RIET | RAJASTHAN INSTITUTE OF ENGINEERING AND TECHNOLOGY

Established in Year 2000 - RTU QIV RANKED A | Approved by AICTE and Affiliated to Rajasthan Technical University, Kota.

LAB MANUAL

COMPUTER GRAPHICS

(Session :2020-21)
B.Tech V Sem (5CS4-21)

Department of Computer Science

Submitted To : Mr. Anil Dhankar
Submitted By : Arya Singh
Roll No. : 18ERECS012
Batch : A1
Session : 2018-22

Sno	Experiment	Page No.
1	Study of basic graphics function defined in “Graphics.h” line,lineto,circle,ellipse,putpixcel,getpixcel,outtextxy,setcolor,arc,closegrap h	
2	Write a program to draw a hut or another geometrical figures(any one).	
3	Write a program to draw a line through Bresenham’s Alogrithm.	
4	Write a program to draw a line through DDA Alogrithm.	
5	Write a program to draw a circle using Mid-point algorithm.	
6	Write a program to draw a Ellipse using Mid-point algorithm.	
7	Write a menu driven program to rotate , scale and translate a line ,point , square, triangle(any one object) about the origin.	
8	Write a program to implement reflection of a point, line.	
9	Write a program to perform shearing on a line.	
10	Write a program to perform line clipping.	
11	Write a program to implement polygon filling(any one).	
12	Write a program to implement transformations in three dimensions(any one).	

EXPERIMENT - 1

AIM : Study of basic graphics function defined in “Graphics.h” Line, lineto, circle, ellipse, putpixcel, getpixcel, outtextxy, setcolor, arc, closegraph

C graphics using graphics.h functions can be used to draw different shapes, display text in different fonts, change colors and many more. Using functions of graphics.h in turbo c compiler you can make graphics programs, animations, projects and games.

Some of the functions are:
Line function in c
line function is used to draw a line from a point(x1,y1) to point(x2,y2) i.e. (x1,y1) and (x2,y2) are end points of the line.The code given below draws a line.
Declaration :- void line(int x1, int y1, int x2, int y2);
C programming code for line

```
#include<graphics.h>
#include<conio.h>
main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TurboC3\\BGI"); line(100, 100, 200,200);
getch();
closegraph();
return 0;
}
```

lineto function in c
lineto function draws a line from current position(CP) to the point(x,y), you can get current position using getx and gety function.
C programming code for lineto:

```
#include<graphics.h>
#include<conio.h>
main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
moveto(100, 100);
lineto(200, 200);
getch();
closegraph();
return 0;
}
```

circle function in c
Declaration :- void circle(int x, int y, int radius);
circle function is used to draw a circle with center (x,y) and third parameter specifies the radius of the circle. The code given below draws a circle.
C program for circle

```
#include<graphics.h>
#include<conio.h>
main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
 circle(100, 100, 50);
getch();
}
closegraph(
);
 return 0;
}
```

ellipse function in c
Declarations of ellipse function :-
void ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);
Ellipse is used to draw an ellipse (x,y) are coordinates of center of the Ellipse, stangle is the starting angle, end angle is the ending angle, and fifth and sixth parameters specifies the X and Y radius of the ellipse. To draw a complete ellipse strangles and end angle should be 0 and 360 respectively.
C programming code for ellipse

```
#include<graphics.h>
#include<conio.h>
main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TurboC3\\BGI"); ellipse(100, 100, 0, 360, 50, 25);
getch();
closegraph();
return 0;
}
```

putpixel function in c
putpixel function plots a pixel at location (x, y) of specified color. Declaration :- void putpixel(int x, int y, int color);

For example if we want to draw a GREEN color pixel at (35, 45) then we will write putpixel(35, 35, GREEN); in our c program, putpixel function can be used to draw circles, lines and ellipses using various algorithms.

C programming code for putpixel

```
#include<graphics.h>
#include<conio.h>
main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
 putpixel(25, 25, RED);
getch();
closegraph();
return 0;
}
```

getpixel function in c

getpixel function returns the color of pixel present at location(x, y).

Declaration :- int getpixel(int x, int y);

C program for getpixel

```
#include<graphics.h>
#include<conio.h>
main()
{
int gd = DETECT, gm,
color; char array[50];
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
color = getpixel(0, 0);
sprintf(array,"color of pixel at (0,0) = %d",color);
outtext(array);
getch();
closegraph();
return 0;
}
```

outtextxy function in c

outtextxy function display text or string at a specified point(x,y) on the screen.

Declaration :- void outtextxy(int x, int y, char *string);

x, y are coordinates of the point and third argument contains the address of string to be displayed.

C programming code for outtextxy

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

```
#include<conio.h>
main()
{
int gd = DETECT, gm;
initgraph(&gd,&gm,"C:\\ TurboC3\\BGI");
outtextxy(100, 100, "Outtextxyfunction");
getch();
closegraph();
return 0;
}
```

setcolor function in c

Declaration :- void setcolor(int color);

In Turbo Graphics each color is assigned a number. Total 16 colors are available.
Strictly speaking number of available colors depends on current graphics mode and driver.For Example :- BLACK is assigned 0, RED is assigned 4 etc. setcolor function is used to change the current drawing color.e.g. setcolor(RED) or setcolor(4) changes the current drawing color to RED. Remember that default drawing color is WHITE.

C programming code for setcolor

```
#include<graphics.h>
#include<conio.h>
main()
{
int gd = DETECT, gm;
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
circle(100,100,50);
setcolor(RED); /* drawn in red color */
Circle(200,200,50); /* drawn in white color
getch();
closegraph();
return 0;
}
```

arc function in c

Declaration :- void arc(int x, int y, int stangle, int endangle, int radius);

arc function is used to draw an arc with center (x,y) and stangle specifies starting angle, endangle specifies the end angle and last parameter specifies the radius of the arc. arc function can also be used to draw a circle but for that starting angle and end angle should be 0 and 360 respectively.

C programming source code for arc

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

```
main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm,"C:\\TurboC3\\BGI");
Arc(100, 100, 0, 135, 50);
getch();
closegraph();
return 0;
}
```

closegraph function in c

closegraph function closes the graphics mode, deallocates all memory allocated by graphics system and restores the screen to the mode it was in before you called initgraph.

Declaration :- void closegraph();

C code of closegraph

```
#include<graphics.h>
#include<conio.h>
main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\ TurboC3\\BGI");
outtext("Press any key to close the graphics mode...");
getch();
closegraph();
return 0;
}
```

EXPERIMENT - 2

AIM : Write a program to draw a hut or another geometrical figures(any one).

Program for creating House shape:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
setcolor(5);
rectangle(60,80,150,200);
rectangle(95,140,120,200);
line(60,80,100,15);
line(100,15,150,80);
circle(100,60,10);
getch();
closegraph();
}
```

EXPERIMENT - 3

AIM : Write a program to draw a line through Bresenham's Alogrithm.

Algorithm:

BRESENHAM'S LINE DRAWING ALGORITHM

```
1. Input the two line end-points, storing the left end-point
        in $(x_0 , y_0)$
2. Plot the point $(x_0 , y_0)$
3. Calculate the constants $\Delta x$, $\Delta y$, $2\Delta y$, and $(2\Delta y - 2\Delta x)$ and
        get the first value for the decision parameter as:
        $p_0 = 2 y - x$
4. At each $x_k$ along the line, starting at $k = 0$, perform the following
        test. If $p_k < 0$, the next point to plot is $(x_k +1, y_k)$ and:
        $p_{k+1} = p_k + 2 y$
        Otherwise, the next point to plot is $(x_k +1, y_k +1)$ and:
        $p_{k+1} = p_k + 2 y - 2 x$
5. Repeat step 4 $(\Delta x - 1)$ times
```

NOTE: The algorithm and derivation above assumes slopes are less than 1. For other slopes we need to adjust the algorithm slightly

Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
intx,y,x1,y1,x2,y2,p,dx,dy;
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
printf("\nEnter the x-coordinate of the first point::");
scanf("%d",&x1);
printf("\nEnter the y-coordinate of the first point ::");
scanf("%d",&y1);
printf("\nEnter the x-coordinate of the second point ::");
scanf("%d",&x2);
printf("\nEnter the y-coordinate of the second point ::");
scanf("%d",&y2);
x=x1;
y=y1;
dx=x2-x1;
dy=y2-y1;
putpixel(x,y,2);
p=(2*dy-dx);
while(x<=x2)
    {
        if(p<0)
           {
                x=x+1;
                p=p+2*dy;
                }
          else
           {
                x=x+1;
                y=y+1;
                p=p+(2*dy)-(2*dx);
                }
        putpixel(x,y,7);
          }
getch();
closegraph();
}
```

EXPERIMENT - 4

AIM : Write a program to draw a line through DDA Alogrithm.

Algorithm:

```
1. Start.
2. Declare variables x,y,x1,y1,x2,y2,k,dx,dy,s,xi,yi and also
        declare gdriver=DETECT, mode.
3. Initialize the graphic mode with the path location in TurboC3 folder.
4. Input the two line end-points and store the left end-points in (x1,y1).
5. Load (x1, y1) into the frame buffer; that is, plot the first point.
        put x=x1,y=y1.
6. Calculate dx=x2-x1 and dy=y2-y1.
7. If abs (dx) > abs (dy), do s=abs(dx).
8. Otherwise s= abs(dy).
9. Then xi=dx/s and yi=dy/s.
10. Start from k=0 and continuing till k<s,the points will be
i. x=x+xi.
ii. Y=y+yi.
11. Plot pixels using putpixel at points (x,y) in specified colour.
12. Close Graph and stop.
```

Program:

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
float round(float a);
void main()
{
        int gd=DETECT,gm; //gd=graphics driver (detects best graphics driver and assigns it as
        default, gm=graphicsmode.
        int x1,y1,x2,y2,steps,k;
        Float xincr,yincr,x,y,dx,dy;
        printf("enter x1,y1");
        scanf("%d%d",&x1,&y1);
        printf("enter x2,y2");
        scanf("%d%d",&x2,&y2);
        initgraph(&gd,&gm,"c:\\turboc3\\BGI");//initializes the graph
        dx=x2-x1;
        dy=y2-y1;
        if(abs(dx)>abs(dy))
        steps=abs(dx);
        else
        steps=abs(dy);
        xincr=dx/steps;
        yincr=dy/steps;
        x=x1;
        y=y1;
        for(k=1;k<=steps;k++)
        {
        delay(100);//for seeing the line drawing process slowly. x+=xincr;
        y+=yincr;
        putpixel(round(x),round(y),WHITE);
        }

        outtextxy(200,20,"DDA"); //for printing text at desired screen
        location.outtextxy(x1+5,y1-5,"(x1,y1)");
        outtextxy(x2+5,y2+5,"(x2,y2)");
        getch();
        closegraph(); // closes the graph and comes back to previous graphic mode.
        }
        float round(float a)
        {
        Int b=a+0.5;
        return b;
}
```

EXPERIMENT - 5

AIM : Write a program to draw a circle using Mid-point algorithm.

Algorithm:

1.Input radius rand circle centre (xc, yc), then set the coordinates for the first point on the circumference of a circle centred on the origin as:
2.Calculate the initial value of the decision parameter as:
3.Starting with k = 0at each position xk, perform the following test. If pk < 0, the next point along the circle centred on (0, 0)is (xk+1, yk)and:Otherwise the next point along the circle is (xk+1, yk-1)and:
4.Determine symmetry points in the other seven octants
5.Move each calculated pixel position (x, y)onto the circular path centred at (xc, yc)to plot the coordinate values:
6.Repeat steps 3 to 5 until x >= y

$X = x + x_c$ $\qquad$ $y = y + y_c$

Program:

```
#include<dos.h>
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void draw_circle(int,int,int);
Void symmetry(int,int,int,int);
void main()
{
int xc,yc,R;
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
printf("Enter the center of the circle:\n");
printf("Xc =");
scanf("%d",&xc);
printf("Yc =");
scanf("%d",&yc);
printf("Enter the radius of the circle:");
scanf("%d",&R);
draw_circle(xc,yc,R);
getch();
closegraph();
}
```

```
void draw_circle(int xc,int yc,int rad)
{
int x = 0;
int y = rad;
int p = 1-rad;
symmetry(x,y,xc,yc);
for(x=0;y>x;x++)
{
if(p<0)
p += 2*x + 3;
else {
p += 2*(x-y) + 5;
y--;
}
symmetry(x,y,xc,yc);
delay(50);
}
}
void symmetry(int x,int y,int xc,int yc)
{
putpixel(xc+x,yc-y,GREEN); //For pixel(x,y)
delay(50);
putpixel(xc+y,yc-x, GREEN); //For pixel(y,x)
delay(50);
putpixel(xc+y,yc+x, GREEN); //For pixel (y,-x)
delay(50);
putpixel(xc+x,yc+y, GREEN); //For pixel (x,-y)
delay(50);
putpixel(xc-x,yc+y, GREEN); //For pixel (-x,-y)
delay(50);
putpixel(xc-y,yc+x, GREEN); //For pixel (-y,-x)
delay(50);
putpixel(xc-y,yc-x, GREEN); //For pixel(-y,x)
delay(50);
putpixel(xc-x,yc-y, GREEN); //For pixel(-x,y)
delay(50);}
```

EXPERIMENT - 6

AIM : Write a program to draw a Ellipse using Mid-point algorithm.

Algorithm:

1.Input rx, ry, and ellipse center (x_c, y_c), and obtain the first point on an ellipse centered on the origin as(x0, y0) = (0, ry)

2.Calculate the initial parameter in region 1 as

3.At each xiposition, starting at i= 0, if $p1_i$< 0, the next point along the ellipse centered on (0, 0) is (xi + 1, yi) andOtherwise, the next point is (xi+ 1, yi–1) andand continue until

4.(x0, y0) is the last position calculated in region 1. Calculate the initial parameter in region 2 as

5.At each yiposition, starting at i= 0, if p2i> 0, the next point along the ellipse centered on (0, 0) is (xi, yi–1) andOtherwise, the next point is (xi+ 1, yi–1) and Use the same incremental calculations as in region 1. Continue until y= 0.

6.For both regions determine symmetry points in the other three quadrants.

7.Move each calculated pixel position (x,y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values

$$x= x+ x_c \text{ , } y= y+ y_c$$

Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void disp();
float x,y;
int xc,yc;
void main()
{
Int gd=DETECT,gm;
int rx,ry;
Float p1,p2;
clrscr();
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
```

```
printf("Enter the center point :");
scanf("%d%d",&xc,&yc);
printf("Enter the value for Rx and Ry:");
scanf("%d%d",&rx,&ry);
x=0;
y=ry;
disp();
p1=(ry*ry)-(rx*rx*ry)+(rx*rx)/4;
while((2.0*ry*ry*x)<=(2.0*rx*rx*y))
{
x++;
if(p1<=0)
p1=p1+(2.0*ry*ry*x)+(ry*ry);
else
{
y--;
p1=p1+(2.0*ry*ry*x)-(2.0*rx*rx*y)+(ry*ry);
}
disp();
x=-x;
disp();
}
x=rx;
y=0;
disp();
p2=(rx*rx)+2.0*(ry*ry*rx)+(ry*ry)/4;
while((2.0*ry*ry*x)>(2.0*rx*rx*y))
{
y++;
if(p2>0)
else
{p2=p2+(rx*rx)-(2.0*rx*rx*y);
 x--;
p2=p2+(2.0*ry*ry*x)-(2.0*rx*rx*y)+(rx*rx);
}
disp();
y=-y;
disp()
};
y=-y;
getch();
closegraph();
}
void disp()
{
delay(50);
putpixel(xc+x,yc+y,10);
putpixel(xc-x,yc+y,10);
putpixel(xc+x,yc-y,10);
putpixel(xc-x,yc-y,10);
}
```

EXPERIMENT - 7

AIM : Write a menu driven program to rotate , scale and translate a line ,point , square, triangle(any one object) about the origin.

ALGORITHM:

```
1.  Start
2.  Initialize the graphics mode.
3.  Construct a 2D object  (use Drawpoly()) e.g. (x,y)
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')
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')
6.  C) Rotation
        a.  Get the Rotation angle
        b.  Rotate the object by the angle ф
                x'=x cos ф -  y sin ф
                    y'=x sin ф  - y cosф
        c.  Plot (x',y')
```

PROGRAM:

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include<math.h>
void main()
{
        int gm;
        int 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,"c:\tc\bg:");
        printf("\t Program for basic transactions");
        printf("\n\t Enter the points of triangle");
        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("\n 1.Transaction\n 2.Rotation\n 3.Scalling\n 4.exit");
        printf("Enter your choice:");
        scanf("%d",&c);
        switch(c)
        {
                case 1:
                        printf("\n Enter the translation factor");
                        scanf("%d%d",&xt,&yt);
                        nx1=x1+xt;
                        ny1=y1+yt;
                        nx2=x2+xt;
                        ny2=y2+yt;
                        nx3=x3+xt;
                        ny3=y3+yt;
                        line(nx1,ny1,nx2,ny2);
                        line(nx2,ny2,nx3,ny3);
                        line(nx3,ny3,nx1,ny1);
                        Getch();

                case 2:
                        printf("\n Enter the angle of rotation");
```

```
            scanf("%d",&r);
            t=3.14*r/180;
            nx1=abs(x1*cos(t)-y1*sin(t));
            ny1=abs(x1*sin(t)+y1*cos(t));
            nx2=abs(x2*cos(t)-y2*sin(t));
            ny2=abs(x2*sin(t)+y2*cos(t));
            nx3=abs(x3*cos(t)-y3*sin(t));
            ny3=abs(x3*sin(t)+y3*cos(t));
            line(nx1,ny1,nx2,ny2);
            line(nx2,ny2,nx3,ny3);
            line(nx3,ny3,nx1,ny1);
            getch();

case 3:
            printf("\n Enter the scalling factor");
            scanf("%d%d",&sx,&sy);
            nx1=x1*sx;
            ny1=y2*sy;
            nx2=x2*sx;
            ny2=y2*sy;
            nx3=x3*sx;
            ny3=y3*sy;
            line(nx1,ny1,nx2,ny2);
            line(nx2,ny2,nx3,ny3);
            line(nx3,ny3,nx1,ny1);
            getch();
case 4:
            break;
default:
            printf("Enter the correct choice");
            }
            closegraph();}
```

EXPERIMENT - 8

AIM : Write a program to implement reflection of a point, line.

ALGORITHM:

Reflection can be done just by rotating the object about given axis of reflection with an angle of 180 degrees.

Program for reflection:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
void refx(int x1,int x2,int x3,int y1,int y2,int y3){
line(320,0,320,430);
line(0,240,640,240);
x1=(320-x1)+320;
x2=(320-x2)+320;
x3=(320-x3)+320;
line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x3,y3,x1,y1);
}

void refy(int x1,int x2,int x3,int y1,int y2,int y3){
line(320,0,320,430);
line(0,240,640,240);
y1=(240-y1)+240;
y2=(240-y2)+240;
y3=(240-y3)+240;
line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x3,y3,x1,y1);
}
void main()
{
int gd=DETECT,gm;
int x1,y1,x2,y2,x3,y3;
clrscr();
initgraph(&gd,&gm,"c://turboc3//bgi");
line(320,0,320,430);
line(0,240,640,240);
x1=150;y1=100;
x2=220;y2=220;
x3=220;y3=110;
line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x3,y3,x1,y1);
getch();
refx(x1,x2,x3,y1,y2,y3);
getch();
refy(x1,x2,x3,y1,y2,y3);
getch();
closegraph();
}
```

EXPERIMENT - 9

AIM : Write a program to perform shearing on a line.

ALGORITHM:

Shear transformation is of 2 types:

a. X-shear: changing x-coordinate value and keeping y constant

    x’=x+shx*y
    y’=y

b.Y-shear: changing y coordinates value and keeping x constant

    x’=x
    y’=y+shy*x
    shx and shy are shear factors along x and y-axis.

Program:

```
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
void main( )
{
/* request auto detection */
int gdriver = DETECT, gmode;
int x1,y1,x2,y2,a,b;
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
printf("Enter the value of line coordinates:");
scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
printf("Enter the value of x-shearing factor:");
scanf("%d",&a);
printf("Enter the value of y-shearing factor:");
scanf("%d",&b);
cleardevice( );
outtextxy(200,20,"LINE BEFORE SHEARING");
line(x1,y1,x2,y2);
x1 = x1+a*y1;
y1 = b*x1+y1;
x2 = x2+a*y2;
y2 = b*x2+y2;
getch( );
cleardevice( );
outtextxy(200,20,"LINE AFTER SHEARING");
line(x1,y1,x2,y2);
getch( );
closegraph( );
restorecrtmode( );
}
```

EXPERIMENT - 10

AIM : Write a program to perform line clipping.

Algorithm:

```
1. Read 2 end points of line as p1(x1,y1) and p2(x2,y2)
2. Read 2 corner points of the clipping window (left-top and
 right-bottom) as (wx1,wy1) and (wx2,wy2)
3. Assign the region codes for 2 endpoints p1 and p2 using following steps:-
        initialize code with 0000
        Set bit 1 if x<wx1
        Set bit 2 if x>wx2
        Set bit 3 if y<wy2
        Set bit 4 if y>wy1
4. Check for visibility of line
        If region codes for both endpoints are zero then line is
    completely visible. Draw the line go to step 9.
        If region codes for endpoints are not zero and logical ANDing
     of them is also nonzero then line is invisible. Discard the line and move to step 9.
        If it does not satisfy 4.a and 4.b then line is partially visible.
5. Determine the intersecting edge of clipping window as follows:-
        If region codes for both endpoints are nonzero find intersection
        ts p1' and p2' with boundary edges.
        If region codes for any one end point is non zero then find
        intersection point p1' or p2'.
6. Divide the line segments considering intersection points.
7. Reject line segment if any end point of line appears outside of
        boundary.
8. Draw the clipped line segment.
9. Stop.
```

Program:

```
#include <iostream.h>
```

```
#include <conio.h>

#include <graphics.h>

#include <dos.h>

class data

{

   int gd, gmode, x, y, xmin,ymin,ymax,xmax;

   int a1,a2;

   float x1, y1,x2,y2,x3,y3;

   int xs, ys, xe, ye;

   float maxx,maxy;

   public:

      void getdata ();

      void find ();

      void clip ();

      void display (float, float,float,float);

      void checkonof (int);

      void showbit (int);

};

void data :: getdata ()

{

   cout<<"Enter the minimum and maximum coordinate of window (x, y) ";

        cin >>xmin>>ymin>>xmax>>ymax;

        cout<<"Enter the end points of the line to be clipped";

        cin >>xs>>ys>>xe>>ye;
```

```
            display (xs, ys, xe,ye);
}
void data :: display (float, xs, float, ys,float xe, float ye)
{
    int gd=DETECT;
    initgraph (&gd,&gmode, "");
    maxx=getmaxx();
    maxy=getmaxy();
    line (maxx/2,0,maxx/2,maxy);
    line (0, maxy/2,maxx,maxy/2);
    rectangle (maxx/2+xmin,maxy/2-ymax,maxx/2+xmax,maxy/2-ymin);
    line (maxx/2+xs,maxy/2-ys,maxx/2+xe,maxy/2-ye);
    getch();
}
void data :: find ()
{     a1=0;
    a2=0;
    if ((ys-ymax)>0)
               a1+=8;
    if ((ymin-ys)>0)
        a1+=4;
    if ((xs-xmax)>0)
         a1+=2;
            if ((xmin-xs)>0)
         a1+=1;
     if ((ye-ymax)>0)
        a2+=8;
            if ((ymin-ye)>0)
                a2+=4;
         if ((xe-xmax)>0)
               a2+=2;
         if ((xmin-xe)>0)
                a2+=1;
         cout<<"\nThe area code of Ist point is ";
                 showbit (a1);
         getch ();
         cout <<"\nThe area code of 2nd point is ";
         showbit (a2);
         getch ();
}
void data :: showbit (int n)
{
        int i,k, and;
        for (i=3;i>=0;i--)
        {
             and =1<<i;
       k = n?
       k ==0?cout<<"0": cout<<"1\"";
```

```
          }
    }
    void data ::clip()
    {
          int j=a1&a2;
          if (j==0)
          {
               cout<<"\nLine is perfect candidate for clipping";
               if (a1==0)
        {
                     else
              {
                    checkonof(a1);
                    x2=x1;y2=y1;
              }
              if (a2=0)
             {
                  x3=xe; y3=ye;
             }
            else
            {
                    checkonof (a2);
                    x3=x1; y3=y1;
             }
             xs=x2; ys=y2;xe=x3;ye=y3;

             cout << endl;

             display (xs,ys,xe,ye);

             cout<<"Line after clipping";

             getch ()

          }

        else if ((a1==0) && (a2=0))

        {

               cout <<"\n Line is in the visible region";

               getch ();

        }

    }

    void data :: checkonof (int i)

    {

          int j, k,l,m;

          1=i&1;

          x1=0;y1=0;

        if (1==1)

          {

              x1=xmin;

              y1=ys+ ((x1-xs)/ (xe-xs))*(ye-ys);

          }

          j=i&8;

       if (j>0)
```

```
{
          y1=ymax;
   x1=xs+(y1-ys)/(ye-ys))*(xe-xs);
 }
 k=i & 4;
 if (k==1)
 {
        y1=ymin;
        x1=xs+((y1-ys)/(ye-ys))*(xe-xs);
 }
 m= i&2;
  if (m==1)
  {
         x1=xmax;
         y1=ys+ ((x1-xs)/ (xe-xs))*(ye-ys);
   }
   main ()
   {
          data s;
          clrscr();
          s.getdata();
          s.find();
          getch();
          closegraph ();
  return ();  }
```

EXPERIMENT - 11

AIM : Write a program to implement polygon filling(any one).

Algorithm:

1. The horizontal scanning of the polygon from its lowermost to its topmost vertex
2. identify the edge intersections of scan line with polygon
3. Build the edge table
    a. Each entry in the table for a particular scan line contains the maximum y value for that edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge.
4. Determine whether any edges need to be splitted or not. If there is need to split, split the edges.
5. Add new edges and build modified edge table.
6. Build Active edge table for each scan line and fill the polygon based on intersection of scanline with polygon edges.

Program:

```
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>
#define maxHt 800
#define maxWd 600
#define maxVer 10000
FILE *fp;

// Start from lower left corner
typedef struct edgebucket
{
   int ymax; //max y-coordinate of edge
   float xofymin; //x-coordinate of lowest edge point updated only in aet
```

```
    float slopeinverse;
}EdgeBucket;
typedef struct edgetabletup
{
    // the array will give the scanline number
    // The edge table (ET) with edges entries sorted
    // in increasing y and x of the lower end
    int countEdgeBucket; //no. of edgebuckets
    EdgeBucket buckets[maxVer];
}EdgeTableTuple;
EdgeTableTuple EdgeTable[maxHt], ActiveEdgeTuple;

// Scanline Function
void initEdgeTable()
{
    int i;
    for (i=0; i<maxHt; i++)
    {
            EdgeTable[i].countEdgeBucket = 0;
    }
    ActiveEdgeTuple.countEdgeBucket = 0;
}
void printTuple(EdgeTableTuple *tup)
{
    int j;
    if (tup->countEdgeBucket)
            printf("\nCount %d-----\n",tup->countEdgeBucket);

            for (j=0; j<tup->countEdgeBucket; j++)
            {
                    printf(" %d+%.2f+%.2f",
                    tup->buckets[j].ymax, tup->buckets[j].xofymin,tup->buckets[j].slopeinverse);
            }
}
```

```
void printTable()
{
   int i,j;
   for (i=0; i<maxHt; i++)
   {
          if (EdgeTable[i].countEdgeBucket)
                   printf("\nScanline %d", i);
          printTuple(&EdgeTable[i]);
   }
}
/* Function to sort an array using insertion sort*/
void insertionSort(EdgeTableTuple *ett)
{
   int i,j;
   EdgeBucket temp;
   for (i = 1; i < ett->countEdgeBucket; i++)
   {
          temp.ymax = ett->buckets[i].ymax;
          temp.xofymin = ett->buckets[i].xofymin;
          temp.slopeinverse = ett->buckets[i].slopeinverse;
          j = i - 1;
   while ((temp.xofymin < ett->buckets[j].xofymin) && (j >= 0))
   {
          ett->buckets[j + 1].ymax = ett->buckets[j].ymax;
          ett->buckets[j + 1].xofymin = ett->buckets[j].xofymin;
          ett->buckets[j + 1].slopeinverse = ett->buckets[j].slopeinverse;
          j = j - 1;
   }
   ett->buckets[j + 1].ymax = temp.ymax;
   ett->buckets[j + 1].xofymin = temp.xofymin;
   ett->buckets[j + 1].slopeinverse = temp.slopeinverse;
   }
}
```

```
void storeEdgeInTuple (EdgeTableTuple *receiver,int ym,int xm,float slopInv)
{
    // both used for edgetable and active edge table..
    // The edge tuple sorted in increasing ymax and x of the lower end.
    (receiver->buckets[(receiver)->countEdgeBucket]).ymax = ym;
    (receiver->buckets[(receiver)->countEdgeBucket]).xofymin = (float)xm;
    (receiver->buckets[(receiver)->countEdgeBucket]).slopeinverse = slopInv;
    // sort the buckets
    insertionSort(receiver);
    (receiver->countEdgeBucket)++;
}
void storeEdgeInTable (int x1,int y1, int x2, int y2)
{
    float m,minv;
    int ymaxTS,xwithyminTS, scanline; //ts stands for to store
    if (x2==x1)
    {
            minv=0.000000;
    }
    else
    {
    m = ((float)(y2-y1))/((float)(x2-x1));
    // horizontal lines are not stored in edge table
    if (y2==y1)
            return;
    minv = (float)1.0/m;
    printf("\nSlope string for %d %d & %d %d: %f",x1,y1,x2,y2,minv);
    }
    if (y1>y2)
    {
            scanline=y2;
            ymaxTS=y1;
            xwithyminTS=x2;
    }
    else
    {
            scanline=y1;
            ymaxTS=y2;
            xwithyminTS=x1;
    }
    // the assignment part is done..now storage..
    storeEdgeInTuple(&EdgeTable[scanline],ymaxTS,xwithyminTS,minv);
}
void removeEdgeByYmax(EdgeTableTuple *Tup,int yy)
{
    int i,j;
    for (i=0; i< Tup->countEdgeBucket; i++)
    {
            if (Tup->buckets[i].ymax == yy)
            {
                        printf("\nRemoved at %d",yy);

                        for ( j = i ; j < Tup->countEdgeBucket -1 ; j++ )
                                {
                                Tup->buckets[j].ymax =Tup->buckets[j+1].ymax;
                                Tup->buckets[j].xofymin =Tup->buckets[j+1].xofymin;
                                Tup->buckets[j].slopeinverse = Tup->buckets[j+1].slopeinverse;
                                }
                                Tup->countEdgeBucket--;
                        i--;
            }
    }
}
void updatexbyslopeinv(EdgeTableTuple *Tup)
{
    int i;
    for (i=0; i<Tup->countEdgeBucket; i++)
    {
            (Tup->buckets[i]).xofymin =(Tup->buckets[i]).xofymin + (Tup->buckets[i]).slopeinverse;
    }
}
void ScanlineFill()
{
    /* Follow the following rules:
    1. Horizontal edges: Do not include in edge table
    2. Horizontal edges: Drawn either on the bottom or on the top.
    3. Vertices: If local max or min, then count twice, else count
            once.
    4. Either vertices at local minima or at local maxima are drawn.*/
    int i, j, x1, ymax1, x2, ymax2, FillFlag = 0, coordCount;
    // we will start from scanline 0;
    // Repeat until last scanline:
    for (i=0; i<maxHt; i++)//4. Increment y by 1 (next scan line)
    {
            // 1. Move from ET bucket y to the
            // AET those edges whose ymin = y (entering edges)
            for (j=0; j<EdgeTable[i].countEdgeBucket; j++)
            {
                    storeEdgeInTuple(&ActiveEdgeTuple,EdgeTable[i].buckets[j].
                                    ymax,EdgeTable[i].buckets[j].xofymin,
                                    EdgeTable[i].buckets[j].slopeinverse);
            }
            printTuple(&ActiveEdgeTuple);

            // 2. Remove from AET those edges for
            // which y=ymax (not involved in next scan line)
            removeEdgeByYmax(&ActiveEdgeTuple, i);

            //sort AET (remember: ET is presorted)
            insertionSort(&ActiveEdgeTuple);

            printTuple(&ActiveEdgeTuple);
```

```
//3. Fill lines on scan line y by using pairs of x-coords from AET
j = 0;
FillFlag = 0;
coordCount = 0;
x1 = 0;
x2 = 0;
ymax1 = 0;
ymax2 = 0;
while (j<ActiveEdgeTuple.countEdgeBucket)
{
        if (coordCount%2==0)
        {
                x1 = (int)(ActiveEdgeTuple.buckets[j].xofymin);
                ymax1 = ActiveEdgeTuple.buckets[j].ymax;
                if (x1==x2)
                {
                /* three cases can arrive-
                        1. lines are towards top of the intersection
                        2. lines are towards bottom
                        3. one line is towards top and other is towards bottom
                */
                        if (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)))
                        {
                                x2 = x1;
                                ymax2 = ymax1;
                        }

                        else
                        {
                                coordCount++;
                        }
                }
                else
                {
                                coordCount++;
                }
        }
        else
        {
                x2 = (int)ActiveEdgeTuple.buckets[j].xofymin;
                ymax2 = ActiveEdgeTuple.buckets[j].ymax;

                FillFlag = 0;

                // checking for intersection...
                if (x1==x2)
                {
                /*three cases can arive-
                        1. lines are towards top of the intersection
                        2. lines are towards bottom
                        3. one line is towards top and other is towards bottom
                */
                        if (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)))
                        {
                                x1 = x2;
                                ymax1 = ymax2;
                        }
                        else
                        {
                                coordCount++;
                                FillFlag = 1;
                        }
                }
                else
                {
                                coordCount++;
```

```
                                                FillFlag = 1;
                                }

                        if(FillFlag)
                        {
                                //drawing actual lines...
                                glColor3f(0.0f,0.7f,0.0f);

                                glBegin(GL_LINES);
                                glVertex2i(x1,i);
                                glVertex2i(x2,i);
                                glEnd();
                                glFlush();

                                // printf("\nLine drawn from %d,%d to %d,%d",x1,i,x2,i);
                        }
                }
                j++;
        }

        // 5. For each nonvertical edge remaining in AET, update x for new y
        updatexbyslopeinv(&ActiveEdgeTuple);
}
printf("\nScanline filling complete");
}
void myInit(void)
{
        glClearColor(1.0,1.0,1.0,0.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0,maxHt,0,maxWd);
        glClear(GL_COLOR_BUFFER_BIT);
}
```

```
void drawPolyDino()
{
   glColor3f(1.0f,0.0f,0.0f);
   int count = 0,x1,y1,x2,y2;
   rewind(fp);
   while(!feof(fp) )
   {
         count++;
         if (count>2)
         {
                  x1 = x2;
                  y1 = y2;
                  count=2;
         }
         if (count==1)
         {
                  fscanf(fp, "%d,%d", &x1, &y1);
         }
         else
         {
                  fscanf(fp, "%d,%d", &x2, &y2);
                  printf("\n%d,%d", x2, y2);
                  glBegin(GL_LINES);
                           glVertex2i( x1, y1);
                           glVertex2i( x2, y2);
                  glEnd();
                  storeEdgeInTable(x1, y1, x2, y2);//storage of edges in edge table.
                  glFlush();
         }
   }
}
void drawDino(void)
{
   initEdgeTable();
```

```
    drawPolyDino();
    printf("\nTable");
    printTable();
    ScanlineFill();//actual calling of scanline filling..
}
void main(int argc, char** argv)
{
    fp=fopen ("PolyDino.txt","r");
    if ( fp == NULL )
    {
            printf( "Could not open file" ) ;
            return;
    }
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(maxHt,maxWd);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("Scanline filled dinosaur");
    myInit();
    glutDisplayFunc(drawDino);
    glutMainLoop();
    fclose(fp);
}
```

EXPERIMENT - 12

AIM : Write a program to implement transformations in three dimensions(any one).

Program:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<process.h>
#include<graphics.h>
Int x1,x2,y1,y2,mx,my,depth;
void draw();
void trans();
void main()
{
int gd=DETECT,gm,c;
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
printf("\n\t\t3D Translation\n\n");
printf("\nEnter 1st top value(x1,y1):");
scanf("%d%d",&x1,&y1);
printf("Enter right bottom value(x2,y2):");
scanf("%d%d",&x2,&y2);
depth=(x2-x1)/4;
mx=(x1+x2)/2;
my=(y1+y2)/2;
draw();
getch();

cleardevice();
trans();
getch();
}
void draw()
{
bar3d(x1,y1,x2,y2,depth,1);
}
void trans()
{
int a1,a2,b1,b2,dep,x,y;
printf("\n Enter the Translation Distances:");
scanf("%d%d",&x,&y);
a1=x1+x;
a2=x2+x;
b1=y1+y;
b2=y2+y;
dep=(a2-a1)/4;
bar3d(a1,b1,a2,b2,dep,1);
setcolor(5);
draw();
}
```

Program for scaling:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<process.h>
#include<graphics.h>
int
x1,x2,y1,y2,mx,my,depth;
void draw();
void scale();
void main()
{
```

```
int gd=DETECT,gm,c;
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
printf("\n\t\t3D Scaling\n\n");
printf("\nEnter 1st top value(x1,y1):");
scanf("%d%d",&x1,&y1);
printf("Enter right bottom value(x2,y2):");
scanf("%d%d",&x2,&y2);
depth=(x2-x1)/4;
mx=(x1+x2)/2;
my=(y1+y2)/2;
draw();
getch();
cleardevice();
scale();
getch();
}
void draw()
{
bar3d(x1,y1,x2,y2,depth,1);
}
void scale()
{
int x,y,a1,a2,b1,b2,dep;
printf("\n\n Enter scaling Factors:");
scanf("%d%d",&x,&y);
a1=mx+(x1-mx)*x;
a2=mx+(x2-mx)*x;
b1=my+(y1-my)*y;
b2=my+(y2-my)*y;
dep=(a2-a1)/4;
bar3d(a1,b1,a2,b2,dep,1);
setcolor(5);
draw();
}
```

Program for Rotation:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<graphics.h>
int x1,x2,y1,y2,mx,my,depth;
void draw();
void rotate();
void main()
{
int gd=DETECT,gm,c;
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
printf("\n3D Transformation Rotating\n\n");
printf("\nEnter 1st top value(x1,y1):");
scanf("%d%d",&x1,&y1);
printf("Enter right bottom value(x2,y2):");
scanf("%d%d",&x2,&y2);
depth=(x2- x1)/4;
mx=(x1+x2)/2;
my=(y1+y2)/2;
draw();
getch();
cleardevice();
rotate();
getch();
}
void draw()
{
bar3d(x1,y1,x2,y2,depth,1);
}
void rotate()
{
```

```
float t;
int a1,b1,a2,b2,dep;
printf("Enter the angle to rotate=");
scanf("%f",&t);
t=t*(3.14/180);
a1=mx+(x1-mx)*cos(t)-(y1-my)*sin(t);
a2=mx+(x2-mx)*cos(t)-(y2-my)*sin(t);
b1=my+(x1-mx)*sin(t)-(y1-my)*cos(t);
b2=my+(x2-mx)*sin(t)-(y2-my)*cos(t);
if(a2>a1)
dep=(a2-a1)/4;
else
dep=(a1-a2)/4;
bar3d(a1,b1,a2,b2,dep,1);
setcolor(5);
}
```