ḥīḥī

Vidyavardhini's
College of Engineering & Technology

Vasai Road (W)

Department of Computer Engineering

Laboratory Manual
[Student Copy]

Semester	III	Class	SE
Course No.	CSL303	Academic Year	2024-25 (Odd Sem.)
Course Name	Computer Graphics Lab		

Vidyavardhini's College of Engineering & Technology

Vision

To be a premier institution of technical education; always aiming at becoming a valuable resource for industry and society.

Mission

- To provide technologically inspiring environment for learning.
- To promote creativity, innovation and professional activities.
- To inculcate ethical and moral values.
- To cater personal, professional and societal needs through quality education.

Department Vision:

To evolve as a center of excellence in the field of Computer Engineering to cater to industrial and societal needs.

Department Mission:

- To provide quality technical education with the aid of modern resources.
- Inculcate creative thinking through innovative ideas and project development.
- To encourage life-long learning, leadership skills, entrepreneurship skills with ethical & moral values.

Program Education Objectives (PEOs):

PEO1: To facilitate learners with a sound foundation in the mathematical, scientific and engineering fundamentals to accomplish professional excellence and succeed in higher studies in Computer Engineering domain
PEO2: To enable learners to use modern tools effectively to solve real-life problems in the field of Computer Engineering.
PEO3: To equip learners with extensive education necessary to understand the impact of computer technology in a global and social context.
PEO4: To inculcate professional and ethical attitude, leadership qualities, commitment to societal responsibilities and prepare the learners for life-long learning to build up a successful career in Computer Engineering.

Program Specific Outcomes (PSOs):

PSO1: Analyze problems and design applications of database, networking, security, web technology, cloud computing, machine learning using mathematical skills, and computational tools.

PSO2: Develop computer-based systems to provide solutions for organizational, societal problems by working in multidisciplinary teams and pursue a career in the IT industry.

Program Outcomes (POs):

Engineering Graduates will be able to:

- **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **PO3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **PO4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **PO6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **PO7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9. Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to

comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

- **PO11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **PO12. Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Objectives

1	To Develop Entity Relationship data model.
2	To develop relational Model
3	To formulate SQL queries.
4	To learn procedural interfaces to SQL queries
5	To learn the concepts of transactions and transaction processing
6	To understand how to handle concurrent transactions and able to access data through front end (using JDBC ODBC connectivity)

Course Outcomes

At the end of the course student will be able to:		PO/PSO	Bloom Level
CSL303.1	Implement output primitive algorithm	Apply	Apply (Level 3)
CSL303.2	Implement filled area primitive algorithm	Apply	Apply (Level 3)
CSL303.3	Implement character generation method	Apply	Apply (Level 3)
CSL303.4	Apply transformation and clipping algorithm on graphical objective	Apply	Apply (Level 3)

CSL303.5	Implement curve and fractal generation method	Apply	Apply (Level 3)
CSL303.6	Develop an animation based graphical application on learned concepts	Construct	Apply (Level 3)

Mapping of Experiments with Course Outcomes

Experiments	Course Outcomes					
	CSL303.1	CSL303.2	CSL303.3	CSL303.4	CSL303.5	CSL303.6
Implement DDA Line drawing algorithm (Simple and dot using switch case)	3					
Implement Bresenham's line-drawing algorithm (Simple and dot using switch case)	3					
Implement a mid-point circle algorithm	3					
Implement a midpoint Ellipse algorithm	3					
Implement Area Filing Algorithm Boundary Fill, Flood Fill, Scan line Polygon Fill		3				
Implement Character generation Bit map Method to generate Characters in computer graphics			3			
Implement 2D transformation Translation, Scaling, Rotation (Equilateral Triangle)				3		

Implement Curve: Bezier Cure for 4 Control points (Midpoint approach)					3	
Implement the Fractal generation method – Koch Curve					3	
Program to perform Animation						3
Mini project to perform using C/C++/java/open GL/Blender/any other tool (3/4 students per group)						3

INDEX

Sr. No.	Name of Experiment	D.O.P.	D.O.C.	Page No.	Remark
1	Implement DDA Line drawing algorithm (Simple and dot using switch case)				
2	Implement Bresenham's line-drawing algorithm (Simple and dot using switch case)				
3	Implement a mid-point circle algorithm				
4	Implement a midpoint Ellipse algorithm				
5	Implement Area Filing Algorithm Boundary Fill, Flood Fill, Scan line Polygon Fill				
6	Implement Character generation Bit map Method to generate Characters in computer graphics				
7	Implement 2D transformation Translation, Scaling, Rotation (Equilateral Triangle)				
8	Implement Curve: Bezier Cure for 4 Control points (Midpoint approach)				
9	Implement the Fractal generation method – Koch Curve				
10	Program to perform Animation				
11	Mini project to perform using C/C++/java/open GL/Blender/any other tool (3/4 students per group)				

D.O.P: Date of performance

Name of Student: karan pawar Class: SE-2

Batch: C Roll No: 61

D.O.C : Date of correction

Experiment No.1
Implement DDA Line drawing algorithm.
Date of Performance: 18/07/24
Date of Submission: 25/07/24

Aim:

Implement DDA line drawing algorithm.

Introduction:

A line connects two points. It is a basic element in graphics. To draw a line, you need two points between which you can draw a line. In the following three algorithms, we refer the one point of line as X_0, Y_0 and the second point of line as X_1, Y_1.

Theory:

DDA Line Drawing Algorithm:

Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm. DDA algorithm is an incremental scan conversion method. Here we perform calculations at each step using the results from the preceding step. The characteristic of the DDA algorithm is to take unit steps along one coordinate and compute the corresponding values along the other coordinate. The unit steps are always along the coordinate of greatest change, e.g. if dx = 10 and dy = 5, then we would take unit steps along x and compute the steps along y.

Procedure:

1. length ←abs(x2-x1);
2. if(abs(y2-y1) > length) then length ← abs(y2-y1);
3. x increment ← (x2-x1) / length;
4. y increment ← (y2-y1) / length;
5. x ←x+0.5; y ←y+0.5;
6. for i ←1 to length follow steps 7 to 9
7. plot(trunc(x),trunc(y));

8. x ←x + x increment;

9. y ←y + y increment;

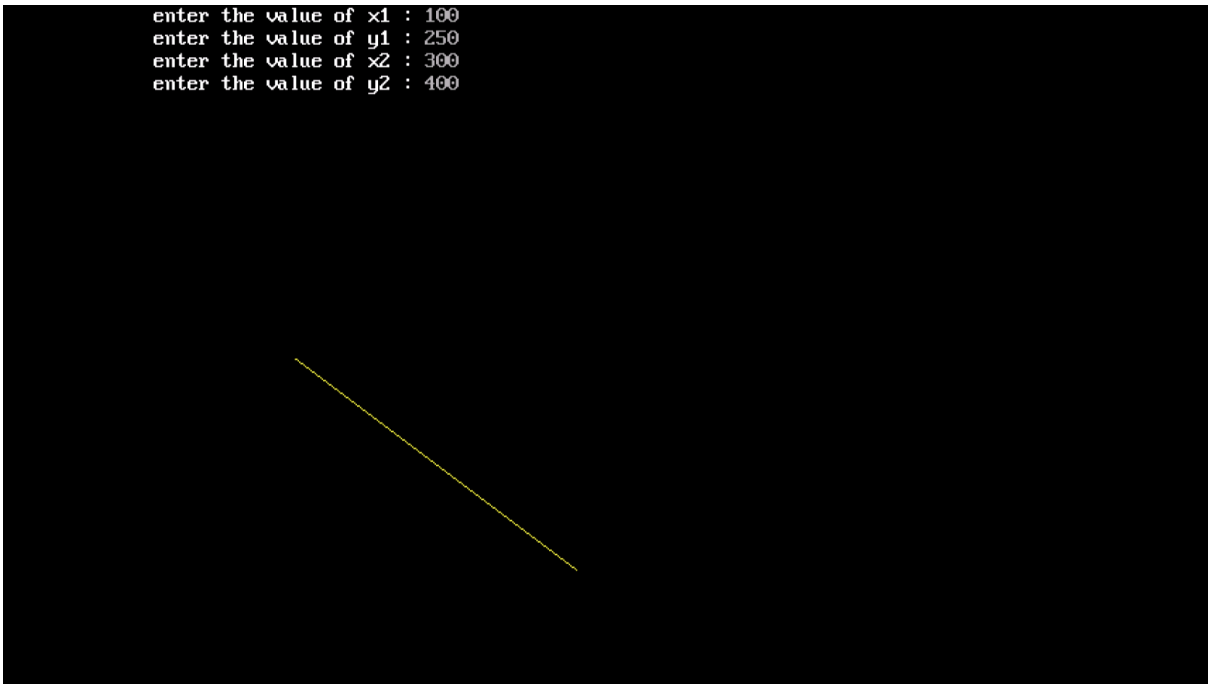
10. Stop.

Program:

```
#include<conio.h>
#include<stdio.h>
#include<graphics.h>
#include<math.h>
#include<dos.h>
int main(){
clrscr();
float x,y,x1,y1,x2,y2,dx,dy,pixel;
int i,gd,gm;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"..\\BGI");
printf("enter the value of x1 : ");
scanf("%f",&x1);
printf("enter the value of y1 : ");
scanf("%f",&y1);
printf("enter the value of x2 : ");
scanf("%f",&x2);
printf("enter the value of y2 : ");
scanf("%f",&y2);
dx=abs(x2-x1);
dy=abs(y2-y1);
if(dx>dy){
pixel=dx;}
else
{pixel=dy;}
dx=dx/pixel;
dy=dy/pixel;
x=x1;
y=y1;
i=1;
while(i<=pixel){
putpixel(x,y,YELLOW);
x=x+dx;
```

```
y=y+dy;
i++;
delay(100);
}
getch();
closegraph();
return 0;
}
```

Output:



Experiment No.2
Implement Bresenham's Line drawing algorithm.
Date of Performance: 25/07/24
Date of Submission: 01/08/24

Aim:

Implement Bresenham's line drawing algorithm.

Bresenham Line Drawing Algorithm:

The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.

Procedure:

1. Input the two line endpoints and store the left endpoint in(x_0, y_0)

2. Load (x_0, y_0) into the frame buffer; that is, plot the first point.

3. Calculate constants Δx, Δy, $2\Delta y$ and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as:

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

4. At each x_k, the next point the line, starting at k=0, perform the following test:

If $p_k < 0$, the next point is (x_{k+1}, y_k) and

$Pk+1 = p_k + 2\Delta y$

Otherwise, the next point is $(xk+1, y_{k+1})$ and

$p_{k+1} = p_k + 2\Delta y - 2\Delta x$

5. Repeat step 4 Δx times

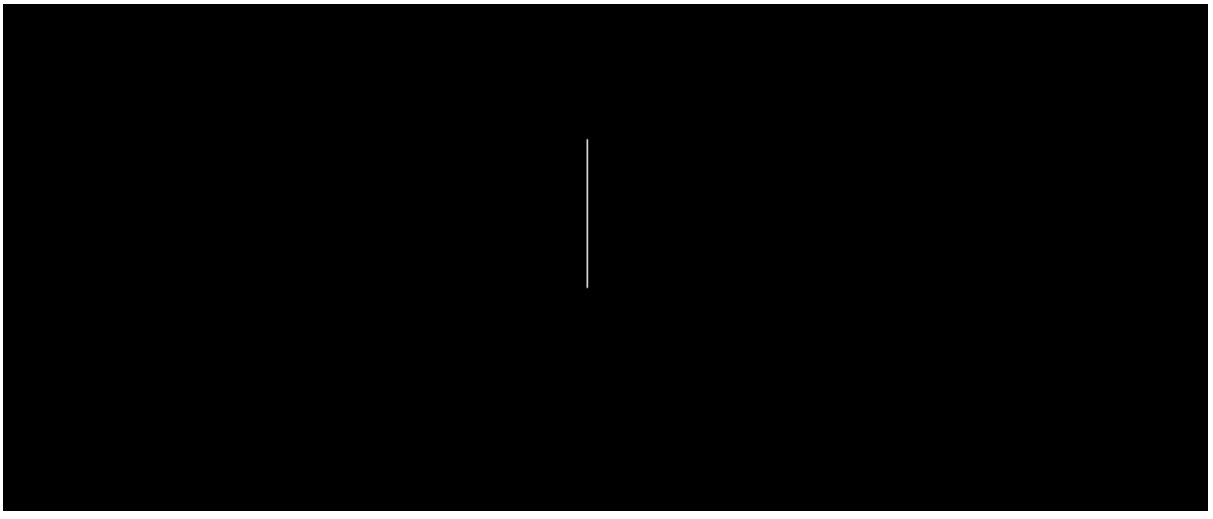
6. STOP

Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
int main(){
clrscr();

int x,y,x1,y1,x2,y2,p0,p1,p2,i,dx,dy;
int gm,gd=DETECT;

printf("enter the value of x1 :\n");
scanf("%d",&x1);

printf("enter the value of x2 :\n ");
scanf("%d",&x2);

printf("enter the value of y1 :\n");
scanf("%d",&y1);

printf("enter the value of y2 :\n");
scanf("%d",&y2);

x=x1;
y=y1;

dx=x2-x1;
dy=y2-y1;

p1=2*dy;
p2=(2*dy)-(2*dx);

p0=(2*dy)-dx;

initgraph(&gd,&gm,"..\\bgi");

for(i=0;i<=dx;i++){
if(p0 < 0){

x=x+1;
y=y;
p0=p0+p1;
```

```
} else {

x=x;
y=y+1;
p0=p0+p1;

}

putpixel(x,y,WHITE);
delay(100);

}

getch();

return 0;

}
```

Output:



Experiment No.3
Implement a mid-point circle algorithm
Date of Performance: 01/08/24
Date of Submission: 08/08/24

Aim:

Implement midpoint Circle algorithm.

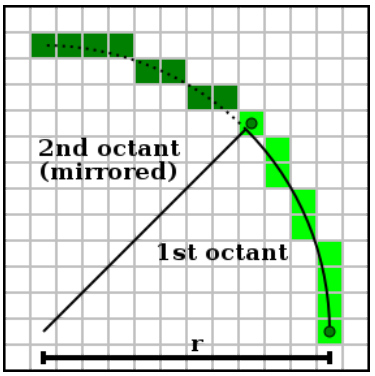
Introduction:

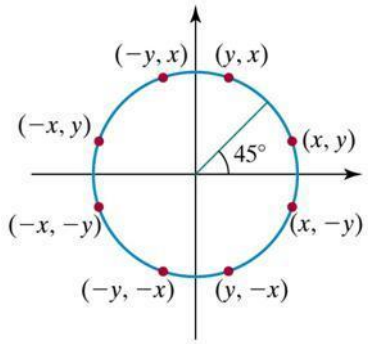
Drawing a circle on the screen is a little complex than drawing a line. There are two popular algorithms for generating a circle – **Bresenham's Algorithm** and **Midpoint Circle Algorithm**. These algorithms are based on the idea of determining the subsequent points required to draw the circle.

Theory:

We use the **Midpoint Circle** algorithm to calculate all the perimeter points of the circle in the **first octant** and then print them along with their mirror points in the other octants. This will work only because a circle is symmetric about it's centre. For any given pixel (x, y), the next pixel to be plotted is either **(x, y+1)** or **(x-1, y+1)**. This can be decided by following the steps below.

1. Find the mid-point **p** of the two possible pixels i.e (x-0.5, y+1)
2. If **p** lies inside or on the circle perimeter, we plot the pixel (x, y+1), otherwise if it's outside we plot the pixel (x-1, y+1)





Procedure:

1. Read the radius (r) of the circle
2. Initialize starting point as

x=0

y=r

3. Calculate initial value of decision parameter as p=1.25-r
4. do

{

Plot(x, y)

If (p<0)

{

x=x+1

y=y

p=p+2x+1

}

Else

{

x=x+1

y=y-1

p=p+2(x-y)+1

}

While (x<y);

5. Determine symmetry point
6. STOP.

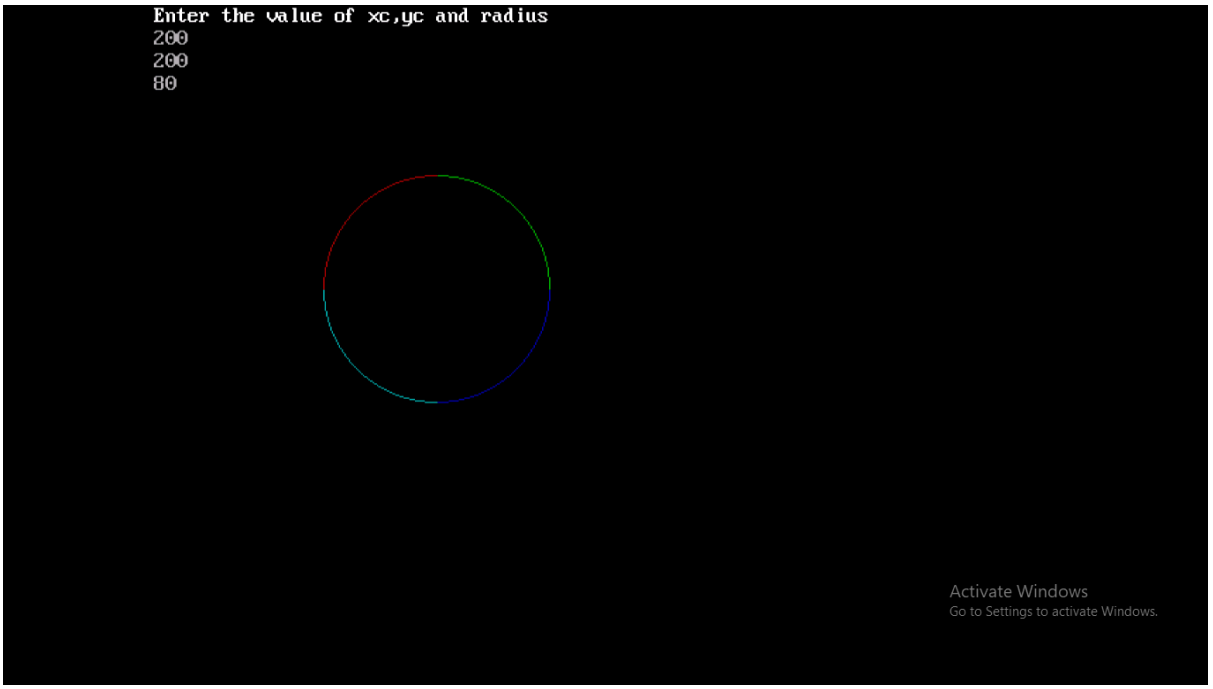
Program:

```
#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<DOS.h>

int main(){

clrscr();

int Xc,Yc,x,y,r;

float P0;


int gd,gm;

gd=DETECT;

initgraph(&gd,&gm,"..\\bgi");


printf("NAME : Karan Pawar\n");

printf("ROll No. : 61\n");

printf("EXP No. : 03 \n");


printf("enter the value of x ,y and r : \n");

scanf("%d%d%d",&Xc,&Yc,&r);
```

```
x=0;

y=r;

P0=1.25-r;




while(x<y){

putpixel(Xc+x,Yc+y,1);

putpixel(Xc+x,Yc-y,2);

putpixel(Xc-x,Yc+y,3);

putpixel(Xc-x,Yc-y,4);

putpixel(Yc+y,Xc+x,5);

putpixel(Yc+y,Xc-x,6);

putpixel(Yc-y,Xc+x,7);

putpixel(Yc-y,Xc-x,8);




}

if(P0<0){

x=x+1;
```

```
y=y;

P0=P0+2*x+1;

}else{

x=x+1;

y=y-1;

P0=P0+2*(x-y)+1;

}

getch();

return 0;

}
```

Output:


Enter the value of xc,yc and radius
200
200
80
Activate Windows
Go to Settings to activate Windows.

Experiment No.4
Implement a mid-point ellipse generation algorithm.
Date of Performance:08/08/24
Date of Submission:22/08/24

Aim:

Implement midpoint Ellipse algorithm.

CSL303: Computer Graphics Lab
Name of Student: karan pawar Class: SE-2

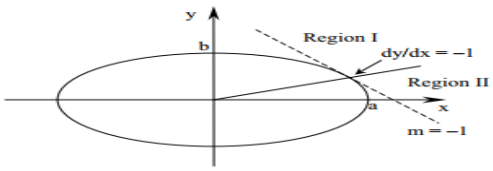
Batch: C Roll No: 61

Introduction:

Ellipse is the set of all points, the sum of whose distances from two fixed points is a constant. Midpoint ellipse algorithm is a method for drawing ellipses in computer graphics. This method is modified from Bresenham's algorithm. The advantage of this modified method is that only addition operations are required in the program loops. This leads to simple and fast implementation in all processors.

Theory:

The curve is divided into two regions. In region I, the slope on the curve is greater than –1 while in region II less than –1.



Consider the general equation of an ellipse, $b^2 x^2 + a^2 y^2 – a^2 b^2 = 0$ where a is the horizontal radius and b is the vertical radius, we can define an function f(x,y) by which the error due to a prediction coordinate (x,y) can be obtained. The appropriate pixels can be selected according to the error so that the required ellipse is formed. The error can be confined within half a pixel.

Procedure:

1. Start.
2. Initialize the graphic system using initgraph function.
3. Get the input of radius of major and minor arc from the user.
4. Store the values of major and minor arc in an another variable.
5. Square the values of major and minor arc.

6. Calculate decision parameter P = (square of minor axis – (square of major axis*minor axis) + (0.25* square of major axis).
7. Put the pixels symmetrically at = (0, length of minor axis).
8. while (2*(square of minor axis *x)<=2*(square of major axis*y)), repeat steps 9 to step 7.
9. increment x axis by 1.
10. If P < 0
11. new P = (P+(square of minor axis* square of major axis)+ square of major axis).
12. Else
13. new P = (P+(square of minor axis*x axis)-(2*square of major axis*y axis)+ square of minor axis).
14. Decrement y by 1.
15. End of step 10 if else structure.
16. Plot symmetric points of ellipse in each quadrant.
17. End of step 8 loop.
18. This will give us ellipse only across minor axis now to draw an ellipse across major axis we proceed further.
19. Get last point of ellipse in 1 st quadrant.
20. Initialize e = square of (x axis+.5)
21. Initialize f = square of (y axis-1).

22. Decision parameter P1 = ((square of minor axis *e)+(square of major axis*f)-(square of minor axis* square of major axis).
23. While y axis != 0 repeat steps 24 to step 32.
24. If P1>0
25. New P1 = (P1+ square of major axis -(2* square of major axis*x axis)).
26. Else
27. New P1 = (P1+(2*square of minor axis*(x ax is+1))-(2* square of major axis*(y axis-1))+square of major axis).
28. Increment x axis by 1.
29. End of step 25 if else structure
30. Decrement y axis by 1.
31. Plot symmetric point in all quadrants
32. End of step 23 while loop.
33. Close the graphic system.
34. Stop.

Program:

```
#include<stdio.h>

#include<dos.h>

#include<conio.h>

#include<graphics.h>
```

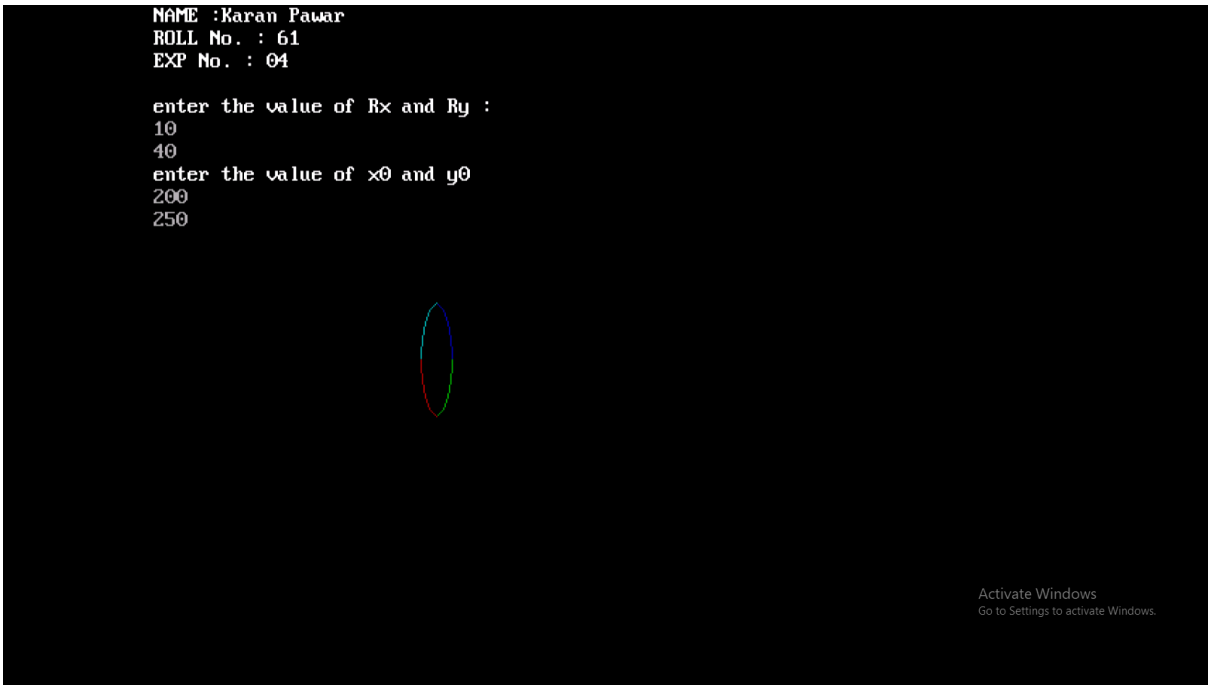
```
void main()
{
   int n, i, j, k, gd = DETECT, gm, dy, dx;
   int x, y, temp;
   int a[20][2], xi[20];
   float slope[20];
     clrscr();

   initgraph(&gd, &gm, "..\\bgi");

   printf("\n\n\tEnter the no. of edges of polygon :");
   scanf("%d", &n);
   printf("\nEnter the cordinates of polygon:\n");
for(i=0;i<n;i++) {
        printf("X%dY%d : ",i,i);
        scanf("%d%d", &a[i][0], &a[i][1]);
   }
   a[n][0] = a[0][0];
   a[n][1] = a[0][1];
```

```
    for (i = 0; i < n; i++)
    {
        line(a[i][0], a[i][1], a[i + 1][0], a[i + 1][1]);
    }
    for (i = 0; i < n; i++)
    {
        dy = a[i + 1][1] - a[i][1];
        dx = a[i + 1][0] - a[i][0];
        if (dy == 0){
           slope[i] = 1.0;}
        if (dx == 0){
           slope[i] = 0.0;}
if((dy!=0)&&(dx!=0))
{
        slope[i] = (float)dx/dy;
}
}
for(y=0; y<480; y++){
    k = 0;
    for (i = 0; i < n; i++)
```

```
{
    if (((a[i][1] <= y) && (a[i + 1][1] > y)) || ((a[i][1] > y) && (a[i + 1][1] <= y)))
    {
       xi[k] = (int)(a[i][0] + slope[i] * (y - a[i][1]));
       k++;
    }
}


    for (j = 0; j < k - 1; j++){
       for (i = 0; i < k - 1; i++)
       {
              if (xi[i] > xi[i + 1])
              {
                     temp = xi[i];
                    xi[i] = xi[i + 1];
                 xi[i + 1] = temp;
              }
       }              }
    setcolor(35);
    for (i = 0; i < k; i += 2)
```

```
        {
            line(xi[i], y, xi[i + 1] + 1, y);
            delay(20);
        }
    }
    getch();
    closegraph();
}
```

Output:


NAME :Karan Pawar
ROLL No. : 61
EXP No. : 04
enter the value of Rx and Ry :
10
40
enter the value of x0 and y0
200
250
Activate Windows
Go to Settings to activate Windows.

Experiment No.5
Implement Area Filing Algorithm Boundary Fill, Flood Fill, Scan Line Polygon Fill
Date of Performance:22/08/24
Date of Submission:29/08/24

Aim: Implement Area Filling Algorithm: Boundary Fill, Flood Fill,Scanline Polygon Fill

Introduction:

Figures on a computer screen can be drawn using polygons. To fill those figures with color, we need to develop some algorithm.There are two famous algorithms for this purpose: Boundary fill and Scanline polygon fill algorithms.

Boundary filling requires a lot of processing and thus encounters few problems in real time. Thus the viable alternative is scanline filling as it is very robust in nature.

Boundary Fill Algorithm:

Theory:

Boundary Fill is seed fill algorithm in which edges of the polygon are drawn. Then starting with some seed any point inside the polygon we examine the neighboring pixels to check whether the boundary pixel is reached. If boundary pixels are not reached, pixels are highlighted and process is continued until boundary pixels are reached. In this to color next pixel, both 4-connected and 8-connected methods can be used.

Procedure:

Step 1: Create a function named as Boundary_Fill with 4 parameters (x,y,B_COLOR,NEW_COLOR).

```
Boundary_Fill(int x, int y, intB_COLOR,int NEW_COLOR)
 {
color = GetPixel(x,y)
if ( (color != B_COLOR) && (color != NEW_COLOR) )
 {
```

```
SetPixel(x,y,NEW_COLOR);

Boundary_Fill(x+1,y,B_COLOR,NEW_COLOR);

Boundary_Fill(x,y+1,B_COLOR,NEW_COLOR);

Boundary_Fill(x-1,y,B_COLOR,NEW_COLOR);

Boundary_Fill(x,y-1,B_COLOR,NEW_COLOR);

 }

 }
```

Step 2: Call it recursively until the boundary pixels are reached.

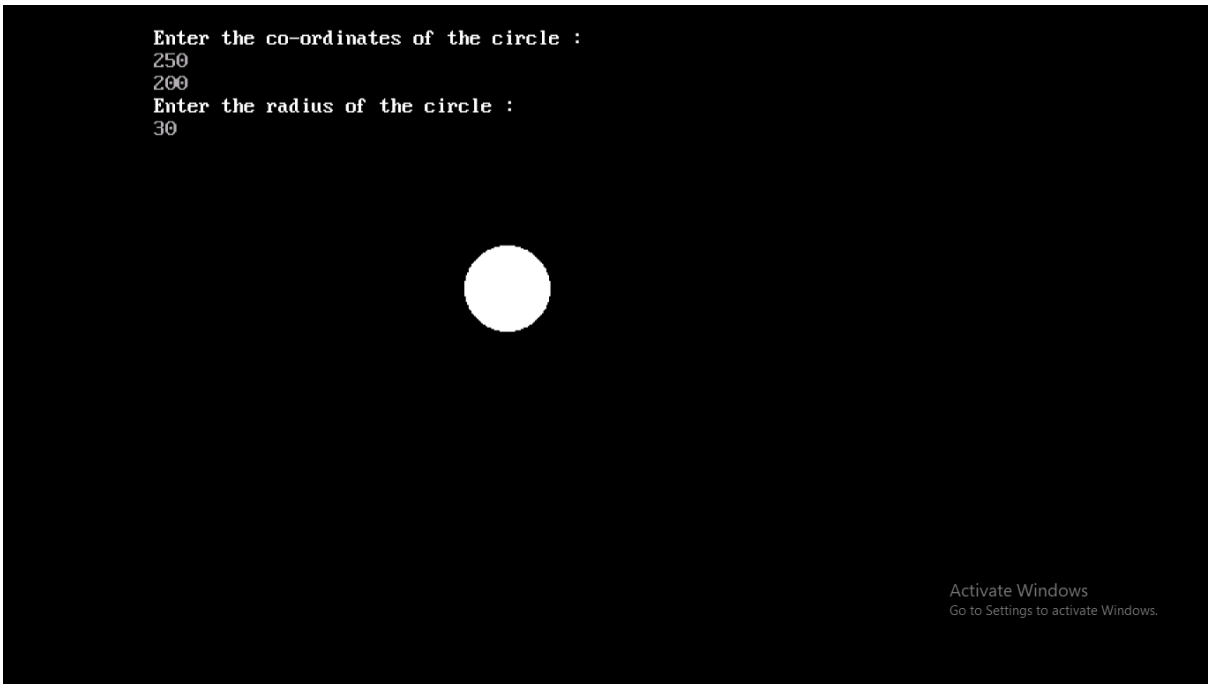
Step 3: Stop.

Program (Boundary Fill):

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <dos.h>
void bf(int xc, int yc, int b, int n)
{
   if (getpixel(xc, yc) != b && getpixel(xc, yc) != n)
   {
      putpixel(xc, yc, n);
      delay(10);
      bf(xc + 1, yc, b, n);
      bf(xc, yc + 1, b, n);
      bf(xc - 1, yc, b, n);
      bf(xc, yc - 1, b, n);
   }
}
int main()
{
   int xc, yc, r;
   int gm;
```

```
    int gd = DETECT;
    initgraph(&gd, &gm, "..\\BGI");

    printf("\nEnter the co-ordinates of the circle : \n");
    scanf("%d %d", &xc, &yc);
    printf("Enter the radius of the circle : \n");
    scanf("%d", &r);
    circle(xc, yc, r);
    bf(xc, yc, 3, 15);
    getch();
    closegraph();
    return 0;
}
```

Output:


Enter the co-ordinates of the circle :
250
200
Enter the radius of the circle :
30
Activate Windows
Go to Settings to activate Windows.

Flood Fill Algorithm:

Theory:

Flood Fill is a seed fill algorithm similar to Boundary Fill algorithm but sometimes when it is required to fill in an area that is not defined within a single color boundary we use flood fill instead of boundary fill.

For this purpose we can create a function or we can use a predefined function in the graphics.h header file which takes 3 arguments:-

floodfill(x,y,color)

Procedure:

Step 1: Create a function called as floodFill (x,y,oldcolor,newcolor)

```
voidfloodFill(intx,inty,intoldcolor,intnewcolor)
{
    if(getpixel(x,y) == oldcolor)
    {
        putpixel(x,y,newcolor);
        floodFill(x+1,y,oldcolor,newcolor);
        floodFill(x,y+1,oldcolor,newcolor);
        floodFill(x-1,y,oldcolor,newcolor);
        floodFill(x,y-1,oldcolor,newcolor);
    }
}
```

//getpixel(x,y) gives the color of specified pixel

Step 2: Repeat until the polygon is completely filled.

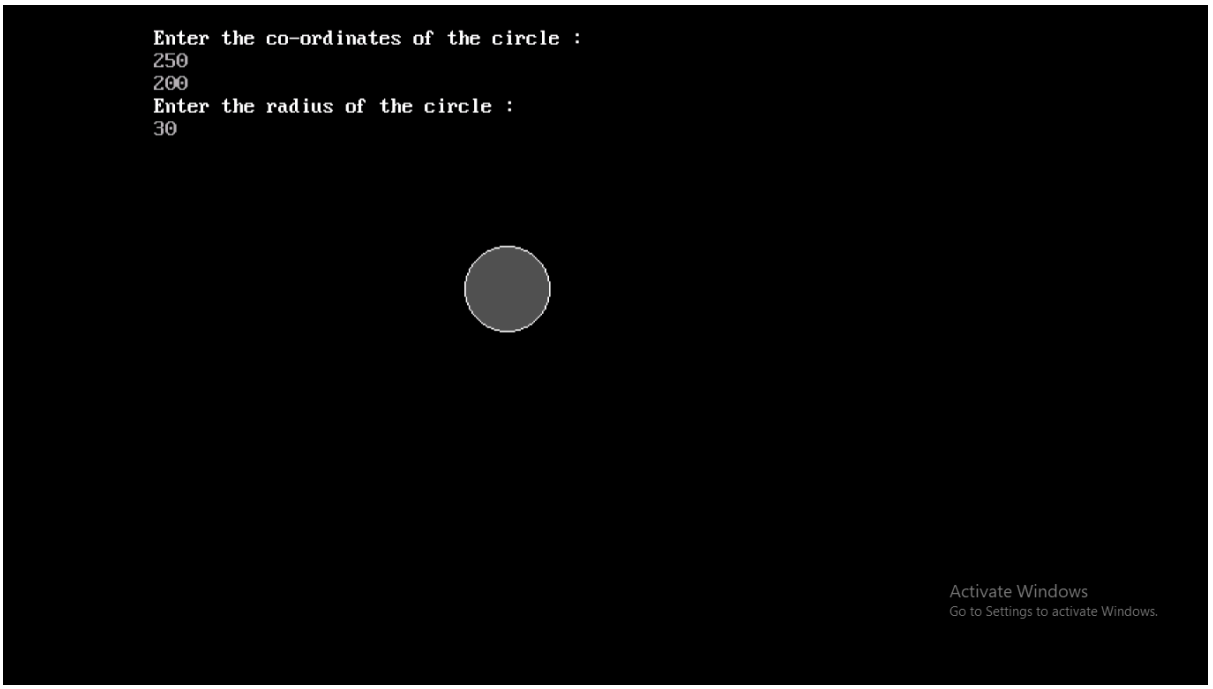
Step 3: Stop.

Program (Flood Fill):

```
#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <dos.h>

void ff(int xc, int yc, int o, int n)

{

   if (getpixel(xc, yc) == o)

   {

      putpixel(xc, yc, n);

      delay(10);

      ff(xc + 1, yc, o, n);

      ff(xc, yc + 1, o, n);

      ff(xc - 1, yc, o, n);

      ff(xc, yc - 1, o, n);

   }

}

int main()

{

   int xc, yc, r;
```

```
    int gm;
    int gd = DETECT;
    initgraph(&gd, &gm, "..\\BGI");

    printf("\nEnter the co-ordinates of the circle : \n");
    scanf("%d %d", &xc, &yc);
    printf("Enter the radius of the circle : \n");
    scanf("%d", &r);
    circle(xc, yc, r);
    ff(xc, yc, 0, 8);
    getch();
    closegraph();
    return 0;
}
```

Output:


Enter the co-ordinates of the circle :
250
200
Enter the radius of the circle :
30
Activate Windows
Go to Settings to activate Windows.

Scan Line Polygon Fill Algorithm:

Theory:

Scanline filling is basically filling up of polygons using horizontal lines or scanlines. The purpose of the SLPF algorithm is to fill (color) the interior pixels of a polygon given only the vertices of the figure. To understand Scanline, think of the image being drawn by a single pen starting from bottom left, continuing to the right, plotting only points where there is a point present in the image, and when the line is complete, start from the next line and continue.

This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections.

Procedure:

Step 1: The algorithm begins with first scan line that the polygon occupies i.eymax and proceeds line by line towards the last scan line i.eymin.

Step 2: Sort Xmax, Xmin, Ymax, Ymin of the edges of polygon along with their slopes.

Step 3: To decide which edges are getting intersected by scan line we are making use of Ymax of particular edge.

Step 4: Every time we are decreasing scan line by 1 from Ymax to Ymin of the polygon.

Step 5: It may happen that the edge which we have selected to find intersection point may get finished i.eYmin goes below the Ymin of selected edge. In that case discard the edge and select next edge from the sorted table and continue to sac line.

Step 6: Decreasing Ymax by 1 unit i.e Ymax-1 to find corresponding value of intersection point. Xnew=Xold+1/m

Step 7: Find the intersection of scan line with every edge of polygon.

Program:

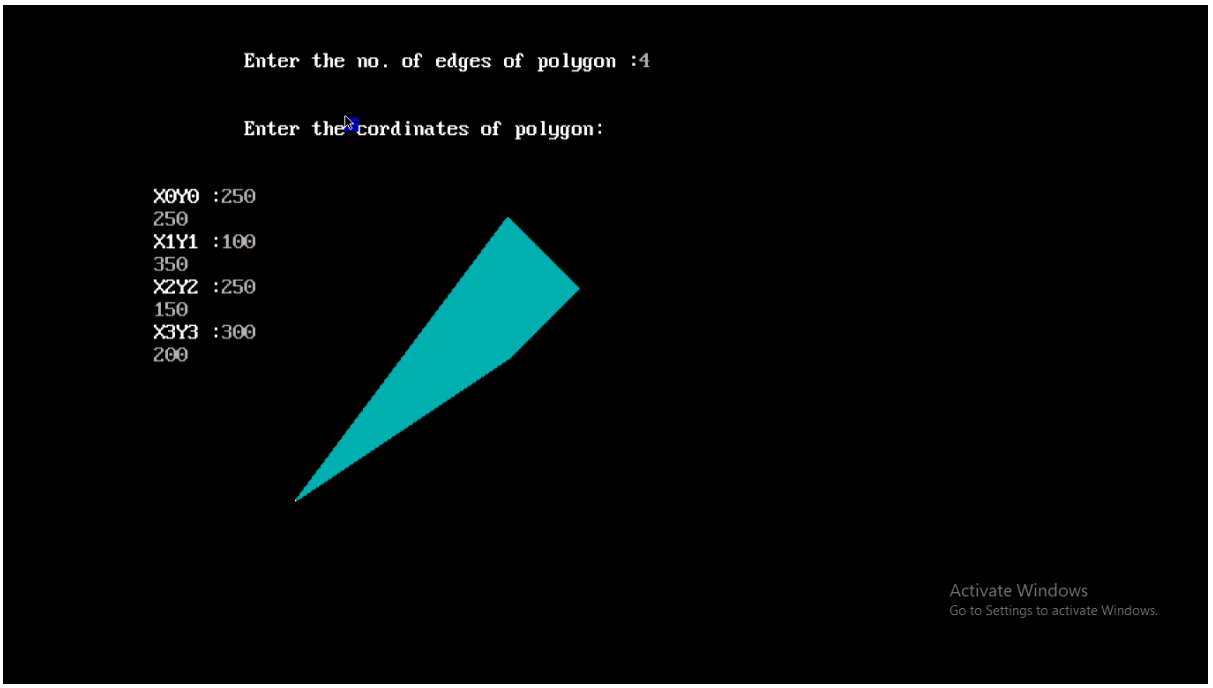
```
#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <dos.h>

void main()

{

   int n, i, j, k, gd = DETECT, gm, dy, dx;



   int x, y, temp;

   int a[20][2], xi[20];

   float slope[20];

   clrscr();

   initgraph(&gd, &gm, "..\\BGI");

   printf("\n\n\tEnter the no. of edges of polygon :");

   scanf("%d", &n);

   printf("\n\n\tEnter the cordinates of polygon:\n\n\n");

   for (i = 0; i < n; i++)

   {

        printf("X%dY%d :", i, i);
```

```
    scanf("%d %d", &a[i][0], &a[i][1]);
}
a[n][0] = a[0][0];
a[n][1] = a[0][1];
/*- draw polygon-*/
for (i = 0; i < n; i++)
{
    line(a[i][0], a[i][1], a[i + 1][0], a[i + 1][1]);
}
for (i = 0; i < n; i++)
{
    dy = a[i + 1][1] - a[i][1];
    dx = a[i + 1][0] - a[i][0];
    if (dy == 0)
        slope[i] = 1.0;
    if (dx == 0)
        slope[i] = 0.0;
    if ((dy != 0) && (dx != 0)) /*- calculate inverse slope-*/
    {
        slope[i] = (float)dx / dy;
```

```
    }
}
for (y = 0; y < 480; y++)
{
    k = 0;
    for (i = 0; i < n; i++)
    {
        if (((a[i][1] <= y) && (a[i + 1][1] > y)) ||
            ((a[i][1] > y) && (a[i + 1][1] <= y)))
        {
            xi[k] = (int)(a[i][0] + slope[i] * (y - a[i][1]));
            k++;
        }
    }

    for (j = 0; j < k - 1; j++) /*- Arrange x-intersections in
            order-*/
        for (i = 0; i < k - 1; i++)
        {
            if (xi[i] > xi[i + 1])
```

```
            {
               temp = xi[i];
               xi[i] = xi[i + 1];
               xi[i + 1] = temp;
            }
         }
      setcolor(35);
      for (i = 0; i < k; i += 2)
      {
         line(xi[i], y, xi[i + 1] + 1, y);
         delay(20);
      }
   }
   getch();
   closegraph();
}
```

Output:


Enter the no. of edges of polygon :4
Enter the cordinates of polygon:
X0Y0 :250
250
X1Y1 :100
350
X2Y2 :250
150
X3Y3 :300
200
Activate Windows
Go to Settings to activate Windows.

Experiment No.6
Implement Character generation Bit map Method to generate Characters in computer graphics.
Date of Performance:29/08/24
Date of Submission:04/09/24

Aim:

Implement character generation: Bitmap method to generate initial characters of your name.

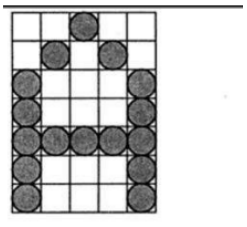
Theory:

Bitmap Method:

· Bitmap method is a called dot-matrix method as the name suggests this method use array of bits for generating a character. These dots are the points for array whose size is fixed.

· In bit matrix method when the dots is stored in the form of array the value 1 in array represent the characters i.e. where the dots appear we represent that position with numerical value 1 and the value where dots are not present is represented by 0 in array.

· It is also called dot matrix because in this method characters are represented by an array of dots in the matrix form. It is a two dimensional array having columns and rows. A 5x7 array is commonly used to represent characters. However 7x9 and 9x13 arrays are also used. Higher resolution devices such as inkjet printer or laser printer may use character arrays that are over 100x100.



Character A in 5 × 7 dot matrix format

Procedure:

1. 2D character generators create two-dimensional characters that can be used in a variety of applications, such as video games or animated films.
2. 3D character generators create three-dimensional characters that can be used in a variety of applications, such as video games or animated films.

Program:

```
#include<stdio.h>

#include<conio.h>

#include<dos.h>

#include<graphics.h>



void main(){

int i,j,k,x,y;

int gd=DETECT,gm;

int ch1[][10]={

{1,1,1,1,0,0,0,0,1,1},

{1,1,1,1,0,0,0,1,1,1},

{0,1,1,1,0,0,1,1,1,0},

{0,1,1,1,0,1,1,1,0,0},

{0,1,1,1,1,1,1,0,0,0},

{0,1,1,1,1,1,1,0,0,0},

{0,1,1,1,0,1,1,1,0,0},
```

```
{0,1,1,1,0,0,1,1,1,0},

{1,1,1,1,0,0,0,1,1,1},

{1,1,1,1,0,0,0,0,1,1},

};
```

```
int ch2[][10]={

{0,0,0,1,1,1,1,0,0,0},

{0,0,1,1,1,1,1,1,0,0},

{0,0,1,1,0,0,1,1,0,0},

{0,0,1,1,0,0,1,1,0,0},

{0,1,1,1,1,1,1,1,1,0},

{0,1,1,1,1,1,1,1,1,0},

{0,1,1,1,0,0,1,1,1,0},

{0,1,1,1,0,0,1,1,1,0},

{1,1,1,0,0,0,0,1,1,1},

{1,1,1,0,0,0,0,1,1,1},

};
```

```
int ch3[][10]={

{1,1,1,1,1,1,0,0,0,0},
```

```
{1,1,1,1,1,1,1,1,0,0},

{1,1,1,0,0,0,1,1,1,0},

{1,1,1,0,0,1,1,1,0,0},

{1,1,1,1,1,1,1,1,0,0},

{1,1,1,1,1,1,0,0,0,0},

{1,1,1,0,1,1,1,0,0,0},

{1,1,1,0,0,1,1,1,0,0},

{1,1,1,0,0,0,1,1,1,0},

{1,1,1,0,0,0,0,1,1,1} };
```

```
int ch4[][10]={

{0,0,0,1,1,1,1,0,0,0},

{0,0,1,1,1,1,1,1,0,0},

{0,0,1,1,0,0,1,1,0,0},

{0,0,1,1,0,0,1,1,0,0},

{0,1,1,1,1,1,1,1,1,0},

{0,1,1,1,1,1,1,1,1,0},

{0,1,1,1,0,0,1,1,1,0},

{0,1,1,1,0,0,1,1,1,0},

{1,1,1,0,0,0,0,1,1,1},

{1,1,1,0,0,0,0,1,1,1} };
```

```
int ch5[][10]={

{1,1,1,0,0,0,0,1,1,1},

{1,1,1,1,0,0,0,1,1,1},

{1,1,1,1,0,0,0,1,1,1},

{1,1,1,1,1,0,0,1,1,1},

{1,1,1,1,1,0,0,1,1,1},

{1,1,1,0,1,1,0,1,1,1},

{1,1,1,0,1,1,0,1,1,1},

{1,1,1,0,0,1,1,1,1,1},

{1,1,1,0,0,1,1,1,1,1},

{1,1,1,0,0,0,1,1,1,1} };


initgraph(&gd,&gm, "..\\BGI");

setbkcolor(BLUE);

for(k=0;k<6;k++)

{

for(i=0;i<10;i++)

{

for(j=0;j<10;j++)

{
```

```
if(k==0)
{
if(ch1[i][j]==1)
{
putpixel(j+200,i+230,YELLOW);
}
}
if(k==1)
{
if(ch2[i][j]==1)
{
putpixel(j+250,i+230,YELLOW);
}
}
if(k==2)
{
if(ch3[i][j]==1)
{
putpixel(j+300,i+230,YELLOW);
}
}
if(k==3)
```

```
{
if(ch4[i][j]==1)
{


putpixel(j+350,i+230,YELLOW);
}
}
if(k==4)
{
if(ch5[i][j]==1)
{
putpixel(j+400,i+230,YELLOW);
}
}
/*if(k==5)
{
if(ch6[i][j]==1)
{
putpixel(j+450,i+230,RED);
}
} */
}
```

```
delay(200);

}

}

getch();

closegraph();









}
```

Output:

CSL303: Computer Graphics Lab
Name of Student: karan pawar Class: SE-2

Batch: C Roll No: 61

Experiment No.7
Implement 2D transformation Translation, Scaling, Rotation (Equilateral Triangle)
Date of Performance:05/09/24
Date of Submission: 12/09/24

Aim:

Implement 2D Transformations: Translation, Scaling, Rotation.

Introduction:

Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation. Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

Theory:

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

X' = X + tx

Y' = Y + ty

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

P=[X][Y] p' = [X′][Y′]T = [tx][ty]

We can write it as −

P' = P + T

Rotation: In rotation, we rotate the object at particular angle θ (theta) from its origin. From the following figure, we can see that the point P(X, Y) is located at angle φ from the horizontal X coordinate with distance r from the origin. Let us suppose you want to rotate it at the angle θ. After rotating it to a new location, you will get a new point P' (X', Y').

Scaling: To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result. Let us assume that the original coordinates are (X, Y), the scaling factors are (SX, SY), and the produced coordinates are (X', Y'). This can be mathematically represented as shown below −

X' = X . SX and Y' = Y . SY

Procedure:

Step 1: Start

Step 2: Initialize the graphics mode.

Step 3: Construct a 2D object (use Drawpoly()) e.g. (x,y)

Step 4: A) Translation

a. Get the translation value tx, ty

b. Move the 2d object with tx, ty (x'=x+tx,y'=y+ty)

c. Plot (x',y')

Step 5: B) Scaling

a. Get the scaling value Sx,Sy

b. Resize the object with Sx,Sy (x'=x*Sx,y'=y*Sy)

c. Plot (x',y')

Step 6: C) Rotation

a. Get the Rotation angle

b. Rotate the object by the angle ϕ

$$x' = x \cos \phi - y \sin \phi$$

$$y' = x \sin \phi - y \cos \phi$$

c. Plot (x',y')

Program:

```
#include <graphics.h>

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>


void main() {

   int gm, gd = DETECT;

   int x1, x2, x3, y1, y2, y3, nx1, nx2, nx3, ny1, ny2, ny3, c;

   int sx, sy, xt, yt, r;

   float t;


   initgraph(&gd, &gm, "..//BGI");


   printf("\tProgram for basic transformations");

   printf("\n\tEnter the points of the triangle (x1, y1, x2, y2, x3, y3):\n");

   setcolor(1);

   scanf("%d%d%d%d%d%d", &x1, &y1, &x2, &y2, &x3, &y3);
```

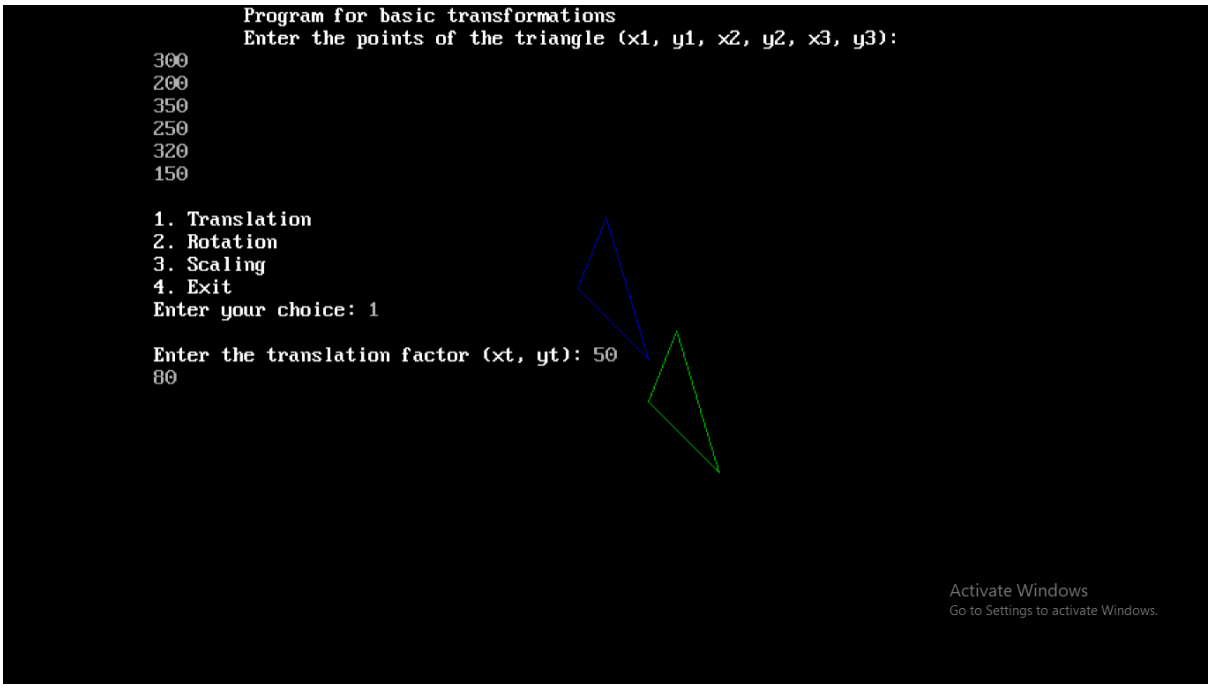
```
line(x1, y1, x2, y2);

line(x2, y2, x3, y3);

line(x3, y3, x1, y1);


getch();


printf("\n1. Translation\n2. Rotation\n3. Scaling\n4. Exit");

printf("\nEnter your choice: ");

scanf("%d", &c);


switch (c) {

   case 1:

      printf("\nEnter the translation factor (xt, yt): ");

      scanf("%d%d", &xt, &yt);


      nx1 = x1 + xt;

      ny1 = y1 + yt;

      nx2 = x2 + xt;

      ny2 = y2 + yt;

      nx3 = x3 + xt;
```

```
    ny3 = y3 + yt;


    setcolor(2);
    line(nx1, ny1, nx2, ny2);
    line(nx2, ny2, nx3, ny3);
    line(nx3, ny3, nx1, ny1);
    getch();
    break;


case 2:
    printf("\nEnter the angle of rotation (in degrees): ");
    scanf("%d", &r);
    t = (3.14159 * r) / 180;


    nx1 = (int)(x1 * cos(t) - y1 * sin(t));
    ny1 = (int)(x1 * sin(t) + y1 * cos(t));
    nx2 = (int)(x2 * cos(t) - y2 * sin(t));
    ny2 = (int)(x2 * sin(t) + y2 * cos(t));
    nx3 = (int)(x3 * cos(t) - y3 * sin(t));
    ny3 = (int)(x3 * sin(t) + y3 * cos(t));
```

```
    setcolor(3);
    line(nx1, ny1, nx2, ny2);
    line(nx2, ny2, nx3, ny3);
    line(nx3, ny3, nx1, ny1);
    getch();
    break;

case 3:
    printf("\nEnter the scaling factors (sx, sy): ");
    scanf("%d%d", &sx, &sy);

    nx1 = x1 * sx;
    ny1 = y1 * sy;
    nx2 = x2 * sx;
    ny2 = y2 * sy;
    nx3 = x3 * sx;
    ny3 = y3 * sy;

    setcolor(4);
```

```
        line(nx1, ny1, nx2, ny2);
        line(nx2, ny2, nx3, ny3);
        line(nx3, ny3, nx1, ny1);
        getch();
        break;

    case 4:
        break;

    default:
        printf("Invalid choice!");
  }

  closegraph();
}
```

Output:


Program for basic transformations
Enter the points of the triangle (x1, y1, x2, y2, x3, y3):
300
200
350
250
320
150
1. Translation
2. Rotation
3. Scaling
4. Exit
Enter your choice: 1
Enter the translation factor (xt, yt): 50
80
Activate Windows
Go to Settings to activate Windows.

Experiment No.8
Implement Curve: Bezier Curve for 4 Control points (Midpoint approach).
Date of Performance:12/09/24
Date of Submission:19/09/24

Aim: Implement Curve: Bezier for n control points.

Introduction:

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.

Bezier Curve:

Theory:

Bezier curve is a mathematically defined curve used in two-dimensional graphic applications like abode illustrator,inkscape etc.The curve is defined by four points: the initial position and the terminating position i.e P0 and P3 respectively (which are called "anchors") and two separate middle points i.e P1 and P2(which are called "handles").Bezier curves are frequently used in computer graphics, animation, modeling etc.

Procedure:

Step 1: Start.

Step 2: Read the number of control points(no) and set the number n=no-1

Step 3: for I as 0 to n do step 5 and 6

Step 4: Read the end point to xk[i] and yk[i]

Step 5: Set i=i+1

Step 6: Set cpx=xk[0] and cpy=yk[0]

Step 7: for u is 0.1 to 0.3 repeat as follows

Step 8: Set xu as 0 and yu as 0

Step 9:for I =n to 0 repeat as follows

Step 10: If I is not equal 0 then set xu=xu+xk[i]*Bezier(n,i) * u^i*(1-u)^n-i

Else set xu=xu+xk[i]*ezier(n,i)*(1-u)^n-i

Step 11:fori=n to 0 do

Step 12: If I is not equal to 0 then set yu=yu+yk[i]*Bezier(n,i)*u^i(1-u)^i-1

Else set yu=yu+yk[i]*Bezier(n,i) (1-u)^n-i

Step 13: Draw the curve based on the values calculated

Step 14: Input the control points and interval d. set n=d-1

Step 15: Inside the Bezier function calculate n!/(i!)*(n-1)! By calling the factorial function for each variable.

Step 16: In the factorial function, initialize fact=1 and get the number into i.

Step 17: Calculate fact=fact*I recursively and decrement I till 1 is reached.

Step 18: End.

Program:

```
// Bezier Curve

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<dos.h>

void main()

{

int gd=DETECT,gm;

int x[4],y[4],px,py,i,n;

double t;

clrscr();


initgraph (&gd, &gm, "..\\bgi");

printf("Enter the no of control points");

scanf("%d",&n);

printf("Enter the control points of bezier curve: ");


for(i=0;i<n;i++)

{ scanf("%d%d",&x[i],&y[i]);
```

```
putpixel(x[i],y[i],4); }

for(t=0.0;t<=1.0;t+=0.001){

px = ((1 - t) * (1 - t) * (1 - t) * x[0]) +

    (3 * t * (1 - t) * (1 - t) * x[1]) +

    (3 * t * t * (1 - t) * x[2]) +

    (t * t * t * x[3]);


py = ((1 - t) * (1 - t) * (1 - t) * y[0]) +

    (3 * t * (1 - t) * (1 - t) * y[1]) +

    (3 * t * t * (1 - t) * y[2]) +

    (t * t * t * y[3]);


putpixel(px,py,WHITE);

delay(2);

}

getch();

closegraph();

}
```

Output:


Enter the no of control points4
Enter the control points of bezier curve: 350
300
250
200
150
100
350
200

Experiment No.9
Implement the Fractal generation method – Koch Curve
Date of Performance:12/09/24
Date of Submission:19/0/24

Name of Student: karan pawar Class: SE-2

Batch: C Roll No: 61

Aim: Implement Fractals (Koch Curve)

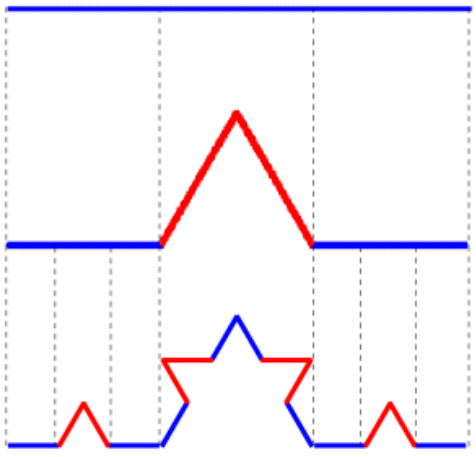
Introduction:

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.

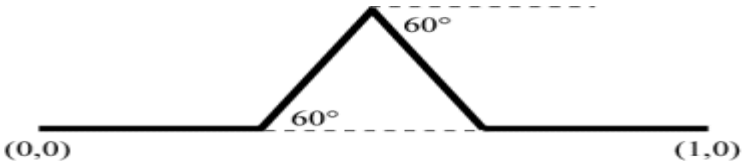
The progression for the area of the snowflake converges to 8/5 times the area of the original triangle, while the progression for the snowflake's perimeter diverges to infinity. Consequently, the snowflake has a finite area bounded by an infinitely long line.

Theory:

Begin with a straight line (the blue segment in the top figure). Divide it into three equal segments and replace the middle segment by the two sides of an equilateral triangle of the same length as the segment being removed (the two red segments in the middle figure). Now repeat, taking each of the four resulting segments, dividing them into three equal parts and replacing each of the middle segments by two sides of an equilateral triangle (the red segments in the bottom figure). Continue this construction.



The Koch curve is the limiting curve obtained by applying this construction an infinite number of times. For a proof that this construction does produce a "limit" that is an actual curve, i.e. the continuous image of the unit interval, see the text by Edgar.



The first iteration for the Koch curve consists of taking four copies of the unit horizontal line segment, each scaled by **r** = 1/3. Two segments must be rotated by 60°, one counterclockwise and one clockwise. Along with the required translations, this yields the following IFS.

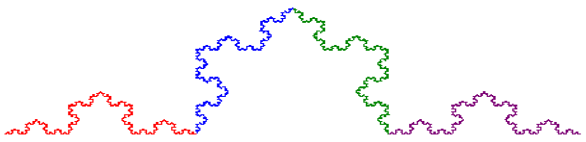
$$f_1(\mathbf{x}) = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \mathbf{x} \qquad \text{scale by } \mathbf{r}$$

$$f_2(\mathbf{x}) = \begin{bmatrix} 1/6 & -\sqrt{3}/6 \\ \sqrt{3}/6 & 1/6 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1/3 \\ 0 \end{bmatrix} \qquad \text{scale by } \mathbf{r}\text{, rotate by } 60°$$

$$f_3(\mathbf{x}) = \begin{bmatrix} 1/6 & \sqrt{3}/6 \\ -\sqrt{3}/6 & 1/6 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1/2 \\ \sqrt{3}/6 \end{bmatrix} \qquad \text{scale by } \mathbf{r}\text{, rotate by } -60°$$

$$f_4(\mathbf{x}) = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/3 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 2/3 \\ 0 \end{bmatrix} \qquad \text{scale by } \mathbf{r}$$

The fixed attractor of this IFS is the Koch curve.

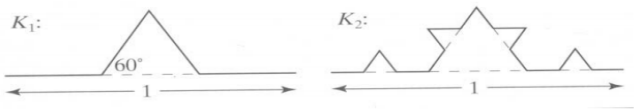


Procedure:

Recursively:

- Divide line into 3 equal parts

- Replace middle section with triangular bump, sides of length 1/3
- New length = 4/3



Pseudo Code:

To draw Kn:

If (n equals 0)

draw straight line

Else

{

Draw Kn-1

Turn left 60°

Draw Kn-1

Turn right 120°

Draw Kn-1

Turn left 60°

Draw Kn-1

}

Program:

```
#include <graphics.h>

#include <conio.h>

#include <iostream.h>

#include <math.h>


//using namespace std;


// Function prototypes

void k_curve(long double x, long double y, long double length, long double angle, int n_order);

void line1(int x1, int y1, int x2, int y2);

void swap(int &a, int &b); // Prototype for swap function


int main() {

   int gd = DETECT, gm;

   long double x = 0;

   long double y = 0;

   long double l = 0;

   long double angle = 0;
```

```
int n = 0;


cout << "Starting point of the line (x, y):" << endl;

cout << "_____" << endl;

cout << "Enter the value of x: ";

cin >> x;

cout << "Enter the value of y: ";

cin >> y;

cout << "Enter the length of the line: ";

cin >> l;

cout << "Angle of the line with x-axis: ";

cin >> angle;

cout << "Enter the order of curve (n): ";

cin >> n;


initgraph(&gd, &gm, "..\\bgi");

k_curve(x, y, l, angle, n);

getch();

closegraph();

return 0;
```

```
}

void k_curve(long double x, long double y, long double length, long double angle, int n_order) {
   if (n_order > 0) {
        length /= 3;
        k_curve(x, y, length, angle, n_order - 1);
      x += (length * cos(angle * (M_PI / 180)));
      y += (length * sin(angle * (M_PI / 180)));

      k_curve(x, y, length, angle - 60, n_order - 1);
      x += (length * cos((angle - 60) * (M_PI / 180)));
      y += (length * sin((angle - 60) * (M_PI / 180)));

      k_curve(x, y, length, angle + 60, n_order - 1);
      x += (length * cos((angle + 60) * (M_PI / 180)));
      y += (length * sin((angle + 60) * (M_PI / 180)));

      k_curve(x, y, length, angle, n_order - 1);
   } else {
```

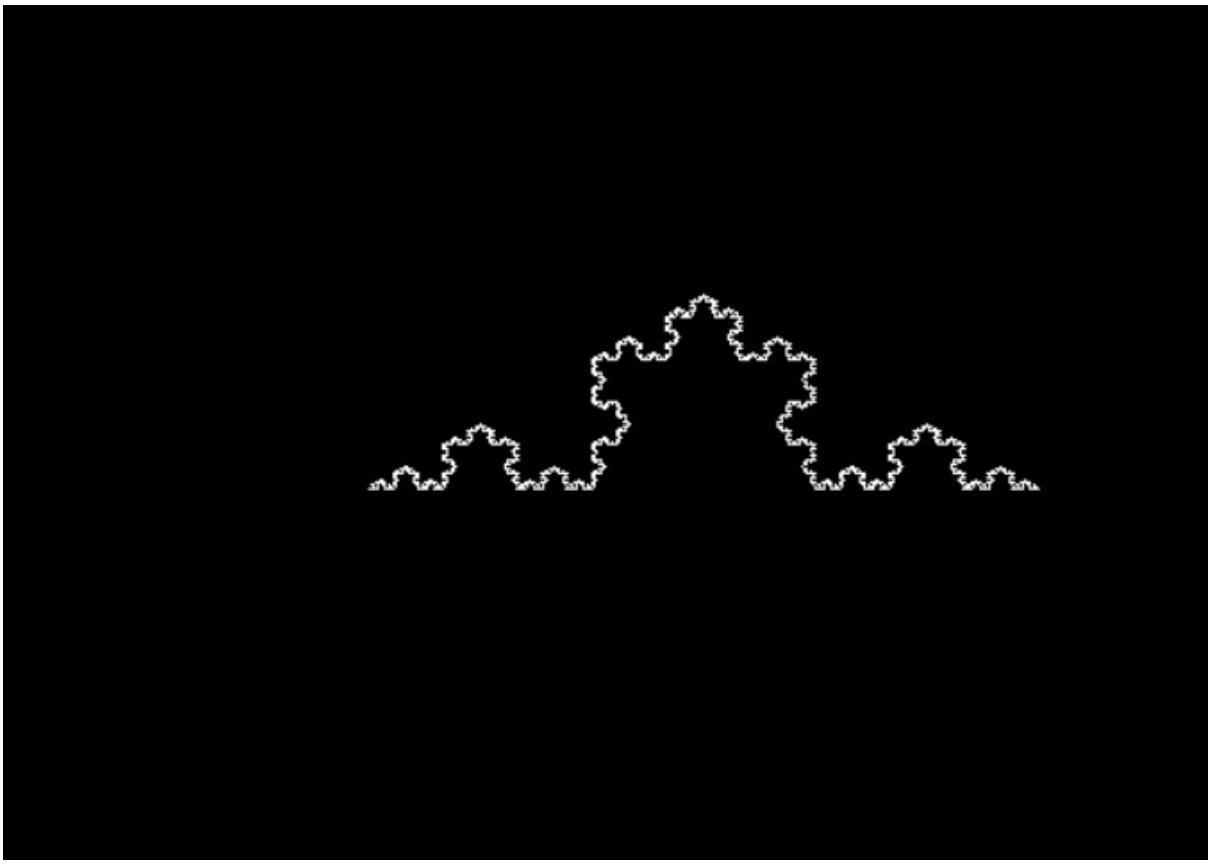
```
        line1(x, y, (int)(x + length * cos(angle * (M_PI / 180)) + 0.5),
                    (int)(y + length * sin(angle * (M_PI / 180))));
    }
}

void line1(int x1, int y1, int x2, int y2) {
    int color = getcolor();
    if (x1 > x2) {
        swap(x1, x2);
        swap(y1, y2);
    }
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);
    int inc_dec = (y2 >= y1) ? 1 : -1;

    if (dx > dy) {
        int two_dy = (2 * dy);
        int two_dy_dx = (2 * (dy - dx));
        int p = (2 * dy) - dx;
        int x = x1, y = y1;
```

```
    putpixel(x, y, color);
    while (x < x2) {
        x++;
        if (p < 0) {
            p += two_dy;
        } else {
            y += inc_dec;
            p += two_dy_dx;
        }
        putpixel(x, y, color);
    }
} else {
    int two_dx = (2 * dx);
    int two_dx_dy = (2 * (dx - dy));
    int p = (2 * dx) - dy;
    int x = x1, y = y1;
    putpixel(x, y, color);
    while (y != y2) {
        y += inc_dec;
        if (p < 0) {
```

```
            p += two_dx;
        } else {
            x++;
            p += two_dx_dy;
        }
        putpixel(x, y, color);
    }
  }
}

// Definition of the swap function
void swap(int &a, int &b) {
  int temp = a;
  a = b;
  b = temp;
}
```

Output:

```
Starting point of the line (x, y):
____________________
Enter the value of x: 350
Enter the value of y: 250
Enter the length of the line: 300
Angle of the line with x-axis: 0
Enter the order of curve (n): 5_
```



Experiment No.10
Program to perform Animation.
Date of Performance:27/09/24
Date of Submission: 02/10/24

Aim: Program to perform animation.

Theory: Animation is a filmmaking technique by which still images are manipulated to create moving images. In traditional animation, images are drawn or painted by hand on transparent celluloid sheets (cels) to be photographed and exhibited on film. Animation has been recognized as an artistic medium, specifically within the entertainment industry. Many animations are computer animations made with computer-generated imagery (CGI).

code:

```
#include<graphics.h>

#include<stdio.h>

#include<conio.h>

#include<dos.h>


void mount();


void main()

{


int a,b,c,d;

int gdriver = DETECT,gmode;

initgraph(&gdriver,&gmode,"C:\\Turboc3\\BGI");
```

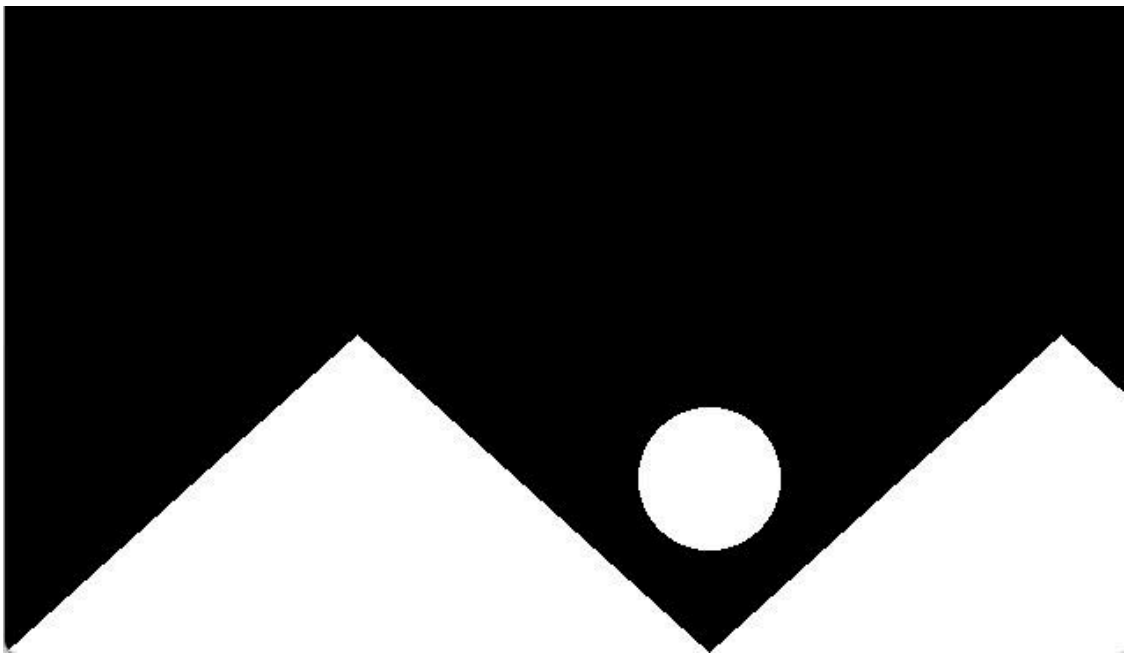
```
mount();


//sun rise
b=0;
for(a=0;a<150;a++)
{
setfillstyle(1,WHITE);
circle(400,600-b,40);
floodfill(400,561-b,WHITE);
b=b+3;
delay(30);
cleardevice();
mount();
}
setfillstyle(1,WHITE);
circle(400,600-b,40);
floodfill(400,561-b,WHITE);
delay(1000);


//circle(400,150,50);
```

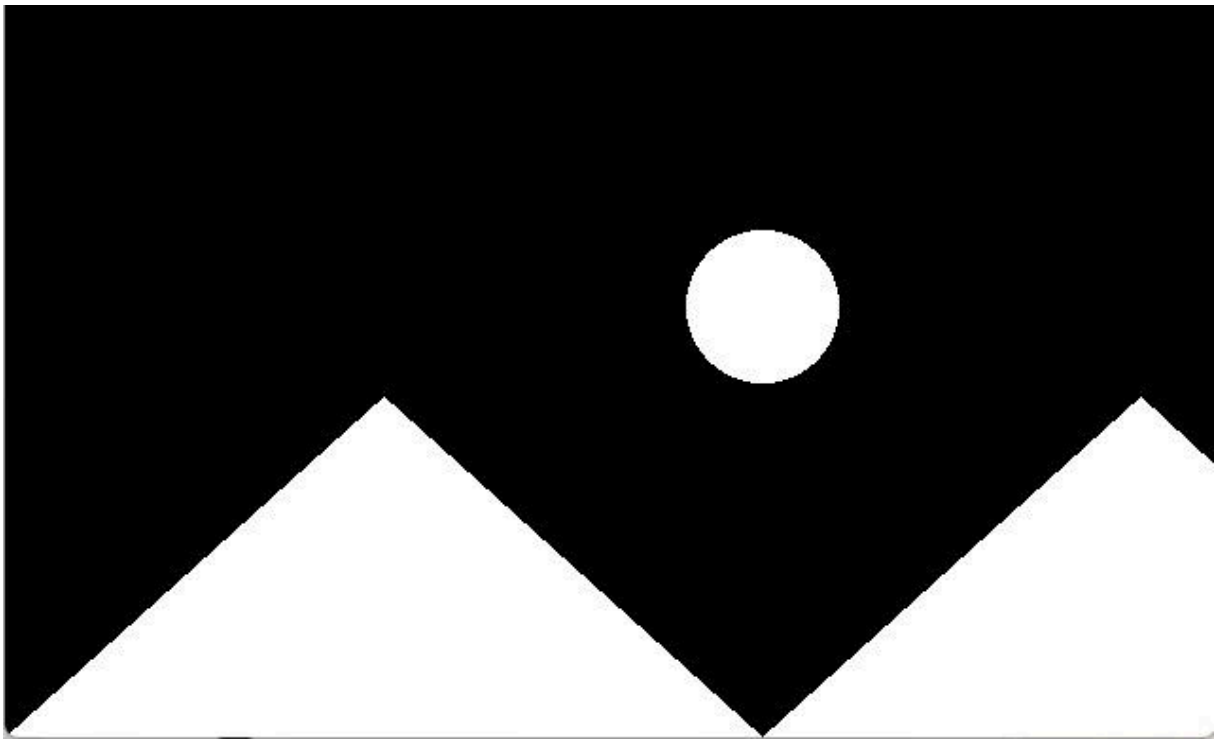
```
line(350,130,330,120);

line(360,115,340,100);

line(378,105,360,85);


line(450,130,470,120);

line(440,115,460,100);

line(420,105,435,85);


line(445,170,470,185);

line(435,185,455,205);

line(420,200,430,222);


line(352,170,325,175);

line(360,185,340,200);

line(380,195,365,215);



line(350,150,325,150);

line(400,100,400,75);
```

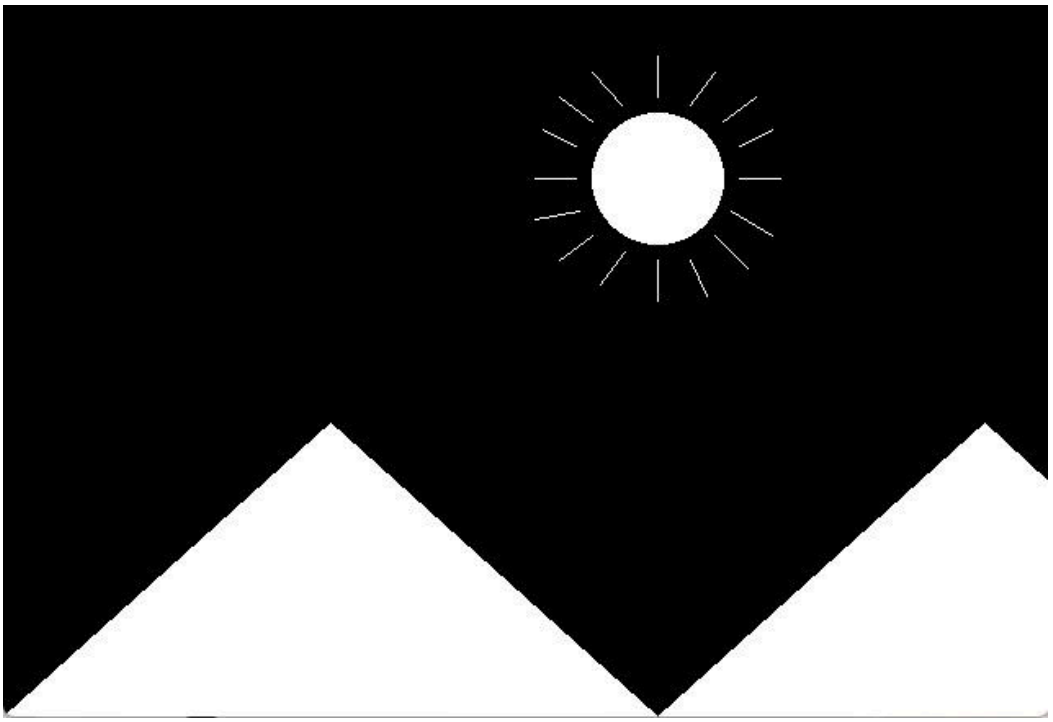
```
line(450,150,475,150);

line(400,200,400,225);


 delay(2000);

//sun set

c=0;

for(a=0;a<150;a++)

{

setfillstyle(1,WHITE);

circle(400,600-b+c,40);

floodfill(400,561-b+c,WHITE);

c=c+3;

delay(30);

cleardevice();

mount();

}




getch();

}
```

```
void mount()

{


setfillstyle(1,WHITE);

//mountain

line(0,480,200,300);

line(200,300,400,480);

line(400,480,600,300);

line(600,300,800,480);

floodfill(100,470,WHITE);

floodfill(550,350,WHITE);


}
```

output:







CSL303: Computer Graphics Lab
Name of Student: karan pawar Class: SE-2

Batch: C Roll No: 61

Experiment No.11
Mini project to perform using C/C++/java/open GL/Blender/any other tool (3/4 students per group)
Date of Performance:
Date of Submission: