**Practical 2**

Implement DDA and Bresenham line drawing algorithm to draw:

* i) Simple Line
* ii) Dotted Line
* iii) Dashed Line

Using mouse interface Divide the screen in four quadrants with center as (0, 0). The line should work for all the slopes positive as well as negative.

**Program Code:-**

#include<iostream>

#include<GL/glut.h>

using namespace std;

int Algo,type;

void Init()

{

glClearColor(0,0,0,0);

glColor3f(0,1,0);

gluOrtho2D(0,640,0,480);

glClear(GL\_COLOR\_BUFFER\_BIT);

}

int sign(float a){

if(a==0){

return 0;

}

if(a>0){

return 1;

}

return -1;

}

void B\_Line(int x\_1,int y\_1,int x\_2,int y\_2,int t){

float dy, dx, m , P;

dy = y\_2 - y\_1;

dx = x\_2 - x\_1;

m = dy/dx;

P = 2\*dy - dx;

int x = x\_1, y = y\_1;

cout<<"\n x1 = "<<x<<" y1 = "<<y;

if(m<1){

int cnt=1;

for(int i=0; i<=dx;i++){

if(t == 1){

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

if(t == 2){

if(i%2==0){

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

}

if(t == 3){

if(cnt <= 10){

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

cnt++;

if(cnt == 15){

cnt =1;

}

}

if(P<0){

x = x +1;

y =y;

P = P + 2\*dy;

}

else{

x= x+1;

y = y+1;

P = P + 2\*dy - 2\*dx;

}

}

}

else{

int cnt = 1;

for(int i=0;i<=dy;i++){

if(t == 1){

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

if(t == 2){

if(i%2==0){

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

}

if(t == 3){

if(cnt <= 10){

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

cnt++;

if(cnt == 15){

cnt =1;

}

}

if(P<0){

x = x;

y =y+1;

P = P + 2\*dx;

}

else{

x= x+1;

y = y+1;

P = P + 2\*dx - 2\*dy;

}

}

}

cout<<"\n xlast = "<<x<<" ylast = "<<y;

glFlush();

}

void DDA\_LINE(int x\_1,int y\_1,int x\_2,int y\_2, int t){

float dx,dy,length;

dx = x\_2-x\_1;

dy = y\_2-y\_1;

if(abs(dx) >= abs(dy)){

length = abs(dx);

}

else{

length = abs(dy);

}

float xin, yin;

xin = dx/length;

yin = dy/length;

float x,y;

x = x\_1 + 0.5 \* sign(xin);

y = y\_1 + 0.5 \* sign(yin);

int i=0;

int cnt =1;

while(i<=length){

if(t == 1){

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

if(t == 2){

if(i%2==0){

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

}

if(t == 3){

if(cnt <= 10){

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

}

cnt++;

if(cnt == 15){

cnt =1;

}

}

x = x + xin;

y = y + yin;

i++ ;

}

glFlush();

}

void display()

{

DDA\_LINE(0,240,640,240,1);

B\_Line(320,0,320,640,1);

glFlush();

}

void mymouse(int b,int s, int x, int y)

{

static int x\_s,y\_s,x\_e,y\_e,pt=0;

if(b==GLUT\_LEFT\_BUTTON && s==GLUT\_DOWN)

{

if(pt==0)

{

x\_s =x;

y\_s =480 - y;

pt++;

glBegin(GL\_POINTS);

glVertex2i(x\_s,y\_s);

glEnd();

}

else

{

x\_e=x;

y\_e=480-y;

cout<<"\n x\_1\_click "<<x\_s<<" y\_1\_click "<<y\_s;

cout<<"\n x\_2\_click "<<x\_e<<" y\_2\_click "<<y\_e<<"\n";

glBegin(GL\_POINTS);

glVertex2i(x\_e,y\_e);

glEnd();

if(Algo == 1){

DDA\_LINE(x\_s,y\_s,x\_e,y\_e,type);

}

if(Algo == 2){

B\_Line(x\_s,y\_s,x\_e,y\_e,type);

}

}

}

else if(b==GLUT\_RIGHT\_BUTTON && s==GLUT\_DOWN)

{

pt=0;

}

glFlush();

}

int main(int argc ,char \*\*argv)

{

cout<<"\n Select the Algorithm \n 1. DDA \n 2. Bresenham's \n";

cin>>Algo;

cout<<"Select the Line Type \n 1. Simple Line \n 2. Dotted Line\n 3. Dashed Line \n";

cin>>type;

if((Algo == 1 || Algo == 2 )&&(type==1 || type==2 || type==3)){

}

else{

cout<<"\n Option enter are wrong \n";

return 0;

}

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowPosition(100,100);

glutInitWindowSize(640,480);

glutCreateWindow("DDA-Line");

Init();

glutDisplayFunc(display);

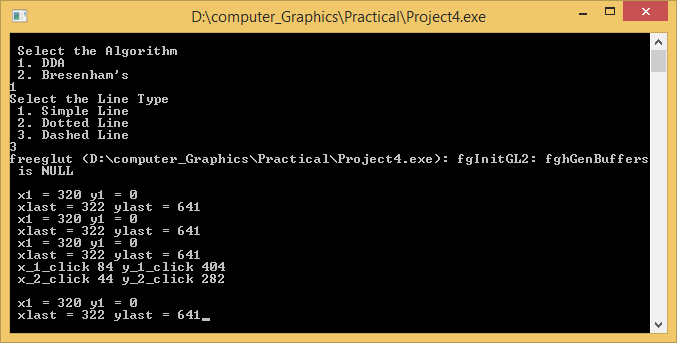
glutMouseFunc(mymouse);

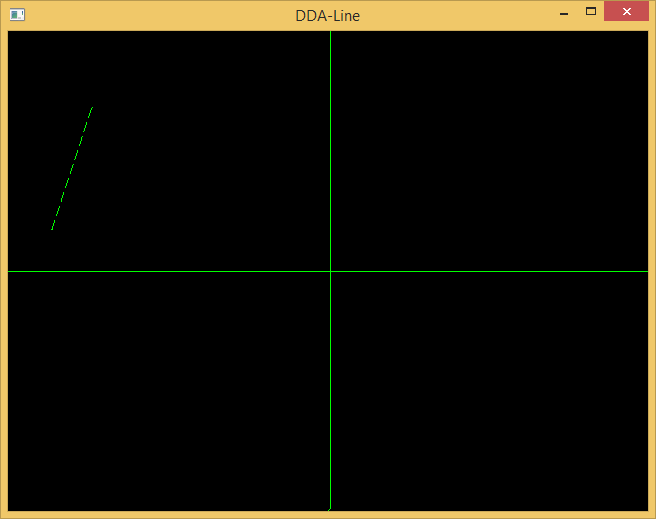
glutMainLoop();

return 0;

}

**Output:-**





**Bresenham’s line Program Code:-**

#include<windows.h>

#include<GL/glu.h>

#include<GL/glut.h>

#include<math.h>

void Draw()

{

GLfloat x1=350,y1=30,x2=350,y2=400;

GLfloat M,p,dx,dy,x,y,t;

glClear(GL\_COLOR\_BUFFER\_BIT);

if((x2-x1)==0)

M = (y2-y1);

else

M = (y2-y1)/(x2-x1);

if(fabs(M)<1)

{

if(x1>x2)

{

t = x1;

x1 = x2;

x2 = t;

t = y1;

y1 = y2;

y2 = t;

}

dx = fabs(x2-x1);

dy = fabs(y2-y1);

p = 2\*dy-dx;

x=x1;

y=y1;

glBegin(GL\_POINTS);

while(x<=x2)

{

glVertex2f(x,y);

x=x+1;

if(p>=0)

{

if(M<1)

y=y+1;

else

y=y-1;

p = p+2\*dy-2\*dx;

}

else

{

y=y;

p = p+2\*dy;

}

}

glEnd();

}

if(fabs(M)>=1)

{

if(y1>y2)

{

t = x1;

x1 = x2;

x2 = t;

t = y1;

y1 = y2;

y2 = t;

}

dx = fabs(x2-x1);

dy = fabs(y2-y1);

p = 2\*dx-dy;

x=x1;

y=y1;

glBegin(GL\_POINTS);

while(y<=y2)

{

glVertex2f(x,y);

y=y+1;

if(p>=0)

{

if(M>=1)

x=x+1;

else

x=x-1;

p = p+2\*dx-2\*dy;

}

else

{

x=x;

p = p+2\*dx;

}

}

glEnd();

}

glFlush();

}

void MyInit()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0,500,0,500);

glMatrixMode(GL\_MODELVIEW);

}

int main(int argC,char \*argV[])

{

glutInit(&argC,argV);

glutInitDisplayMode(GLUT\_RGB|GLUT\_SINGLE);

glutInitWindowPosition(0,0);

glutInitWindowSize(500,500);

glutCreateWindow("Brenham’s Line Drawing Algo");

MyInit();

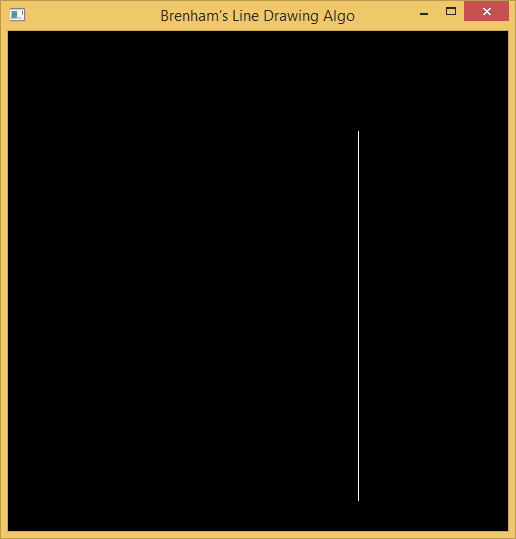
glutDisplayFunc(Draw);

glutMainLoop();

return 0;

}

**Output:-**

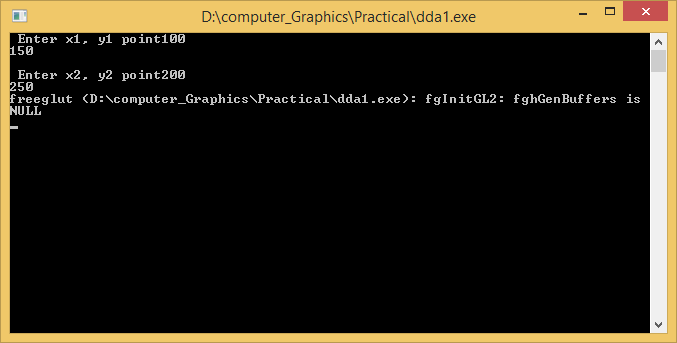
****

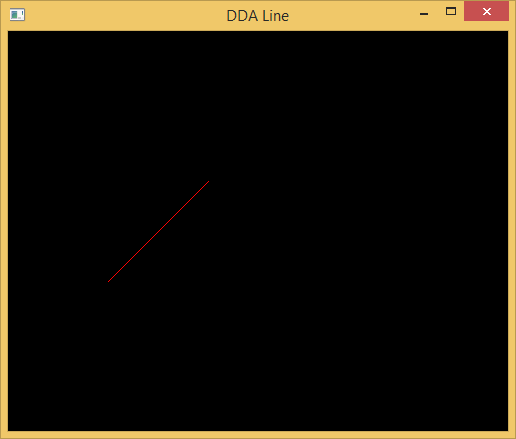
**DDA Line Program Code:-**

#include<GL/glut.h>

 #include<math.h>  
#include<iostream>  
using namespace std;  
  
float x\_1, x\_2, y\_1 ,y\_2;  
  
int sgn(float a){  
  
    if(a==0){  
      
        return 0;  
    }  
    if(a<0){  
      
        return -1;  
    }  
    else  
        return 1;  
      
}  
void Line(){  
    float dy,dx, length;  
    dy = y\_2 - y\_1;  
    dx = x\_2 - x\_1;  
    if(abs(dx)>=abs(dy)){  
      
        length = abs(dx);  
    }  
    else{  
        length = abs(dy);  
    }     
    float xin,yin;  
    xin = (x\_2-x\_1)/length;  
    yin = (y\_2-y\_1)/length;  
    float x,y;  
      
    x = x\_1 + 0.5 \* sgn(xin);  
    y = y\_1 + 0.5 \* sgn(yin);  
      
    int i = 0;  
    while(i<=length){  
       
        glBegin(GL\_POINTS);  
            glVertex2i(x,y);  
        glEnd();  
        x = x + xin;  
        y = y + yin;  
        i++;  
    }  
         
    glFlush();  
}  
  
void init(void)  
{  
        glClearColor(0,0,0,0);  
        glColor3f(1.0,0.0,0.0);  
        gluOrtho2D(0,500,0,400);  
        glClear(GL\_COLOR\_BUFFER\_BIT);  
}  
int main(int argc,char\*\* argv ){  
  
    cout<<" Enter x1, y1 point";  
    cin>>x\_1>>y\_1;  
    cout<<"\n Enter x2, y2 point";  
    cin>>x\_2>>y\_2;  
    glutInit(&argc, argv);  
    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);     
    glutInitWindowSize(500,400);  
    glutCreateWindow("DDA Line");  
    init();     
    glutDisplayFunc(Line);  
    glutMainLoop();  
    return 0;  
}

**Output:-**

****

****

**Practical 3**

Implement Bresenham circle drawing algorithm to draw any object. The object should be displayed in all the quadrants with respect to center and radius

**Program Code:-**

#include<GL/glut.h>

#include<iostream>

using namespace std;

int r;

void E\_way(int x, int y){

glBegin(GL\_POINTS);

glVertex2i(x+320,y+240);

glVertex2i(y+320,x+240);

glVertex2i(y+320, -x+240);

glVertex2i(x+320, -y+240);

glVertex2i(-x+320,-y+240);

glVertex2i(-y+320,-x+240);

glVertex2i(-y+320,x+240);

glVertex2i(-x+320,y+240);

glEnd();

glFlush();

}

void B\_circle(){

float d;

d = 3 - 2\*r;

int x,y;

x = 0 ;

y = r ;

do{

E\_way(x,y);

if(d<0){

d=d+4\*x+6;

}

else{

d= d+4\*(x-y)+10;

y=y-1;

}

x=x+1;

}while(x<y);

}

void init(){

glClearColor(1,1,1,0);

glColor3f(1,0,0);

gluOrtho2D(0,640,0,480);

glClear(GL\_COLOR\_BUFFER\_BIT);

}

int main(int argc, char \*\*argv){

cout<<"\n Enter Radius \t ";

cin>>r;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowPosition(100,100);

glutInitWindowSize(640,480);

glutCreateWindow("Circle");

init();

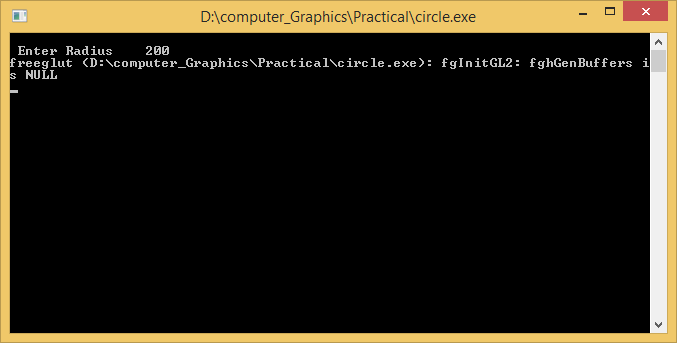
glutDisplayFunc(B\_circle);

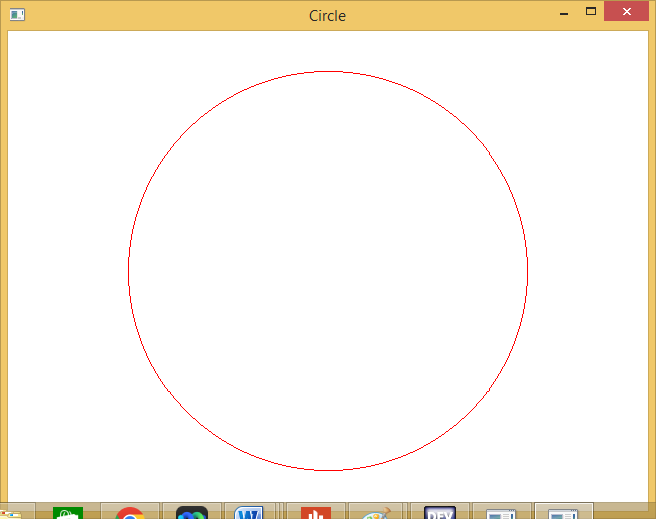
glutMainLoop();

return 0;

}

**Output:-**





**Practical 4**

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

**Program Code:-**

**1)Boundary Fill**

#include <iostream>

#include <math.h>

#include <time.h>

#include <GL/glut.h>

using namespace std;

void delay(float ms){

clock\_t goal = ms + clock();

while(goal>clock());

}

void init(){

glClearColor(1.0,1.0,1.0,0.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0,640,0,480);

}

void bound\_it(int x, int y, float\* fillColor, float\* bc){

float color[3];

glReadPixels(x,y,1.0,1.0,GL\_RGB,GL\_FLOAT,color);

if((color[0]!=bc[0] || color[1]!=bc[1] || color[2]!=bc[2])&&(

color[0]!=fillColor[0] || color[1]!=fillColor[1] || color[2]!=fillColor[2])){

glColor3f(fillColor[0],fillColor[1],fillColor[2]);

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

glFlush();

bound\_it(x+1,y,fillColor,bc);

bound\_it(x-2,y,fillColor,bc);

bound\_it(x,y+2,fillColor,bc);

bound\_it(x,y-2,fillColor,bc);

}

}

void mouse(int btn, int state, int x, int y){

y = 480-y;

if(btn==GLUT\_LEFT\_BUTTON)

{

if(state==GLUT\_DOWN)

{

float bCol[] = {1,0,0};

float color[] = {0,0,1};

bound\_it(x,y,color,bCol);

}

}

}

void world(){

glLineWidth(3);

glPointSize(2);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2i(150,100);

glVertex2i(300,300);

glVertex2i(450,100);

glEnd();

glFlush();

}

int main(int argc, char\*\* argv){

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(640,480);

glutInitWindowPosition(200,200);

glutCreateWindow("Polygon Boundary Fill");

glutDisplayFunc(world);

glutMouseFunc(mouse);

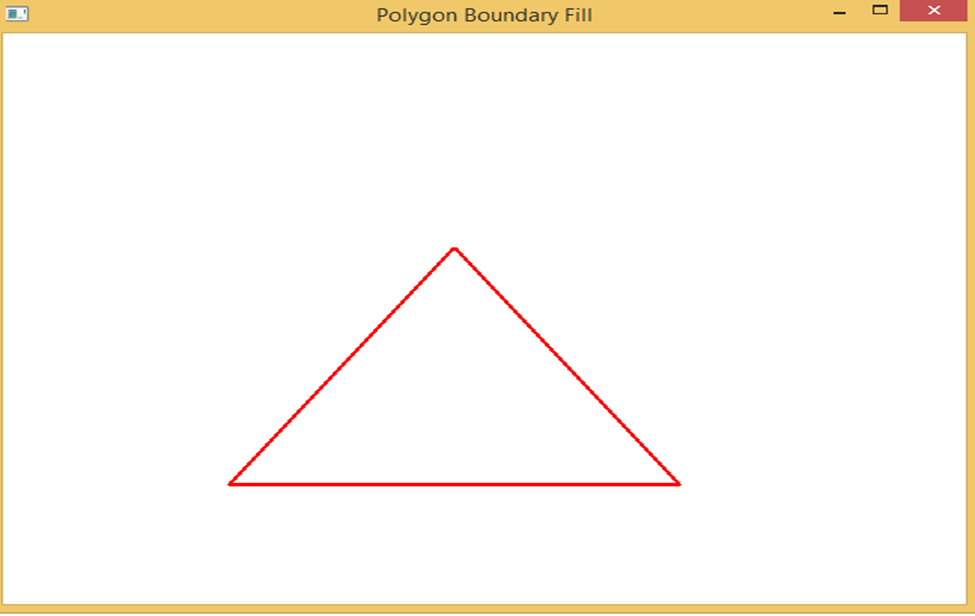
init();

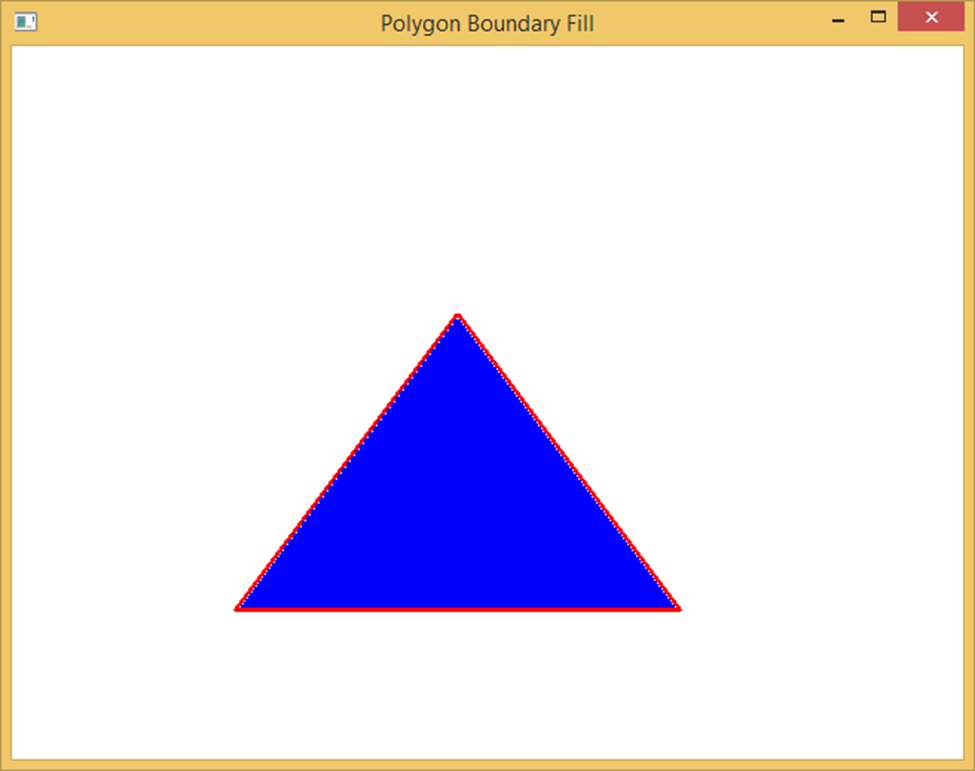
glutMainLoop();

return 0;

}

**Output:-**

****

****

**Practical 5**

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

**Program Code:-**

#include<windows.h>

#include<GL/glu.h>

#include<GL/glut.h>

GLfloat xMin=-0.5,xMax=0.5,yMin=-0.5,yMax=0.5;

GLfloat x1=-0.8,y1=-0.6,x2=0.7,y2=0.4;

int Left=1,Right=2,Bot=4,Top=8;

int C1,C2;

int Clip\_Flag = 0, Flag = 1;;

int Get\_Code(GLfloat x,GLfloat y)

{

int Code = 0;

if(x<xMin)

Code = Code | Left;

if(x>xMax)

Code = Code | Right;

if(y<yMin)

Code = Code | Bot;

if(y>yMax)

Code = Code | Top;

return Code;

}

void Clip()

{

int C;

GLfloat x,y;

if(C1)

C = C1;

else

C = C2;

if(C & Left)

{

x = xMin;

y = y1+(y2-y1)\*((xMin-x1)/(x2-x1));

}

if(C & Right)

{

x = xMax;

y = y1+(y2-y1)\*((xMax-x1)/(x2-x1));

}

if(C & Bot)

{

y = yMin;

x = x1+(x2-x1)\*((yMin-y1)/(y2-y1));

}

if(C & Top)

{

y = yMax;

x = x1+(x2-x1)\*((yMax-y1)/(y2-y1));

}

if(C == C1)

{

x1 = x;

y1 = y;

}

else

{

x2 = x;

y2 = y;

}

}

void Draw()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1,1,1);

glBegin(GL\_LINE\_LOOP);

glVertex2f(xMin,yMin);

glVertex2f(xMax,yMin);

glVertex2f(xMax,yMax);

glVertex2f(xMin,yMax);

glEnd();

glColor3f(1,0,0);

if(Flag == 1)

{

glBegin(GL\_LINES);

glVertex2f(x1,y1);

glVertex2f(x2,y2);

glEnd();

}

while(1 & Clip\_Flag == 1)

{

C1 = Get\_Code(x1,y1);

C2 = Get\_Code(x2,y2);

if((C1|C2) == 0)

break;

else if((C1&C2)!=0)

{

Flag = 0;

break;

}

else

Clip();

}

glFlush();

}

void Key(unsigned char ch,int x,int y)

{

Clip\_Flag = 1;

glutPostRedisplay();

}

int main(int argC,char \*argV[])

{

glutInit(&argC,argV);

glutInitWindowSize(500,500);

glutInitWindowPosition(100,100);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutCreateWindow("Cohen-Sutherland Algorithm");

glutDisplayFunc(Draw);

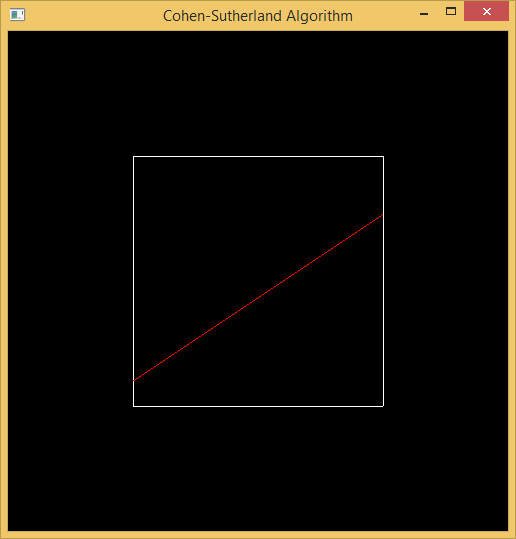
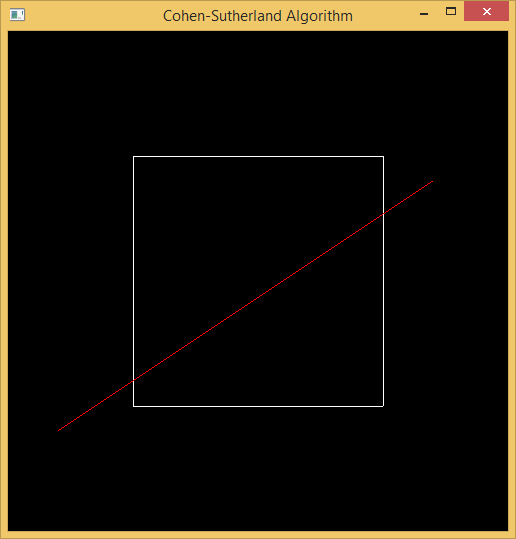
glutKeyboardFunc(Key);

glutMainLoop();

return 0;

}

**Output:-**

****

**Practical 6**

Implement following 2D transformations on the object with respect to axis : –

i) Scaling ii) Rotation about arbitrary point iii) Reflection

**Program Code:-**

#include <stdio.h>

#include <math.h>

#include <iostream>

#include <vector>

#include <GL/glut.h>

using namespace std;

int pntX1, pntY1, choice = 0, edges;

vector<int> pntX;

vector<int> pntY;

int transX, transY;

double scaleX, scaleY;

double angle, angleRad;

char reflectionAxis, shearingAxis;

int shearingX, shearingY;

double round(double d)

{

return floor(d + 0.5);

}

void drawPolygon()

{

glBegin(GL\_POLYGON);

glColor3f(1.0, 0.0, 0.0);

for (int i = 0; i < edges; i++)

{

glVertex2i(pntX[i], pntY[i]);

}

glEnd();

}

void drawPolygonTrans(int x, int y)

{

glBegin(GL\_POLYGON);

glColor3f(0.0, 1.0, 0.0);

for (int i = 0; i < edges; i++)

{

glVertex2i(pntX[i] + x, pntY[i] + y);

}

glEnd();

}

void drawPolygonScale(double x, double y)

{

glBegin(GL\_POLYGON);

glColor3f(0.0, 0.0, 1.0);

for (int i = 0; i < edges; i++)

{

glVertex2i(round(pntX[i] \* x), round(pntY[i] \* y));

}

glEnd();

}

void drawPolygonRotation(double angleRad)

{

glBegin(GL\_POLYGON);

glColor3f(0.0, 0.0, 1.0);

for (int i = 0; i < edges; i++)

{

glVertex2i(round((pntX[i] \* cos(angleRad)) - (pntY[i] \* sin(angleRad))), round((pntX[i] \* sin(angleRad)) + (pntY[i] \* cos(angleRad))));

}

glEnd();

}

void drawPolygonMirrorReflection(char reflectionAxis)

{

glBegin(GL\_POLYGON);

glColor3f(0.0, 0.0, 1.0);

if (reflectionAxis == 'x' || reflectionAxis == 'X')

{

for (int i = 0; i < edges; i++)

{

glVertex2i(round(pntX[i]), round(pntY[i] \* -1));

}

}

else if (reflectionAxis == 'y' || reflectionAxis == 'Y')

{

for (int i = 0; i < edges; i++)

{

glVertex2i(round(pntX[i] \* -1), round(pntY[i]));

}

}

glEnd();

}

void drawPolygonShearing()

{

glBegin(GL\_POLYGON);

glColor3f(0.0, 0.0, 1.0);

if (shearingAxis == 'x' || shearingAxis == 'X')

{

glVertex2i(pntX[0], pntY[0]);

glVertex2i(pntX[1] + shearingX, pntY[1]);

glVertex2i(pntX[2] + shearingX, pntY[2]);

glVertex2i(pntX[3], pntY[3]);

}

else if (shearingAxis == 'y' || shearingAxis == 'Y')

{

glVertex2i(pntX[0], pntY[0]);

glVertex2i(pntX[1], pntY[1]);

glVertex2i(pntX[2], pntY[2] + shearingY);

glVertex2i(pntX[3], pntY[3] + shearingY);

}

glEnd();

}

void myInit(void)

{

glClearColor(1.0, 1.0, 1.0, 0.0);

glColor3f(0.0f, 0.0f, 0.0f);

glPointSize(4.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-640.0, 640.0, -480.0, 480.0);

}

void myDisplay(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 0.0);

if (choice == 1)

{

drawPolygon();

drawPolygonTrans(transX, transY);

}

else if (choice == 2)

{

drawPolygon();

drawPolygonScale(scaleX, scaleY);

}

else if (choice == 3)

{

drawPolygon();

drawPolygonRotation(angleRad);

}

else if (choice == 4)

{

drawPolygon();

drawPolygonMirrorReflection(reflectionAxis);

}

else if (choice == 5)

{

drawPolygon();

drawPolygonShearing();

}

glFlush();

}

int main(int argc, char\*\* argv)

{

cout << "Enter your choice:\n\n" << endl;

cout << "1. Translation" << endl;

cout << "2. Scaling" << endl;

cout << "3. Rotation" << endl;

cout << "4. Mirror Reflection" << endl;

cout << "5. Shearing" << endl;

cout << "6. Exit" << endl;

cin >> choice;

if (choice == 6) {

return choice;

}

cout << "\n\nFor Polygon:\n" << endl;

cout << "Enter no of edges: "; cin >> edges;

for (int i = 0; i < edges; i++)

{

cout << "Enter co-ordinates for vertex " << i + 1 << " : "; cin >> pntX1 >> pntY1;

pntX.push\_back(pntX1);

pntY.push\_back(pntY1);

}

if (choice == 1)

{

cout << "Enter the translation factor for X and Y: "; cin >> transX >> transY;

}

else if (choice == 2)

{

cout << "Enter the scaling factor for X and Y: "; cin >> scaleX >> scaleY;

}

else if (choice == 3)

{

cout << "Enter the angle for rotation: "; cin >> angle;

angleRad = angle \* 3.1416 / 180;

}

else if (choice == 4)

{

cout << "Enter reflection axis ( x or y ): "; cin >> reflectionAxis;

}

else if (choice == 5)

{

cout << "Enter reflection axis ( x or y ): "; cin >> shearingAxis;

if (shearingAxis == 'x' || shearingAxis == 'X')

{

cout << "Enter the shearing factor for X: "; cin >> shearingX;

}

else

{

cout << "Enter the shearing factor for Y: "; cin >> shearingY;

}

}

//cout << "\n\nPoints:" << pntX[0] << ", " << pntY[0] << endl;

//cout << angleRad;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(640, 480);

glutInitWindowPosition(100, 150);

glutCreateWindow("Extended Basic Transformations");

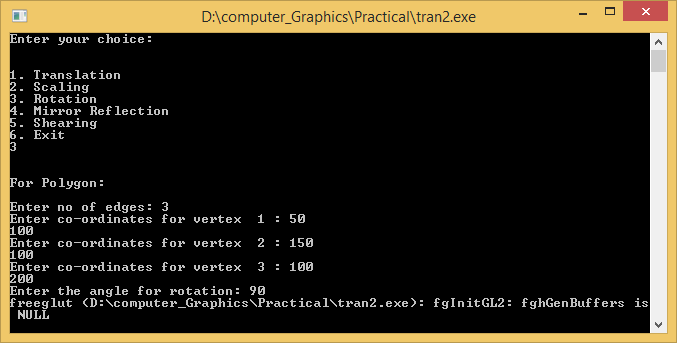
glutDisplayFunc(myDisplay);

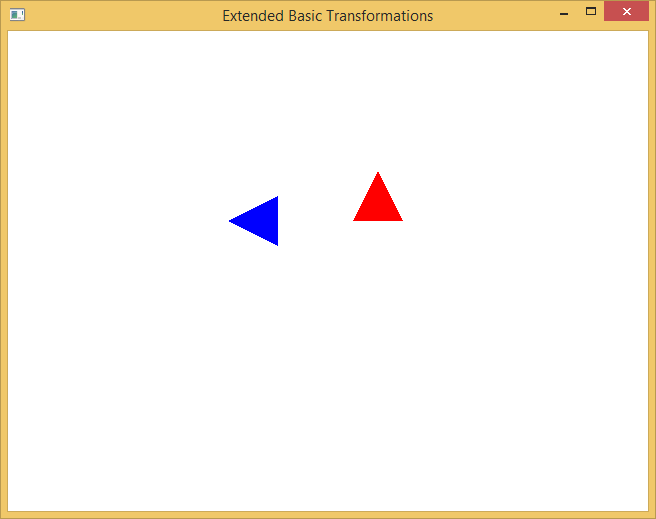
myInit();

glutMainLoop();

**}**

**Output:-**





**Practical 7**

Generate fractal patterns using i) Bezier ii) Koch Curve

**Program Code:-**

**Code for Koch Curve -**

#include <GL/glut.h>

#include <math.h>

GLfloat oldx=-0.7,oldy=0.5;

void drawkoch(GLfloat dir,GLfloat len,GLint iter) {

GLdouble dirRad = 0.0174533 \* dir;

GLfloat newX = oldx + len \* cos(dirRad);

GLfloat newY = oldy + len \* sin(dirRad);

if (iter==0) {

glVertex2f(oldx, oldy);

glVertex2f(newX, newY);

oldx = newX;

oldy = newY;

}

else {

iter--;

//draw the four parts of the side \_/\\_

drawkoch(dir, len, iter);

dir += 60.0;

drawkoch(dir, len, iter);

dir -= 120.0;

drawkoch(dir, len, iter);

dir += 60.0;

drawkoch(dir, len, iter);

}

}

void display(){

glClear( GL\_COLOR\_BUFFER\_BIT );

glBegin(GL\_LINES);

glColor3f(0.0, 1.0, 0.0);

drawkoch(0.0,0.04,3);

drawkoch(-120.0, 0.04, 3);

drawkoch(120.0,0.04,3);

glEnd();

glFlush();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(500,500);

glutInitWindowPosition(0,0);

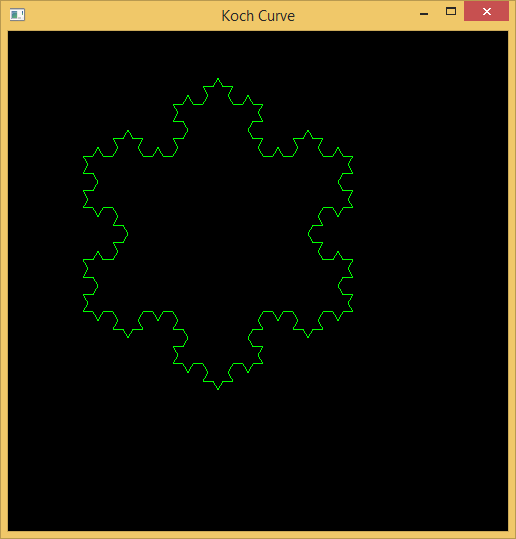
glutCreateWindow("Koch Curve");

glutDisplayFunc(display);

glutMainLoop();

}

**Output:-**

****

**Code for Bezier Curve -**

#include <iostream>

#include <stdlib.h>

#include <GL/glut.h>

#include <math.h>

using namespace std;

//Point class for taking the points

class Point {

public:

float x, y;

void setxy(float x2, float y2)

{

x = x2; y = y2;

}

//operator overloading for '=' sign

const Point & operator=(const Point &rPoint)

{

x = rPoint.x;

y = rPoint.y;

return \*this;

}

};

int factorial(int n)

{

if (n<=1)

return(1);

else

n=n\*factorial(n-1);

return n;

}

float binomial\_coff(float n,float k)

{

float ans;

ans = factorial(n) / (factorial(k)\*factorial(n-k));

return ans;

}

Point abc[20];

int SCREEN\_HEIGHT = 500;

int points = 0;

int clicks = 4;

void myInit() {

glClearColor(1.0,1.0,1.0,0.0);

glColor3f(0.0,0.0,0.0);

glPointSize(3);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0,640.0,0.0,500.0);

}

void drawDot(int x, int y) {

glBegin(GL\_POINTS);

glVertex2i(x,y);

glEnd();

glFlush();

}

void drawLine(Point p1, Point p2) {

glBegin(GL\_LINES);

glVertex2f(p1.x, p1.y);

glVertex2f(p2.x, p2.y);

glEnd();

glFlush();

}

//Calculate the bezier point

Point drawBezier(Point PT[], double t) {

Point P;

P.x = pow((1 - t), 3) \* PT[0].x + 3 \* t \* pow((1 -t), 2) \* PT[1].x + 3 \* (1-t) \* pow(t, 2)\* PT[2].x + pow (t, 3)\* PT[3].x;

P.y = pow((1 - t), 3) \* PT[0].y + 3 \* t \* pow((1 -t), 2) \* PT[1].y + 3 \* (1-t) \* pow(t, 2)\* PT[2].y + pow (t, 3)\* PT[3].y;

return P;

}

//Calculate the bezier point [generalized]

Point drawBezierGeneralized(Point PT[], double t) {

Point P;

P.x = 0; P.y = 0;

for (int i = 0; i<clicks; i++)

{

P.x = P.x + binomial\_coff((float)(clicks - 1), (float)i) \* pow(t, (double)i) \* pow((1 - t), (clicks - 1 - i)) \* PT[i].x;

P.y = P.y + binomial\_coff((float)(clicks - 1), (float)i) \* pow(t, (double)i) \* pow((1 - t), (clicks - 1 - i)) \* PT[i].y;

}

//cout<<P.x<<endl<<P.y;

//cout<<endl<<endl;

return P;

}

void myMouse(int button, int state, int x, int y) {

// If left button was clicked

if(button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) {

// Store where mouse was clicked, Y is backwards.

abc[points].setxy((float)x,(float)(SCREEN\_HEIGHT - y));

points++;

// Draw the red dot.

drawDot(x, SCREEN\_HEIGHT - y);

// If (click-amout) points are drawn do the curve.

if(points == clicks)

{

glColor3f(0.2,1.0,0.0);

// Drawing the control lines

for(int k=0;k<clicks-1;k++)

drawLine(abc[k], abc[k+1]);

Point p1 = abc[0];

/\* Draw each segment of the curve.Make t increment in smaller amounts for a more detailed curve.\*/

for(double t = 0.0;t <= 1.0; t += 0.02)

{

Point p2 = drawBezierGeneralized(abc,t);

cout<<p1.x<<" , "<<p1.y<<endl;

cout<<p2.x<<" , "<<p2.y<<endl;

cout<<endl;

drawLine(p1, p2);

p1 = p2;

}

glColor3f(0.0,0.0,0.0);

points = 0;

}

}

}

void myDisplay() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

}

int main(int argc, char \*argv[]) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(640,500);

glutInitWindowPosition(100,150);

glutCreateWindow("Bezier Curve");

glutMouseFunc(myMouse);

glutDisplayFunc(myDisplay);

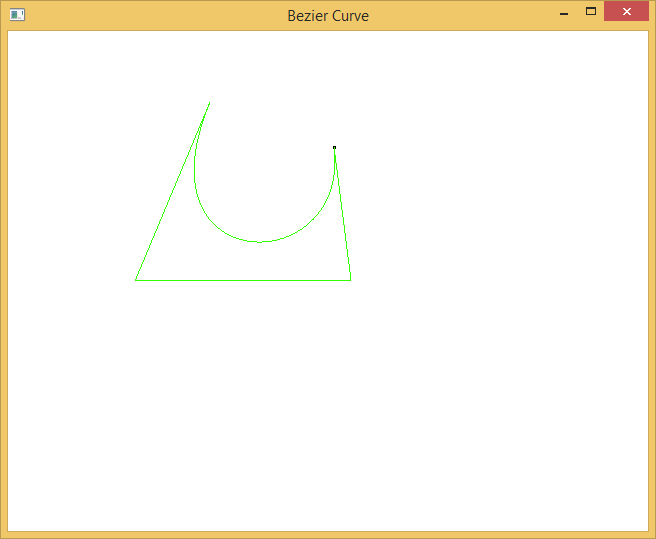
myInit();

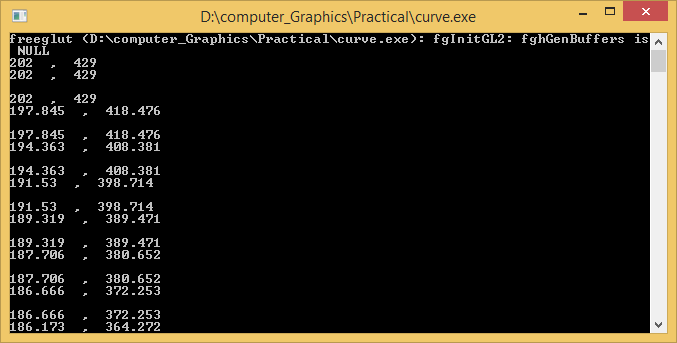
glutMainLoop();

return 0;

}

**Output:-**

****

****

**Practical 8**

Implement animation principles for any object.

**Program Code:-**

**Bouncing ball code:-**

#include <GL/glut.h>

#include <GL/gl.h>

#include <GL/glu.h>

#include <math.h>

#include <stdio.h>

#include <string.h>

#define WIDTH 640

#define HEIGHT 480

void reshape(int width, int height){

glViewport(0,0,width,height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-WIDTH/2,WIDTH/2-1,-HEIGHT/2,HEIGHT/2-1,-1,1);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void init(void){

glClearColor(0.0,0.0,0.0,1.0);

glPointSize(2.0);

}

void Timer(int ex)

{

glutPostRedisplay();

glutTimerFunc(30,Timer,0);

}

int k=20;

void circle(int p,int m)

{

int x=0,y,d,r;

r=m;

y=r;

d=3-2\*r;

while(x<=y)

{

glVertex2i(x,y+p);

glVertex2i(y,x+p);

glVertex2i(-x,y+p);

glVertex2i(-y,x+p);

glVertex2i(-x,-y+p);

glVertex2i(-y,-x+p);

glVertex2i(y,-x+p);

glVertex2i(x,-y+p);

if(d<0)

d=d+4\*x+6;

else

{

d=d+4\*(x-y)+10;

y--;

}

x++;

}

}

int r=100,flag=0;

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glColor4f(0.0,1.0,1.0,1.0);

glBegin(GL\_POINTS);

circle(k,r);

if(flag==0)

{

if((k+r)<=240)

{

k=k+10;

}

if((k+r)>=240){

flag=1;

}

}

if(flag==1)

{

k=k-10;

if((k-r)<=-240)

{

flag=0;

if(r!=10)

r=r-10;

}

}

glEnd();

glutSwapBuffers();

}

void idle(void){

/\* do nothing \*/

}

int main(int argc, char \*\*argv){

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH);

glutInitWindowPosition(0,0);

glutInitWindowSize(WIDTH,HEIGHT);

glutCreateWindow(argv[0]);

init();

glutIdleFunc(idle);

glutReshapeFunc(reshape);

glutDisplayFunc(display);

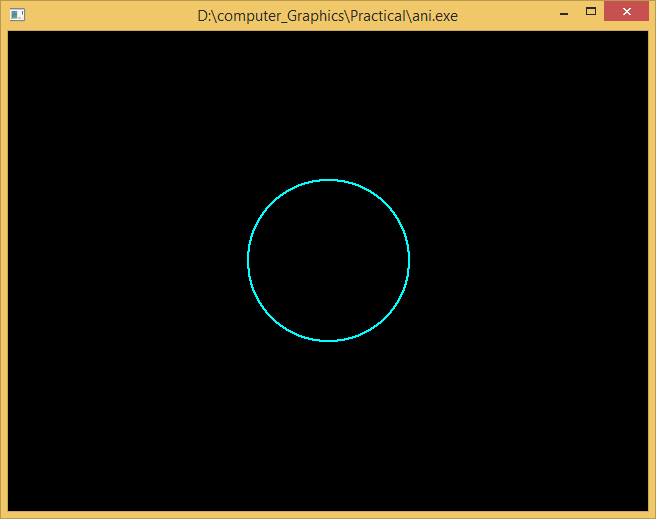
glutTimerFunc(0,Timer,0);

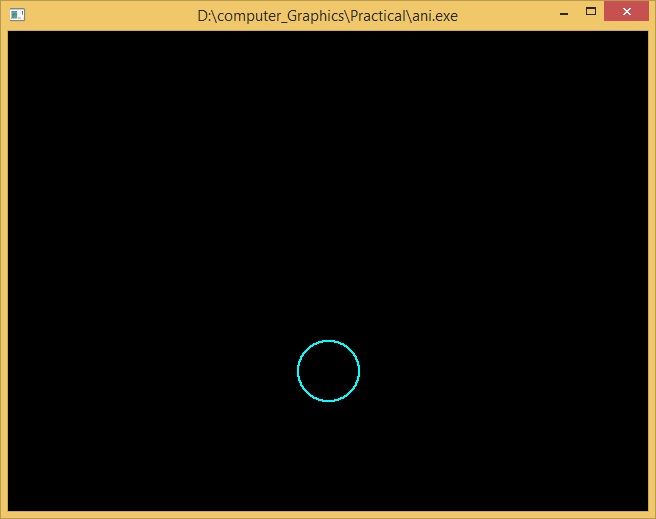
glutMainLoop();

return(1);

}

**Output:-**

****

****