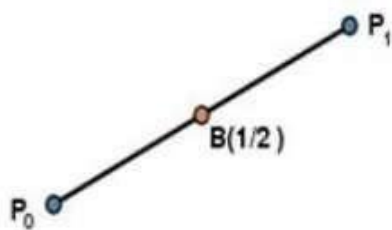
Bezier Curves:

A Bezier curve is a parametric curve frequently used in computer graphics and related fields. Generalizations of Bezier curves to higher dimensions are called Bezier surfaces, of which the Bezier triangle is a special case.

In vector graphics, Bezier curves are used to model smooth curves that can be scaled indefinitely. "Paths," as they are commonly referred to in image manipulation programs are combinations of linked Bezier curves. Paths are not bound by the limits of rasterized images and are intuitive to modify. Bezier curves are also used in animation as a tool to control motion Four points P0, P1, P2 and P3 in the plane or in higher dimensional space define a cubic Bezier curve. The curve starts at P0 going toward P1and arrives at P3 coming from the direction of P2. Usually, it will not pass through P1or P2; these points are only there to provide directional information. The distance between P0 and P1determines "how long" the curve moves into direction P2 before turning towards P3.

The explicit form of the curve is:

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \ , \ t \in [0,1].$$

Bezier curve is discovered by the French engineer **Pierre Bezier**. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as −

Simple Bezier Curve   Quadratic Bazier Curve   Cubic Bazier Curve
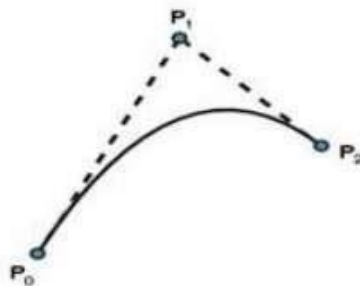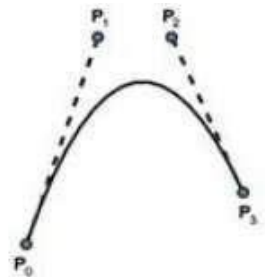
Properties of Bezier Curves:

- They generally follow the shape of the control polygon, which consists of the segments joining the control points.
- They always pass through the first and last control points.

- The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.

- A Bezier curve generally follows the shape of the defining polygon.

B-Spline Curves:

The Bezier-curve produced by the Bernstein basis function has limited flexibility.

- First, the number of specified polygon vertices fixes the order of the resulting polynomial which defines the curve.
- The second limiting characteristic is that the value of the blending function is nonzero for all parameter values over the entire curve.

Properties of B-spline Curve

B-spline curves have the following properties −

- The sum of the B-spline basis functions for any parameter value is 1.

- Each basis function is positive or zero for all parameter values.
- Each basis function has precisely one maximum value, except for k=1.
- The maximum order of the curve is equal to the number of vertices of defining polygon.

- The degree of B-spline polynomial is independent on the number of vertices of defining polygon.

- The curve line within the convex hull of its defining polygon.

Algorithm:
Step 1: Select a value t Î [0,1]. This value remains constant for the rest of the steps.

Step 2: Set P_i [0] (t) = P_i, for i = 0,...,n.

Step 3: For j= 0,...,n, set $P_i^{[j]}(t) = (1-t)P_{i-1}^{[j-1]}(t) + tP_i^{[j-1]}(t)$ for i = j,...,n.

Step 4: g (t) = P_n[n] (t)

Koch Curve-
The Koch snowflake (also known as the Koch curve, Koch star, or Koch Island) is a mathematical curve and one of the earliest fractal curves to have been described. It is based on the Koch curve, which appeared in a 1904 paper titled "On a continuous curve without tangents, constructible from elementary geometry" by the Swedish mathematician Helge von Koch.

The progression for the area of the snowflake converges to 8/5 times the area of the original triangle, while the progression for the snowflake's perimeter diverges to infinity. Consequently, the snowflake has a finite area bounded by an infinitely long line.

Construction of Koch curve-

Step1:
Draw an equilateral triangle. You can draw it with a compass or protractor, or just eyeball it if you don't want to spend too much time drawing the snowflake.

☐ It's best if the length of the sides is divisible by 3, because of the nature of this fractal. This will become clear in the next few steps.
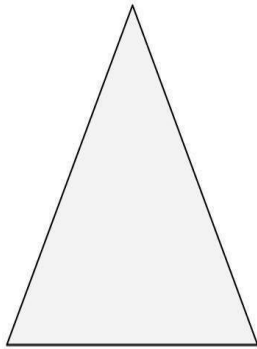
Step2: Divide each side in three equal parts. This is why it is handy to have the sides divisible by three.
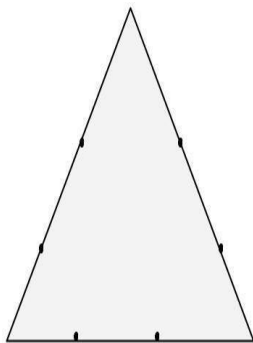
Step3:
Draw an equilateral triangle on each middle part. Measure the length of the middle third to know the length of the sides of these new triangles.
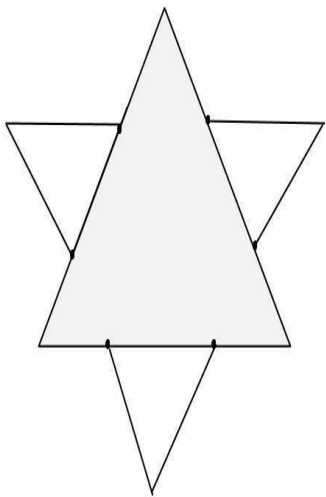
Step4:
Divide each outer side into thirds. You can see the 2nd generation of triangles covers a bit of the first. These three-line segments shouldn't be parted in three.
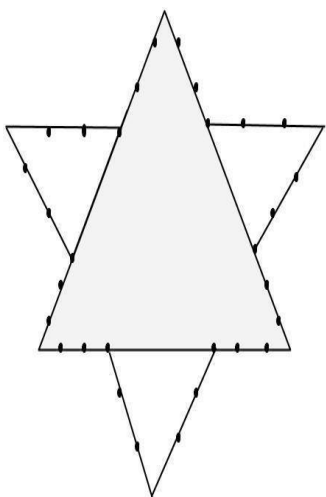
Step5:
Draw an equilateral triangle on each middle part.

☐ Note how you draw each next generation of parts that are one 3rd of the mast one.
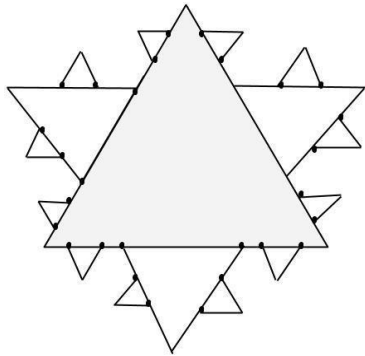
Conclusion:

Assignment No: 08

Title: Animation

Problem Statement: Implement animation principles for any object

Objectives: 1. To learn and understand the concept of animation.
                2. To learn and understand animation principles.

Outcomes: 1. Students will be able to understand the concept of animation.
              2. Students will be able to understand various animation principles.

Theory:

The 12 Principles of Animation-

1. Squash and Stretch

Squash and stretch refer to distorting or deforming an object or character to give the illusion of weight and flexibility. As a character moves, its body stretches and squashes to create the impression of mass and elasticity. For example, when a ball bounces, it squashes on impact and then stretches back into shape. The same tips apply to character animation. As a character runs, its body squashes down with footsteps and stretches back up between steps.

2. Anticipation

Anticipation prepares the audience for an upcoming movement or action. It builds anticipation through subtle movements that lead to the main action. For example, a golfer might waggle their club back and forth before swinging, or a pitcher might wind up their throw by bringing the ball backward first.

3. Staging

Staging refers to the composition and framing of a scene to focus the audience's attention on the most important parts of the action. Animators stage scenes with appropriate camera angles, lighting, and character placement to highlight the main ideas and points of interest.

4. Straight Ahead and Pose to Pose

There are two main approaches to creating animation: straight-ahead and pose-to-pose animation.

Straight-ahead animation means drawing out the frames one by one in order from start to finish. Animators taking this approach don't plan ahead but rather figure out each subsequent frame as they go. This creates a very organic, fluid feel and allows for experimentation. However, it can also result in inconsistent volumes and proportions.

Pose-to-pose animation involves planning out specific keyframes and poses first, then filling in the in-betweens afterward. This allows for greater control in terms of volumes and proportions. However, the motion can feel more mechanical if the in-betweens aren't filled in carefully.

5. Follow-Through and Overlapping Action

Using follow-through and overlapping action makes animation more natural and lifelike. **Follow-through** refers to the continuation of motion after the character has stopped applying force. For example, when a character swings their arm, the arm, hair, clothing, etc., will continue to move after the character stops actively swinging. Without follow-through, the motion would look unnatural since things don't just stop instantly in real life.

6. Slow In and Slow Out

The principle of slow in and out means animating with more drawings near the start and end of an action. This exaggerates the extreme poses and makes the movement feel smoother. For example, if a character is jumping, more frames would be used at the highest and lowest parts of the jump arc to accentuate those poses. Fewer frames would be used in the middle.

7. Arcs

Animators follow arched trajectories when animating actions like jumping and throwing. Rather than moving in straight lines, characters swing arms and bodies in expressive curve paths. Arms don't just move up and down—they follow sweeping arcs for greater fluidity.

8. Secondary Action

Secondary actions are subtle motions that complement or support the main action. For example, a character telling an energetic story may swing their arms around as they talk. Or if a character jumps down from a ledge, they may readjust their clothing when landing. These secondary motions reinforce what's happening in the scene without distracting from the core action.

9. Timing

Timing refers to the number of frames allotted to each action. It determines the pace and rhythm of the animation. Timing impacts how movements feel—the more frames, the slower and more exaggerated the action becomes. For example, a ball that drops quickly has fewer frames than one that bounces slowly to a stop.

10. Exaggeration

Exaggeration in animation means overstating movements and actions beyond normal proportions. A fist bump becomes a full-body motion. Eyes pop out of an astonished character's head. Exaggerating the squash and stretch is one form, but **exaggeration** applies to all aspects of movement, from arcs to timing.

11. Solid Drawing

Solid drawing means creating figures and objects with volume, weight, and convincing form. Characters feel believable when drawn with anatomically correct proportions and dimensions—when they have substance and mass. Solid drawing especially applies to key poses where characters demonstrate their shape based on the <u>shape language technique</u>.

12. Appeal

Appeal in animation refers to creating characters and motions that are compelling, entertaining, and aesthetically pleasing. Animators aim to give characters charm and whimsy through their design and movement. Disney stressed the principle of appeal—animation should above all be charming and captivating.

Conclusion:

Graphics Functions

Following table gives us various commands and functions with short description, syntax and example.

Sr. No.	Command/ Function Name	Description	Syntax	Example
1.	Circle	Circle drawsa circle.	Circle(int x, inty, int radius).	circle(50,50,100)
2.	Arc	Arc draws an arc.	Arc(int x, int y, int stangle, int endangle,int radius).	arc(35,70,45,45,50)
3.	Pieslice	It draws a pieslice.	pieslice(int x,int y, int stangle, int endangle,int radius).	pieslice(midx.midy,stangle,endangle,radius)
4.	Getmaxx	Returns maximum x screen coordinates.	Getmaxx().	Midx=getmaxx()/2.
5.	Getmaxy	Returns maximum x screen coordinates.	Getmaxy().	Midy=getmaxy()/2.
6.	Ellipse	Ellipse draws an ellipse,	Ellipse (int x, inty, int stangle, int endangle, int xradius, intyradius).	Ellipse (25,25,0,180,50,50)
7.	Setcolor	Setcolor sets the current drawing colour.	Setcolor(int color).	Setcolor(red).
8.	Getmaxcolor	It returns maximum colour value.	Getmaxcolor(void).	Getmaxcolor(void).
9.	Outtextxy	Displays a string in the graphics mode	Outtextxy(char far *textstring)	

10.	Putpixel	Putpixel plots a pixel at a specified point.	Putpixel(int x,int y,int color)	Putpixel(20,20,RED).
11.	Bar	Draws a bar.	Bar(int left,int top,int right,int bottom)	Bar(25,30,25,30)
12	polygon	Draws a polygon	Polygon()	
13	closegraph	For closing the graphics mode.	Closegraph().	Closegraph().
14	floodfill	Fille a bounded region.	Floodfill(int x,int y, int border).	Floodfill(2,2,getmaxcolor()).
15	getcolor	Getcolor returns the current drawing color.	int far Getcolor()	Getcolor();
16	drawpoly	It draws the outtline of the polygon.	Drawpoly(int numpoints,int far *polypoints)	Drawpoly(5,poly).
17	Line	It draws the line between two specified points.	Line(int x, inty)	Line(10,20)
18	getpixel	It gets the colour of the specified pixel.	Getpixel(int x,int y)	Getpixel(45,78).
19	fillellipse	Fillellipse draws an ellipse,that fills the ellipse with the current fill color and fill pattern.	Fillellipse(int x,int y,xradius,yradius).	Fillellipse(25,75,20,30)

20	Bar3d	Draws a 3D bar.	Bar3d(int left,int top,int right,int bottom,int depth,int topflag).	Bar3d(20,30,40,20,30,40).
21	getbkcolor	Getbkcolor returns the current background colour.	Getbkcolor(void).	Getbkcolor(void).
22	setfillstyle	Set the fill pattern and colour.	Setfillstyle(int pattern,int color).	Setfillstyle(SOLID_FILL,getmaxcolor()).
23	Textheight	It returns the height of the string in pixels.	Textheight(char far *textstring).	Textheight(msg).
24	Textwidth	It returns the width of the string in pixels.	Textwidth(char far *textstring).	Textwidth(msg).
25	Lineto	Lineto drawsa line from the CP to (x,y), then moves the CP to (x,y).	Lineto(int x,inty).	Lineto(20,40).
26	Sector	Sector draws an elliptical pislice.	Sector(int x,int y,int stangle,int endangle, int xradius,int yradius).	Sector(20,30,40,20,30,40).
27	outtext	outtext displays a string in the viewport (graphics mode)	outtext(char *textstring);	outtext("this ").
28	Setbkcolor	Setbkcolor sets the current background colour using the pallet.	setbkcolor(int color).	setbkcolor().

29	getcolor	getcolor returns the current drawing colour.	getcolor(void).	getcolor(void).
30	Rectangle	Draw a rectangle	Rectangle(int left, int top, int right,int bottom)	Rectangle(269,189,369,289)
31	Fillpoly	Draw and fills the polygon	Fillpoly(int numpoint, int far *polypoints)	Fillpoly(5,poly)
32	Ellipse	Draw the elliptical arc	Ellipse(int x, int y, int stangle, int endangle,int xradius, int yradius)	Ellipse(450,350,0,360,50,100)
33.	Sector	Draw and fills the elliptical pie slice	Sector(int x, int y, int stangle, int endangle,int xradius, int yradius)	Sector(200,325,0,360,20,45)
34	Cleardevice	Clear the graphics screen	Void far cleardevice(void)	Cleardevice()
35	Setfillstyle	Set the fill pattern and color	Setfillstyle(int pattern, int color)	Setfillstyle(1,3)
36	Setlinestyle	Sets the current line style and width or pattern	Setlinestyle(int linestyle, int unsigned pattern,int thickness)	Setlinestyle(1,1,1)
37	Outtextxy	Displays a string at the specified location	Void far outtextxy(int x,int y, char far *textstring);	Void far outtextxy(int 230, int 240, char far* hello);