Installation

- **sudo apt-get update**
- **sudo apt-get install mesa-common-dev**
- **sudo apt-get install freeglut3 freeglut3-dev**

Compile

gcc -o filename1 filename2.c -lGLU -lGL -lglut

Run

./filename1

Experiment no:1

DDA ALGORITHM

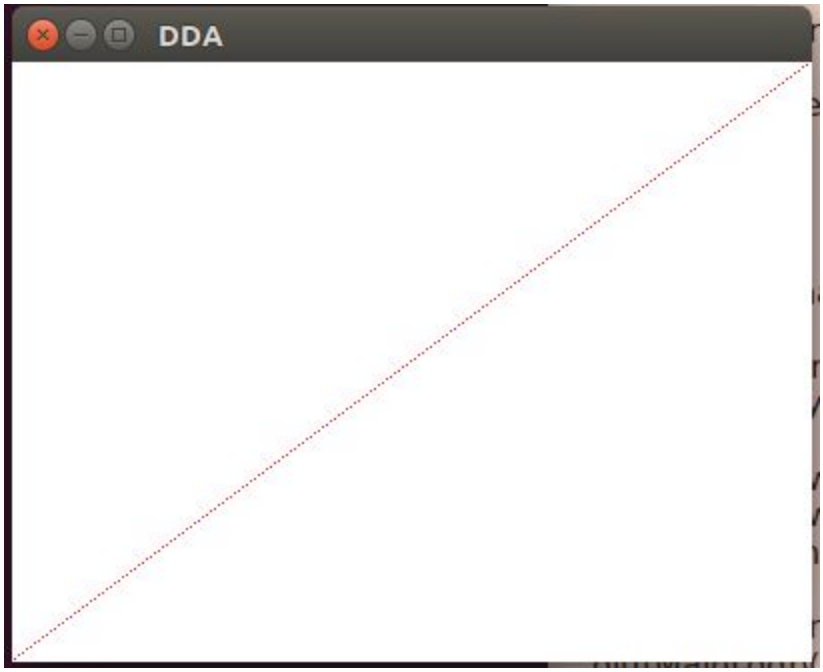
Aim: To draw a line using DDA line drawing algorithm.

Program

```
#include<GL/glut.h>
void init()
{
   glClearColor(1.0,1.0,1.0,0.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0.0,200.0,0.0,150.0);
}
void dda()
{
   float steps,Xa=0.0,Ya=0.0,Xb=400.0,Yb=300.0;
   float Dx=Xb-Xa;
   float Dy=Yb-Ya;
   float X=Xa,Y=Ya;
   float Xinc,Yinc;
   int k;
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0,0.0,0.0);
   glBegin(GL_POINTS);
   if(abs(Dx)>abs(Dy))
   {
         steps=abs(Dx);
   }
   else
   {
         steps=abs(Dy);
```

```
    }
    Xinc=Dx/steps;
    Yinc=Dy/steps;
    glVertex2f(X,Y);
    for(k=0;k<steps;k++)
    {
            X+=Xinc;
            Y+=Yinc;
            k++;
            glVertex2f(X,Y);
    }
    glEnd();
    glFlush();
}
void main(int argc,char **argv)
{
                    glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowPosition(50,50);
    glutInitWindowSize(400,300);
    glutCreateWindow("DDA");
    init();
    glutDisplayFunc(dda);
    glutMainLoop();
}
```

OUTPUT



Experiment No:2

Integer Bresenham Algorithm

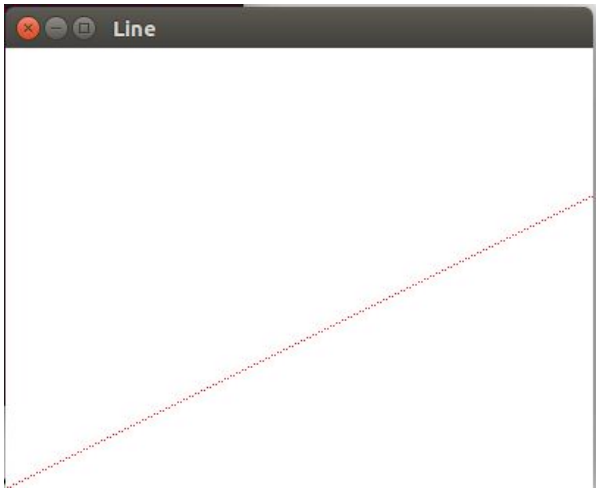
Aim: To draw a line using Integer Bresenham Algorithm

Program

```
#include<GL/glut.h>
void init()
{
        glClearColor(1.0,1.0,1.0,0.0);
        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(0.0,200.0,0.0,150.0);
}
void brescen()
{
```

```
	float x1=0.0,y1=0.0,x2=400.0,y2=200.0;
	float x=x1,y=y1,e,d1,d2;
	float i;
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1.0,0.0,0.0);
	glBegin(GL_POINTS);
	d1=x2-x1;
	d2=y2-y1;
	e=2*y-d1;
	glVertex2f(x,y);
	for(i=0.0;i<d1;i=i+0.1)
	{
		glVertex2f(x,y);
		while(e>0)
		{
			y=y+1;
			e=e-2*d1;
		}
		x=x+1;
		e=e+2*d2;
	}



	glEnd();
	glFlush();
}


void main(int argc,char **argv)
{
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(400,300);
	glutCreateWindow("Line");
	init();
	glutDisplayFunc(brescen);
	glutMainLoop();
}
```

OUTPUT



Experiment No:3

General Bresenham Algorithm

Aim: To draw a line using General Bresenham Algorithm

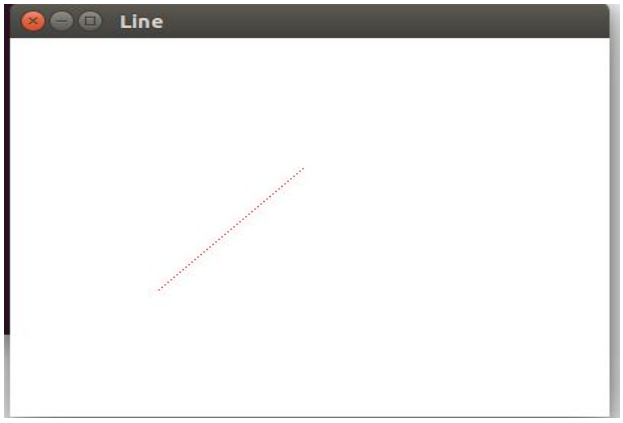
Program

```
#include<stdio.h>
#include<GL/glut.h>

/* GLOBAL DECLARATION */
float x1;
float Y1;
```

```
float x2;
float y2;

/* READ DATA */
void data()
{
	printf("Enter the first coordinate: ");
	scanf("%f %f",&x1,&Y1);
	printf("Enter the second coordinate: ");
	scanf("%f %f",&x2,&y2);
}


/* FIND SIGN */
int sign(float a,float b)
{
	if((b-a)>0)
		return (1);
	else if((b-a)<0)
		return (-1);
	else
		return (0);
}


void init()
{
	glClearColor(1.0,1.0,1.0,0.0);
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(0.0,200.0,0.0,150.0);
}


/* GENERAL BRESCENHAMS */
void general()
{
	float x,y,dx,dy,e=0.0;
	int s1,s2,i,temp;
	int flag=0;
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1.0,0.0,0.0);
	glBegin(GL_POINTS);
	x=x1;
	y=Y1;
	dx=abs(x2-x1);
	dy=abs(y2-Y1);
	s1=sign(x1,x2);
	s2=sign(Y1,y2);

	if(dy>dx)
	{
		temp=dx;
		dx=dy;
		dy=temp;
```

```
		flag=1;
	}
	else
		flag=0;

	e=2*dy-dx;

	for(i=1;i<dx;i++)
	{
		glVertex2f(x,y);
		while(e>0)
		{
			if(flag==1)
			{
				x=x+s1;
			}
			else
				y=y+s2;
			e=e-2*dx;
		}

		if(flag==1)
			y=y+s2;
		else
			x=x+s1;
		e=e+2*dy;
	}

	glEnd();
	glFlush();
}

/* MAIN FUNCTION */
void main(int argc,char **argv)
{
	data();
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutInitWindowPosition(50,50);
	glutInitWindowSize(400,300);
	glutCreateWindow("Line");
	init();
	glutDisplayFunc(general);
	glutMainLoop();
}
```

OUTPUT



Experiment No:4

Mid Point Circle Algorithm

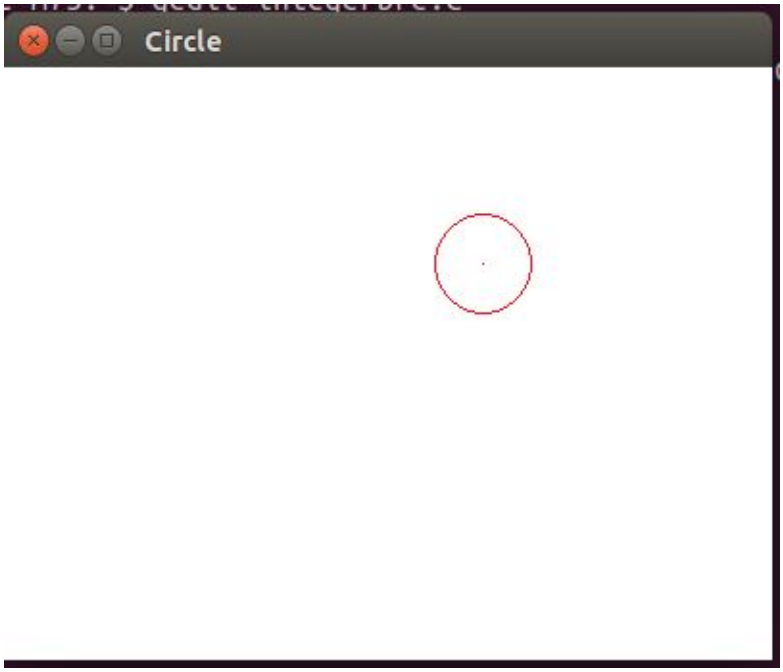
Aim: To draw a circle using Mid Point Circle Algorithm

Program

```
#include<stdio.h>
#include<GL/glut.h>

float r,x,y;

void data()
{
	printf("Enter the radius(r), center of circle(x,y): ");
	scanf("%f %f %f",&r,&x,&y);
}

void init()
{
	glClearColor(1.0,1.0,1.0,0.0);
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(-200.0,200.0,-150.0,150.0);
}

void circle()
{
	float x0,y0,p0,pk,xk,yk,x1,y1;

	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1.0,0.0,0.0);
	glBegin(GL_POINTS);
	glVertex2f(x,y);
	//1st positon on the circumferance
	x0=0;
	y0=r;
	//decision parameter
	p0=1-r;
	pk=p0;
	xk=x0;
	yk=y0;
	while(xk<=yk)
	{
		if(pk<0)
		{
			xk+=1;
			pk=pk+2*xk+2;
		}
		else
		{
			xk+=1;
			yk-=1;
			pk=pk+2*xk+3-2*yk-1;
		}
		x1=x+xk;y1=y+yk;
		glVertex2f(x1,y1);
		glVertex2f(y1,x1);
		glVertex2f(x-xk,y-yk);
		glVertex2f(y-yk,x-xk);
```

```
                glVertex2f(x-xk,y1);
                glVertex2f(y1,x-xk);
                glVertex2f(x1,y-yk);
                glVertex2f(y-yk,x1);
                glVertex2f(x,y+r);
                glVertex2f(x,y-r);
                glVertex2f(x+r,y);
                glVertex2f(x-r,y);

        }
        glEnd();
        glFlush();
}

void main(int argc,char **argv)
{
        data();
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowPosition(50,50);
        glutInitWindowSize(400,300);
        glutCreateWindow("Circle");
        init();
        glutDisplayFunc(circle);
        glutMainLoop();
}
```

OUTPUT



Experiment No:5

Ellipse Drawing Algorithm

Aim: To draw an ellipse using Ellipse Drawing Algorithm

Program

```
#include<GL/glut.h>
#include<stdio.h>
int rx,ry,xc,yc,x,y;
void Input()
{
	printf("\n enter the centre of the ellipse:");
	scanf("%d%d",&xc,&yc);
	printf("\n enter the radius of the ellipse:");
	scanf("%d%d",&rx,&ry);
}
void Init()
{
	glClearColor(1.0,1.0,1.0,0.0);
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(0 , 640 , 0 , 480);
}
void Ellipse()
{
	int p1k,p2k;
	glClear(GL_COLOR_BUFFER_BIT);
	x=0;
	y=ry;
	int rysqr=(ry*ry);
	int rxsqr=(rx*rx);
	p1k=((rysqr)-(rxsqr*ry)+((1/4)*rxsqr));
	glColor3f(1.0,0.0,0.0);
	glBegin(GL_POINTS);
	do
	{
		if(p1k<0)
		{
			x=x+1;
			p1k=(p1k+(2*rysqr*x)+rysqr);
		}
		else
		{
			y--; x=x+1;
			p1k=(p1k+(2*rysqr*x)+rysqr-(2*rxsqr*y));
		}
		glVertex2f(xc+x,yc+y);
glVertex2f(xc+x,yc-y);
glVertex2f(xc-x,yc-y);
glVertex2f(xc-x,yc+y);
	} while((2*rysqr*x)<(2*rxsqr*y));
	glEnd();
	glFlush();
```

```
        p2k=(rysqr*((x+(1/2))*(x+(1/2)))+(rxsqr*((y-1)*(y-1)))-(rxsqr*rysqr));
        do
        {
                if(p2k>0)
                {
                        y=y-1;
                        p2k=(p2k-(2*rxsqr*y)+rxsqr);

                }
                else
                {
                        x=x+1;
                        y=y-1;
                        p2k=(p2k+(2*rysqr*x)-(2*rxsqr*y)+rxsqr);
                }
                glBegin(GL_POINTS);
                glVertex2f(xc+x,yc+y);
                glVertex2f(xc+x,yc-y);
                glVertex2f(xc-x,yc-y);
                glVertex2f(xc-x,yc+y);
        }while(y>=0);
         glEnd();
         glFlush();

}
void main(int argc,char **argv)
{
        Input();
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE||GLUT_RGB);
        glutInitWindowPosition(0,0);
        glutInitWindowSize(640,480);
        glutCreateWindow("ELLIPSE");
        Init();
        Ellipse();
        glutMainLoop();
}
```

OUTPUT

Experiment No:6

Olympic Ring

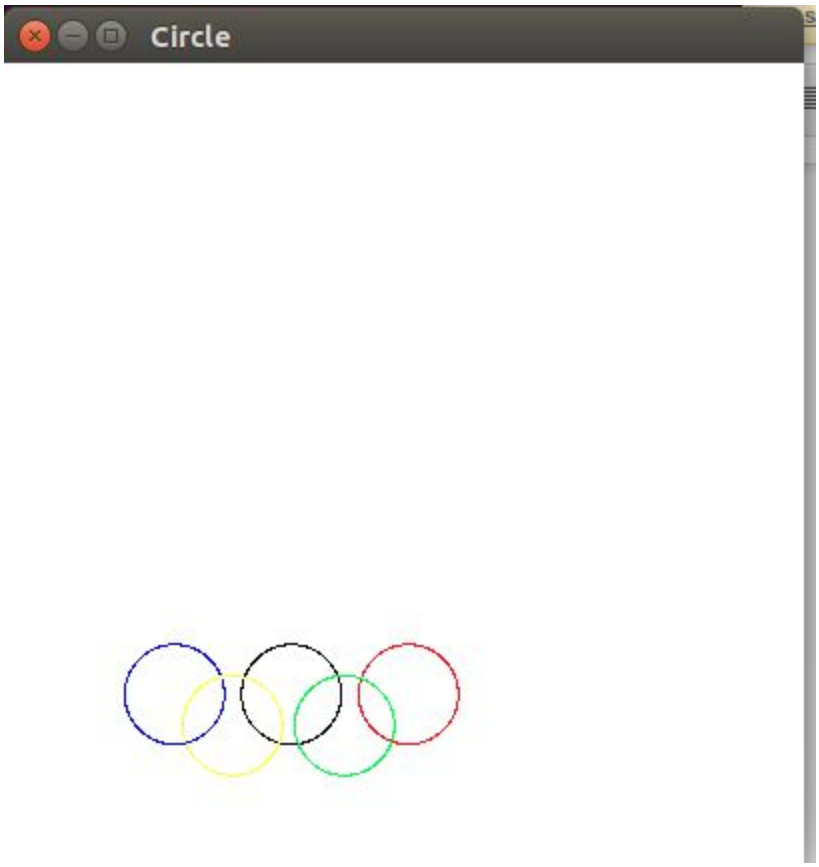
Aim: To draw an olympic ring using mid point circle algorithm.

Program

```
#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
float x,y,r;
void circle();
void data()
{
	printf("Enter the radius(r), center of circle(x,y): ");
	scanf("%f %f %f",&r,&x,&y);
}
void init()
{
	glClearColor(1.0,1.0,1.0,0.0);
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(0.0,800.0,0.0,800.0);
}
void circle()
{
	float x0,y0,p0,pk,xk,yk,x1,y1;
	int i,count=0,temp;
	glClear(GL_COLOR_BUFFER_BIT);
	glBegin(GL_POINTS);
	temp=x;
	for(i=0;i<5;i++)
	{
		if(count==0)
			glColor3f(0.0,0.0,1.0);
		else if(count==1)
			glColor3f(0.0,0.0,0.0);
		else if(count==2)
			glColor3f(1.0,0.0,0.0);
		else if(count==3)
			glColor3f(1.0,1.0,0.0);
		else if(count==4)
			glColor3f(0.0,1.0,0.0);
		x0=0;
		y0=r;
		//decision parameter
		p0=1-r;
		pk=p0;
		xk=x0;
		yk=y0;
		while(xk<=yk)
		{
			if(pk<0)
```

```
                        {
                                xk+=1;
                                pk=pk+2*xk+2;
                        }
                        else
                        {
                                xk+=1;
                                yk-=1;
                                pk=pk+2*xk+3-2*yk-1;
                        }

                        x1=x+xk;y1=y+yk;
                        glVertex2f(x1,y1);
                        glVertex2f(x+yk,y+xk);
                        glVertex2f(x-xk,y+yk);
                        glVertex2f(x-yk,y+xk);
                        glVertex2f(x+xk,y-yk);
                        glVertex2f(x+yk,y-xk);
                        glVertex2f(x-xk,y-yk);
                        glVertex2f(x-yk,y-xk);
                        glVertex2f(x,y+r);
                        glVertex2f(x,y-r);
                        glVertex2f(x+r,y);
                        glVertex2f(x-r,y);


                }
                if(i<2)
                        x=x+2*r+(r/3);
                else if(i==2)
                {
                        x=temp+(2*r+(r/3))/2;
                        y=y-r+(((2*r+(r/3))/2)*tan(60));
                }
                else if(i==3)
                        x=x+2*r+(r/4);

                ++count;
        }
                glEnd();
                glFlush();
}
void main(int argc,char **argv)
{
        data();
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowPosition(50,50);
        glutInitWindowSize(400,400);
        glutCreateWindow("Circle");
        init();
        glutDisplayFunc(circle);
        glutMainLoop();
}
```

OUTPUT



Experiment No:7

Solar System

Aim: To draw a solar system using mid point circle algorithm and ellipse drawing algorithm.

Program

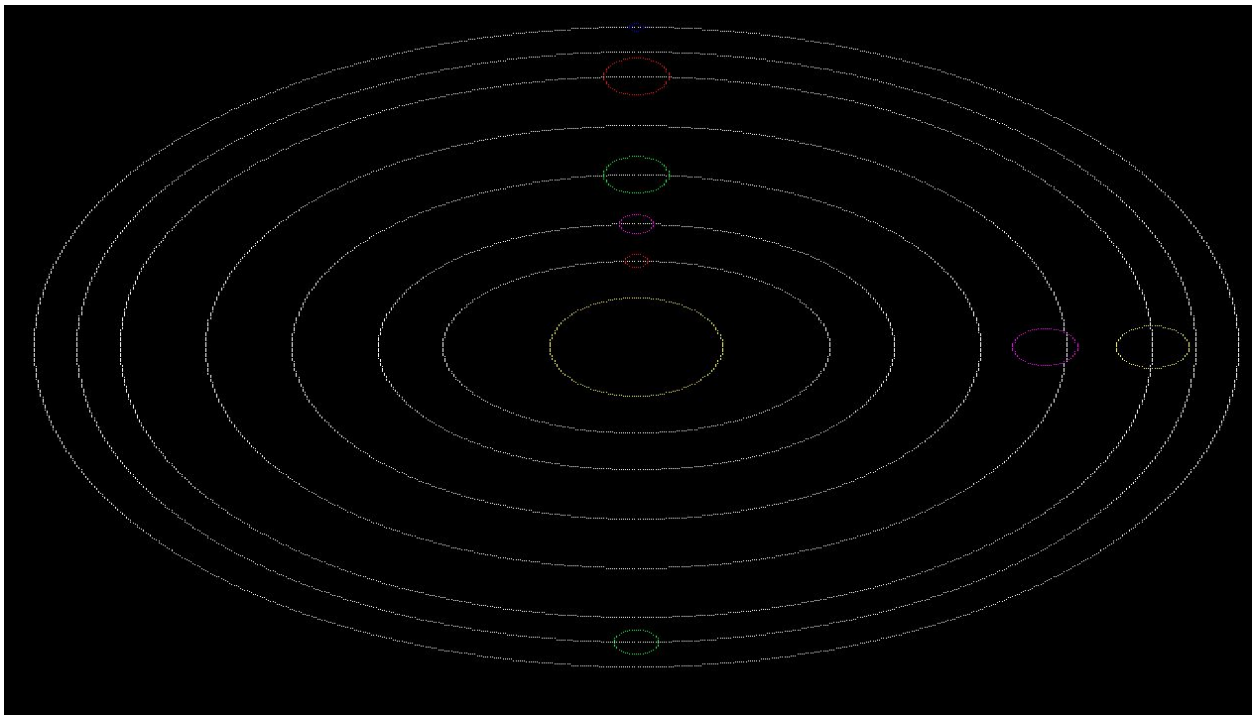
```
#include<GL/glut.h>
#include<stdio.h>
float xc,yc,r,rx,ry,pa,pk,pb,a,b,x,y;
char c;
void init()
{
        glClearColor(0.0,0.0,0.0,0.0);
        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(-300.0,300.0,-300.0,300.0);
}

void cir(int xc,int yc,int r,char c)
{
glClear(GL_COLOR_BUFFER_BIT);
if(c=='y')
glColor3f(1.0,1.0,0.0);
if(c=='b')
glColor3f(0.0,0.0,1.0);
if(c=='r')
glColor3f(1.0,0.0,0.0);
if(c=='g')
glColor3f(0.0,1.0,0.0);
if(c=='w')
glColor3f(1.0,0.0,1.0);
glBegin(GL_POINTS);
pk=(5/4)-r;
x=0;y=r;
glVertex2f(xc+x,yc+y);
glVertex2f(xc+y,yc+x);
glVertex2f(xc-x,yc+y);
glVertex2f(xc-y,yc+x);
glVertex2f(xc+x,yc-y);
glVertex2f(xc+y,yc-x);
glVertex2f(xc-x,yc-y);
glVertex2f(xc-y,yc-x);
while(x<=y)
{
if(pk<0)
{
x=x+1;
pk=pk+(2*x)+1;
}
```

```
else
{
x+=1;
y-=1;
pk=pk+(2*x)+1-(2*y);
}
glVertex2f(xc+x,yc+y);
glVertex2f(xc+y,yc+x);
glVertex2f(xc-x,yc+y);
glVertex2f(xc-y,yc+x);
glVertex2f(xc+x,yc-y);
glVertex2f(xc+y,yc-x);
glVertex2f(xc-x,yc-y);
glVertex2f(xc-y,yc-x);
}
}
void ell(float xc,float yc,float ry,float rx)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,1.0,1.0);
a=ry*ry;
b=rx*rx;
x=0;y=ry;
pa=a-(b*ry)+((1/4)*b);void ell()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0,0.0,0.0);
a=ry*ry;
b=rx*rx;
x=0;y=ry;
pa=a-(b*ry)+((1/4)*b);
glBegin(GL_POINTS);
while((2*a*x)<(2*b*y))
{
if(pa<0)
{
x++;
pa+=2*(a*x)+a;
}
else
{
x++;y--;
pa+=(2*a*x)-(2*b*y)+a;
}
glVertex2f(xc+x,yc+y);
glVertex2f(xc-x,yc+y);
glVertex2f(xc+x,yc-y);
glVertex2f(xc-x,yc-y);
}
pb=(a*((x+0.5)*(x+0.5)))+(b*((y-1)*(y-1)))-(a*b);
while(y>0)
{
if(pb>0)
{
```

```
y--;
pb+=b-2*b*y;
}
else
{
x++;
y--;
pb+=b-2*b*y+2*a*x;
}
glVertex2f(xc+x,yc+y);
glVertex2f(xc-x,yc-y);
glVertex2f(xc-x,yc+y);
glVertex2f(xc+x,yc-y);
}glEnd();
glFlush();
}
glBegin(GL_POINTS);
while((2*a*x)<(2*b*y))
{
if(pa<0)
{
x++;
pa+=2*(a*x)+a;
}
else
{
x++;y--;
pa+=(2*a*x)-(2*b*y)+a;
}
glVertex2f(xc+x,yc+y);
glVertex2f(xc-x,yc+y);
glVertex2f(xc+x,yc-y);
glVertex2f(xc-x,yc-y);
}
pb=(a*((x+0.5)*(x+0.5)))+(b*((y-1)*(y-1)))-(a*b);
while(y>0)
{
if(pb>0)
{
y--;
pb+=b-2*b*y;
}
else
{
x++;
y--;
pb+=b-2*b*y+2*a*x;
}
glVertex2f(xc+x,yc+y);
glVertex2f(xc-x,yc-y);
glVertex2f(xc-x,yc+y);
glVertex2f(xc+x,yc-y);
}
}
```

```
void sol()
{
cir(0,0,40,'y');
ell(0,0,70,90);//1
cir(0,70,5,'r');
ell(0,0,100,120);//2
cir(0,100,8,'w');
ell(0,0,140,160);//3
cir(0,140,15,'g');
ell(0,0,180,200);//4
cir(190,0,15,'w');
ell(0,0,220,240);//5
cir(0,220,15,'r');
ell(0,0,220,240);//6
cir(240,0,17,'y');
ell(0,0,220,240);//7
cir(0,-240,10,'g');
ell(0,0,240,260);//8
//cir(260,0,5,'b');
ell(0,0,260,280);//9
cir(0,260,3,'b');
glEnd();
glFlush();
}

void main(int argc,char** argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowPosition(50,100);
glutInitWindowSize(1900,1900);
glutCreateWindow("solar_system");
init();
glutDisplayFunc(sol);
glutMainLoop();
}
```

OUTPUT



Experiment No:8

Two Dimensional Transformations

Aim: To implement two dimensional transformations rotation,scaling,translation,shearing and

Program

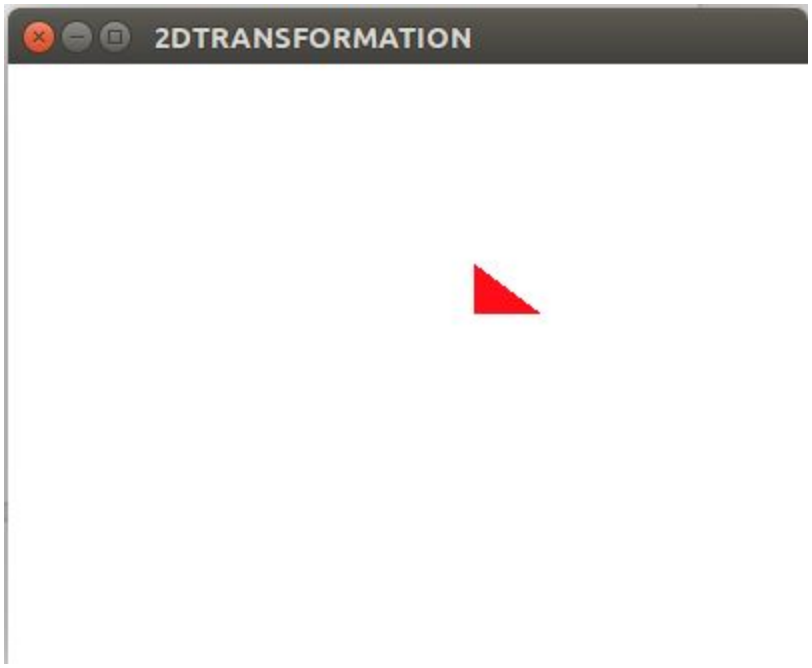
```
#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
float a[3][2],t[2][2],r[10][2],o,d,x00,y00,x01,y01,x02,y02;
void init()
{
        glClearColor(1.0,1.0,1.0,0.0);
        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(-300.0,300.0,-300.0,300.0);
}
void draw()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);
glBegin(GL_TRIANGLES);
glVertex2f(x00,y00);
glVertex2f(x01,y01);
glVertex2f(x02,y02);
glEnd();
glFlush();
}
void input()
{
        printf(" Enter the Coordinates of the Triangle: ");
        printf(" x1_y1 ");
        scanf("%f%f", &x00,&y00);
        printf(" x2_y2 ");
scanf("%f%f", &x01,&y01);
        printf(" x3_y3 ");
scanf("%f%f", &x02,&y02);
        a[0][0]=x00;
        a[0][1]=y00;
        a[1][0]=x01;
        a[1][1]=y01;
        a[2][0]=x02;
        a[2][1]=y02;
        draw();
}
void mul()
{
int i,j,p;
for(i=0;i<3;i++)
{
```

```
for(j=0;j<2;j++)
{
r[i][j]=0;
for(p=0;p<2;p++)
{
r[i][j]=r[i][j]+a[i][p]*t[p][j];
}
}
}
        x00=r[0][0];
        y00=r[0][1];
        x01=r[1][0];
        y01=r[1][1];
        x02=r[2][0];
        y02=r[2][1];
}
void sel()
{
        int ch,ch1,para;
        while(1>0)
{
        printf("MENU\n1.Rotation\n2.Reflection\n3.Scaling\n4.Shearing\n5.Translation\n");
        printf("Choice: ");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:
                printf("Enter the Rotation Degree");
                scanf("%f",&o);
                d=o*(3.14/180);
                 t[0][0]=cos(d);
                 t[0][1]=sin(d);
                 t[1][0]=-sin(d);
                 t[1][1]=cos(d);
                 mul();
                 draw();
                 break;
                case 2:
         printf(" Reflection about 1.x-Axis 2.y-Axis ");
                scanf("%d",&ch1);
                if(ch1==1)
                        {
                                t[0][0]=1;
                                t[0][1]=0;
                t[1][0]=0;
                t[1][1]=-1;
                        }
                else if(ch1==2)
                        {
                                t[0][0]=-1;
t[0][1]=0;
t[1][0]=0;
t[1][1]=1;
                        }
```

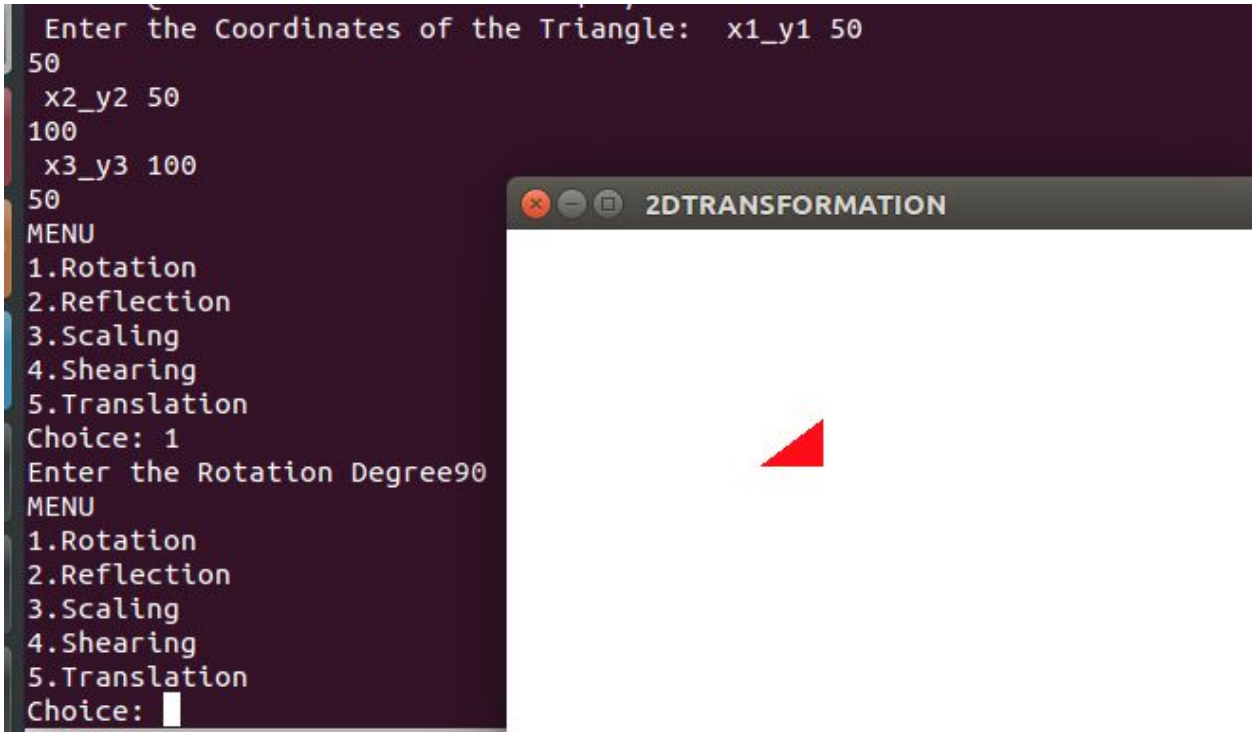
```
                else
                        {
                                printf(" Wrong Option ");
                        }
                mul();
                draw();
                break;
                case 3:
printf(" Uniform Scaling ");
        printf(" Enter the Parameter: ");
                scanf("%d",&para);
if(para>1)
                        {
                                printf(" Expansion ");
                        }
                else
                        {
                                printf(" Compression ");
                        }
t[0][0]=para;
t[0][1]=0;
t[1][0]=0;
t[1][1]=para;
        mul();
draw();
break;
                case 4:
                printf(" Shearing ");
printf(" Enter the Shear Factor: ");
scanf("%d",&para);
x02+=para;
                y02+=para;
draw();
break;
                case 5:
                printf(" Translation ");
printf(" Enter the Translation Degree: ");
scanf("%d",&para);
x00+=para;
y00+=para;
                x01+=para;
y01+=para;
                x02+=para;
y02+=para;
draw();
break;
                case 6:

                exit(0);

                }


        }
```

```
}

void main(int argc,char **argv)
{
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE||GLUT_RGB);
	glutInitWindowPosition(50,100);
	glutInitWindowSize(400,300);
	glutCreateWindow("2DTRANSFORMATION");
	init();
	input();
	sel();
	glutDisplayFunc(draw);
	glutMainLoop();
}
```

OUTPUT



Rotation

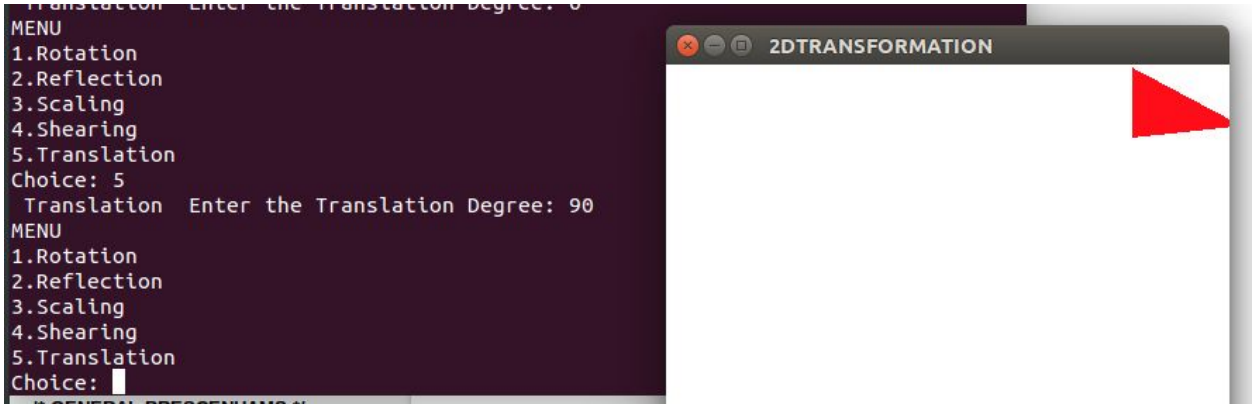


Reflection

Scaling

Shearing

Translation



Experiment No:9

Line Clipping

Aim: To implement line clipping algorithm.

Program

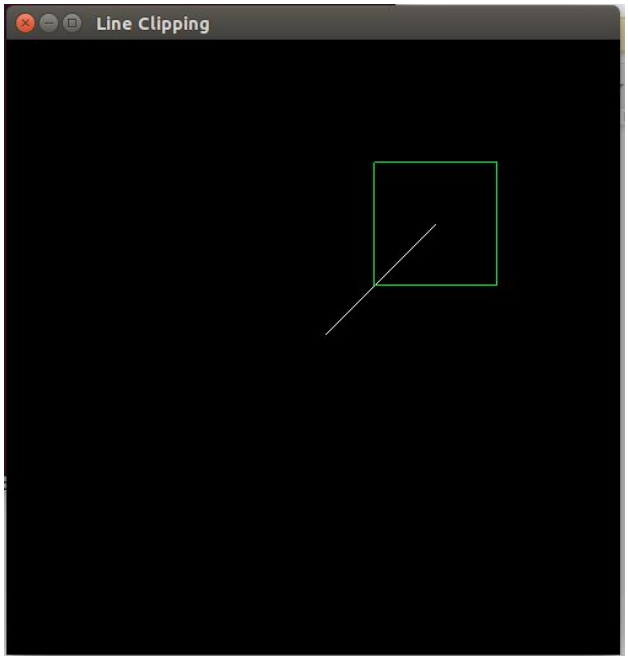
```
#include<GL/glut.h>
#include<stdio.h>
int lx1,ly1,lx2,ly2,bxmin,bymin,bxmax,bymax,rr1,rr2;
void init() {
	glClearColor(0.0,0.0,0.0,0.0);
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(-250.0,250.0,-250.0,250.0);
}
int finder(int x, int y,int xmin,int ymin,int xmax, int ymax){
	int op=0,oo=0,top=1000,bottom=0100,left=0001,right=0010;
	if (y > ymax)
	op = op+1000;
else if (y < ymin)
	op = op+100;
if (x > xmax)
	op =op+ 10;
else if (x < xmin)
	op = op+1;
	return op;
}
void redraw(){
	float x1=lx1,a,b,c,d,x2=lx2,y1=ly1,y2=ly2,m,x,yr1,yrr1,xrr1,xr1,y,xr2,yr2,xrr2,yrr2,k,l;
	m=((y2-y1)/(x2-x1));
	yr1=y1+(m*(bxmin-x1));

	yrr1=y1+(m*(bxmax-x1));
	xr1=x1+((bymin-y1)/m);
	xrr1=x1+((bymax-y1)/m);

	yr2=y2+(m*(bxmin-x2));

	yrr2=y2+(m*(bxmax-x2));
	xr2=x2+((bymin-y2)/m);
	xrr2=x2+((bymax-y2)/m);k=((yr2-yr1)/(xr2-xr1));
	l=((yrr2-yrr1)/(xrr2-xrr1));
		{
		switch(rr1){
			case 0:a=lx1;b=ly1;break;
			case 1:a=bxmin;b=yr1;break;
			case 10:a=bxmax;b=yrr1;break;
			case 1000:a=xrr1;b=bymax;break;
			case 100:a=xr1;b=bymin;break;
			case 1001:a=xrr1;b=yr1;if(xr1<bxmin) a=bxmin; if(yr1<bymin) b=bymin;break;
			case 1010:a=xrr1;b=yrr1;if(xr1<bxmin) a=bxmin; if(yr1<bymin) b=bymin;break;
			case 101:a=yr1;b=yr1;if(xr1<bxmin) a=bxmin; if(yr1<bymin) b=bymin;break;
```

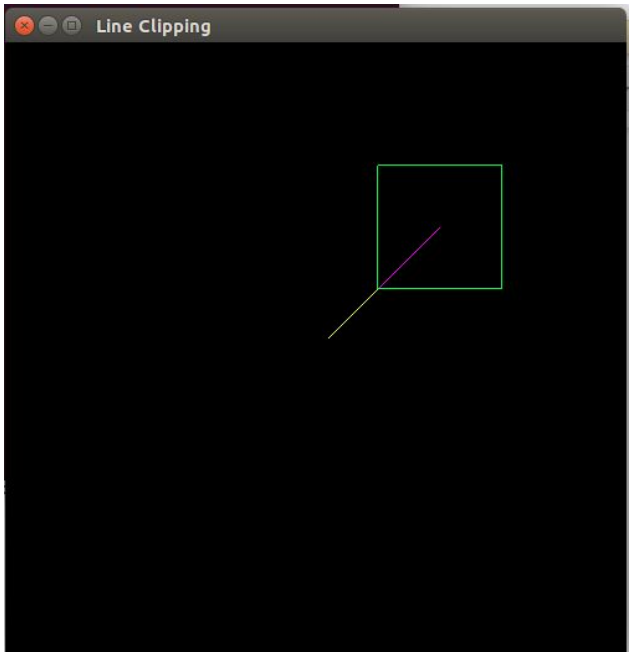
```
                        case 110:a=yrr1;b=xr1;if(xr1<bxmin) a=bxmin; if(yr1<bymin) b=bymin;break;
                        default:printf("error swtich");
                        }
                switch(rr2){
                        case 0:c=lx2;d=ly2;break;
                        case 1:c=bxmin;d=yr2;break;
                        case 10:c=bxmax;d=yrr2;break;
                        case 1000:c=xrr2;d=bymax;break;
                        case 100:c=xr2;d=bymin;break;
                        case 1001:c=xrr2;d=yr2;if(xrr2>bxmax) c=bxmax; if(yrr2>bymax) d=bymax;break;
                        case 1010:c=xrr2;d=yrr2;if(xrr2>bxmax) c=bxmax; if(yrr2>bymax) d=bymax;break;
                        case 101:c=yr2;d=yr2;if(xrr2>bxmax) c=bxmax; if(yrr2>bymax) d=bymax;break;
                        case 110:c=yrr2;d=xr2;if(xrr2>bxmax) c=bxmax; if(yrr2>bymax) d=bymax;break;
                        default:printf("error swtich");
                        }
                }
        glColor3f(1.0,0.0,1.0);
        glBegin(GL_LINES);
        glVertex2i(a,b);
        glVertex2i(c,d);
        glEnd();
        glFlush();
        glColor3f(0.0,1.0,0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2i(bxmin,bymin);
        glVertex2i(bxmin,bymax);
        glVertex2i(bxmax,bymax);
        glVertex2i(bxmax,bymin);
        glEnd();
        glFlush();
        glColor3f(1.0,1.0,0.0);
        glBegin(GL_LINES);
        glVertex2i(lx1,ly1);
        glVertex2i(a,b);
        glVertex2i(lx2,ly2);
        glVertex2i(c,d);
        glEnd();
        glFlush();

}
void equality(){
        int r1=0,r2=0,t1,t2,e=0,t=1;
        r1=finder(lx1,ly1,bxmin,bymin,bxmax,bymax);
        r2=finder(lx2,ly2,bxmin,bymin,bxmax,bymax);
        rr1=r1;rr2=r2;
        printf("\nLine is :");
        do{
                e=e+1;
                t1=r1%10;
                t2=r2%10;
                if(t1==t2&&t1==1&&t2==1){
                        switch(e){
                                case 1:printf("Left ");t=0;break;
                                case 2:printf("Right ");t=0;break;
```

```
                                        case 3:printf("Bottom ");t=0;break;
                                        case 4:printf("Top ");t=0;break;
                                        deafult:t=1;break;
                                        }
                                }
                        r1=r1/10;
                        r2=r2/10;
                        }while(e<=4);
                if(t==1)
                        {
                        printf("Inside %d\t%d\nEnter any key to continue",rr1,rr2);
                        int a;
                        scanf("%d",&a);
                        redraw();
                        }
                else
                        printf("| Outside\n");
                }

void draw(){
                glClear(GL_COLOR_BUFFER_BIT);
                glColor3f(1.0,1.0,1.0);
                glBegin(GL_LINES);
                glVertex2i(lx1,ly1);
                glVertex2i(lx2,ly2);
                glEnd();
                glFlush();
                glColor3f(0.0,1.0,0.0);
                glBegin(GL_LINE_LOOP);
                glVertex2i(bxmin,bymin);
                glVertex2i(bxmin,bymax);
                glVertex2i(bxmax,bymax);
                glVertex2i(bxmax,bymin);
                glEnd();
                glFlush();
}
void reader(){
                printf("Enter line vertices:");
                scanf("%d%d%d%d",&lx1,&ly1,&lx2,&ly2);
                bxmin=50;
                bymin=50;
                bxmax=150;
                bymax=150;
}
void main(int argc, char ** argv){
                reader();
                glutInit(&argc,argv);
                glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
                glutInitWindowPosition(200,100);
                glutInitWindowSize(500,500);
                glutCreateWindow("Line Clipping");
                init();
                glClear(GL_COLOR_BUFFER_BIT);
                draw();
```

```
        glutDisplayFunc(equality);
        glutMainLoop();
}
```

OUTPUT





Experiment No:10

Polygon Clipping

Aim: To implement polygon clipping algorithm.

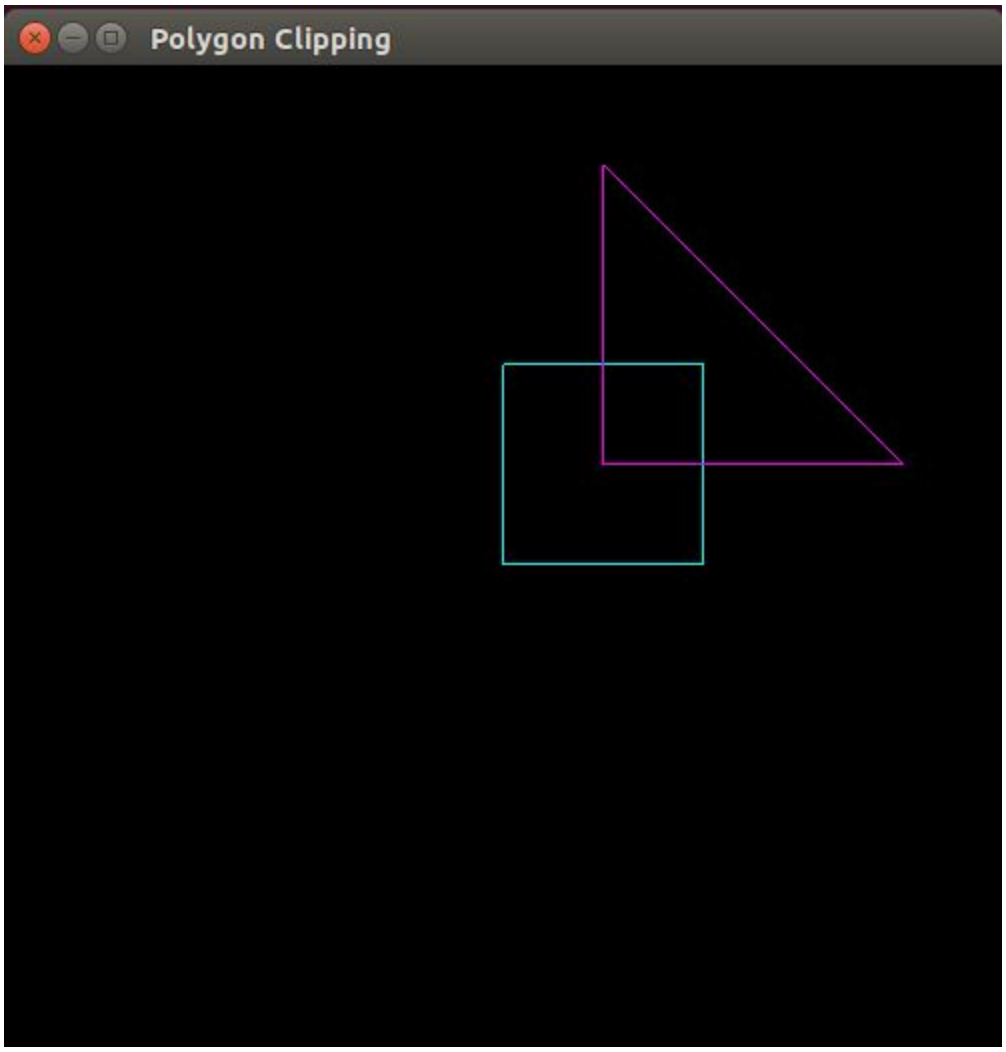
Program

```
#include<GL/glut.h>
#include<stdio.h>
int i,n,xmin,xmax,ymin,ymax,side1[10],side2[10],ch;
void init()
{
	glClearColor(0.0,0.0,0.0,0.0);
	glMatrixMode(GL_PROJECTION);
	gluOrtho2D(-250.0,250.0,-250.0,250.0);
}
void getdata()
{
	printf("Enter the no of vertices:");
	scanf("%d",&n);
	printf("Enter the sides:");
	for(i=0;i<n;i++)
		scanf("%d%d",&side1[i],&side2[i]);
}
void polygon1()
{
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(0.0,1.0,1.0);
	glBegin(GL_LINE_LOOP);
		glVertex2i(0,0);
		glVertex2i(100,0);
		glVertex2i(100,100);
		glVertex2i(0,100);
	glEnd();
	glFlush();
		glColor3f(1.0,0.0,1.0);
		glBegin(GL_LINE_LOOP);
		for(i=0;i<n;i++)
			glVertex2i(side1[i],side2[i]);

	glEnd();
	glFlush();
	while(1)
	{
		printf("Do u want to see the clipped polygon:");
		printf("1.YES\n2.NO\n");
		scanf("%d",&ch);
		switch(ch)
		{
```

```
                        case 1:glClear(GL_COLOR_BUFFER_BIT);
                                glColor3f(0.0,0.0,1.0);
                                glBegin(GL_LINE_LOOP);
                                        glVertex2i(0,0);
                                        glVertex2i(100,0);
                                        glVertex2i(100,100);
                                        glVertex2i(0,100);
glEnd();
                                glFlush();
                                glColor3f(1.0,0.0,1.0);
                                glBegin(GL_LINE_LOOP);
                                for(i=0;i<n;i++)
                                {
                                        if(side1[i]<=100 && side1[i]>=0 && side2[i]<=100 && side2[i]>=0)
                                        {
                                                glVertex2i(side1[i],side2[i]);
                                        }
                                        else if(side1[i]<=100 && side1[i]>=0 && side2[i]>100 )
                                        {
                                                glVertex2i(side1[i],100);
                                        }
                                        else if(side1[i]<=100 && side1[i]>=0 && side2[i]<0 )
                                        {
                                                glVertex2i(side1[i],0);
                                        }
                                        else if(side2[i]<=100 && side2[i]>=0 && side1[i]>100 )
                                        {
                                                glVertex2i(100,side2[i]);
                                        }
                                        else if(side2[i]<=100 && side2[i]>=0 && side1[i]<0 )
                                        {
                                                glVertex2i(0,side2[i]);
                                        }
                                        else if(side1[i]>100 && side2[i]>100)
                                        {
                                                glVertex2i(100,100);
                                        }
                                        else
                                        {
                                                glVertex2i(0,0);
                                        }

                                }
                                glEnd();
                                glFlush();
                                break;
                        case 2:exit(1);
                                break;
                }
        }
}
void main(int argc,char**argv)
{
getdata();
```

```
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowPosition(50,100);
glutInitWindowSize(500,500);
glutCreateWindow("Polygon Clipping");
glClear(GL_COLOR_BUFFER_BIT);
init();
glutDisplayFunc(polygon1);
glutMainLoop();
}
```

OUTPUT



Experiment No:11

Bouncing Ball

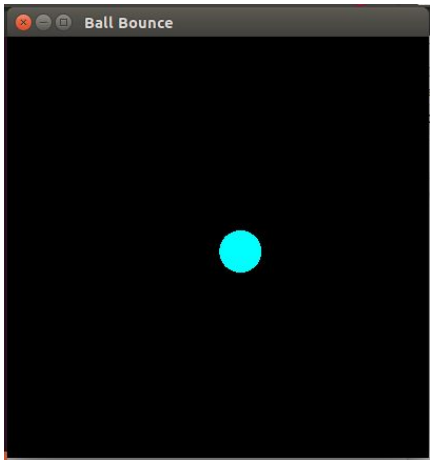
Aim: To illustrate the bouncing of a ball.

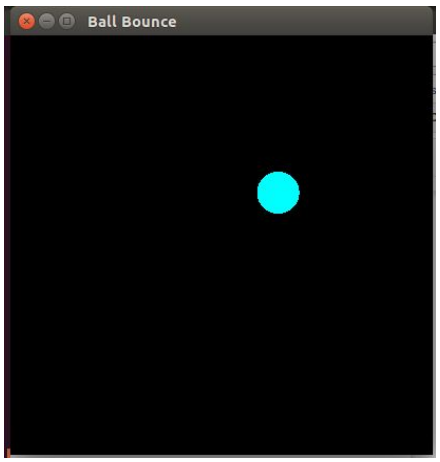
Program

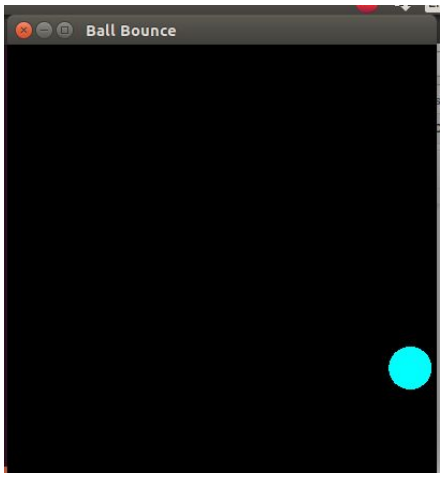
```
#include<stdlib.h>
#include<GL/glut.h>
float ballX = -0.8f;
float ballY = 0.5f;
float ballZ = -1.0f;
static int flag=1;
static int flagger=1;
void drawBall(void) {
glColor3f(0.0, 1.0, 1.0);
glTranslatef(ballX,ballY,ballZ);
glutSolidSphere (0.1, 100, 10); //Objects
}
void keyPress(int key, int x, int y)
{
if(key==GLUT_KEY_UP)
ballY += 0.05f;
if(key==GLUT_KEY_DOWN)
ballY -= 0.05f;
        if(key==GLUT_KEY_LEFT)
ballX -= 0.05f;
if(key==GLUT_KEY_RIGHT)
ballX += 0.05f;
glutPostRedisplay();
}
void initRendering()
{
glEnable(GL_DEPTH_TEST);
}
void drawScene()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();//Identity Matrix
drawBall();
glutSwapBuffers();
}
float rt=0.5;
float i=0.006f;
void updater(int value){
        if(flagger)
        {
ballX += i;
if(ballX>0.9)
{
         i -= 0.001f;
```

```
flagger=0;
}
}
if (!flagger)
{
ballX -= i;
if(ballX<-0.9)
{
        i -= 0.001f;
flagger=1;
}
}
glutPostRedisplay();
glutTimerFunc(1, updater, 5000);
        }
void update(int value) {
if(flag)
{
ballY += 0.01f;
if(ballY>rt)
{
flag=0;
}
}
        rt=rt-0.001;
if (!flag)
{
ballY -= 0.01f;
if(ballY<-0.5)
{
flag=1;
}
}
glutPostRedisplay();
glutTimerFunc(1, update, 5000);
}
int main(int argc,char** argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(400,400);
glutCreateWindow("Ball Bounce");
initRendering();
glutDisplayFunc(drawScene);
glutSpecialFunc(keyPress);
glutTimerFunc(300, update, 5000);
glutTimerFunc(300, updater, 5000);
glutMainLoop();
return(0);
}
```

OUTPUT







Experiment No:12

Flying Kite

Aim: To illustrate the flying of a kite.

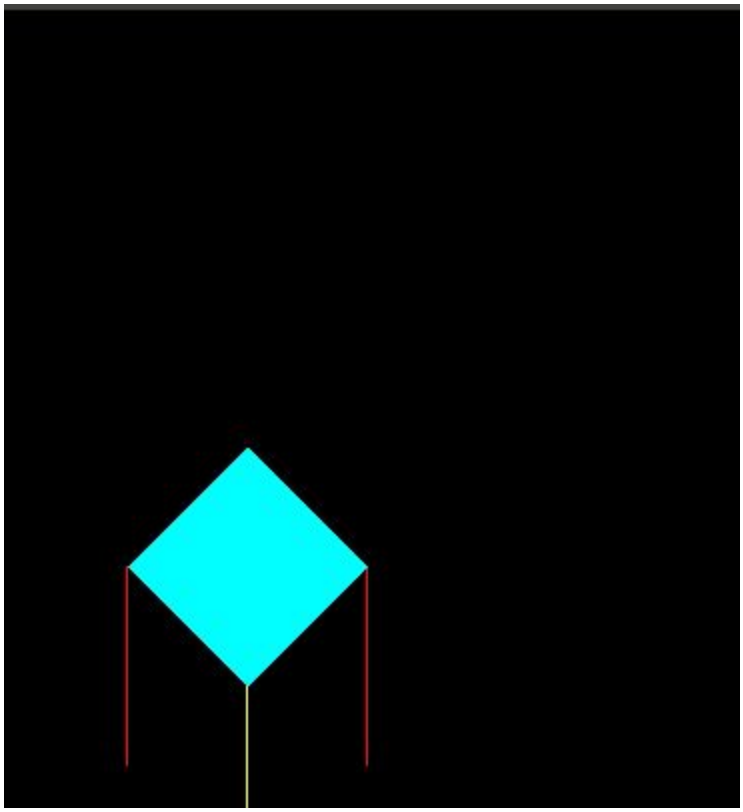
Program

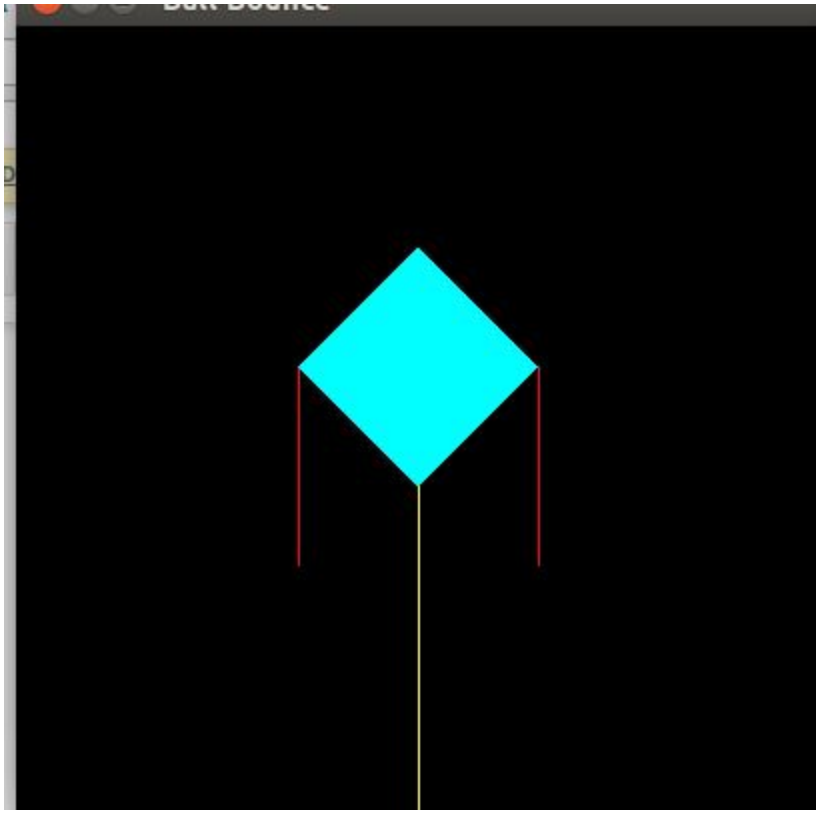
```
#include<stdlib.h>
#include<GL/glut.h>
float kiteX = -0.8f;
float kiteY = -0.5f;
float kiteZ = -1.0f;
static int flag=1;
static int flagger=1;
void drawKite(void) {
glColor3f(0.0, 1.0, 1.0);
glTranslatef(kiteX,kiteY,kiteZ);
glBegin(GL_POLYGON);
glVertex2f(0.0,-0.3);
glVertex2f(-0.3,0.0);
glVertex2f(0.0,0.3);
glVertex2f(0.3,0.0);
glEnd();
glFlush();
glBegin(GL_LINES);
glColor3f(1.0,0.0,0.0);
glVertex2f(-0.3,0.0);
glVertex2f(-0.3,-0.5);
glVertex2f(0.3,0.0);
glVertex2f(0.3,-0.5);
glColor3f(1.0,1.0,0.0);
glVertex2f(0.0,-0.3);
glVertex2f(0.0,-5.0);
glEnd();
glFlush();
}

void keyPress(int key, int x, int y)
{
if(key==GLUT_KEY_UP)
kiteY += 0.05f;
if(key==GLUT_KEY_DOWN)
kiteY -= 0.05f;
        if(key==GLUT_KEY_LEFT)
kiteX -= 0.05f;
if(key==GLUT_KEY_RIGHT)
kiteX += 0.05f;
glutPostRedisplay();
}
void initRendering()
{
glEnable(GL_DEPTH_TEST);
}
```

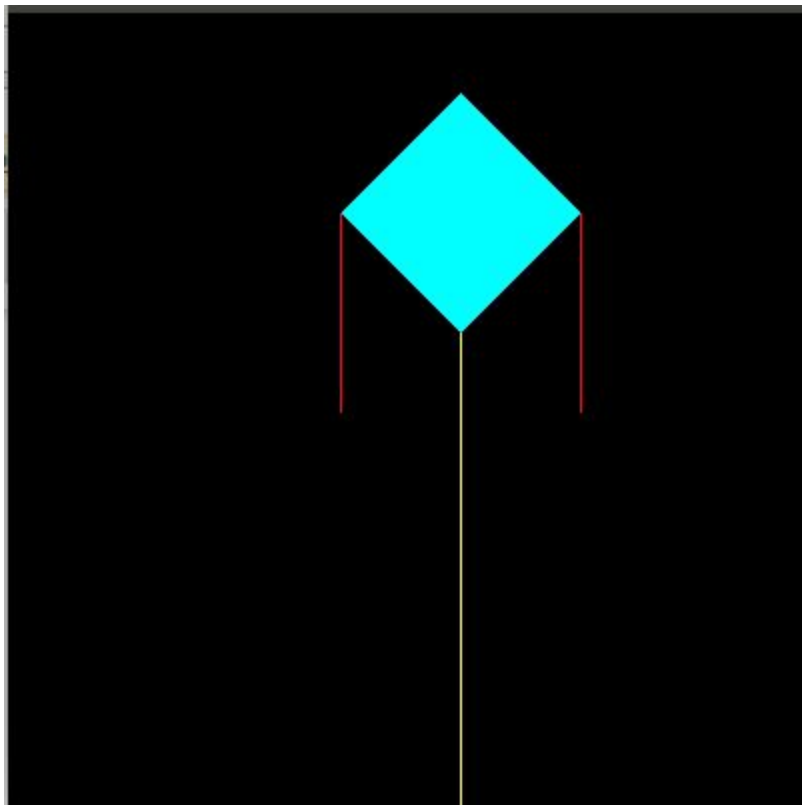
```
void drawScene()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();//Identity Matrix
drawKite();
glutSwapBuffers();
}
float rt=0.5;
float x=0.9;
void updater(int value){
	if(flag==0)
		x=0.4;
	if(flagger)
	{
kiteX += 0.002f;
if(kiteX>x)
{
flagger=0;
}
}
if (!flagger)
{
kiteX -= 0.002f;
if(kiteX<-x)
{
flagger=1;
}
}
glutPostRedisplay();
glutTimerFunc(1, updater, 5000);
	}
void update(int value)
{
if(flag)
{
kiteY += 0.0005f;
if(kiteY>rt)
{
flag=0;
}
}
glutPostRedisplay();
glutTimerFunc(1, update, 5000);
}
int main(int argc,char** argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(400,400);
glutCreateWindow("Ball Bounce");
initRendering();
glutDisplayFunc(drawScene);
```

```
glutSpecialFunc(keyPress);
glutTimerFunc(30, update, 5000);
glutTimerFunc(30, updater, 5000);
glutMainLoop();
return(0);
}
```

OUTPUT







Experiment No:13

Moving Boat

Aim: To illustrate the movement of a boat.

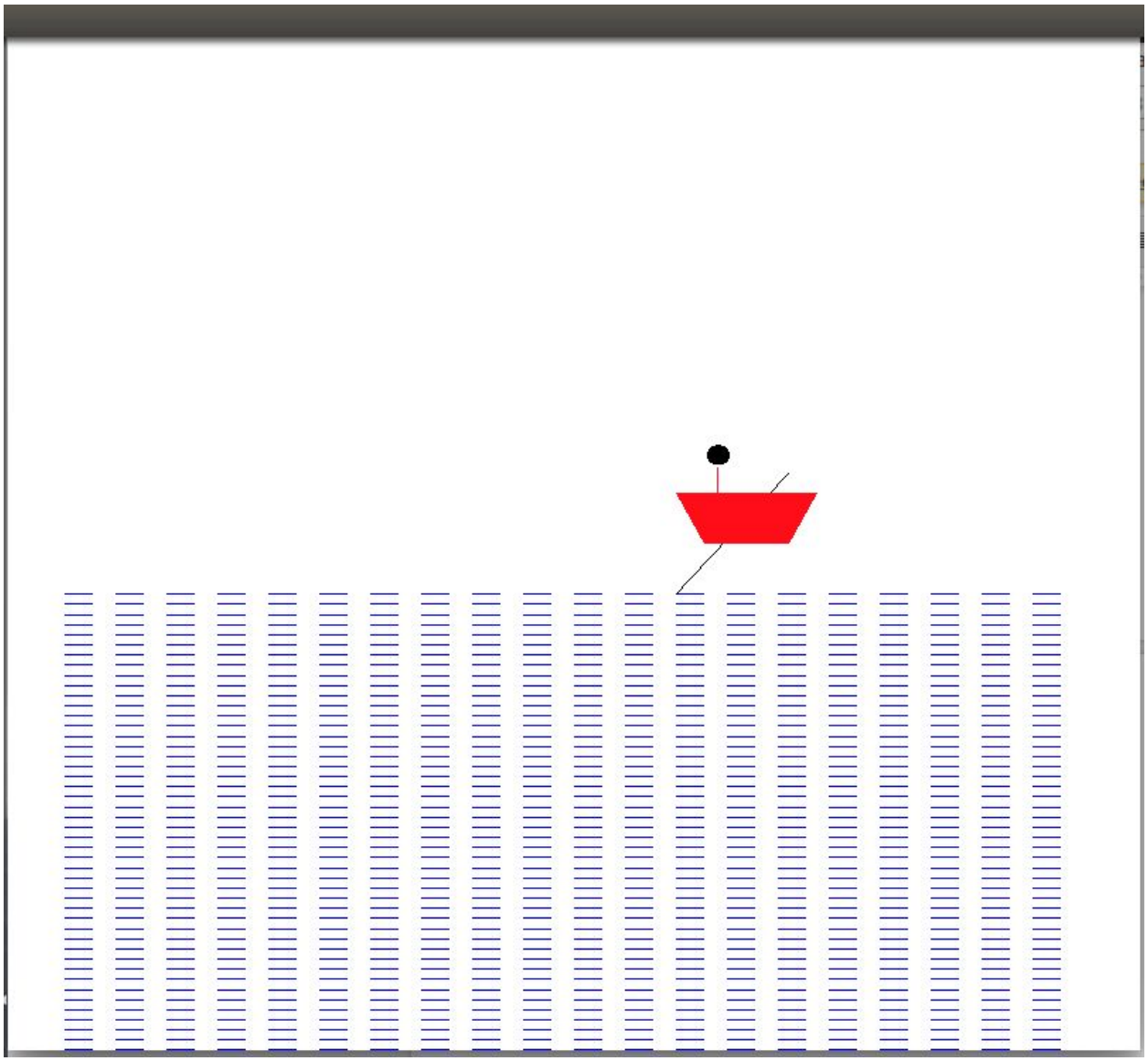
Program

```
#include<stdlib.h>
#include<GL/glut.h>
float boatX = -0.2f;
float boatY = 0.0f;
float boatZ = -1.0f;
static int flag=1;
static int flagger=1;
void triangle();
void water()
{
	float x1=-0.9,x2=-0.85,y=-0.1;
	int i,j;
	glColor3f(0.0,0.0,1.0);
				glBegin(GL_LINES);
				for(i=0;i<50;i++)
					{
						for(j=0;j<20;j++)
						{ glVertex2f(x1,y);
x1+=0.09;
						glVertex2f(x2,y);
x2+=0.09;
						}
						y-=0.02;
						x1=-0.9;
						x2=-0.85;
					}
glEnd();
glFlush();
}
void triangle()
{
	glColor3f(0.0, 0.0, 0.0);
	glColor3f(1.0,0.0,0.0);//boat
	glTranslatef(boatX,boatY,boatZ);
	glBegin(GL_POLYGON);
	glVertex2f(0.0,0.1);
	glVertex2f(0.25,0.1);
	glVertex2f(0.2,0.0);
	glVertex2f(0.05,0.0);
	glEnd();
	glFlush();
	glColor3f(1.0, 0.0, 0.0);//mans body line
```

```
        glBegin(GL_LINES);
        glVertex2f(0.075,0.15);
        glVertex2f(0.075,0.1);
        glColor3f(0.0, 0.0, 0.0);//stick
        glVertex2f(0.2,0.14);
        glVertex2f(0.0,-0.1);
        glEnd();
        glFlush();
        glColor3f(0.0, 0.0, 0.0);//mans haed
        glTranslatef(0.075,0.175,0);
        glutSolidSphere (0.02, 50,100);
}
void keyPress(int key, int x, int y)
{
if(key==GLUT_KEY_UP)
boatY += 1.0f;
if(key==GLUT_KEY_DOWN)
boatY -= 1.0f;
        if(key==GLUT_KEY_LEFT)
boatX -= 1.0f;
if(key==GLUT_KEY_RIGHT)
boatX += 1.0f;
glutPostRedisplay();
}
void initRendering()
{
glEnable(GL_DEPTH_TEST);
}
void drawScene()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glClearColor(1.0,1.0,1.0,0.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
water();

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();//Identity Matrix
triangle();
glutSwapBuffers();
}
float rt=0.5;
void updater(int value){
        if(flag==1)
        {
boatX += 0.004f;
if(boatX>1.2)
{
flag=0;
}
}
glutPostRedisplay();
glutTimerFunc(1, updater, 5000);
        }
```

```
void update(int value) {
if(flag)
{
boatY += 0.01f;
if(boatY>rt)
{
flag=0;
}
}
            rt=rt-0.001;
if (!flag)
{
boatY -= 0.01f;
if(boatY<-0.5)
{
flag=1;
}
}
glutPostRedisplay();
glutTimerFunc(1, update, 5000);
}
int main(int argc,char** argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(800,800);
glutCreateWindow("BOAT");
initRendering();
glutDisplayFunc(drawScene);
glutSpecialFunc(keyPress);
glutTimerFunc(30, updater, 5000);
glutMainLoop();
return(0);
}
```

OUTPUT



Experiment No:14

Moving Car

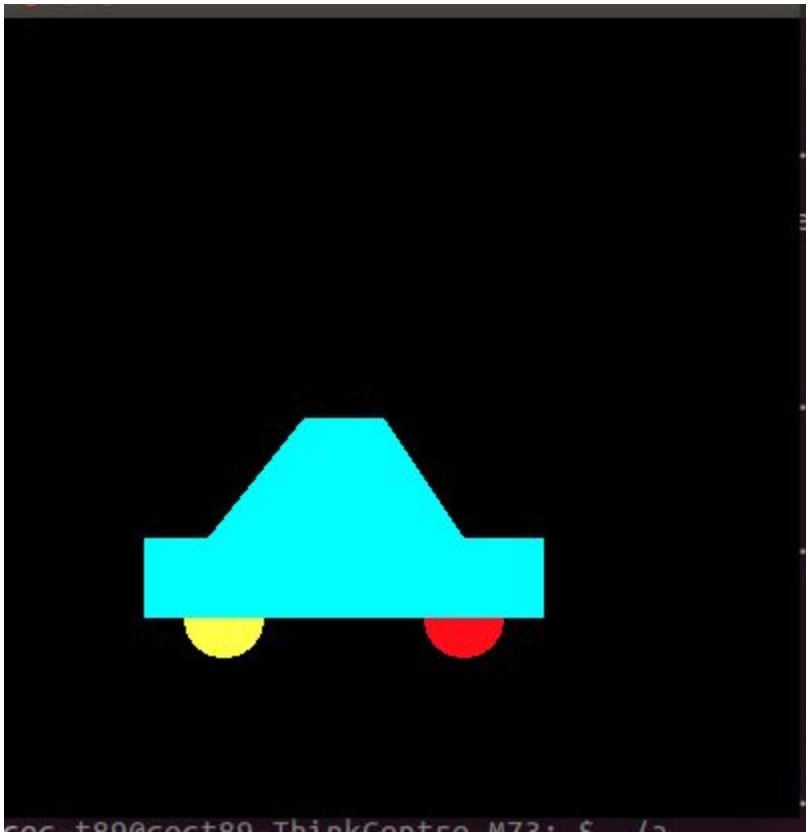
Aim: To illustrate the movement of a car.

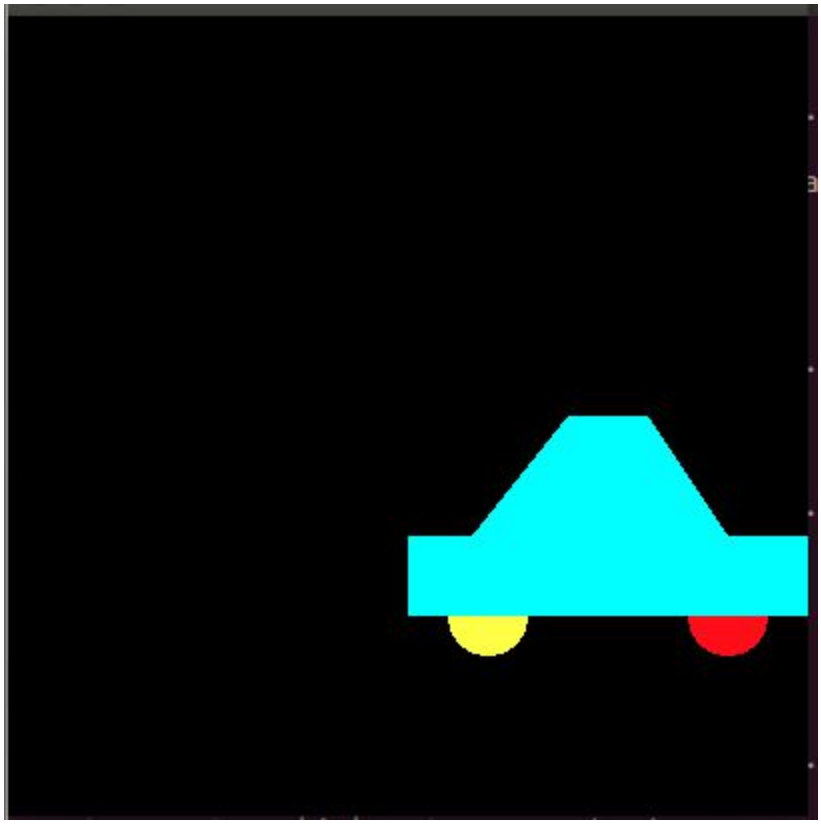
Program

```
#include<stdlib.h>
#include<GL/glut.h>
float carX = -0.8f;
float carY = -0.5f;
float carZ = -1.0f;
float wheel1X=-0.7,wheel1Y=-0.3,wheel2X=-0.3,wheel2Y=-0.3;
static int flag=1;
static int flagger=1;
void drawCar(void) {
	glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0, 1.0, 1.0);
glTranslatef(carX,carY,carZ);
glBegin(GL_POLYGON);
		glVertex2f(-0.5,0);
		glVertex2f(-0.5,0.2);
		glVertex2f(-0.3,0.2);
		glVertex2f(-0.1,0.5);
		glVertex2f(0.1,0.5);
		glVertex2f(0.3,0.2);
		glVertex2f(0.5,0.2);
		glVertex2f(0.5,0);
	glEnd();
	glFlush();
	glColor3f(1.0, 0.0, 0.0);
	glTranslatef(0.3,0,0);
	glutSolidSphere(0.1,100,10);
	glColor3f(1.0, 1.0, 0.0);
	glTranslatef(-0.6,0,0);
	glutSolidSphere(0.1,100,10);
}
void keyPress(int key, int x, int y)
{
if(key==GLUT_KEY_UP)
carY += 0.05f;
if(key==GLUT_KEY_DOWN)
carY -= 0.05f;
	if(key==GLUT_KEY_LEFT)
carX -= 0.05f;
if(key==GLUT_KEY_RIGHT)
carX += 0.05f;
glutPostRedisplay();
```

```
}
void initRendering()
{
glEnable(GL_DEPTH_TEST);
}
void drawScene()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();//Identity Matrix
drawCar();
glutSwapBuffers();
}
float rt=0.5;
void updater(int value){
        if(flag==1)
        {
carX += 0.004f;
if(carX>1.5)
{
flag=0;
}
}
glutPostRedisplay();
glutTimerFunc(1, updater, 5000);
}
int main(int argc,char** argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowPosition(0,0);
glutInitWindowSize(400,400);
glutCreateWindow("car Bounce");
initRendering();
glutDisplayFunc(drawScene);
glutSpecialFunc(keyPress);
glutTimerFunc(30, updater, 5000);
glutMainLoop();
return(0);
}
```

OUTPUT





Experiment No:15

Tree Climbing

Aim: To illustrate the Coconut tree climbing.

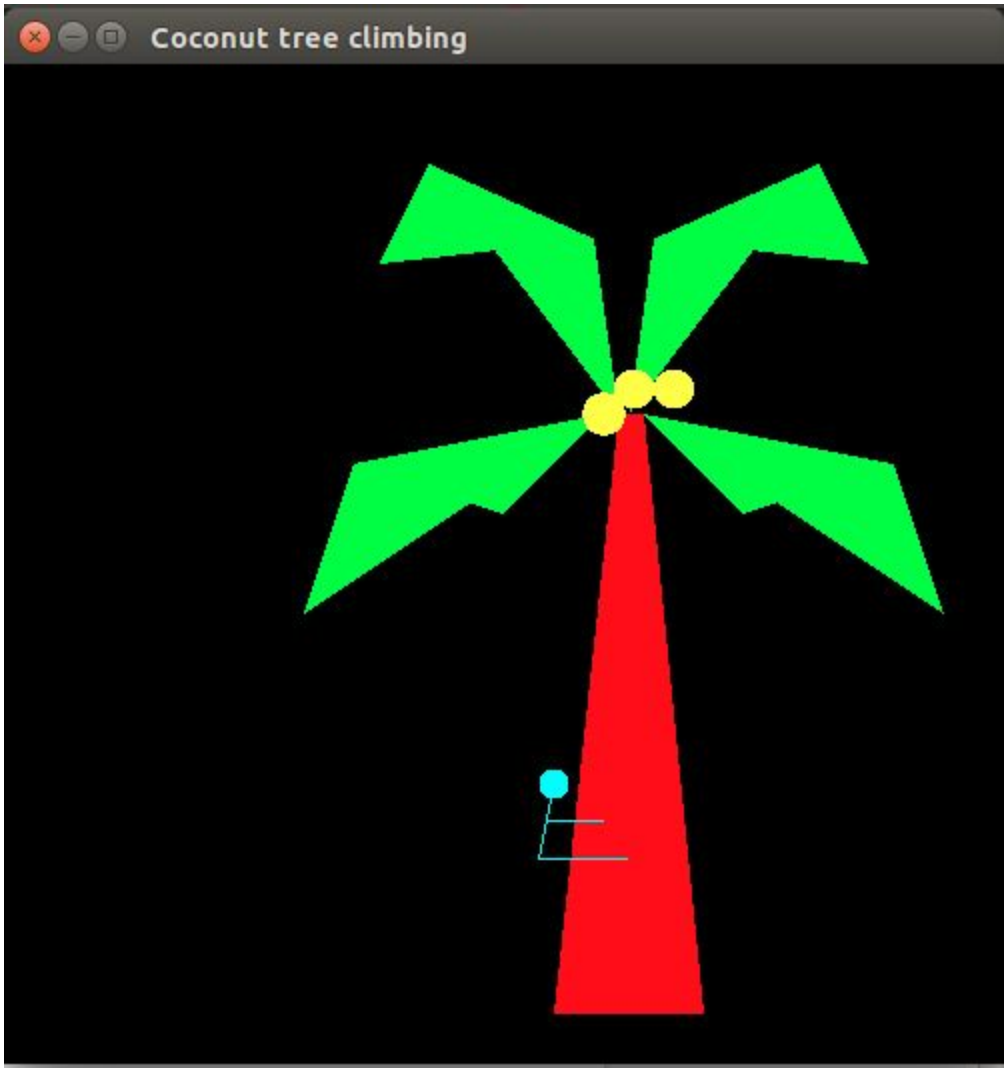
Program

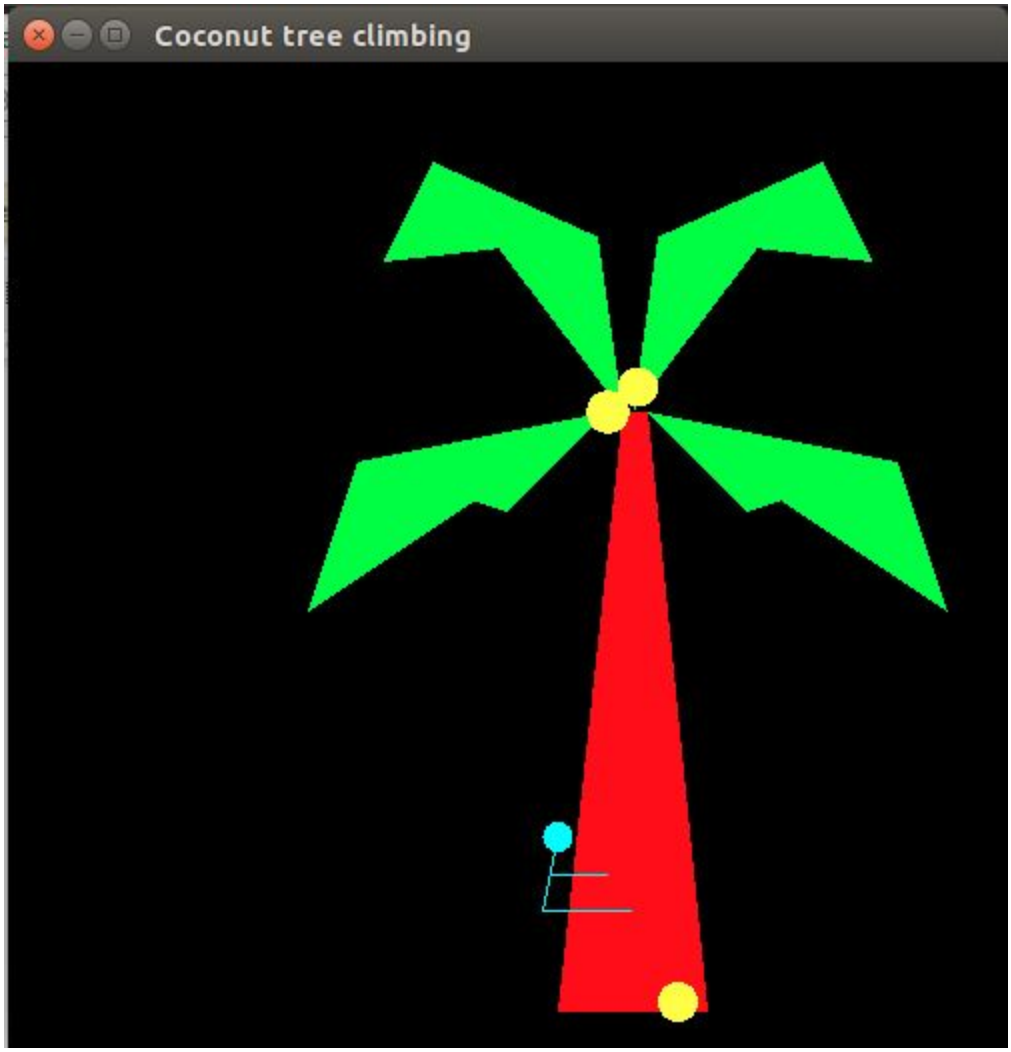
```
#include<stdlib.h>
#include<GL/glut.h>
float manX = 0.1f;
float manY = -1.0f;
float manZ = -1.0f;
float cocoX = 0.08f;
float cocoY = 0.0f;
static int flag=1;
static int flagger=1;
void drawTree(void) {
	/*TRUNK*/
glColor3f(1.0,0.0,0.0);

	glBegin(GL_POLYGON);
		glVertex2f(0.1,-0.9);
		glVertex2f(0.4,-0.9);
		glVertex2f(0.28,0.3);
		glVertex2f(0.23,0.3);
	glEnd();
	glFlush();
	/*LEAF*/
	glColor3f(0.0,1.0,0.0);
	glBegin(GL_POLYGON);
		glVertex2f(0.2,0.3);
		glVertex2f(0.0,0.1);
		glVertex2f(-0.3,0.2);
		glVertex2f(-0.4,-0.1);
	glEnd();
	glFlush();
	glBegin(GL_POLYGON);
		glVertex2f(0.18,0.65);
		glVertex2f(0.23,0.3);
		glVertex2f(-0.15,0.8);
		glVertex2f(-0.25,0.6);
	glEnd();
	glFlush();
	glBegin(GL_POLYGON);
		glVertex2f(0.28,0.3);
		glVertex2f(0.48,0.1);
		glVertex2f(0.78,0.2);
```

```
		glVertex2f(0.88,-0.1);
	glEnd();
	glFlush();
	glBegin(GL_POLYGON);
		glVertex2f(0.3,0.65);
		glVertex2f(0.25,0.3);
		glVertex2f(0.63,0.8);
		glVertex2f(0.73,0.6);
	glEnd();
	glFlush();/*leaf end*/

	/*COCONUT*/
	glColor3f(1.0, 1.0, 0.0);
	glTranslatef(0.2,0.3,0);
	glutSolidSphere(0.043,100,10);
	glTranslatef(0.06,0.05,0);
	glutSolidSphere(0.04,100,10);
	glTranslatef(cocoX,cocoY,0);
	glutSolidSphere(0.04,100,10);
}
void drawMan()
{
	/*MAN HEAD*/
	glColor3f(0.0, 1.0, 1.0);
glTranslatef(manX,manY-0.2,0);
	glutSolidSphere(0.03,100,10);

	/*MAN BODY*/
	glBegin(GL_LINES);
		glVertex2f(0.0,-0.0);
		glVertex2f(-0.03,-0.15);
	glEnd();
	glFlush();
	glBegin(GL_LINES);
		glVertex2f(-0.03,-0.15);
		glVertex2f(0.15,-0.15);
	glEnd();
	glFlush();
	glBegin(GL_LINES);
	glVertex2f(-0.015,-0.075);
	glVertex2f(0.1,-0.075);
	glEnd();
	glFlush();
}
void initRendering()
{
glEnable(GL_DEPTH_TEST);
}
void drawScene()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();//Identity Matrix
drawMan();
```

```
//glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
drawTree();
glutSwapBuffers();
}
float rt=0.48;


void update(int value) {
if(flag==1)
{
manY += 0.01f;
if(manY>rt)
{
flag=0;
}
}
if(flag==0)
{
        cocoY-= 0.03f;
        if(cocoY<-1.2)
        flag=2;
}
if (flag==2)
{
manY -= 0.01f;
if(manY<=-0.7)
{
flag=5;
}
}
glutPostRedisplay();
glutTimerFunc(1, update, 5000);
}
int main(int argc,char** argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(500,500);
glutCreateWindow("Coconut tree climbing");
initRendering();
glutDisplayFunc(drawScene);
glutTimerFunc(30, update, 5000);
glutMainLoop();
return(0);
}
```

OUTPUT




Coconut tree climbing

Experiment No:16

Pacman

Aim: To illustrate the working of the game pacman.

Program

```
#include<stdlib.h>
#include<GL/glut.h>
float pacX = 0.0f;
float pacY = 0.0f;
static int flag=1;
void line(void)
{
	glColor3f(1.0,1.0,1.0);
	glBegin(GL_LINE_STRIP);
	glVertex2f(-.2,0);
	glVertex2f(-.2,-.2);
	glVertex2f(.2,-.2);
	glVertex2f(.2,.2);
	glVertex2f(-.4,.2);
	glVertex2f(-.4,-.4);
	glVertex2f(.4,-.4);
	glVertex2f(.4,-.1);
	glEnd();
	glColor3f(1.0,0.0,0.0);
	glBegin(GL_LINE_STRIP);
	glVertex2f(.4,.1);
	glVertex2f(.4,.4);
	glVertex2f(-.6,.4);
	glVertex2f(-.6,-.4);
	glEnd();
	glColor3f(0.0,1.0,0.0);
	glBegin(GL_LINE_STRIP);
	glVertex2f(-.6,-.5);
	glVertex2f(-.6,-.6);
	glVertex2f(.6,-.6);
	glVertex2f(.6,.6);
	glVertex2f(-.8,.6);
	glVertex2f(-.8,-.8);
	glVertex2f(.8,-.8);
	glVertex2f(.8,.8);
	glEnd();
	glFlush();
}
void drawBall(void)
{
line();
glColor3f(0.0, 1.0, 0.0);
```

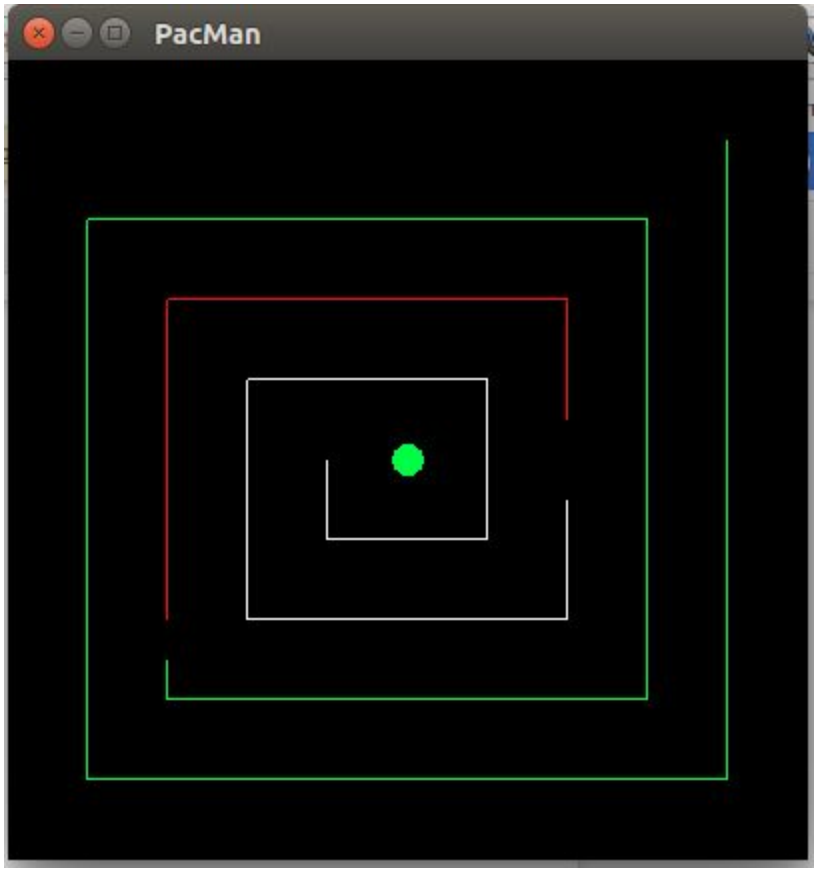
```
glTranslatef(pacX,pacY,0.0);
glutSolidSphere (0.04, 10, 10);
}

void keyPress(int key, int x, int y)
{
	if(key==GLUT_KEY_RIGHT)
	{
		if(pacX<0.15 && pacX>-0.15)
			pacX += 0.01f;
		/* left of Y axis */
		else if(pacX<-0.25 && pacX>-0.35)
			pacX += 0.01f;
		else if(pacX<-0.45 && pacX>-0.55)
			pacX += 0.01f;
		else if(pacX<-0.65 && pacX>-0.75)
			pacX += 0.01f;
		/* right of Y axis */
		else if(pacX<0.35 && pacX>0.25)
			pacX += 0.01f;
		else if(pacX<0.55 && pacX>0.45)
			pacX += 0.01f;
		else if(pacX<0.75 && pacX>0.65)
			pacX += 0.01f;
		/* above X axis */
		else if(pacX<0.15 && pacX>-0.35 && pacY>0.05)
			pacX += 0.01f;
		else if(pacX<0.35 && pacX>-0.55 && pacY>0.25 && pacY<0.35)
			pacX += 0.01f;
		else if(pacX<0.55 && pacX>-0.75 && pacY>0.45 && pacY<0.55)
			pacX += 0.01f;
		/* below X axis */
		else if(pacX<0.35 && pacX>-0.35 && pacY>-0.35 && pacY<-0.25)
			pacX += 0.01f;
		else if(pacX<0.55 && pacX>-0.55 && pacY>-0.55 && pacY<-0.45)
			pacX += 0.01f;
		else if(pacX<0.75 && pacX>-0.75 && pacY>-0.75 && pacY<-0.65)
			pacX += 0.01f;
		/* openings */
		if(pacX>-0.75 && pacX<-0.45 && pacY<-0.4 && pacY>-0.5)
			pacX += 0.01f;
		if(pacX>0.25 && pacX<0.55 && pacY<0.1 && pacY>-0.1)
			pacX += 0.01f;
	}
	if(key==GLUT_KEY_LEFT)
	{
		if(pacX<0.15 && pacX>-0.15)
			pacX -= 0.01f;
		/* left of Y axis */
		else if(pacX<-0.25 && pacX>-0.35)
			pacX -= 0.01f;
		else if(pacX<-0.45 && pacX>-0.55)
			pacX -= 0.01f;
		else if(pacX<-0.65 && pacX>-0.75)
```

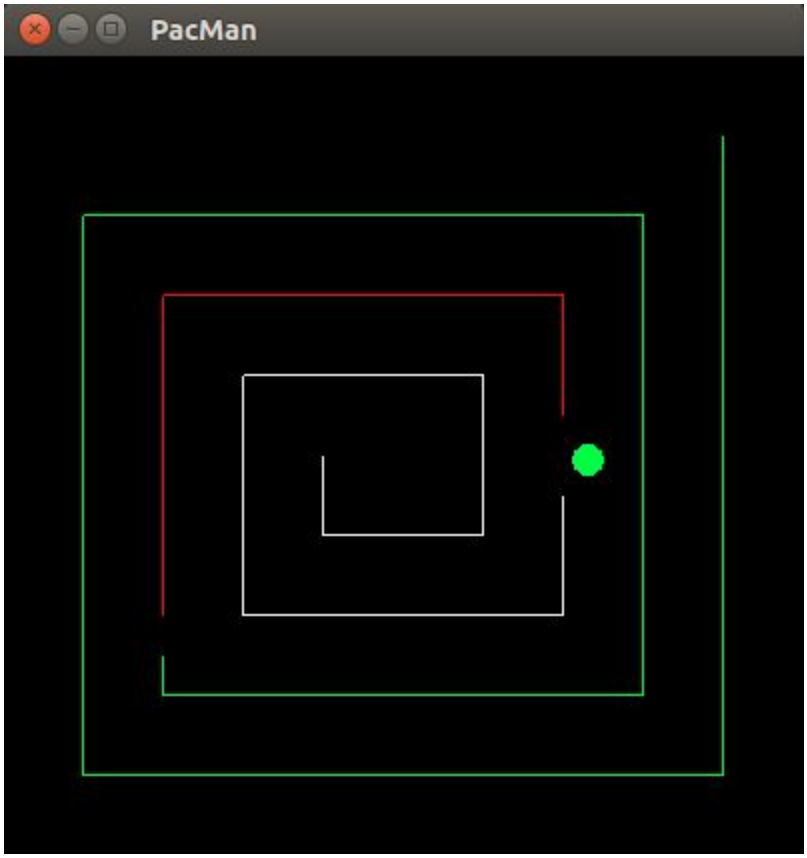
```
		pacX -= 0.01f;
	/* right of Y axis */
	else if(pacX<0.35 && pacX>0.25)
		pacX -= 0.01f;
	else if(pacX<0.55 && pacX>0.45)
		pacX -= 0.01f;
	else if(pacX<0.75 && pacX>0.65)
		pacX -= 0.01f;
	/* above X axis */
	else if(pacX<0.15 && pacX>-0.35 && pacY>0.05)
		pacX -= 0.01f;
	else if(pacX<0.35 && pacX>-0.55 && pacY>0.25 && pacY<0.35)
		pacX -= 0.01f;
	else if(pacX<0.55 && pacX>-0.75 && pacY>0.45 && pacY<0.55)
		pacX -= 0.01f;
	/* below X axis */
	else if(pacX<0.35 && pacX>-0.35 && pacY>-0.35 && pacY<-0.25)
		pacX -= 0.01f;
	else if(pacX<0.55 && pacX>-0.55 && pacY>-0.55 && pacY<-0.45)
		pacX -= 0.01f;
	else if(pacX<0.75 && pacX>-0.75 && pacY>-0.75 && pacY<-0.65)
		pacX -= 0.01f;
	/* openings */
	if(pacX>-0.75 && pacX<-0.45 && pacY<-0.4 && pacY>-0.5)
		pacX -= 0.01f;
	if(pacX>0.25 && pacX<0.55 && pacY<0.1 && pacY>-0.1)
		pacX -= 0.01f;
}
if(key==GLUT_KEY_DOWN)
{
	/* above X axis */
	if(pacY>-0.15 && pacX>-0.25 && pacX<0.15)
	pacY -= 0.01f;
	else if(pacY<0.35 && pacY>0.25)
	pacY -= 0.01f;
	else if(pacY<0.55 && pacY>0.45)
	pacY -= 0.01f;
	/* belove X axis */
	else if(pacY<-0.25 && pacY>-0.35)
	pacY -= 0.01f;
	else if(pacY<-0.45 && pacY>-0.55)
	pacY -= 0.01f;
	else if(pacY<-0.65 && pacY>-0.75)
	pacY -= 0.01f;
	/* left of Y axis */
	else if(pacY<0.15 && pacY>-0.35 && pacX>-0.35 && pacX<-0.25)
	pacY -= 0.01f;
	else if(pacY<0.35 && pacY>-0.55 && pacX>-0.55 && pacX<-0.45)
	pacY -= 0.01f;
	else if(pacY<0.55 && pacY>-0.75 && pacX>-0.75 && pacX<-0.65)
	pacY -= 0.01f;
	/* right of Y axis */
	else if(pacY<0.35 && pacY>-0.45 && pacX>0.25 && pacX<0.35)
	pacY -= 0.01f;
```

```
		else if(pacY<0.55 && pacY>-0.65 && pacX>0.45 && pacX<0.55)
		pacY -= 0.01f;
		else if(pacY<0.8 && pacY>-0.75 && pacX>0.65 && pacX<0.75)
		pacY -= 0.01f;
		/* openings */
		else if(pacY<-0.45 && pacY>-0.55 && pacX>-0.75 && pacX<-0.45)
		pacY -= 0.01f;
		else if(pacY<0.05 && pacY>-0.05 && pacX>0.25 && pacX<0.55)
		pacY -= 0.01f;

	}
	if(key==GLUT_KEY_UP)
	{
		/* above X axis */
		if(pacY<0.15)
		pacY += 0.01f;
		else if(pacY<0.35 && pacY>0.25)
		pacY += 0.01f;
		else if(pacY<0.55 && pacY>0.45)
		pacY += 0.01f;
		/* belove X axis */
		else if(pacY<-0.25 && pacY>-0.35)
		pacY += 0.01f;
		else if(pacY<-0.45 && pacY>-0.55)
		pacY += 0.01f;
		else if(pacY<-0.65 && pacY>-0.75)
		pacY += 0.01f;
		/* left of Y axis */
		else if(pacY<0.15 && pacY>-0.35 && pacX>-0.35 && pacX<-0.25)
		pacY += 0.01f;
		else if(pacY<0.35 && pacY>-0.55 && pacX>-0.55 && pacX<-0.45)
		pacY += 0.01f;
		else if(pacY<0.55 && pacY>-0.75 && pacX>-0.75 && pacX<-0.65)
		pacY += 0.01f;
		/* right of Y axis */
		else if(pacY<0.35 && pacY>-0.45 && pacX>0.25 && pacX<0.35)
		pacY += 0.01f;
		else if(pacY<0.55 && pacY>-0.65 && pacX>0.45 && pacX<0.55)
		pacY += 0.01f;
		else if(pacY<0.8 && pacY>-0.75 && pacX>0.65 && pacX<0.75)
		pacY += 0.01f;
		/* openings */
		else if(pacY<-0.45 && pacY>-0.55 && pacX>-0.75 && pacX<-0.45)
		pacY += 0.01f;
		else if(pacY<0.05 && pacY>-0.05 && pacX>0.25 && pacX<0.55)
		pacY += 0.01f;
	}
	glutPostRedisplay();
}
void initRendering()
{	glEnable(GL_DEPTH_TEST);}
void drawScene()
{
	glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
```

```
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        drawBall();
        glutSwapBuffers();
}
int main(int argc,char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowSize(400,400);
        glutCreateWindow("PacMan");
        initRendering();
        glutDisplayFunc(line);
        glutDisplayFunc(drawScene);
        glutSpecialFunc(keyPress);
        glutMainLoop();
        return(0);
}
```

OUTPUT




PacMan