NAME: SINGH SUDHAM DHARMENDRA

ROLL NO.: AIMLD50

BRANCH: CSE (AI & ML)

SUBJECT: COMPUTER GRAPHICS

TOPIC: EXPERIMENT NO. 1

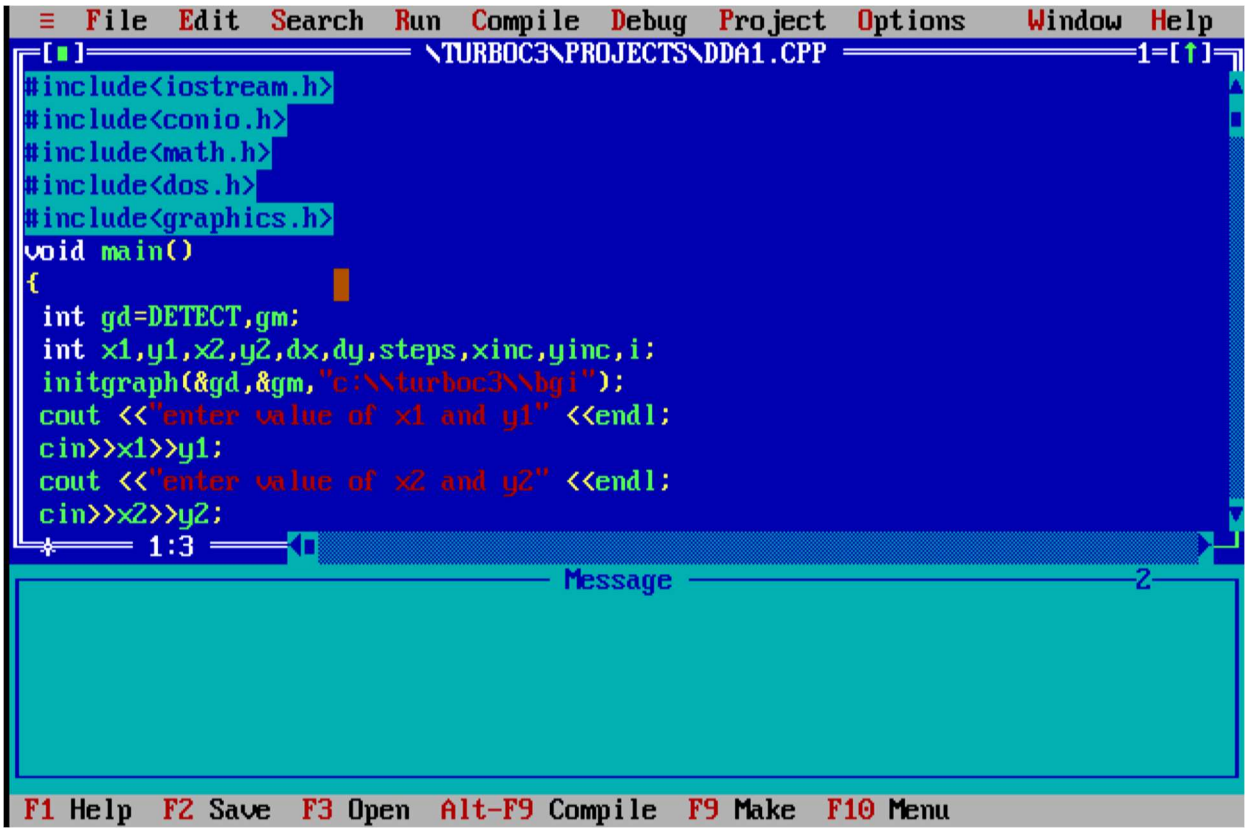
EXPERIMENT NO. 1

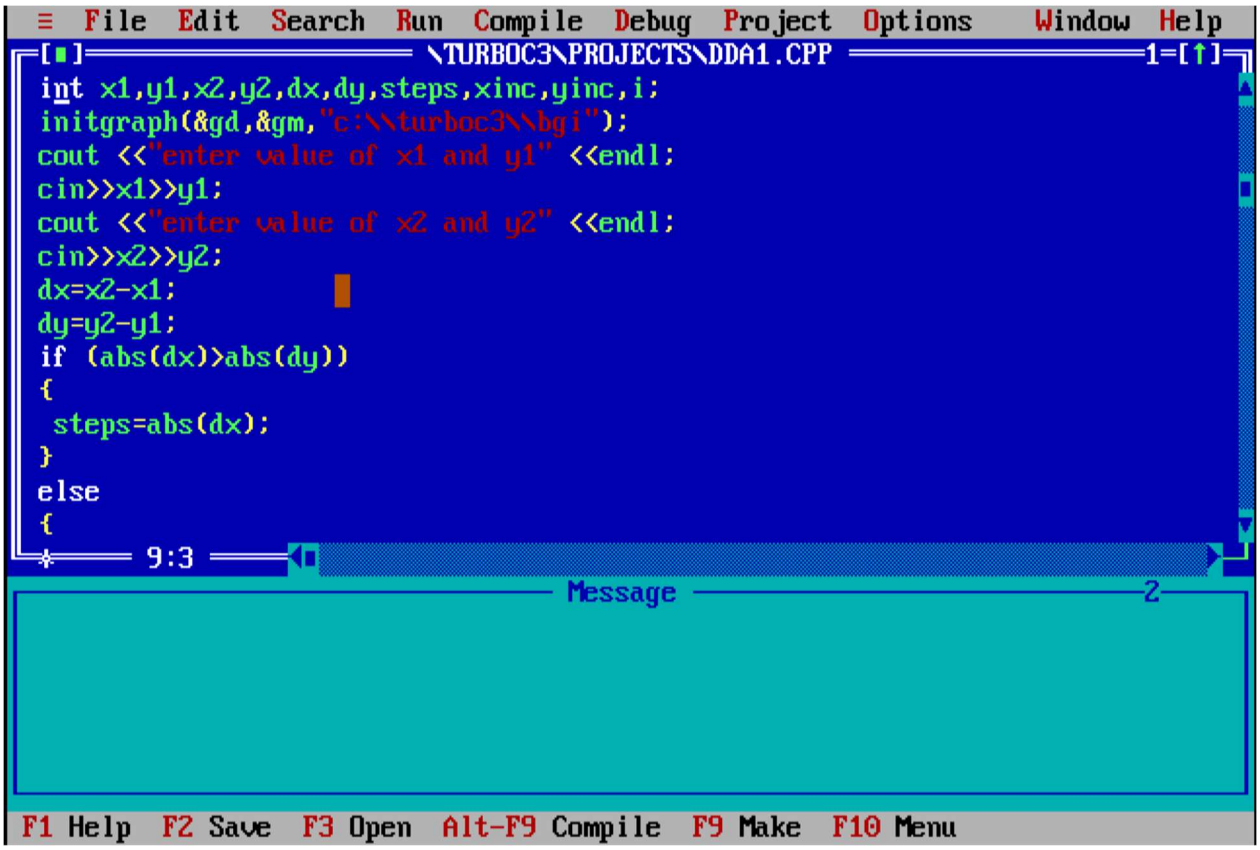
AIM: WRITE A PROGRAM IN C++ FOR DDA LINE DRAWING ALGORITHM.

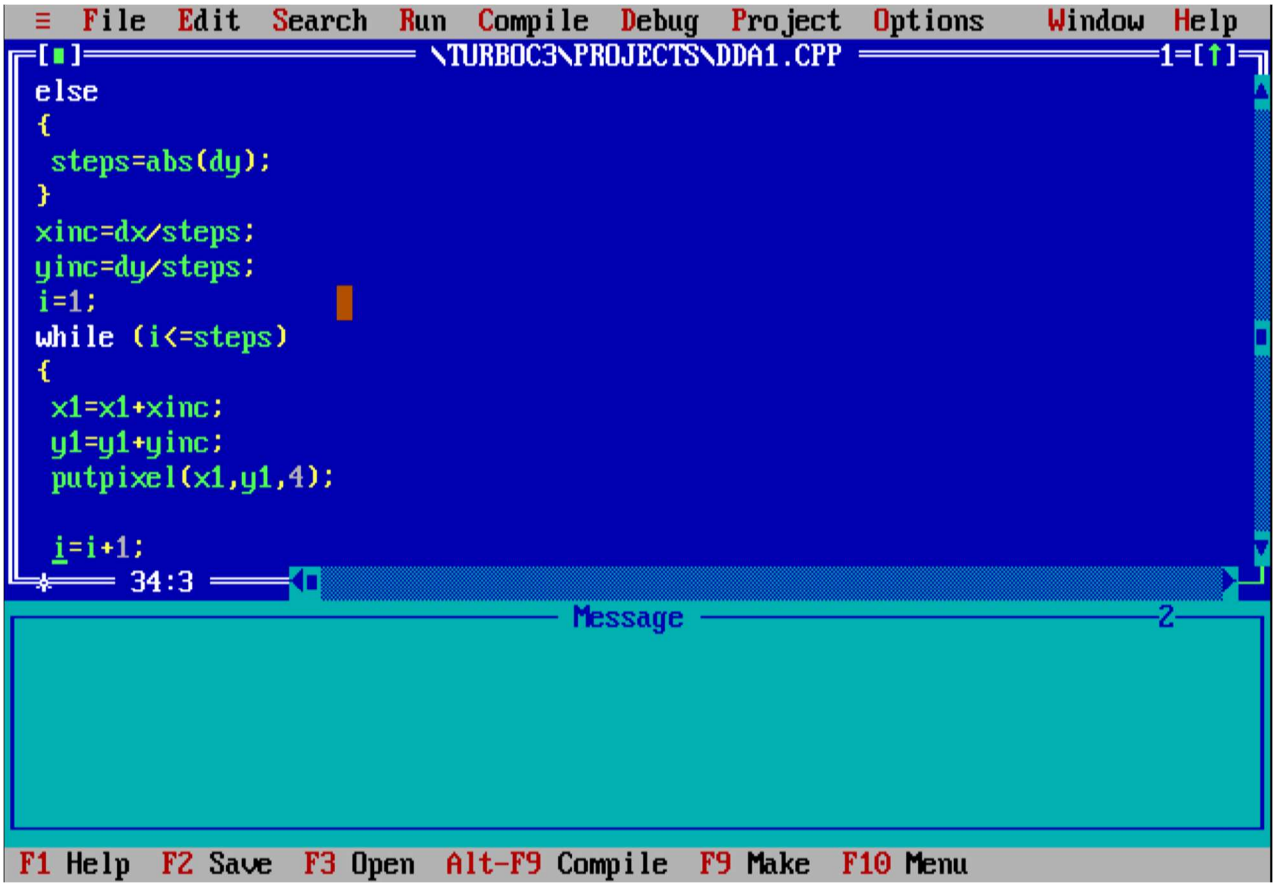
(i)TO DRAW A THIN LINE.

SOFTWARE USED: TURBO C++

SOURCE CODE:

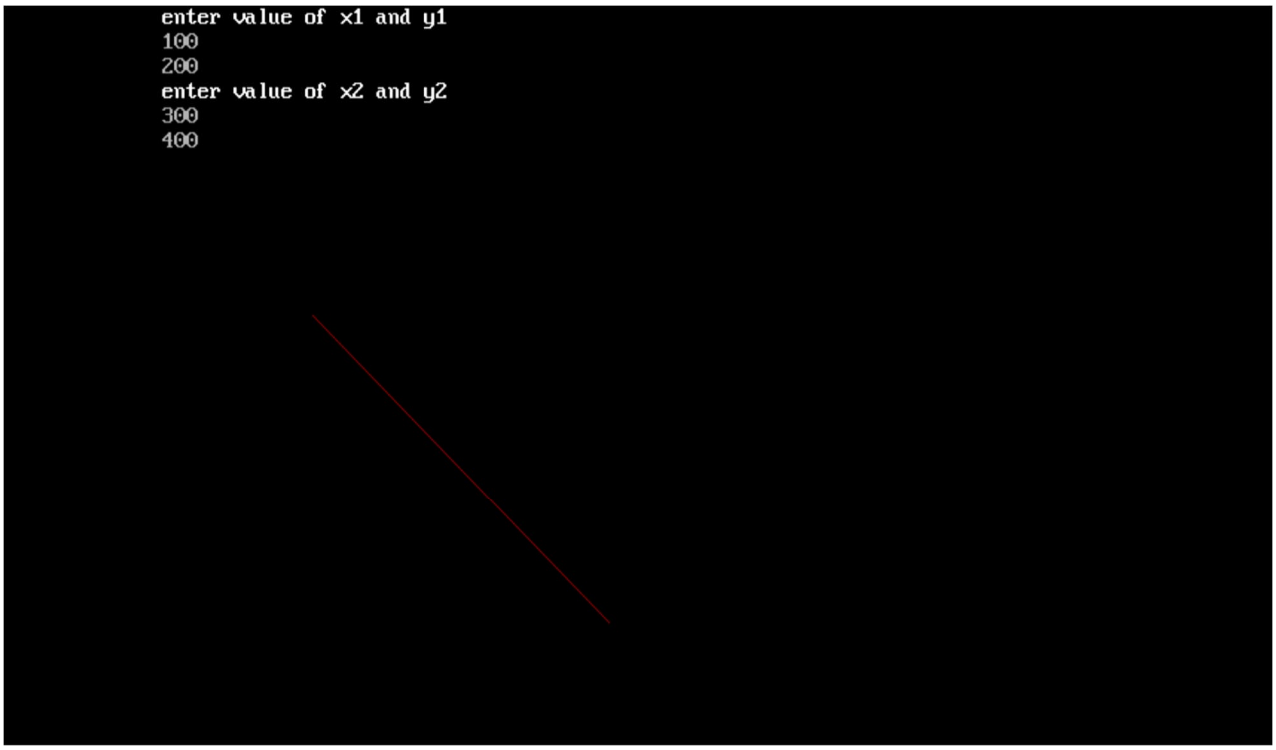

File Edit Search Run Compile Debug Project Options Window Help
\TURBOC3\PROJECTS\DDA1.CPP
1=[↑]
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<dos.h>
#include<graphics.h>
void main()
{
int gd=DETECT,gm;
int x1,y1,x2,y2,dx,dy,steps,xinc,yinc,i;
initgraph(&gd,&gm,"c:\\turboc3\\bgi");
cout <<"enter value of x1 and y1" <<endl;
cin>>x1>>y1;
cout <<"enter value of x2 and y2" <<endl;
cin>>x2>>y2;
1:3
Message
2
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu


File Edit Search Run Compile Debug Project Options Window Help
\TURBOC3\PROJECTS\DDA1.CPP
int x1,y1,x2,y2,dx,dy,steps,xinc,yinc,i;
initgraph(&gd,&gm,"c:\\turboc3\\bgi");
cout <<"enter value of x1 and y1" <<endl;
cin>>x1>>y1;
cout <<"enter value of x2 and y2" <<endl;
cin>>x2>>y2;
dx=x2-x1;
dy=y2-y1;
if (abs(dx)>abs(dy))
{
steps=abs(dx);
}
else
{
9:3
Message
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu


File Edit Search Run Compile Debug Project Options Window Help
\TURBOC3\PROJECTS\DDA1.CPP
1=[↑]
else
{
steps=abs(dy);
}
xinc=dx/steps;
yinc=dy/steps;
i=1;
while (i<=steps)
{
x1=x1+xinc;
y1=y1+yinc;
putpixel(x1,y1,4);
i=i+1;
34:3
Message
2
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu


File Edit Search Run Compile Debug Project Options Window Help
\TURBOC3\PROJECTS\DDA1.CPP
1=[↑]
i=1;
while (i<=steps)
{
x1=x1+xinc;
y1=y1+yinc;
putpixel(x1,y1,4);
i=i+1;
delay(50);
}
getch();
closegraph();
}
40:3
Message
2
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

OUTPUT:



CONCLUSION:

WITH THE HELP OF TURBO C++ FOR DDA LINE DRAWING ALGORITHM, WE CAN TAKE INPUT FROM THE USER FOR X AND Y CO-ORDINATE AND CAN DRAW AN DESIRED STRAIGHT LINE.

NAME:- SINGH SUDHAM DHARMENDRA

BRANCH:- CSE(AIML)

ROLL NO.:- AIMLD50

SUBJECT:- COMPUTER GRAPHICS

TOPIC:-EXPERIMENT NO:-2

EXPERIMENT NO. 2

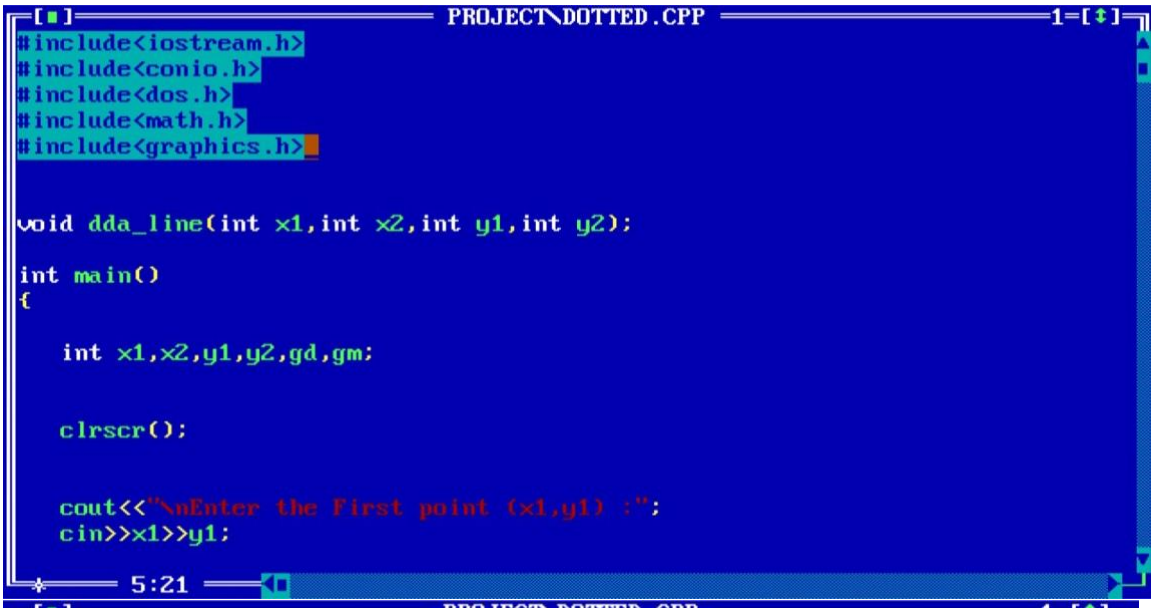
AIM: - A program in C/C++ for DDA line drawing algorithm.

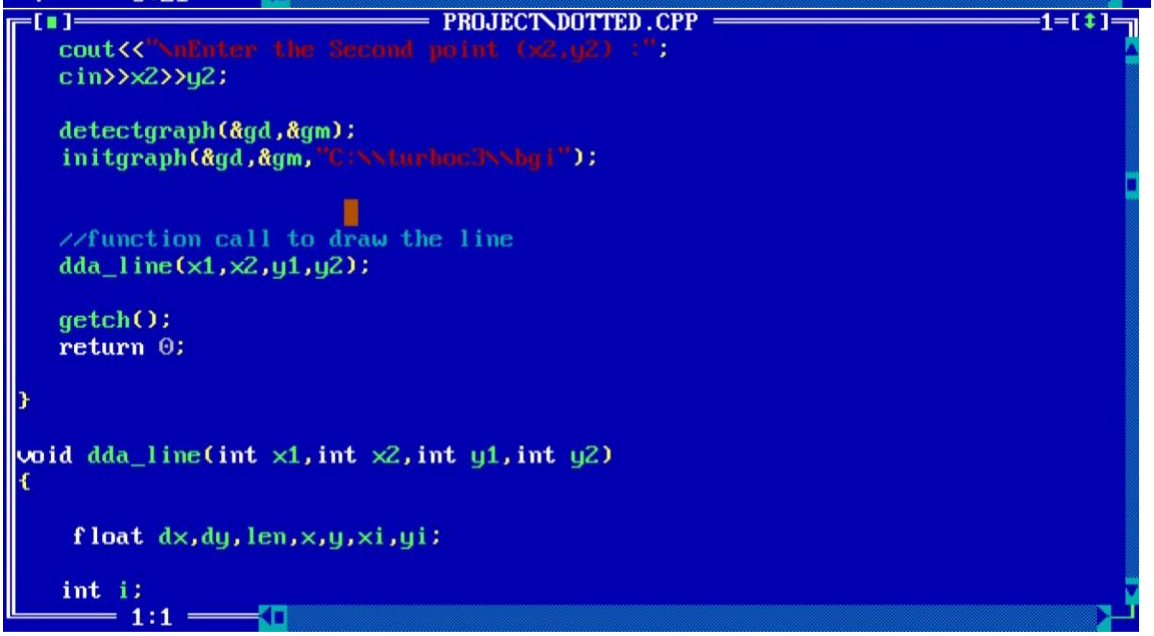
i) **To draw a dotted line.**

ii) **To draw a dashed line.**

SOFTWARE USED: - Turbo C++.

SOURCE CODE: - 1) To draw a dotted line.


PROJECT\DOTTED.CPP
1
#include<iostream.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#include<graphics.h>
void dda_line(int x1,int x2,int y1,int y2);
int main()
{
int x1,x2,y1,y2,gd,gm;
clrscr();
cout<<"\nEnter the First point (x1,y1) :";
cin>>x1>>y1;
5:21

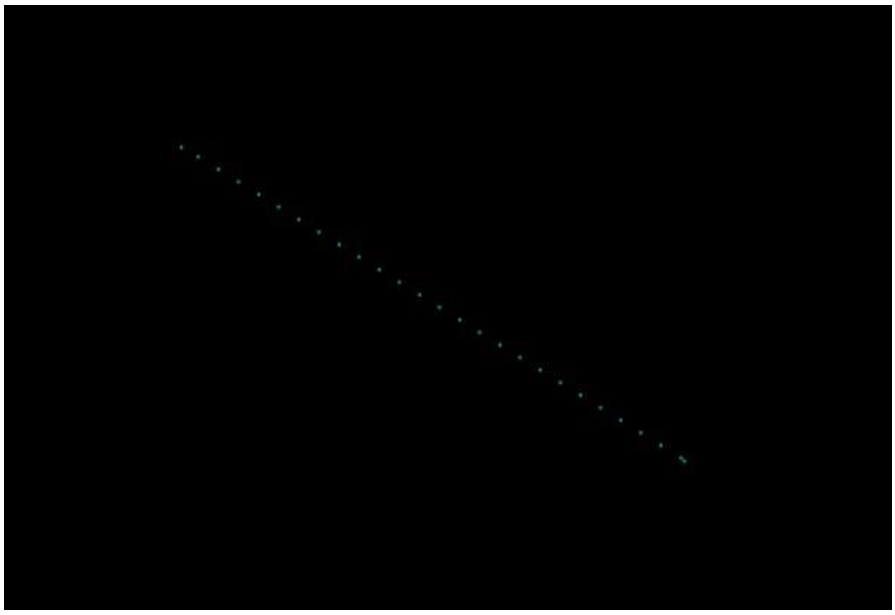

PROJECT\DOTTED.CPP
1
cout<<"\nEnter the Second point (x2,y2) :";
cin>>x2>>y2;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"C:\\turboc3\\bgi");
//function call to draw the line
dda_line(x1,x2,y1,y2);
getch();
return 0;
}
void dda_line(int x1,int x2,int y1,int y2)
{
float dx,dy,len,x,y,xi,yi;
int i;
1:1

```
PROJECT\DOTTED.CPP                                    1=[↕]
   dx = abs(x2-x1);

   dy = abs(y2-y1);


   if(dx>=dy)

      len=dx;

          else

          len=dy;


   dx = (x2-x1)/len;

   dy = (y2-y1)/len;

1:1
```

```
PROJECT\DOTTED.CPP                                    1=[↕]
   x = x1 + 0.5;

   y = y1 + 0.5;

   putpixel(x1,y1,3);

   putpixel(x2,y2,3);

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

    {  if(i%6>4)

       putpixel(x,y,3);

       x += dx;

       y += dy;

    }
}
1:1
```

OUTPUT: -

```
Enter the First point (x1,y1) :100
200

Enter the Second point (x2,y2) :250
300_
```

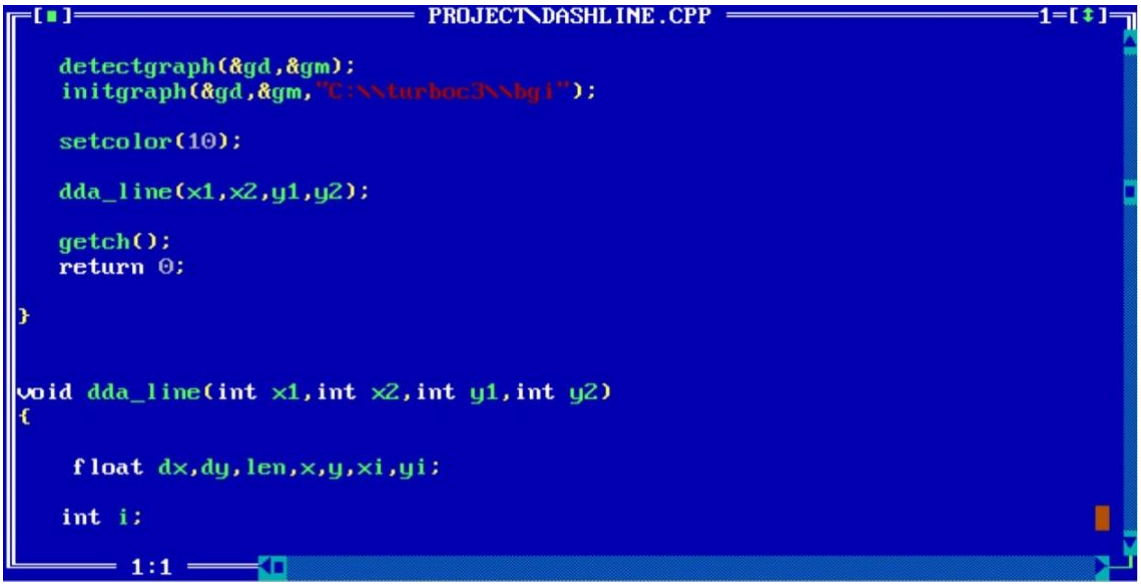


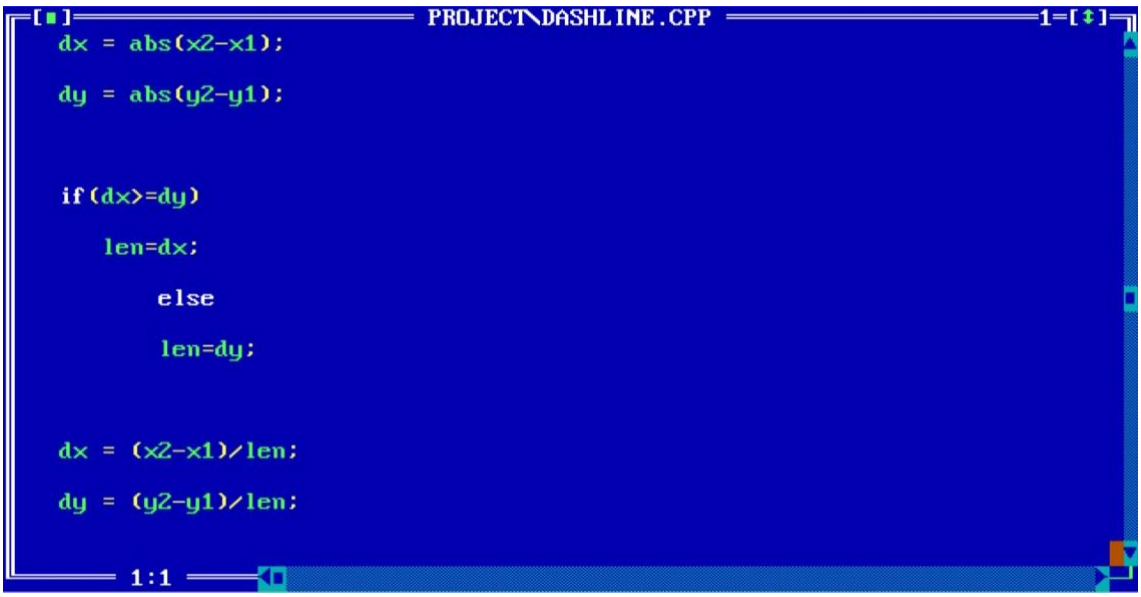
2) To draw dashed line.

PROJECT\DASHLINE.CPP

```
#include<iostream.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#include<graphics.h>


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

int main()
{

   int x1,x2,y1,y2,gd,gm;

   clrscr();

   cout<<"\nEnter the First point (x1,y1) :";
   cin>>x1>>y1;

   cout<<"\nEnter the Second point (x2,y2) :";
   cin>>x2>>y2;
```

5:21

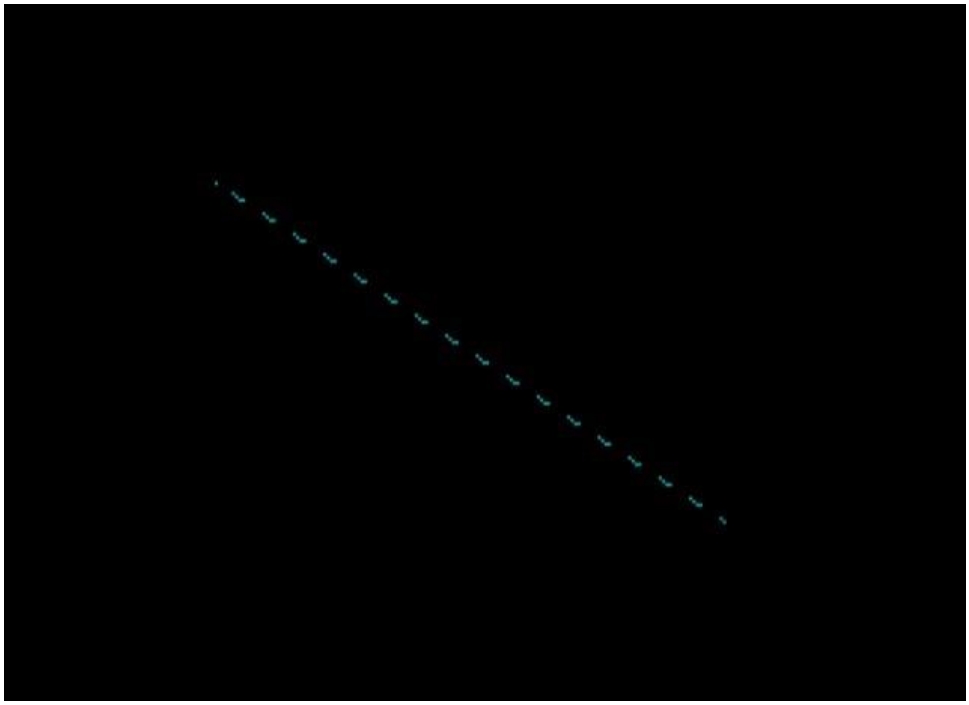

PROJECT\DASHLINE.CPP
1=[↕]
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"C:\\turboc3\\bgi");
setcolor(10);
dda_line(x1,x2,y1,y2);
getch();
return 0;
}
void dda_line(int x1,int x2,int y1,int y2)
{
float dx,dy,len,x,y,xi,yi;
int i;
1:1


PROJECT\DASHLINE.CPP
1=[↕]
dx = abs(x2-x1);
dy = abs(y2-y1);
if(dx>=dy)
len=dx;
else
len=dy;
dx = (x2-x1)/len;
dy = (y2-y1)/len;
1:1

```
PROJECT\DASHLINE.CPP                                    1=[↕]
   x = x1 + 0.5;
   y = y1 + 0.5;
   putpixel(x1,y1,3);
   putpixel(x2,y2,3);
   for(i=0;i<=len;i++)
    {  if(i%9>4)
       putpixel(x,y,3);
       x += dx;
       y += dy;
    }
}
1:1
```

OUTPUT:-

```
Enter the First point (x1,y1) :100
200

Enter the Second point (x2,y2) :250
300_
```



CONCLUSION: -

Hence by taking input from user dotted and dashed line can be drawn by using DDA line drawing algorithm.

NAME:- SINGH SUDHAM DHARMENDRA

BRANCH:- CSE(AIML)

ROLL NO.:- AIMLD50

SUBJECT:- COMPUTER GRAPHICS

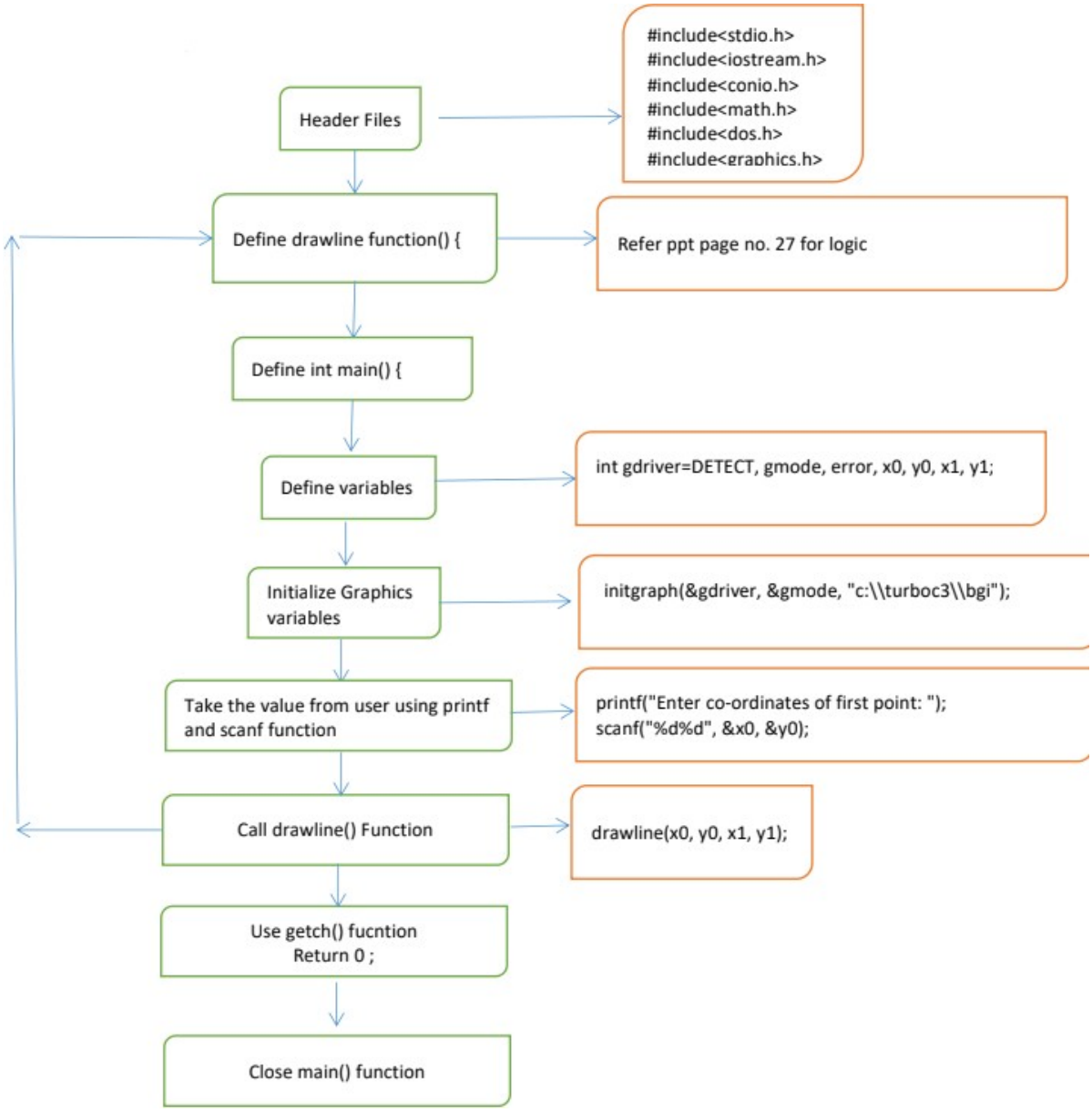
TOPIC:-EXPERIMENT NO:-3

EXPERIMENT:3

Aim : Write a program in c/ c++ to implement Bresenham's line drawing algorithm.

Software used : Turbo C++

Flow chart :



i) To draw a dotted line.

Code:

```
#include <stdio.h>
#include <graphics.h>
#include <math.h>
void main()
{
   int gd = DETECT, gm, x, y, x1, y1, x2, y2, dx, dy, i, e;float
   xinc, yinc;
   initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
   printf("Enter the starting coordinate of line\n");
   scanf("%d%d", &x1, &y1);
   printf("Enter the ending coordinates of line\n");
   scanf("%d%d", &x2, &y2);
   dx = x2 - x1;
   dy = y2 - y1;
   if (x1 < x2)
      xinc = 1;
   else
      xinc = -1;
   if (y1 < y2)
      yinc = 1;
   else
      yinc = -1;
   x = x1;
   y = y1;
   if (dx >= dy)
   {
      e = (2 * dy) - dx;
      while (x != x2)
      {
         if (e < 0)
            e = e + (2 * dy);
         else
         {
            e = e + (2 * (dy - dx));
```

```
            y = y + yinc;
            y = y + yinc;
        }
        x = x + xinc;
        x = x + xinc;
        putpixel(x, y, WHITE);
    }
}
else
{
    e = (2 * dx) - dy;
    while (y != y2)
    {
        if (e < 0)
            e = e + (2 * dx);
        else
        {
            e = e + (2 * (dx - dy));x
            = x + xinc;
            x = x + xinc;
        }
        y = y + yinc;
        y = y + yinc;
        putpixel(x, y, WHITE);
    }
}
getch();
closegraph();
}
```

OUTPUT:

CONCLUSION: In this experiment we have successfully generated a DOTTED Line using Bresenham's line algorithm.

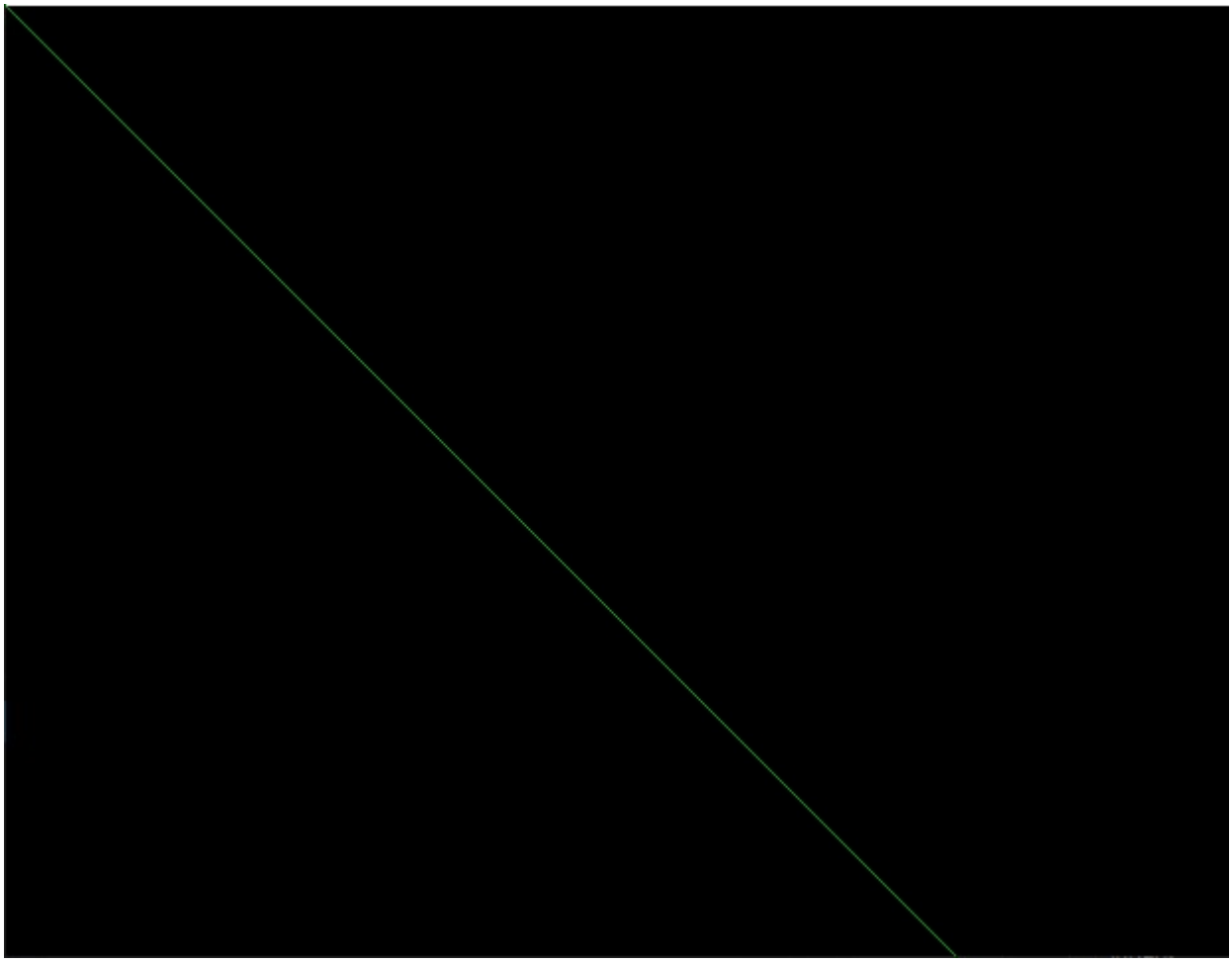
ii) To draw a thin line

Code:

```
#include <conio.h>
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
   int gdriver = DETECT, gmode;
   int i, x, y, x1, y1, x2, y2, dx, dy, length, P0; printf("Enter
   the starting coordinate of line\n");scanf("%d%d", &x1,
   &y1);
   printf("Enter the ending coordinates of line\n");
   scanf("%d%d", &x2, &y2);
   initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");
   dx = abs(x2 - x1);
   dy = abs(y2 - y1);
   if (dx > dy)
      length = dx;
   else
      length = dy;
   putpixel(x1, y1, 2);
   x = x1;
   y = y1;
   P0 = 2 * dy - dx;
   for (i = 0; i % 6 < 4; i++)
      for (i = 0; i <= length; i++)
      {
         if (P0 < 0)
         {
            x = x + 5;
```

```
            y = y + 5;
            putpixel(x, y, 2);
            P0 = P0 + 2 * y;
        }
        else
        {
            x = x + 5;
            y = y + 5;
            putpixel(x, y, 2);
            P0 = P0 + 2 * y - 2 * dx;
        }
    }
  getch();
  closegraph();
  return 0;
}
```

OUTPUT:



CONCLUSION: In this experiment we have successfully generated a THIN Line using Bresenham's line algorithm.

NAME:- SINGH SUDHAM DHARMENDRA

BRANCH:- CSE(AIML)

ROLL NO.:- AIMLD50

SUBJECT:- COMPUTER GRAPHICS

TOPIC:-EXPERIMENT NO:-4

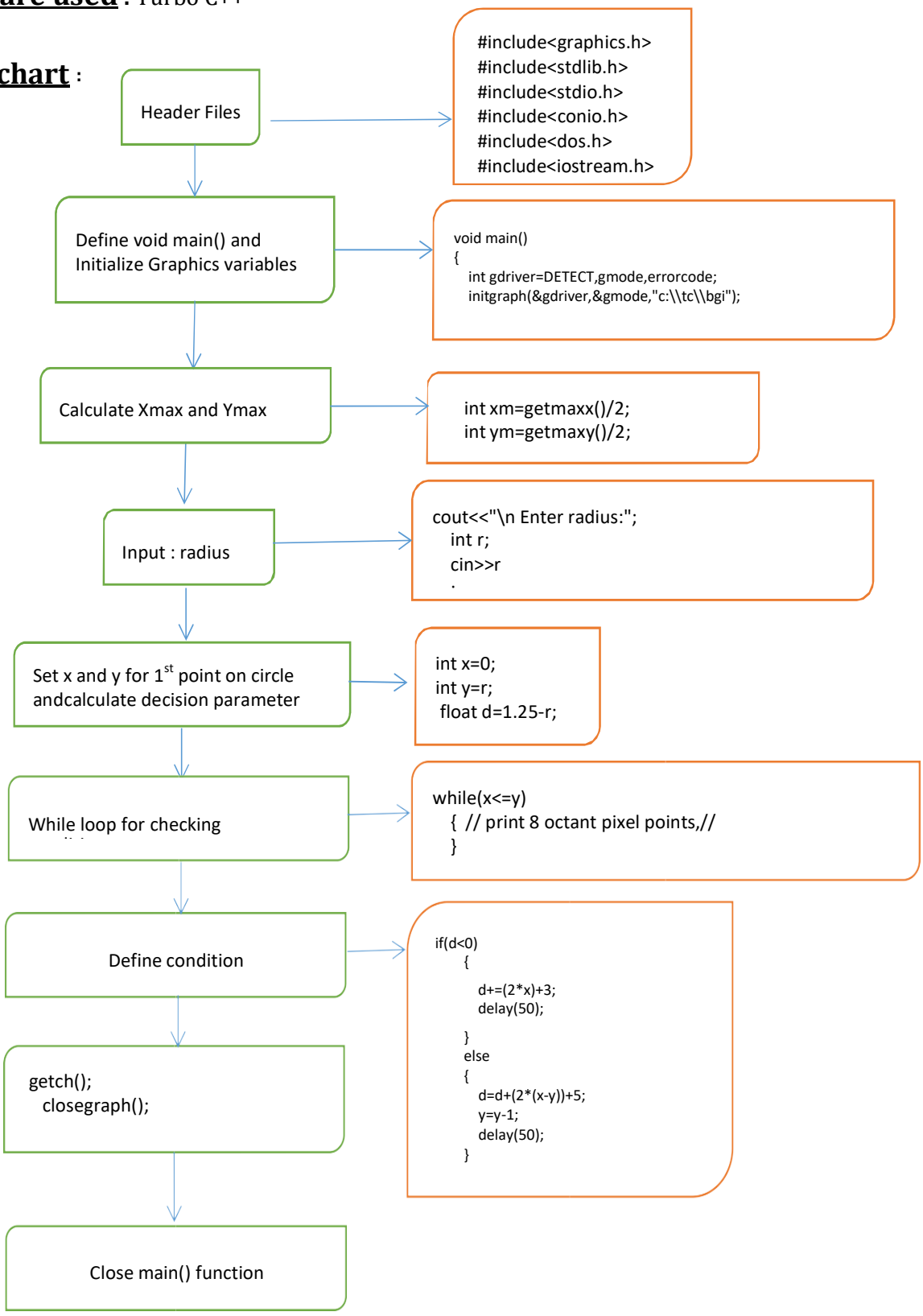
EXPERIMENT : 4

Aim :

Write a program in c to implement Mid-Point circle generating algorithm

Software used : Turbo C++

Flow chart :



Source Code:

```
#include<graphics.h>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<iostream.h>

void main()
{
 int gdriver = DETECT, gmode,errorcode;
 initgraph(&gdriver, &gmode,"c:\\TURBOC3\\BGI");
 int xm = getmaxx()/2;
 int ym = getmaxx()/2;
 int r;
 printf("enter radius\n");
 scanf("%d",&r)
 int x = 0;
 int y = r;
 float d = 1.25 -r;
 while(x<=y)
 {
     putpixel(xm+x,ym-y,15);
     putpixel(xm+y,ym-x,15);
     putpixel(xm+y,ym+x,15);
     putpixel(xm+x,ym+y,15);
     putpixel(xm-x,ym+y,15);
     putpixel(xm-y,ym+x,15);
     putpixel(xm-y,ym-x,15);
     putpixel(xm-x,ym-y,15);
     if(d<0)
     {
        d+=(2*x)+3;
        delay(50);
     }
     else
     {
        d=d+(2*(x-y))+5;
        y=y-1;
        delay(50);
     }
     x=x+1;
 }
```

```
  getch();
  closegraph();
}
```

Output:

Conclusion:

From this experiment we have learnt to draw a circle using mid – point algorithm.

NAME: SINGH SUDHAM DHARMENDRA

ROLL NO.: AIMLD50
BRANCH: CSE (AI & ML)
SUBJECT: COMPUTER GRAPHICS
TOPIC: EXPERIMENT NO. 5

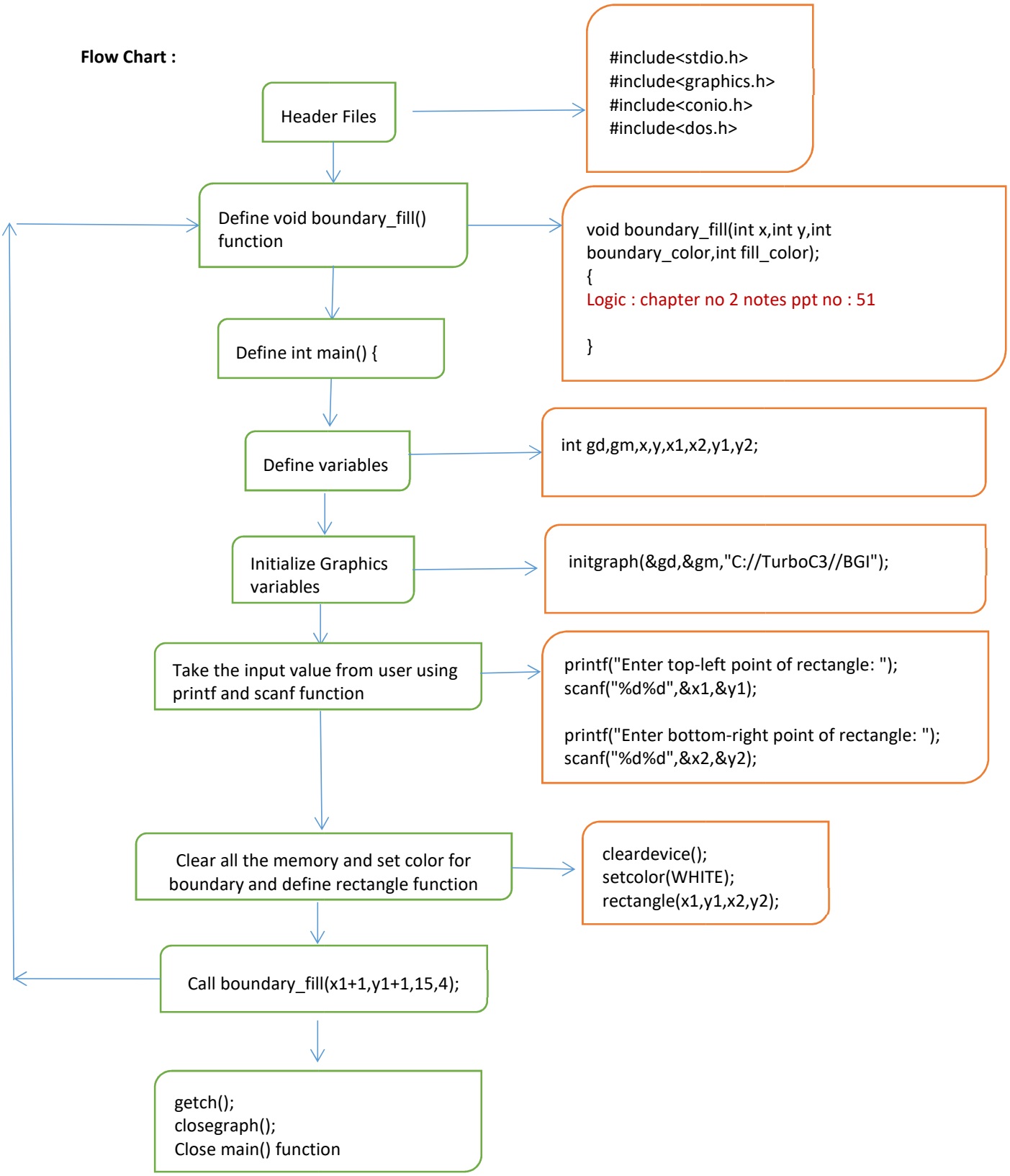
Experiment No: 5

Aim: Write a program in c to implement

i) Boundary Fill algorithm ii) Flood Fill algorithm.

Software used : Turbo C++

Flow Chart :

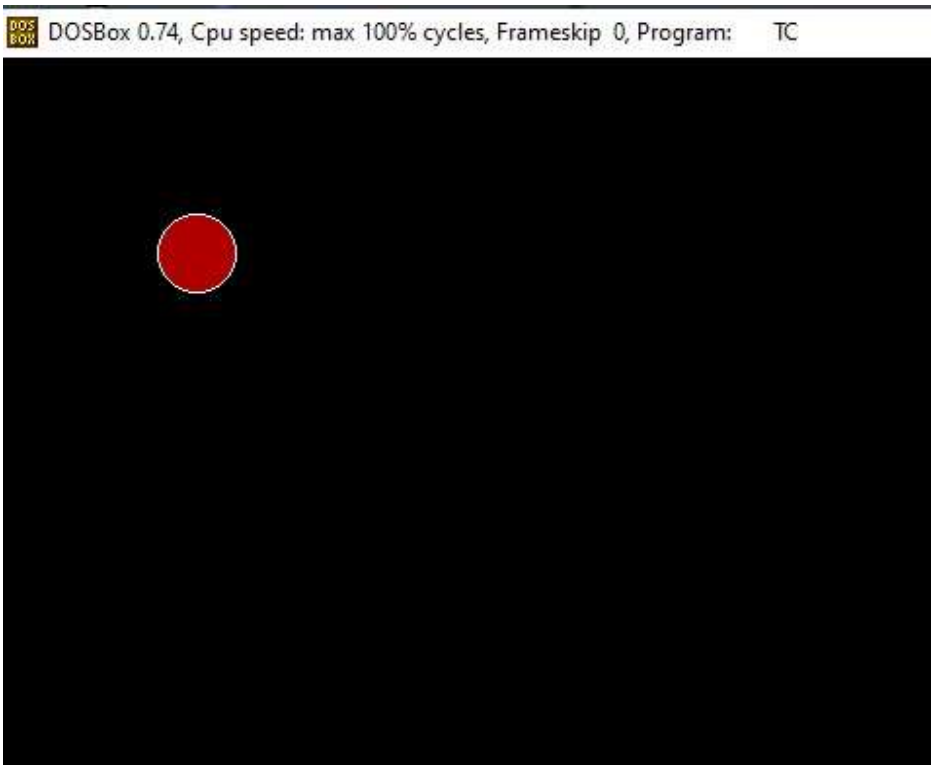


1) Boundary Fill algorithm

Source Code:

```
#include<stdio.h>
#include<graphics.h>

void boundaryfill(int x,int y,int f_color,int b_color)
{ if(getpixel(x,y)!=b_color && getpixel(x,y)!=f_color)
   { putpixel(x,y,f_color);
      boundaryfill(x+1, y,
      f_color,b_color); boundaryfill(x,
      y+1, f_color,b_color);
      boundaryfill(x-1, y, f_color,b_color);
      boundaryfill(x, y-1, f_color,b_color);
   }
} int
main()
{
   int gm,gd=DETECT,radius;
   int x,y;

   printf("Enter x and y positions for circle\n");
   scanf("%d%d",&x,&y);
   printf("Enter radius of circle\n");
   scanf("%d",&radius);

   initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
   circle(x,y,radius); boundaryfill(x,y,4,15);
   delay(5000); closegraph();

   return 0;
}
```

Output:



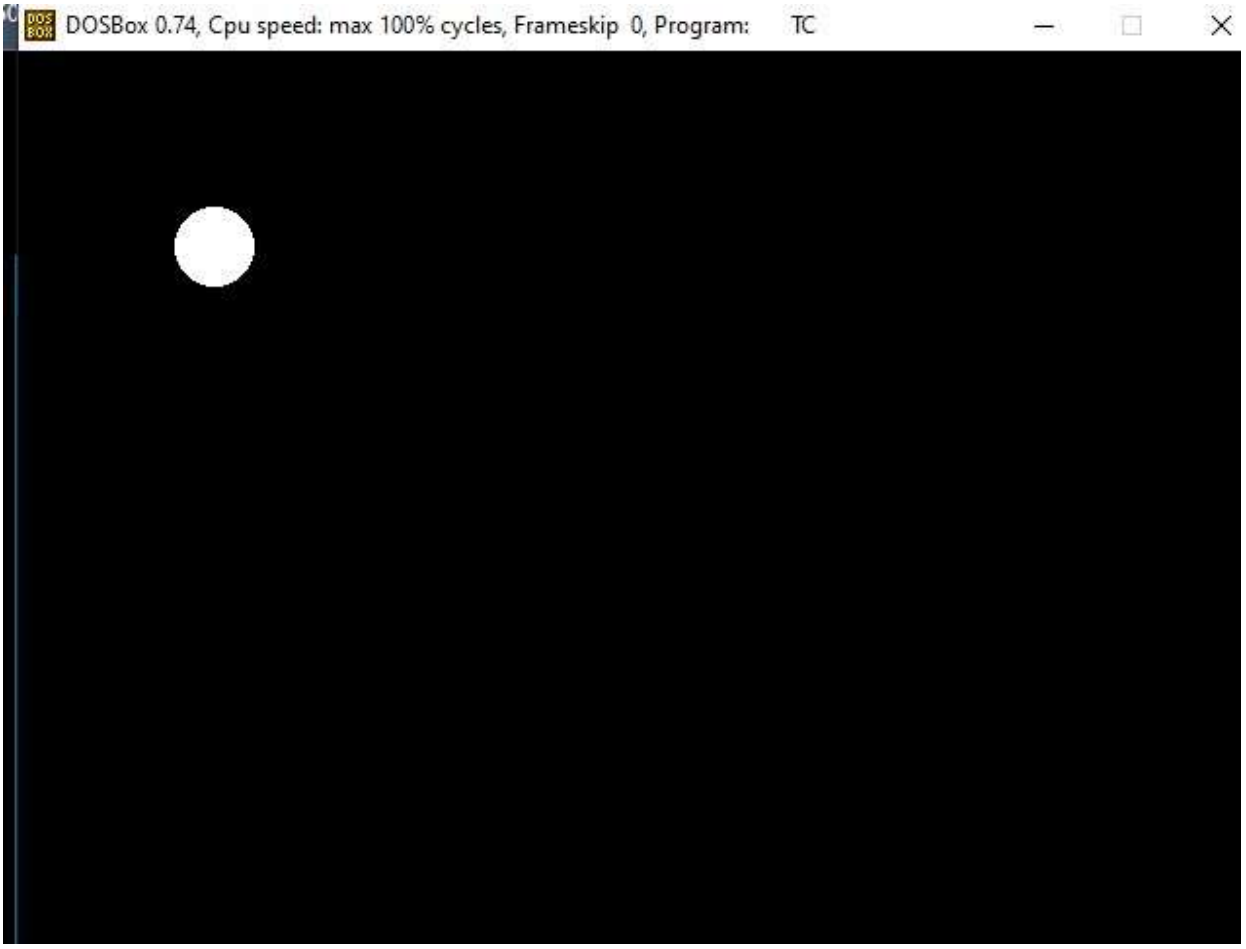
Conclusion: Here's a circle obtained by using the BOUNDARY FILL algorithm.

2) Flood Fill algorithm

Source Code:

```
#include<stdio.h>
#include<graphics.h>
#include<dos.h>

void floodFill(int x,int y,int oldcolor, int newcolor)
{ 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

int main()
{
        int gm,gd=DETECT,radius;
        int x,y;

        printf("Enter x and y positions for circle\n");
        scanf("%d%d",&x,&y);
        printf("Enter radius of circle\n");
        scanf("%d",&radius);

        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI")
        ; circle(x,y,radius); floodFill(x,y,0,15);

        delay(5000);
        closegraph();
        return 0 ;
}
```

Output:



Conclusion: Here's a circle obtained by using the FLOOD FILL algorithm.

NAME:- SINGH SUDHAM DHARMENDRA

BRANCH:- CSE(AIML)

ROLL NO.:- AIMLD50

SUBJECT:- COMPUTER GRAPHICS

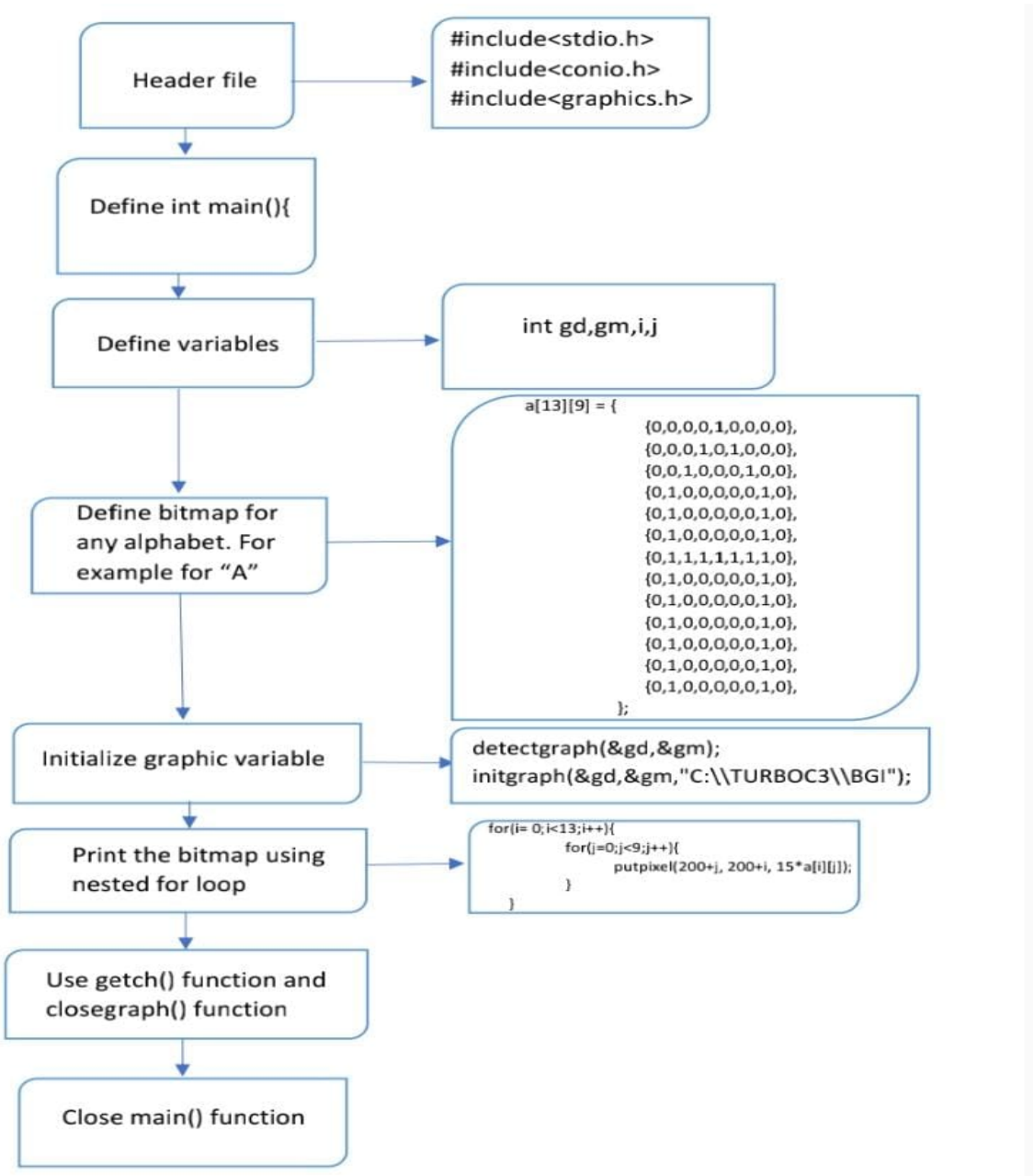
TOPIC:-EXPERIMENT NO:-6

EXPERIMENT NO. 6

AIM: Write a program in c for character generation -bitmap method.

SOFTWARE USED: Turbo C++

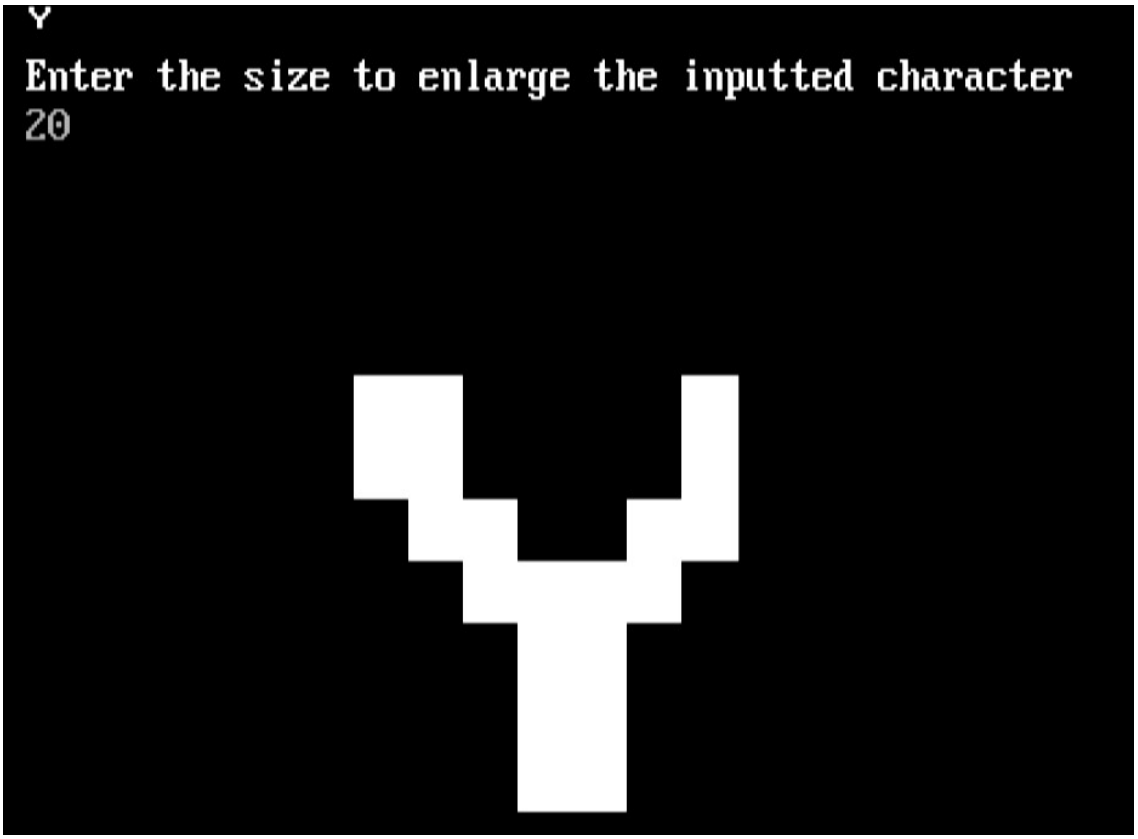
FLOWCHART:



SOURCE CODE:

```
#include<conio.h>
#include<stdio.h>
#include<graphics.h>
void main()
{
int gd= DETECT, gm,i,j,m;
char c[8][8],s[1];  // needed to delcare 's' as string inorder to use it with
outtextxy()
initgraph(&gd,&gm,"..\\bgi");
printf("Enter a Character:");
scanf("%s",s);
clearviewport();
outtextxy(1,1,s);
for(i=0;i<textwidth("S");i++)      // any  capital character can be given
instead of S
{
for(j=0;j<textheight("S");j++)
{
c[i][j]=getpixel(i,j);
}
}
printf("Enter the size to enlarge the inputted character \n");
scanf("%d",&m);
for(i=0;i<8;i++)
{
for(j=0;j<8;j++)
{
setfillstyle(SOLID_FILL,c[i][j]);
bar(100+(i*m),100+(j*m),100+(i+1)*m,100+(j+1)*m);
}
}
getch();
closegraph();
}
```

OUTPUT:



CONCLUSION:

From this experiment we successfully generated a character using Bitmap method.

NAME:- SINGH SUDHAM DHARMENDRA

BRANCH:- CSE(AIML)

ROLL NO.:- AIMLD50

SUBJECT:- COMPUTER GRAPHICS

TOPIC:-EXPERIMENT NO:-7

Experiment No : 7

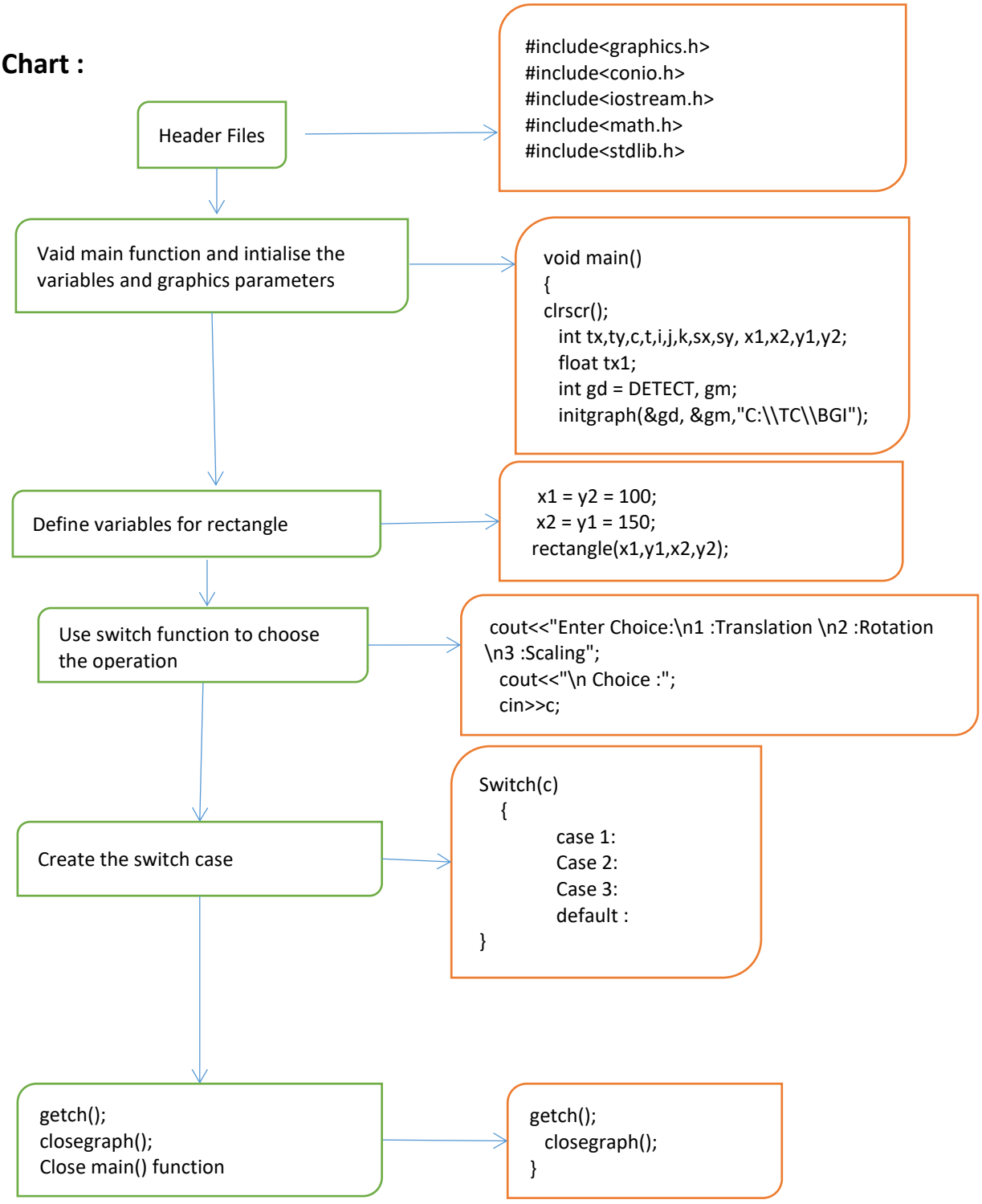
Aim :

Write a program in c to implement 2D transformations

i) Translation ii) Rotation iii) Scaling

Software used : Turbo C++

Flow Chart :

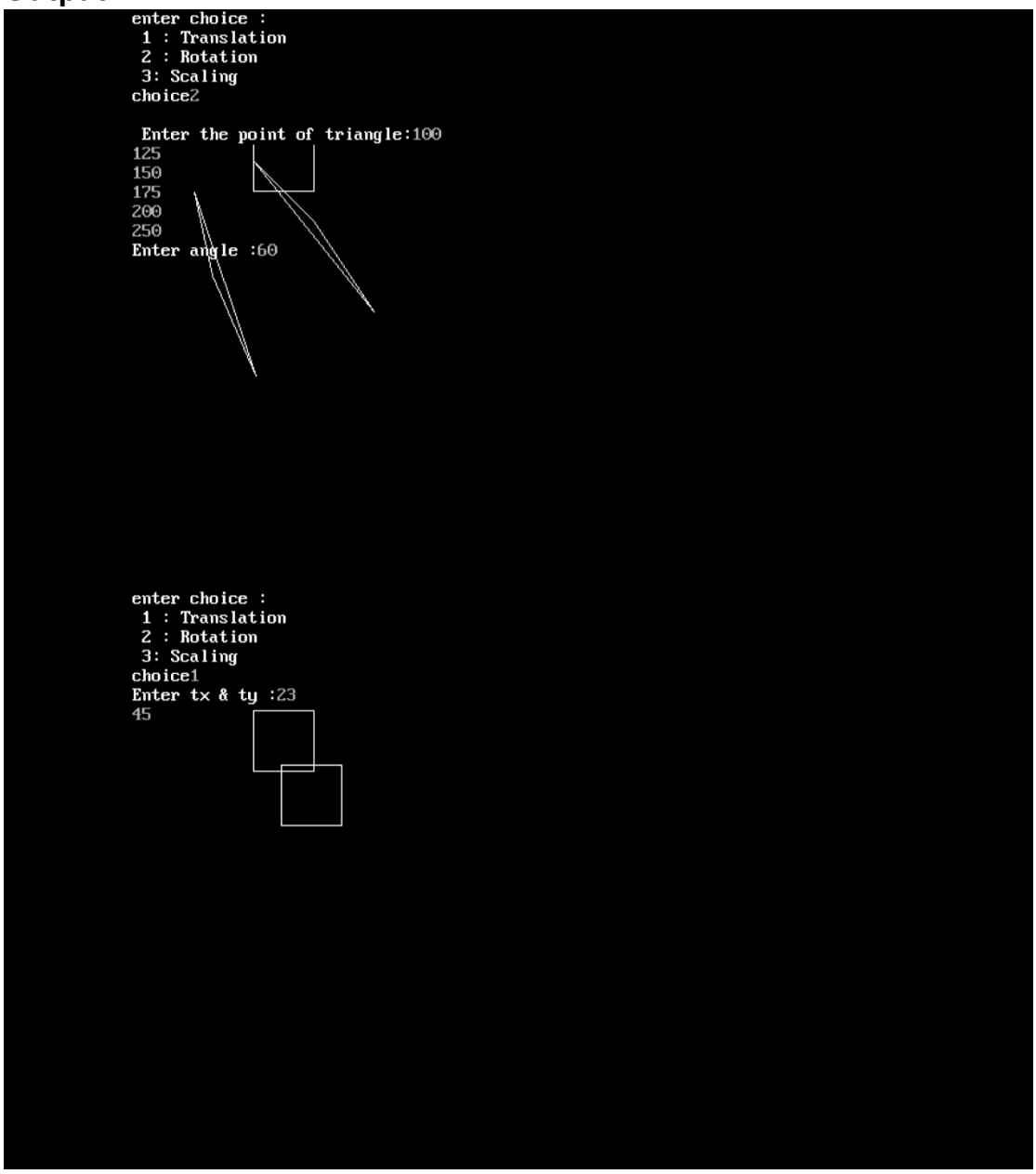


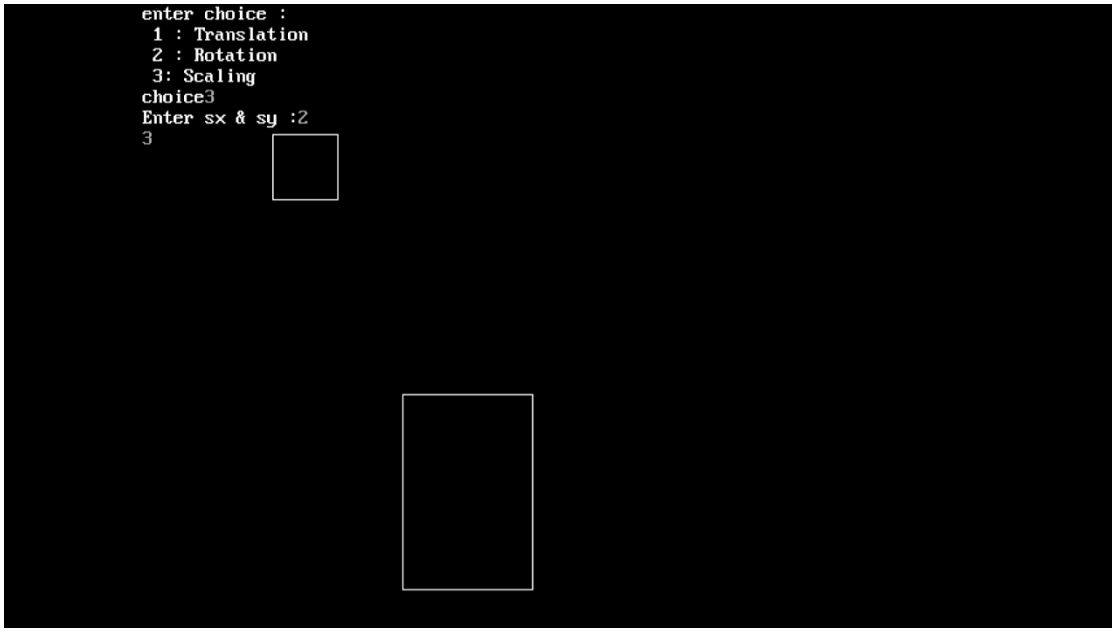
Source Code:

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
#include<stdlib.h>
#include<dos.h>

int main()
{
   int tx,ty,c,t,i,j,k,sx,sy,x1,x2,y1,y2,t1,t2,t3,t4X1,Y1,X2,Y2,X3,Y3,rrx,nx1,ny1,nx2,ny2,nx3,ny3;
   float tx1;
   int gd = DETECT, gm;
   initgraph(&gd,&gm,"C://Turboc3//BGI");
   x1 = y2 = 100;
   x2 = y1 = 150;
   rectangle(x1,y1,x2,y2);
   printf("enter choice :\n 1 : Translation\n 2 : Rotation\n 3: Scaling");
   printf("\nchoice");
   scanf("%d",&c);
   switch(c){
          case '1':
            printf("Enter tx & ty :");
            scanf("%d%d",&tx,&ty);
            t1=x1+tx;
            t2=y1+ty;
            t3=x2+tx;
            t4=y2+ty;
            rectangle(t1,t2,t3,t4);
            break;
     case '2' :
                                printf(" \n Enter the point of triangle:");
                                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);
                                int r;
                                printf("Enter angle :");
                                scanf("%d",&r);
                                rx=r*(3.14/180);

                                nx1 =abs( X1* cos(rx) - Y1*sin(rx));
                                ny1 =abs( Y1* cos(rx) + X1*sin(rx));

                                nx2 = abs( X2* cos(rx) -Y2*sin(rx));
                                ny2 = abs( Y2* cos(rx) + X2*sin(rx));

                                nx3 = abs (X3* cos(rx) - Y3*sin(rx));
                                ny3 = abs (Y3* cos(rx) + X3*sin(rx));

                                line(nx1,ny1,nx2,ny2);
                                line(nx2,ny2,nx3,ny3);
                                line(nx3,ny3,nx1,ny1);

                                break;
     case '3' :
                        printf("Enter sx & sy :");
```

```
                    scanf("%d%d",&sx,&sy);
                    rectangle(x1*sx, y1*sy, x2*sx, y2*sy);
                    break;
    default :
                    printf("Not a valid choice");
  }
  getch();
  closegraph();
  return 0;
}
```

Output:





Conclusion :

From this experiment we have learned the algorithm for scaling, translating and rotating a 2-Dimensional object.

NAME: SINGH SUDHAM DHARMENDRA

ROLL NO.: AIMLD50

BRANCH: CSE (AI & ML)

SUBJECT: COMPUTER GRAPHICS

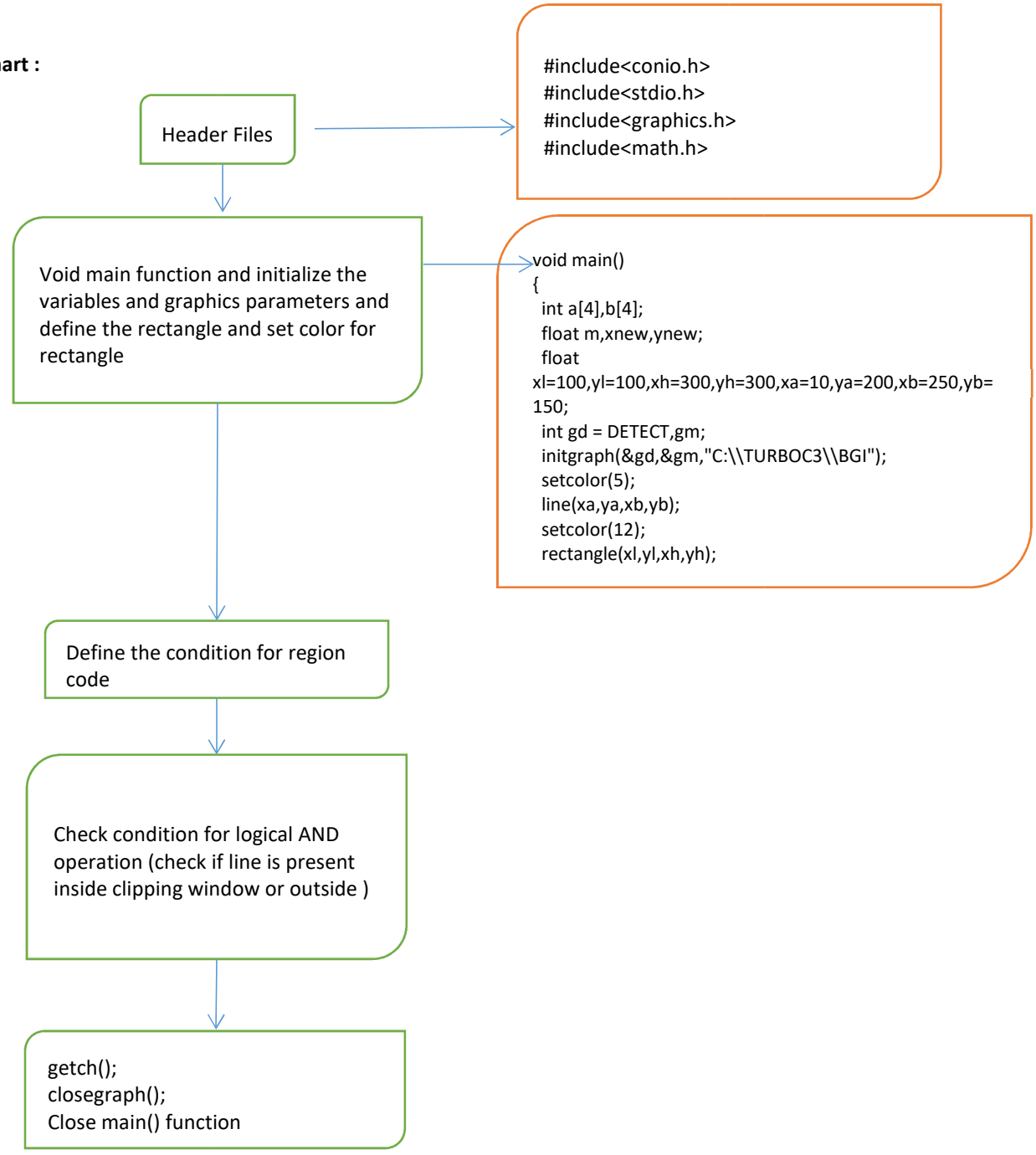
TOPIC: EXPERIMENT NO. 8

Experiment No: 8

Aim: Write a program in c to implement Line clipping algorithm Cohen Sutherland

Software used : Turbo C++

Flow Chart :



SOURCE CODE:

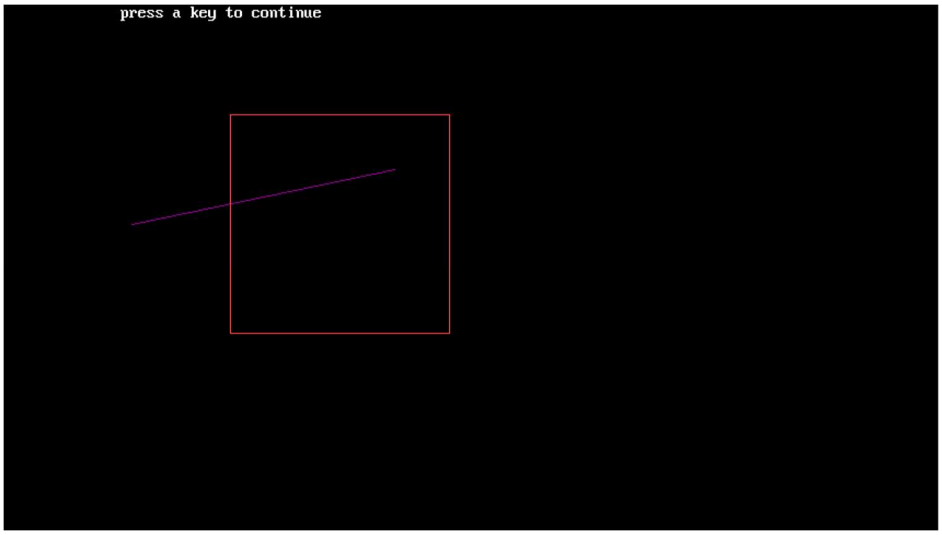
```
#include<conio.h>
#include<stdio.h>
#include<graphics.h>
#include<math.h>

void main()
{
  int a[4],b[4];
  float m,xnew,ynew;
  float xl=100,yl=100,xh=300,yh=300,xa=10,ya=200,xb=250,yb=150;
  int gd = DETECT,gm;
  initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
  setcolor(5);
  line(xa,ya,xb,yb);
  setcolor(12);
  rectangle(xl,yl,xh,yh);
  m = (yb-ya)/(xb-xa);

  if(xa < xl)
    a[3] = 1;
  else a[3] = 0;

  if(xa>xh)
   a[2] = 1;
  else a[2] = 0;

  if(ya < yl)
   a[1] = 1;
  else a[1] = 0;

  if (ya > yh)
```

```
 a[0] = 1;
 else a[0] = 0;

if(xb < xl)
  b[3] = 1;
else b[3] = 0;

if(xb>xh)
 b[2] = 1;
else b[2] = 0;

if(yb < yl)
 b[1] = 1;
else b[1] = 0;

if (yb > yh)
 b[0] = 1;
 else b[0] = 0;

printf("press a key to continue");
getch();
if(a[0] == 0 && a[1] == 0 && a[2] == 0 && a[3] == 0 && b[0] == 0 &&
b[1] == 0 && b[2] == 0 && b[3] == 0 )
{

 printf("no clipping");
 line(xa,ya,xb,yb);
}
else if(a[0]&&b[0] || a[1]&&b[1] || a[2]&&b[2] || a[3]&&b[3])
{
 clrscr();
 printf("line discarded");
 rectangle(xl,yl,xh,yh);
}
```

```
else
{
 if(a[3] == 1 && b[3]==0)
 {
  ynew  = (m * (xl-xa)) + ya;
  setcolor(12);
  rectangle(xl,yl,xh,yh);
  setcolor(0);
  line(xa,ya,xb,yb);
  setcolor(15);
  line(xl,ynew,xb,yb);
 }
 else if(a[2] == 1 && b[2] == 0)
 {
  ynew  = (m * (xh-xa)) + ya;
  setcolor(12);
  rectangle(xl,yl,xh,yh);
  setcolor(0);
  line(xa,ya,xb,yb);
  setcolor(15);
  line(xl,ynew,xb,yb);
 }
 else if(a[1] == 1 && b[1] == 0)
 {
  xnew = xa + (yl-ya)/m;
  setcolor(0);
  line(xa,ya,xb,yb);
  setcolor(15);
  line(xnew,yh,xb,yb);
 }

 else if(a[0] == 1 && b[0] == 0)
 {
  xnew = xa + (yh-ya)/m;
```

```
    setcolor(0);
    line(xa,ya,xb,yb);
    setcolor(15);
    line(xnew,yh,xb,yb);
   }
  }
  getch();
  closegraph();
}
```

OUTPUT:



CONCLUSION: IN THIS EXPERIMENT WE HAVE SUCCESSFULLY IMPLEMENTED LINE CLIPPING USING Cohen Sutherland ALGORITHM.

NAME: SINGH SUDHAM DHARMENDRA

ROLL NO.: AIMLD50

BRANCH: CSE (AI & ML)

SUBJECT: COMPUTER GRAPHICS

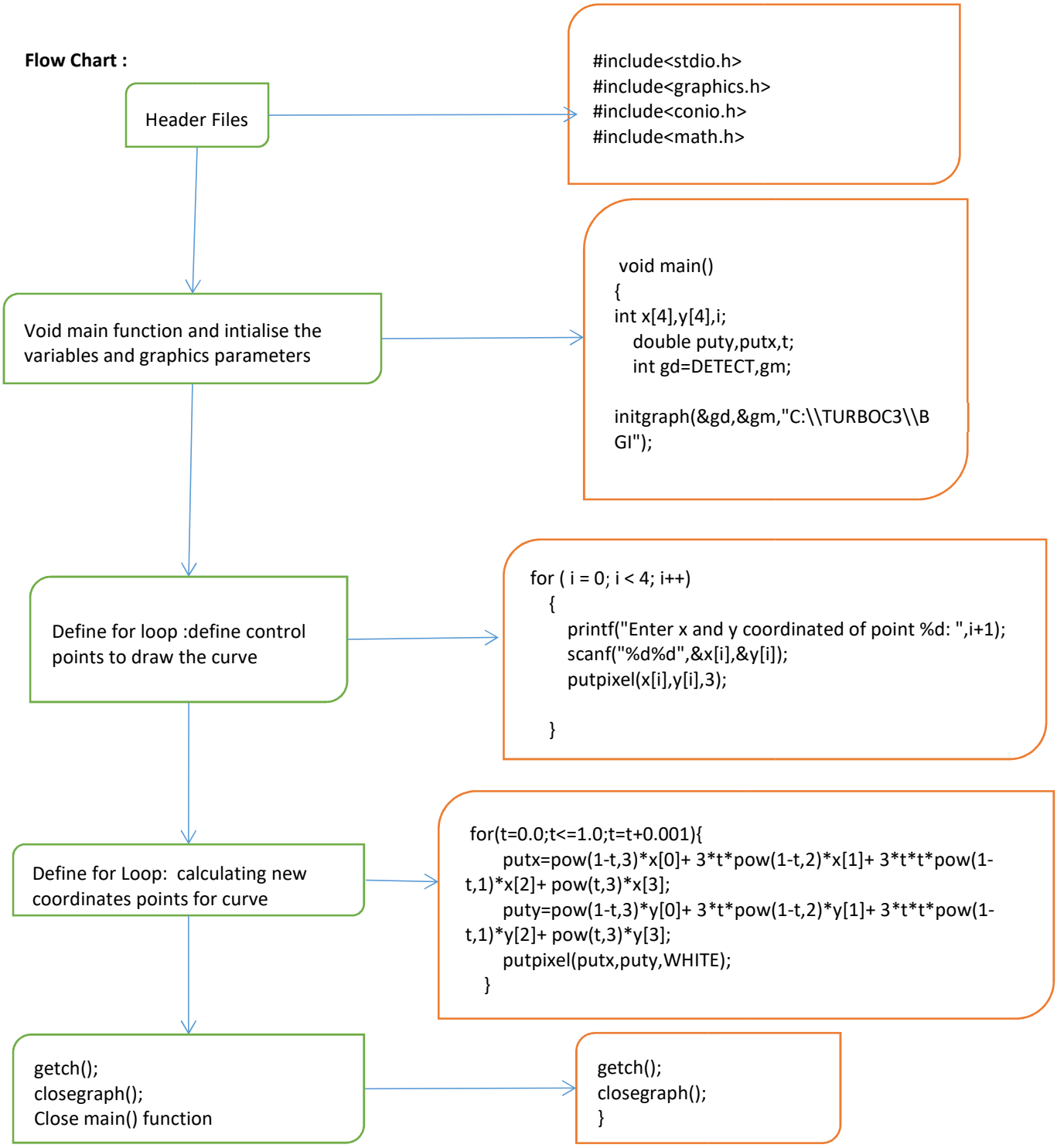
TOPIC: EXPERIMENT NO. 9

Experiment No: 9

Aim: Write a program in c to implement Bezier curve.

Software used : Turbo C++

Flow Chart :

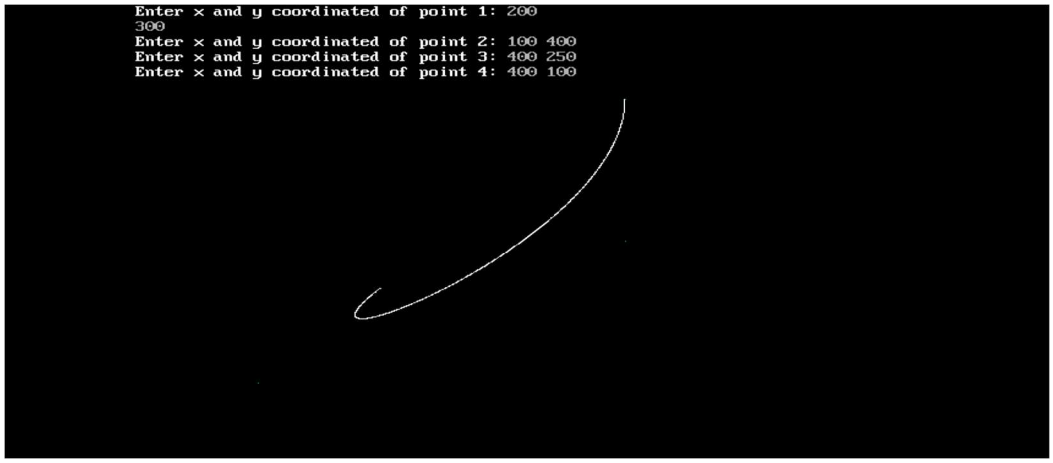


Source Code:

```
#include<stdio.h>

#include<graphics.h>

#include<conio.h>

#include<math.h>

void main(){

   int x[4],y[4],i;

   double puty,putx,t;

   int gd=DETECT,gm;

   initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

   for ( i = 0; i < 4; i++)

   {

      printf("Enter x and y coordinated of point %d:
",i+1);

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

      putpixel(x[i],y[i],3);



   }
```

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

      putx=pow(1-t,3)*x[0]+ 3*t*pow(1-t,2)*x[1]+
3*t*t*pow(1-t,1)*x[2]+ pow(t,3)*x[3];

      puty=pow(1-t,3)*y[0]+ 3*t*pow(1-t,2)*y[1]+
3*t*t*pow(1-t,1)*y[2]+ pow(t,3)*y[3];

      putpixel(putx,puty,WHITE);

   }

   getch();

   closegraph();



}
```

OUTPUT:



CONCLUSION:

IN THIS EXPERIMENT WE HAVE SUCCESSFULLY IMPLEMENTED BEZIER CURVE USING TURBO C++.

NAME: SINGH SUDHAM DHARMENDRA

ROLL NO.: AIMLD50

BRANCH: CSE (AI & ML)

SUBJECT: COMPUTER GRAPHICS

EXPERIMENT NO. 10

MINI PROJECT

MINI PROJECT TITLE: VISUALIZATION OF SIGNAL TRANSMISSION

GROUP MEMBERS: 1. RUPARELIYA AKSHAR JAYANTI (41)

2. SINGH SUDHAM DHARMENDRA (50)

3. ANSARI HANZALA MOHD IMAMUDDIN (03)

4. CHIKANKAR PRATHAMESH SHIVAJI (08)

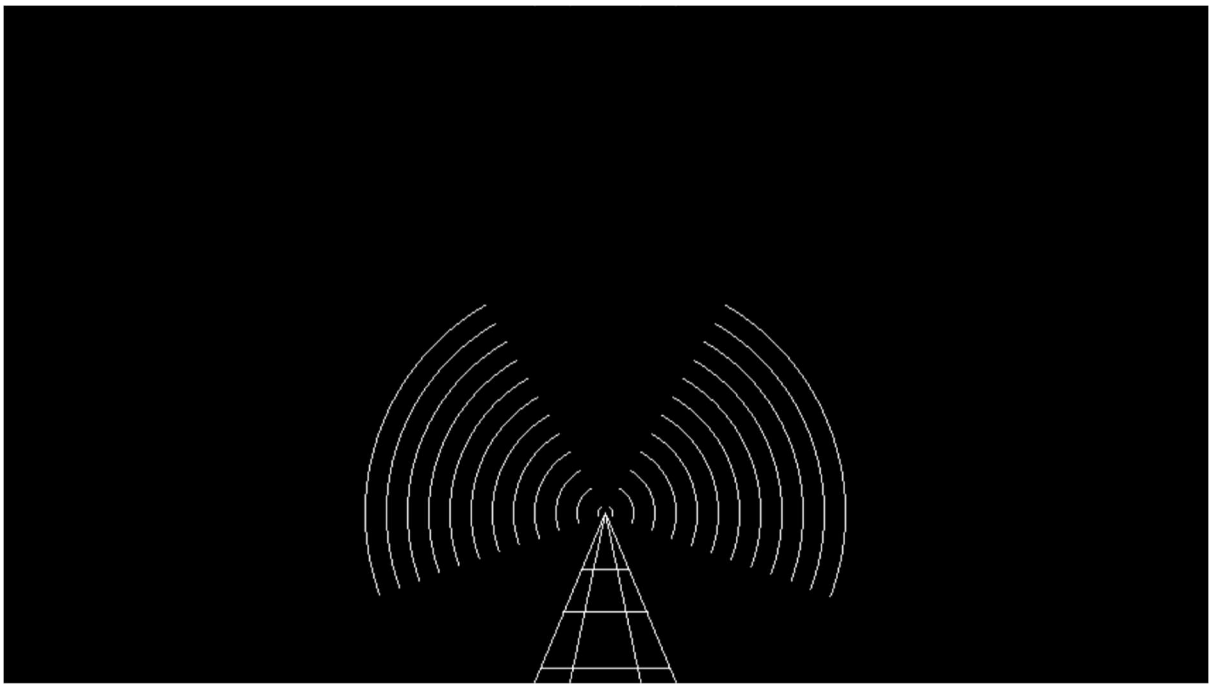
SOFTWARE USED: TURBO C++

SOURCE CODE:

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <dos.h>
int main()
{
    int gdriver = DETECT, gmode, err;
    int midx, y, radius = 5;
    int k = 0, stangle, endangle;

    initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

    err = graphresult();
    if (err != grOk)
    {   /* error occurred */
        printf("Graphics Error: %s\n",grapherrormsg(err));
        getch();
        return 0;
    }
    /* mid position in x-axis */
    midx = getmaxx() / 2;

    /* calculating the position of y */
    y = (3 * getmaxy()) / 4;

    /* start and end angles of signals */
    stangle = 120;
    endangle = 200;

    while (!kbhit())
    {
        k = 0, radius = 5;
        /* clears graphic screen */
        cleardevice();

        /* construct antenna using lines */
        line(midx - 50, getmaxy(), midx, y);
        line(midx + 50, getmaxy(), midx, y);
        line(midx - 25, getmaxy(), midx, y);
        line(midx + 25, getmaxy(), midx, y);
        line(midx - 45, getmaxy() - 10, midx + 45, getmaxy() - 10);
        line(midx - 30, getmaxy() - 50, midx + 30, getmaxy() - 50);
        line(midx - 16, getmaxy() - 80, midx + 16, getmaxy() - 80);
        /* signals from the antenna */

        while (k < 18)
        {
            /* signal at the left side */
            arc(midx, y, stangle, endangle, radius);
            /* signal at the right side */
            arc(midx, y, 0, 60, radius);
```

```
            arc(midx, y, 340, 360, radius);
            radius = radius + 15;
            delay(50);
            k++;
        }
    }
    getch();

    /* deallocate memory allocated for graphic screen */
    closegraph();

    return 0;
}
```

OUTPUT:



CONCLUSION:

FROM THE ABOVE PROGRAM AND OUTPUT WE CONCLUDE THAT THE PROJECT IS ABLE TO PERFORM VISUALIZATION OF SIGNAL TRANSMISSION I.E. THE TRANSMISSION OF ELECTROMAGNETIC WAVES FROM AN ANTENNAS OR SATELLITE.

Computer Graphics Mini Project

- GUIDED BY VAIBHAV PALAV

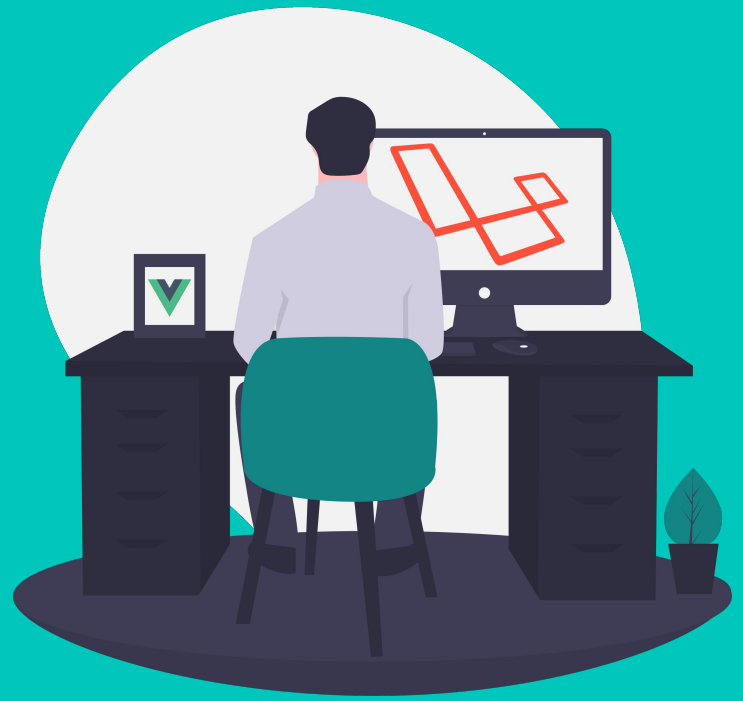
Team Members :

1. AIMLD50_SINGH SUDHAM DHARMENDRA
2. AIMLD03_ANSARI HANZALA MOHD IMAMUDDIN
3. AIMLD41_RUPARELIYA AKSHAR JAYANTI
4. AIMLD08_CHIKANKAR PRATHAMESH SHIVAJI

Visualization Of Signal Transmission

OUTLINE :

1. Introduction
2. Program
3. Output
4. Conclusion



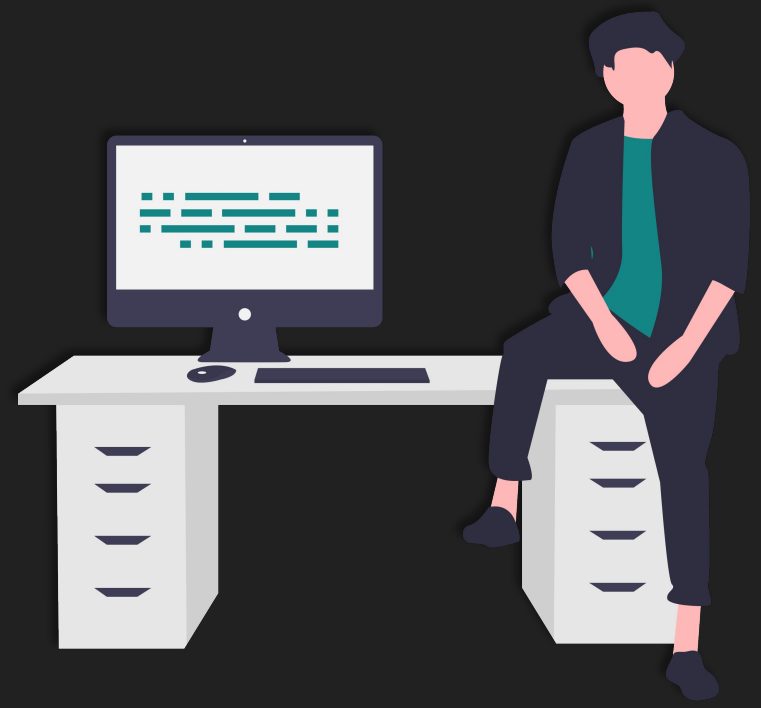
Introduction

1. In this Mini Project, We are going to develop an Animation using Computer Graphics for Visualization of Signal Transmission .
2. Wireless Communication is the fastest growing and most vibrant technological areas in the communication field.
3. Wireless Communication is a method of transmitting information from one point to other, without using any connection like wires, cables or any physical medium.
4. Antennas are electrical devices that transform the electrical signals to radio signals in the form of Electromagnetic (EM) Waves and vice versa.
5. These Electromagnetic Waves propagates through space. Hence, both transmitter and receiver consists of an antenna.

Introduction

Some Built-in Functions of Graphics :

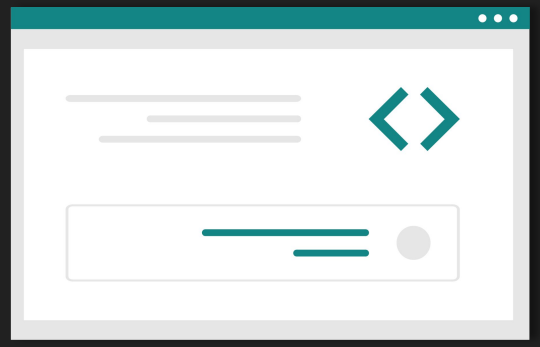
1. **initgraph() :** initgraph initializes the graphics system by loading a graphics driver from disk and putting the system into graphics mode.
2. **graphresult() :** graphresult returns the error code for the last graphics operation that reported an error and resets the error level to grOk.
3. **getmaxx() :** getmaxx returns the maximum x screen coordinate for the current graphics driver and mode. getmaxx is invaluable for centering, determining the boundaries of a region onscreen, and so on.
4. **getmaxy() :** getmaxx returns the maximum x screen coordinate



Program

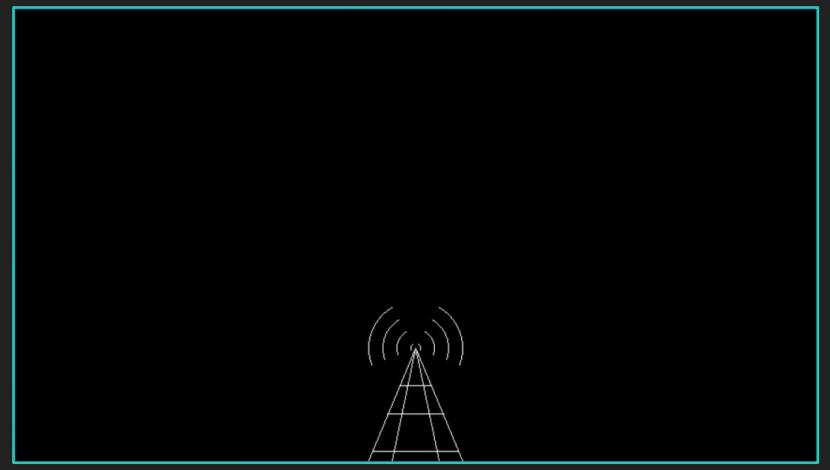
```c
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <dos.h>
int main()
{
    int gdriver = DETECT, gmode, err;
    int midx, y, radius = 5;
    int k = 0, stangle, endangle;

    initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

    err = graphresult();
    if (err != grOk)
    {   /* error occurred */
        printf("Graphics Error: %s\n",grapherrormsg(err));
        getch();
        return 0;
    }
    /* mid position in x-axis */
    midx = getmaxx() / 2;

    /* calculating the position of y */
    y = (3 * getmaxy()) / 4;

    /* start and end angles of signals */
    stangle = 120;
    endangle = 200;
```

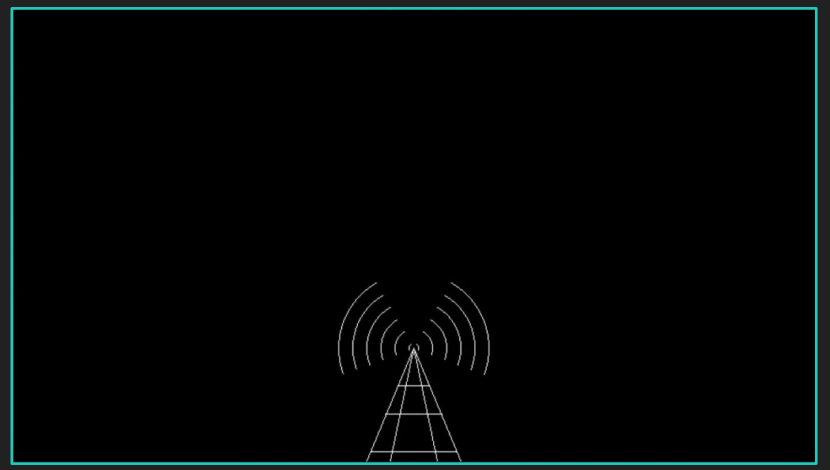
Program

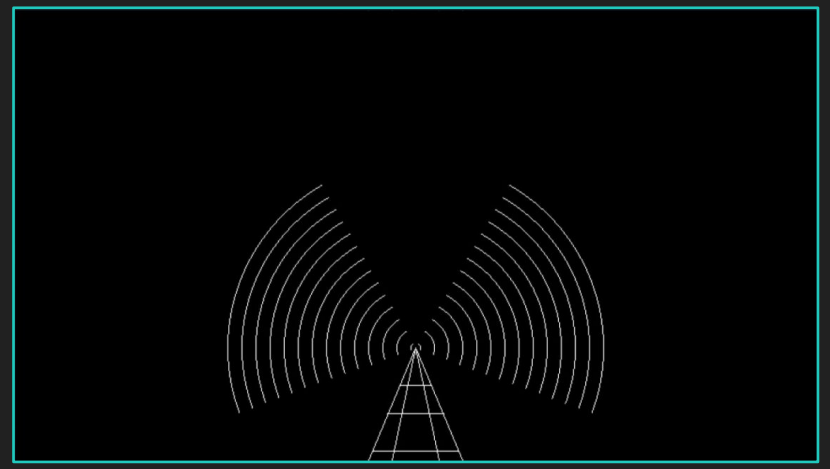


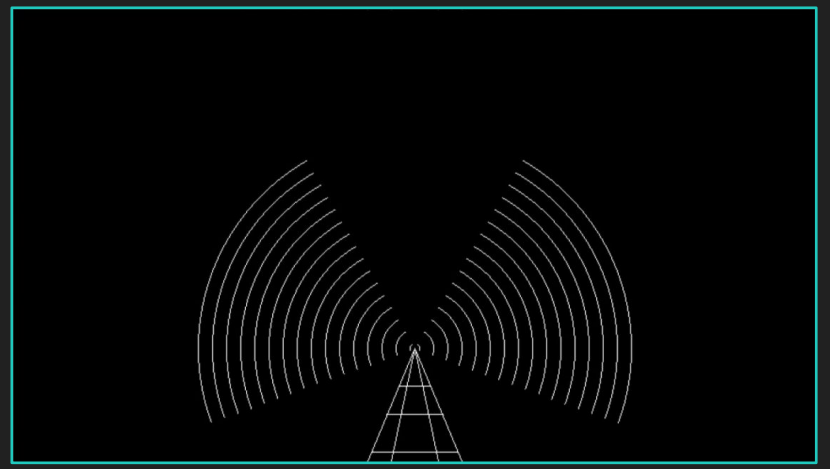
```
    while (!kbhit())
    {
        k = 0, radius = 5;
        /* clears graphic screen */
        cleardevice();

        /* construct antenna using lines */
        line(midx - 50, getmaxy(), midx, y);
        line(midx + 50, getmaxy(), midx, y);
        line(midx - 25, getmaxy(), midx, y);
        line(midx + 25, getmaxy(), midx, y);
        line(midx - 45, getmaxy() - 10, midx + 45, getmaxy() - 10);
        line(midx - 30, getmaxy() - 50, midx + 30, getmaxy() - 50);
        line(midx - 16, getmaxy() - 80, midx + 16, getmaxy() - 80);
        /* signals from the antenna */

        while (k < 18)
        {
            /* signal at the left side */
            arc(midx, y, stangle, endangle, radius);
            /* signal at the right side */
            arc(midx, y, 0, 60, radius);
            arc(midx, y, 340, 360, radius);
            radius = radius + 15;
            delay(50);
            k++;
        }
    }
    getch();

    /* deallocate memory allocated for graphic screen */
    closegraph();

    return 0;
}
```

Output



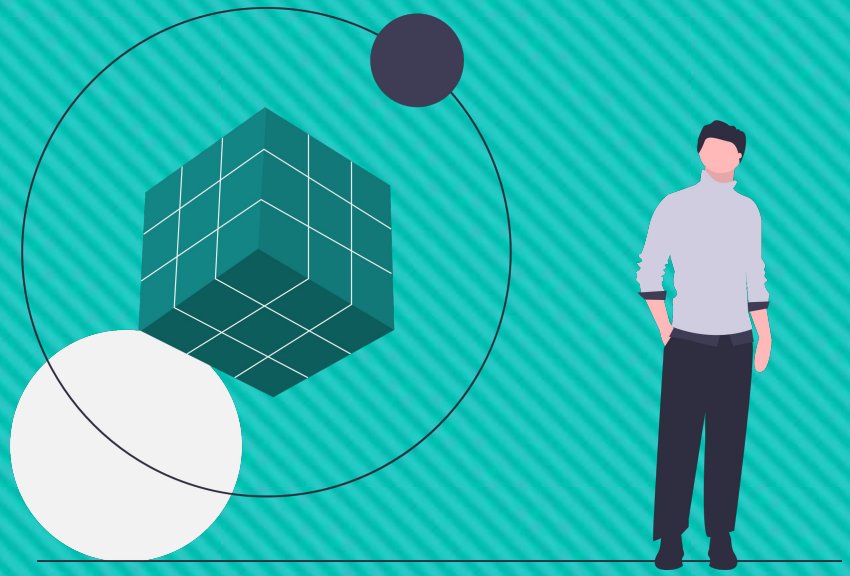






Conclusion

- ✔ From the above program and output, we conclude that the project is able to perform Visualization of Signal Transmission i.e. the Transmission of Electromagnetic Waves from an antennas or Satellites .
- ✔ Aim for the Project is Successfully Achieved !!!



Thank You

Name :- Singh Sudham Dharmendra

Roll no :- AIML D - 50

Branch :- CSE (AI & ML)

Subject :- Computer Graphics

Topic :- Assignment No. 1

Date of Submission :- 20/11/2021

Signature :-

Q.1) Differentiate between Raster scan & Random scan devices.

→

Raster scan	Random scan
1. Electron beam - Swept across the screen and handles one row at a time and in downward direction.	1. Electron beam - Directed to the portion of the screen where a picture is to be rendered.
2. Resolution Poor, since it generates meander lines which are organized as distinct point sets.	2. Resolution Good as their produces even line drawings
3. Picture definition - Stored as a combination of intensity values for all screen points	3. Picture definition - stored as a group of line drawing instructions in a display file.
4. Realistic display - Effectively displays realistic scenes.	4. Realistic display - Unable to display realistic shaded scenes

Q.2) Explain DDA line drawing algorithm and plots the points for line AB A (10, 15) & B (5, 25) using it.

→ Step 1: Start.

Step 2: Starting coordinates of the line segment are

$P(x_1, y_1) = P_1(10, 15)$

Ending co-ordinates of the line segment are

$P_2(x_2, y_2) = P_2(5, 25)$

Step 3:: Calculate $dx = x_2 - x_1 = 5 - 10$

$\therefore dx = -5$

Step 4: Calculate $dy = y_2 - y_1$

$= 25 - 15$

$dy = 10$

Step 5: Absolute value

As abs $(dy) >$ abs (dx)

abs $(10) >$ abs (5)

length $= 10$

Step 6: $x_{increment} = dx / length$

$= -5/10$

$= -0.5$

$y_{increment} = dy / length$

$= 10/10$

$= 1$

Step 7: Starting point position ie,

Plot (10, 15, WHITE)

Step 8: For (k=1)

Pass 1

$x_2 = x_{1+1} = x_1 + x_{increment}$

$= 10 + (-0.5)$

$x_2 = 9.5$ Round $(x_2) = 10$

$y_2 = y_1 + y_{increment}$

$= 15 + 1$

$y_2 = 16$

Plot (10, 16, WHITE)

For (k=2)

Pass 2

$x_3 = x_2 + x_{increment}$

$= 9.5 + (-0.5)$

$x_3 = 9$

$y_3 = y_2 + y_{increment}$

$= 16 + 1$

$y_3 = 17$

Plot (9, 17, WHITE)

Pass 3 (k=3)

$x_4 = x_3 + x_{increment}$
$= 9 + (-0.5)$
$x_4 = 8.5$ Round $(x_4) = 9$

$y_4 = y_3 + y_{increment}$
$= 17 + 1$
$y_4 = 18$

Plot (9, 18, WHITE)

Pass 4 (k=4)

$x_5 = x_4 + x_{increment}$
$= 8.5 + (-0.5)$
$x_5 = 8$

$y_5 = y_4 + y_{increment}$
$= 18 + 1$
$y_5 = 19$

Plot (8, 19, WHITE)

Pass 5 (k=5)

$x_6 = x_5 + x_{increment}$
$= 8 + (-0.5)$
$x_6 = 7.5$ Round $(x_6) = 8$

$y_6 = y_5 + y_{increment}$
$= 19 + 1$
$y_6 = 20$

Plot (8, 20, WHITE)

Pass 6 (k=6)

$x_7 = x_6 + x_{increment}$
$= 7.5 + (-0.5)$
$x_7 = 7$

$y_7 = y_6 + y_{increment}$
$= 20 + 1$
$y_7 = 21$

Plot (7, 21, WHITE)

Pass 7 (k=7)

$x_8 = x_7 + x_{increment}$
$= 7 + (-0.5)$
$x_8 = 6.5$ Round $(x_8) = 7$

$y_8 = y_7 + y_{increment}$
$= 21 + 1$
$y_8 = 22$

Plot (7, 22, WHITE)

Pass 8 (k=8)

$x_9 = x_8 + x_{increment}$
$= 6.5 + (-0.5)$
$x_9 = 6$

$y_9 = y_8 + y_{increment}$
$= 22 + 1$
$y_9 = 23$

Plot (6, 23, WHITE)
Pass 9 (k = 9)

$x_{10} = x_9 + x_{increment}$
$= 6 + (-0.5)$
$x_{10} = 5.5$ Round $(x_{10}) = 6$

$y_{10} = y_9 + y_{increment}$
$= 23 + 1$
$y_{10} = 24$

Plot (6, 24, WHITE)
Pass 10 (k = 10)

$x_{11} = x_{10} + x_{increment}$
$= 5.5 + (-0.5)$
$x_{11} = 5$

$y_{11} = y_{10} + y_{increment}$
$= 24 + 1$
$y_{11} = 25$

Plot (5, 25, WHITE)
Step 9 : Stop.

Q.3) Derive the composite ~~function~~ transformation matrix to scale an object with respect to a fixed position.

→ The procedure of scaling with respect to an ordinary arbitary fixed ~~positions~~ point is :-

1. Translate the object ~~so~~ that the fixed point coincides with origin
2. Scale the object with respect to the origin.
3. Use the inverse translation of step 1 to return the object to its original position.

The corresponding composite transformation matrix :-

$$\begin{bmatrix} 1 & 0 & X_F \\ 0 & 1 & Y_F \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -X_F \\ 0 & 1 & -Y_F \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_x & 0 & X_F(1 - S_x) \\ 0 & S_y & Y_F(1 - S_y) \\ 0 & 0 & 1 \end{bmatrix}$$

Q.4) Derive the 2D transformation matrix to reflect a figure about the line $y=mx$

→ $y = mx$

Hence, slope m & y-intercept = 0

we can relate slope m to angle θ by equation

$$m = \tan\theta$$

$$\theta = \tan^{-1} m$$

θ is the inclination of line with respect to X-axis

Translation matrix can be given as

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

Rotation matrix to match the given line with X-axis can be obtained as.

$$R_2 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflection matrix about X-axis

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

Inverse transformation matrices

$$R_2^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Final transformation matrix can be obtained as

$$R_T = T \cdot R_2 \cdot M \cdot R_2^{-1} \cdot T^{-1}$$

As we have $\tan\theta = m$, using trignometric identities

$\sin\theta = \frac{m}{\sqrt{m^3+1}}$ $\quad \cos\theta = \frac{m}{\sqrt{m^3+1}}$

$$R_T = \begin{bmatrix} \cos 2\theta & \sin 2\theta & 0 \\ \sin 2\theta & -\cos 2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & 0 \\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(Q.5) Specify midpoint circle algorithm using the same plot circle whose radius is 5 units and centre is at (10,10)

→ Circle drawing algorithm

Step 1: The centre co-ordinates of the circle are $(x_c, y_c) = (10,10)$ and radius = 5 units.
The fixed point on the circle boundary will be
$(x_0, y_0) = (0,5) = (0,8)$

Step 2: Plot the first point $(0,5) \equiv (0,8)$

Step 3: Consider the symmetry property of circle and find the remaining points in other quadrants.

Step 4: Intial decision parameter at $k=0$
$P_0 = 1-r$

Step 5: Starting $k=0$ at X_k position perform the following steps

If $(P_k < 0)$ then midpoint is inside the boundary of circle -

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

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

Plot (x_{k+1}, y_{k+1})

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

If $(P_k \geq 0)$ the point is outside the boundary of the circle.

$x_{next} = x_{k+1}$ $\quad$ $y_{next} = y_k - 1$

Plot $(x_k + 1, y_k - 1)$

$P_{k+1} = P_k + 2x_k - 2y_k + 5$

Step 6: Translate each calculated fixed position by $T(x_6, y_6)$ to new position.

$X = x_{next} + x_6$ $\quad$ $Y = y_{next} + Y_6$

Step 7: Repeat the steps until $x \geq y$

Step 8: Stop.

(2)

Step 1: Accept the centre co-ordinates (10, 10) and radius 5 units.
The first point will be $(x_0, y_0) = (0, 5)$

Step 2: Plot the first point (0, 5)

Step 3: Remaining points will be (0, 5) (0, -5) (5, 0) (-5, 0)

Step 4: Initial decision parameter at $k = 0$

$P_0 = 1 - r = 1 - 5 = -4$

Step 5: at each x_k position.

Pass 1 at k=1

($P_0 < 0$) point is inside bandary of circle

$x_1 = x_0 + 1 = 1$

$y_1 = y_0 = 5$

$P_1 = P_0 + 2x_0 + 3$

$= -4 + 3 = -1$

Translate

$X = 1 + 10 = 11$

$y = 5 + 10 = 15$

Pass 2 at k=2

($P_1 < 0$) Point is inside the circle boundary.

$x_2 = x_1 + 1 = 2$ $\quad y_2 = y_1 = 5$

Plot (2,5)

$P_2 = P_1 + 2x_1 + 3$

$= -1 + 2 + 3$

$P_2 = 4$

Translate

$X = 2 + 10 = 12$ $\quad Y = 5 + 10 = 15$

Pass 3 k=3

($P_2 >= 0$) Point is outside circle boundary

$x_3 = x_2 + 1 = 3$

$y_3 = y_2 - 1 = 4$

Plot 3,4

$P_3 = P_2 + 2x_2 - 2y_2 + 5$

$= 4 + 4 - 10 + 5$

$= 3$

Translate

$x = 3 + 10 = 13$ $\quad$ $Y = 4 + 10 = 14$

Pass 4 at k=4

($P_3 > 0$) Point is outside circle boundary.

$x_4 = x_3 + 1 = 4$

$y_4 = y_3 - 1 = 3$

Plot (4,3)

$P_4 = P_3 + 2x_3 - 2y_3 + 5$

$= 3 + 6 - 8 + 5$

$P_4 = 6$

Translate

$x = 4 + 10 = 14$ $\quad$ $Y = 3 + 10 = 13$

Step 6: Determining points from other octant

Pass 1:	(1,5)	(-1,5)	(-1,-5)	(5,-1)
	(-5,1)	(5,-1)	(5,1)	(-5,-1)
Pass 2:	(2,5)	(-2,5)	(-2,-5)	(2,-5)
	(5,2)	(-5,2)	(-5,-2)	(5,-2)
Pass 3:	(3,4)	(-3,4)	(-3,-4)	(3,-4)
	(4,3)	(-4,3)	(-4,-3)	(4,-3)
Pass 4:	(4,3)	(-4,3)	(-4,-3)	(4,-3)
	(3,4)	(-3,4)	(-3,-4)	(3,-4)

Step 7: Since $x = 4 >= y = 3$

Stop

Name :- Singh Sudham Dharmendra

Roll no :- AIMLD-50

Branch :- CSE (AI & ML)

Subject :- Computer Graphics

Topic :- Assignment No. 2

Date of Submission :- 25/11/2021

Signature :- Sudham

Sudham D. Singh
AIML D50

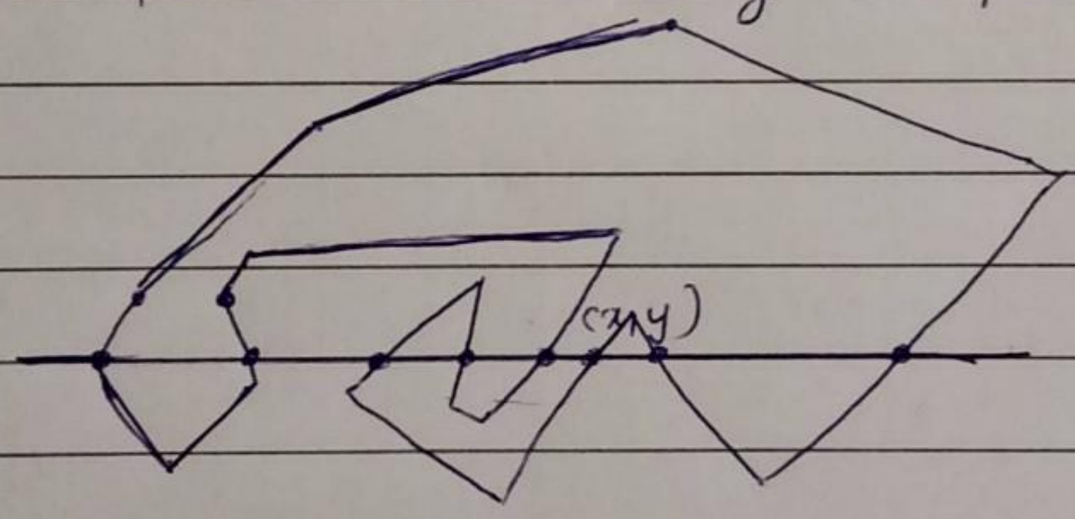
2.i) What is the purpose of Indside - outside Test, explain any one method.

→ This method is also known as counting number method while filling an object we often need to identify whether particular point is inside the object or outside it. These method by which we identify whether particular point is inside or outside an object.

① Odd - Even Rule
② Non zero winding number rule.

Odd - Even Rule :-

In this technique, we will count the edge crossing along the line from any point (x,y) is an interior point, and if the number of interaction is even, then the point (x,y) is an exterior point. The following example depicts the concept.



From the above fig, we can see that from the point (x,y) the number of interactions point on the left side is 5 and the right side is 3. From both ends, the min of interaction point is odd, so the point is considered with in the object.

Q.2) Explain Area subdivision algorithm for hidden surface removal.

→ Area subdivisional method is :-

The area - subdivision method takes advantage by locating those view area that represents part of a single surface by dividing the total viewing area into smaller and smaller rectangles until each small area is projection ~~by~~ of part of a single visible surface or no surface at all.
The algorithm is a recursive procedure based on a 2-step strategy.

(i) Decide which ~~program~~ polygon overlaps the given area on the screen.

(ii) Which polygon are visible in that area.

So the area subdivision method checks whether the polygon is ~~partially~~ pratially or fully visible in the given area.

Hence according to Screen area the polygons are classify into 4 categories -

1) Surrounding polygon
2) intersecting polygon
3) Contained polygon
4) Disjoined polygon.

Q.3) Explain Liang Barsky line clipping algorithm, what is its benefit over Cohen Sutherland algorithm? Clip the line with co-ordinates (5,10) and (35,30) against the window $(x_{min}, y_{min}) = (10,10)$ and $(x_{max}, y_{max}) = (20,20)$

→ Liang Barsky line clipping algorithm is faster line algorithm based on analysis of the parametric eq. of a line segment

→ $X = X_1 + U\Delta X$ & $Y = Y_1 + U\Delta Y$

This method uses the following concepts -

① The parametric eqn of the line

② The inequalities describing the range of the clipping window which is used to determine the intersections between the line and the clip window.

→ $X_{wmin} \leq X_1 + U(X_2 - X_1) \leq X_{wmax}$

$Y_{wmin} \leq Y_1 + U\Delta Y \leq Y_{wmax}$

Each of these four inequalities can be expressed as

$$\mu p_k \leq q_k \text{ for } k = 1, 2, 3, 4$$

The parameters p and q are defined as :-

$p_1 = -\Delta x$ & $q_1 = x_1 - x_{min}$ (Left boundary)

$p_2 = \Delta x$ & $q_2 = x_{max} - x_1$ (Right boundary)

$p_3 = -\Delta y$ & $q_3 = y_1 - y_{min}$ (Bottom boundary)

$p_4 = \Delta y$ & $q_4 = y_{max} - y_1$ (Top boundary)

If a line is || to a view window boundary, the p values for the boundary is 0. And if line is || to X axis, assume $p_1 = 0$ & $p_2 = 0$

→ Given $p_k = 0$ if $q_k < 0$ the line is trivially invisible because it is outside new window

→ Given $p_k = 0$ if $q_k > 0$ then the line is inside the corresponding window boundary.

When $pk < 0$, as U increases line goes from outside to inside ie; entering and if $pk > 0$, lines goes from inside to outside ie, exiting.

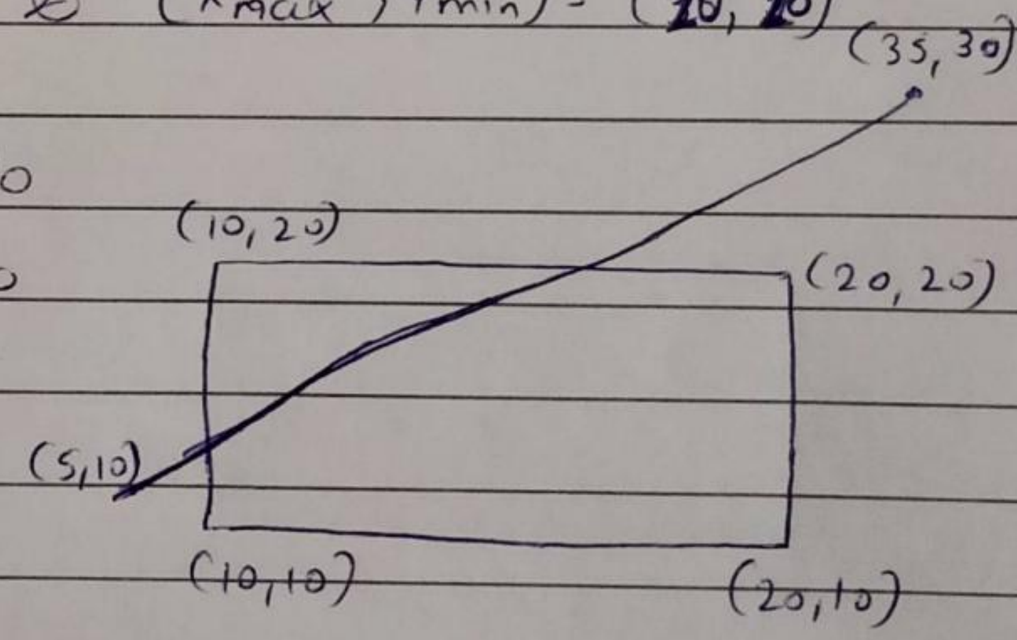
If ~~there~~ is a segment of line ~~within~~ inside the clip region, a sequence of infinite line intersection must go entering, entering, exiting & exiting.

~~If there is a seg.~~ The Liang-Barsky algorithm is more efficient than Cohen-Sutherland line Clipping algorithm & can be extended to 3-Dimensional Clipping.

Given:-

line with co-ordinate (5,10) & (35,30)

Window (x_{min}, y_{max}) = (10,10) & (x_{max}, y_{min}) = (20,20)



$X_L = 10$ $Y_B = 10$ $X_R = 20$ $Y_T = 20$

& $x_1 = 5$ $Y_1 = 10$ $x_2 = 35$ $Y_2 = 30$

$\Delta x = x_2 - x_1$

$= 35 - 5 = 30$

$\Delta y = y_2 - y_1$

$= 30 - 10 = 20$

Calculate P & Q

$P_1 = -\Delta x = -30$ $P_3 = -\Delta y = -20$

$P_2 = \Delta x = 30$ $P_4 = \Delta y = 20$

$Q_1 = x_1 - x_L = 5 - 10 = -5$

$Q_2 = x_R - x_1 = 20 - 5 = 15$

$Q_3 = y_1 - y_B = 10 - 10 = 0$

$Q_4 = y_T - y_1 = 20 - 10 = 10$

Now find out values for P for i = 1 to 4

$P_1 = \frac{Q_1}{P_1} = \frac{-5}{-30} = \frac{1}{6}$ $\quad P_2 = \frac{Q_2}{P_2} = \frac{15}{30} = \frac{1}{2}$

$P_3 = \frac{Q_3}{P_3} = \frac{0}{-20} = 0$ $\quad P_4 = \frac{Q_4}{P_4} = \frac{10}{20} = \frac{1}{2}$

Then let $t_1 = 0$ & $t_2 = 1$

$\therefore$ New, $t_1 = \text{Max}(1/6, 0, 0) = 1/6$

$t_2 = \text{Min}(\frac{1}{2}, \frac{1}{2}, 0) = 1/2$

$\therefore x_1' = x_1 + \Delta x * t_1 = 5 + (30)(1/6)$

$x_1' = 10$

$y_1' = y_1 + \Delta y * t_1 = 10 + 20(1/6)$

$y_1' = 13.33$

$x_2' = x_2 + \Delta x * t_2 = 5 + (30 \times 1/2)$

$x_2' = 20$

$y_2' = y_1 + \Delta y * t_2 = 10 + (20 \times 1/2)$

$y_2' = 20$

From this we will come to know that a point (20, 20) is an intersection point with respect to the edge of the window boundary. So we need ~~to~~ discard the line from (5, 10) to (10, 13.33) & consider line from (10, 13.33) to (20, 20).

Q.4) Define Window, Viewport, Object in object space, Image in image space, world co-ordinate system, physical device co-ordinate system, obtain viewing transforming matrix.

→ i) Window :-

A world-co-ordinate area selected for display is called a window.

(ii) Viewport :-

An area on a display device to which a window, is mapped is called viewport.

(iii) Object in object space :-

The space where object model reside is called object space.

(iv) Image in image space :-

The optical space co-ordinating the visual representation or component of a scene.

(v) World space co-ordinate system :-

The object space contains the dimensions actual which is called as world co-ordinate system.

(vi) Physical device co-ordinate :-

Co-ordinate system on the display device where the image of the picture is display device where the image of the picture is display.

(vii) viewing transformation matrix :-

Mapping of a part of world co-ordinate system scene to device co-ordinate system is refered as viewing transformation also known as viewport transformation

Viewing transformation contain 3 steps :-

step - i) Translate window to origin

$T_x = -X_{wmin}$ $T_y = -Y_{min}$

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X_{wmin} & -Y_{min} & 1 \end{bmatrix}$$

Step 2) Scaling of the window to match it's size to the viewport.

$$S_x = \frac{(X_{wmax} - X_{vmin})}{(X_{wmax} - X_{wmin})} - \frac{\text{Width of new port}}{\text{width of window}}$$

$$S_y = \frac{(Y_{vmax} - Y_{vmin})}{(Y_{wmax} - Y_{wmin})} - \frac{\text{height of view port}}{\text{height of window}}$$

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

Step 3) Again translate viewport to its correct position on screen.

Where $T_x = X_{vmin}$ $\quad T_y = X_{vmin}$

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ X_{vmin} & Y_{xmin} & 0 \end{bmatrix}$$

Viewing transformation = T.S.T

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X_{vmin} & -Y_{vmin} & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ X_{vmin} & Y_{min} & 1 \end{bmatrix}$$

(Q.5) What is meant by parallel and perspective projection ? Derive matrix for perspective projection.

→ Parallel Projection :-

In parallel projection, z co-ordinate is discarded and parallel line from each vertex on the object are extended until they intersect the view plane. The point of intersection is the projection of the vertex. We connect the projected vertices by line segments which correspond to connection on the original object. There are two types of parallel projection :-

(i) Orthographic parallel projection
(ii) Oblique parallel Projection.

Perspective Projection :-

The perspective project produces realistic view but does not preserve relative proportions. In perspective projection, the lines of projections are not parallel. Instead they all converge at a single point called as the centre of projection. The object positions are transformed to the view plane along these converged projection lines and the projected view of an object is determine by calculating the intersection of the converged projection line with the view plane. There are 4 types of perspective projections

i) Co-ordinate description.
ii) one-point perspective projection of cube
iii) Two-point of perspective projection of cube.
iv) Three point perspective of a cube.

The perspective projection transformation can be obtained by performing two operations.

1) Make the centre line of the frustum perpendicular to the view plane by shearing the view volume.
2) Scale the view volume with a scaling factor that depends on $1/z$

Hence, shearing matrix =

$$[x' \; y' \; z \; 1] = [x \; y \; z \; 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ a & b & 1 & 0 \\ -a z_{prp} & -b z_{prp} & 0 & 1 \end{bmatrix}$$

Where, $a = -\left(\frac{x_{prp} - (x_{wmin} + x_{wmax})/2}{z_{prp}}\right)$

$$b = -\left(\frac{y_{prp} - (y_{wmin} + y_{wmax})/2}{z_{prp}}\right)$$

By matrix multiplication,

$$x' = x + a(z - z_{prp})$$
$$y' = y + b(z - z_{prp})$$
$$z' = z$$

Scaling of the object

$$[x'' \; y'' \; z \; 1] = [x' \; y' \; z \; 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-x_{prp}}{z_{prp} - z_{vp}} & \frac{-y_{prp}}{z_{prp} - z_{vp}} & 1 & \frac{-1}{z_{prp} - z_{vp}} \\ \frac{x_{prp} z_{vp}}{z_{prp} - z_{vp}} & \frac{y_{prp} z_{vp}}{z_{prp} - z_{vp}} & 0 & \frac{z_{vp}}{z_{prp} - z_{vp}} \end{bmatrix}$$

By matrix multiplication,

$$x'' = x' \left[\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right] + x_{prp} \left[\frac{z_{vp} - z}{z_{prp} - z}\right]$$

$$y'' = y' \left[\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right] + y_{prp} \left[\frac{z_{vp} - z}{z_{prp} - z}\right]$$

Therefore the perspective projection translation can be expressed in matrix form as,

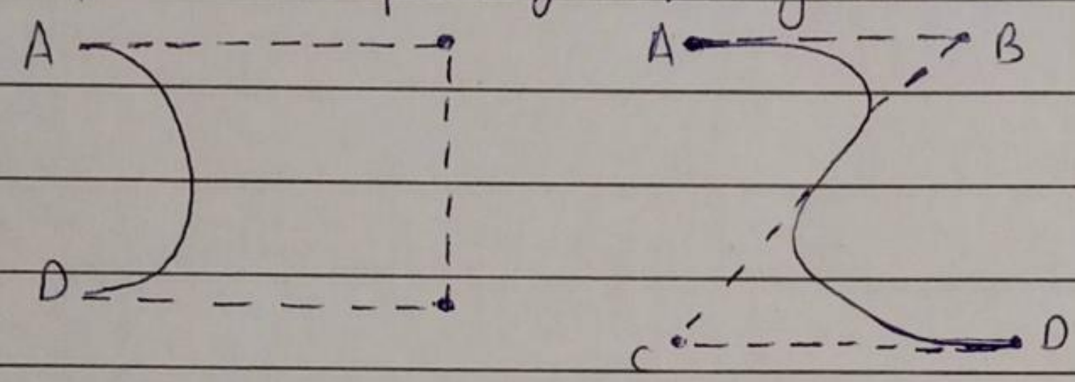
$$M_{perspective} = M_{shear} \cdot M_{scale}$$

Sudham D. Singh
AIML D 50

(Q6) Explain Bezier Curve and Bspline curve? State the various properties of Bezier curve.

→ (1) Bezier Curve :-

(i) It is a different way of specifying a curve, rather same shapes can be represented by B-spline & Bezier curves. The cubic Bezier curve require four sample points, these points completely specify the curve.



(ii) The curve begins at the first sample point and ends at fourth point. If we need another Beizer curve then we need another four sample points. But if we need two Beizer curves connected to each other, then with six sample points we can achieve it for should this, the third and fourth point of first curve should be made same as first and second point of curve.

(iii) The equation for the Beizer curve are as follows :-

$$X = X_4 a^3 + 3X_3 a^2(1-a) + 3x_2 a(1-a)^2 + X_1(1-a)^3$$

$$Y = X_4 a^3 + 3Y_3 a^2(1-a) + 3Z_2 a(1-a)^2 + Y_1(1-a)^3$$

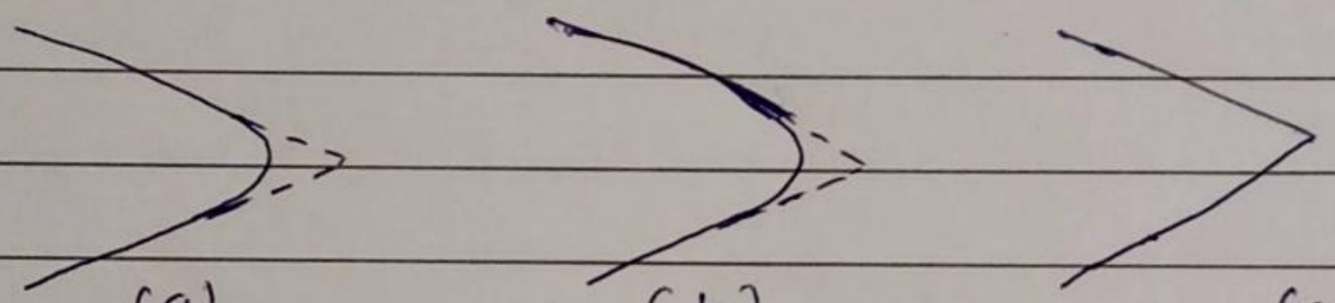
$$z = z_4 a^3 + 3Z_3 a^2(1-a) + 3Z_2 a(1-a)^2 + Z_1(1-a)^3$$

(iv) Here as the value of 'a' moves 0 to 1, the curve travels from the first to fourth sample point. But we can construct a Beziex curve without referencing to the above expression It is constructed by simply taking midpoints.

(2) B-Spline Curve:-

(i) A set of ~~bel~~ blending curve functions which takes this approach is called B-splines. Basically spline means a strip, which we have to move around the sample point.

(ii) Generally the B-spline bending functions were designed to elimirate share corners in the curve and the curve does not usually pass through the sample points. But if we need sharp coorners we can produce it by using many identical sample points.



(3) Properties of Bezier Curve are as follows:-

(1) The bais function are real in nature.
(2) Bezier curve always pases through the first & last control point i.e; curve has same end points as the guiding polygon.
(3) The degree of polynomial defining the curve segment is one less than the number of defining polygon point.
(4) The curve gererally follows the shape of the ~~following~~ defining polygon.
(5) The direction of the tangent vector at the end points is the same as that of the vector determined by first and last segments.
(6) The curve lies entirely within the coorner null formed by for control points.
(7) The curve exhibits the variation diminishing property. This means that the curve does not oscillate about any stracight line more than the defining polygon.
(8) The curve is invariant under an affine transformation.

Sudham D. Singh
AIMLD50

(Q.7) Explain 3D rotations about an arbitrary axis.

→ (1) When the object is rotated about an axis that is not parallel to any one of co-ordinate axis, ie. x, y, z. Then additional transformation are required. First of all, alignment is needed, and then the object is being back to the original position.

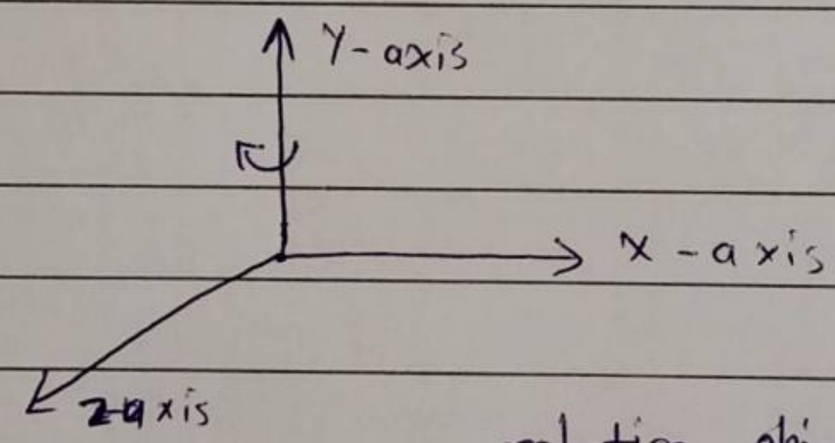
(2) Following steps are required:-

(i) Translate the object to the origin.

(ii) Rotate object so that axis of object coincide with any of co-ordinate axis.

(iii) Perform rotation about co-ordinate axis with whom coinciding is done.

(iv) Apply inverse rotation to bring rotation back to the original position.



rotation object Y-axis clockwise.

Matrix form representing 3D rotations about the Z axis.

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

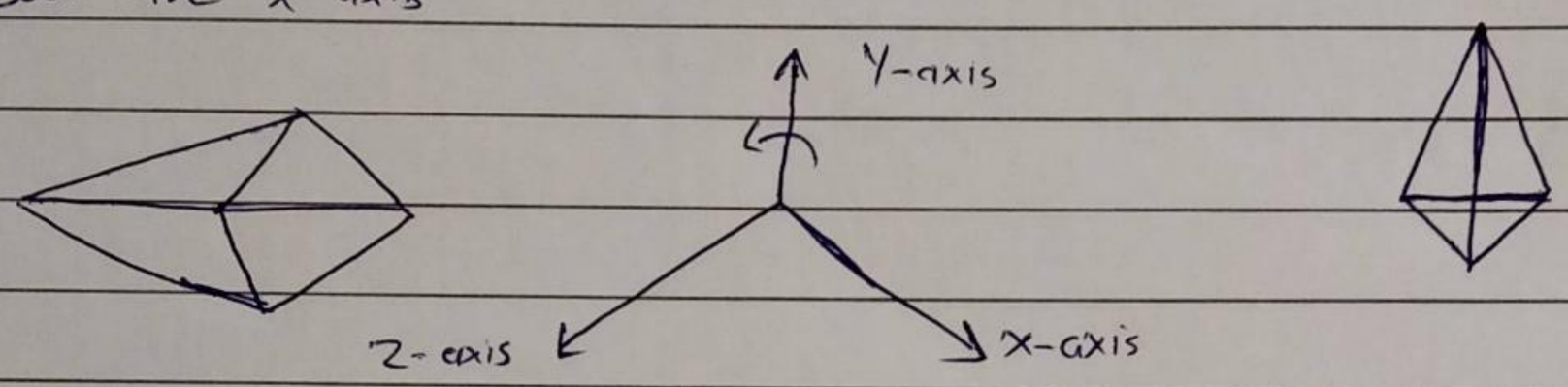
Matrix for representing 3D rotations about the x-axis,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix for representing 3D rotations about Y-axis.

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Following figure show the original position of object after rotation about the X-axis



(v) Apply inverse translation to bring rotation axis to the original position.

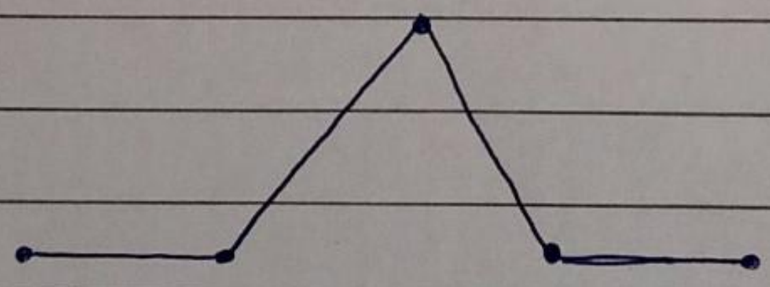
~~for sixth~~

Q.8) Explain koach Curve & Factals.

→ A] Koach Curve :-

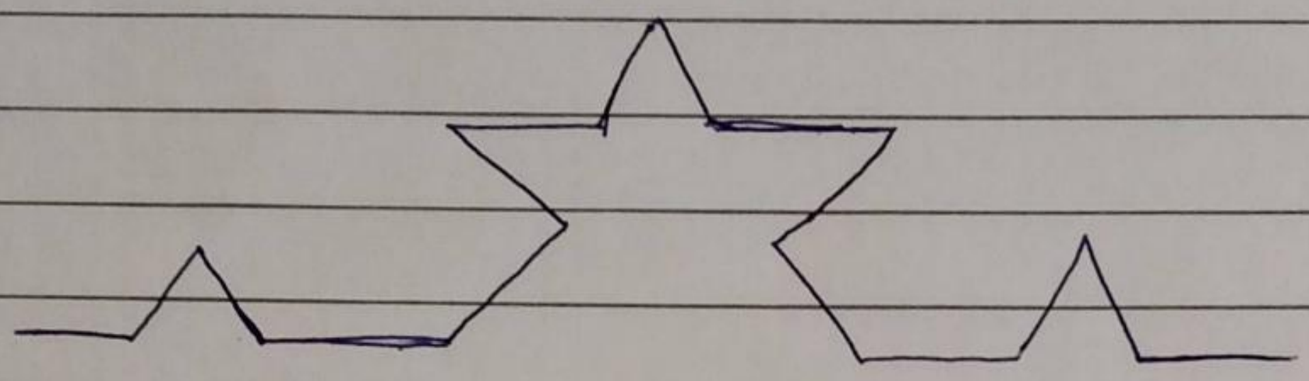
① The koach curve can be drawn by dividing line into 4 equal segments with scaling factor 1/3 and middle two segments are so adjusted that they form adjacent sides of an equilateral triangle is shown in the fig(a).

This is the first approximation to the koach curve.



(a) First approximation of the koach curve.

(2) To apply second approximation to the koch curve we have to repeat the above process for each of the four segments. The resultant curve is shown in fig (b).



(b) The second approximation to koch curve.

(3) The resultant curve has more wiggles and its length is 16/9 times the original length.

(4) From the above figure we can easily note following point about the koch curve :-

(i) Each repetition increases the length of the curve by factor 4/3.
(ii) Length of curve is infinite.
(iii) Unlike Hibert's curve, it doesn't fill an area
(iv) It doesn't deviate much from its original shape.
(v) If we reduce the scale of the curve by 3 we find the curve that looks just like the original one, but we must assemble 4 such curves to make the original, so we have,

$$4 = 3$$

solving for D, we get;

$$D = \log_3 4 = \log 4 / \log 3$$

$$= 1.2618$$

$$\therefore \boxed{D = 1.2618}$$

(5) Therefore for koch curve topological dimension is 1 but fractal dimension is 1.2618

B] Fractals :-

(1) A fractal is defined as a rough for fragmented geometric shape that can be split into parts, each of which is approximately a reduced - size reproduction of the complete shape based on the property known as self - similarity. It was derived from the latin word 'fractus' which means broken or fractured. Natural objects can be realistically described using fractal geometry methods

(2) Fractal methods use procedures rather than equations to model objects, so it uses procedural modelling.

(3) The major characteristics of any procedural model is that the model is not based on data, but rather on the implementation of the procedure following a particular set of rules.

(4) Its parts have the same form or structure as a whole, except that they are at a different scale and may be slightly deformed.

(5) Its form is extremely irregular or fragmented and remains so, whatever the scale of examination.

(6) It is formed by iteration ie, the procedure is used repeatedly (recursively).