UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No. 1 Study of Basic Output Primitives in C++ using OpenGL

Date: 19/7/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

a). To create an output window using OPENGL and to draw the following basic output primitives – POINTS, LINES, LINE_STRIP, LINE_LOOP, TRIANGLES, QUADS, QUAD_STRIP, POLYGON.

b) To create an output window and draw a checkerboard using OpenGL.

c) To create an output window and draw a house using POINTS,LINES,TRAINGLES and QUADS/POLYGON.

Code:

A) Shapes:

```
#include<GL/glut.h>
#include<stdio.h>
void myInit()
{
   glClearColor(0.0,0.0,0.0,1.0);
   gluOrtho2D(-1.0,1.0,-1.0,1.0);
}
void myDisplay()
{
   glClear(GL_COLOR_BUFFER_BIT);

   // 1) POINTS

   glColor3f(1.0,1.0,1.0);
   glBegin(GL_POINTS);
       glVertex2f(-0.1,-0.7);
   glEnd();

   // 2) LINES

   glColor3f(1.0,1.0,1.0);
   glBegin(GL_LINES);
       glVertex2f(-0.3,-0.3);
       glVertex2f(0.3,-0.3);
   glEnd();

   // 3) LINE_STRIP

   glColor3f(0.0,0.0,1.0);
```

```
glBegin(GL_LINE_STRIP);
    glVertex2f(-0.1,-0.3);
    glVertex2f(0.0,0.2);
    glVertex2f(0.2,0.5);
glEnd();

// 4) LINE_LOOP

glColor3f(1.0,0.0,0.0);
glBegin(GL_LINE_LOOP);
    glVertex2f(-0.1 - 0.15,-0.3);
    glVertex2f(0.0 - 0.15,0.2);
    glVertex2f(0.2 - 0.15,0.5);
glEnd();

// 5) TRIANGLES

glColor3f(0.3,0.7,0.4);
glBegin(GL_TRIANGLES);
    glVertex2f(-0.1 - 0.4,-0.3 + 0.5);
    glVertex2f(0.0 - 0.4,0.2 + 0.5);
    glVertex2f(0.2 - 0.4,0.5 + 0.5);
glEnd();

// 6) TRIANGLE STRIP

glColor3f(0.0,0.7,0.8);
glBegin(GL_TRIANGLE_STRIP);
    glVertex2f(-0.1 +0.4,-0.3 - 0.2);
    glVertex2f(0.0 +0.4,0.2 - 0.2);
    glVertex2f(0.2 +0.4,0.5 - 0.2);
    glVertex2f(0.4 +0.4, 0.6 - 0.2);
glEnd();

// 7) TRIANGLE FAN

glColor3f(0.7,0.1,0.8);
glBegin(GL_TRIANGLE_FAN);
    glVertex2f(-0.1 -0.75,-0.3 - 0.55);
    glVertex2f(0.0 -0.75,0.2 - 0.55);
    glVertex2f(0.2 -0.75,0.5 - 0.55);
    glVertex2f(0.4 -0.75, 0.6 - 0.55);
glEnd();

// 8) QUADS

glColor3f(0.1,0.1,0.8);
glBegin(GL_QUADS);
```

```
        glVertex2f(-0.1, -0.1 - 0.6);
        glVertex2f(-0.1, 0.1 - 0.6);
        glVertex2f(0.1, 0.1 - 0.6);
        glVertex2f(0.1, -0.1 - 0.6);
    glEnd();

    // 9) QUAD_STRIP

    glColor3f(0.6,0.1,0.5);
    glBegin(GL_QUAD_STRIP);
        glVertex2f(-0.1 + 0.5, -0.1 - 0.6);
        glVertex2f(-0.1 + 0.5, 0.1 - 0.6);
        glVertex2f(0.1 + 0.5, -0.1 - 0.6);
        glVertex2f(0.1 + 0.5, 0.1 - 0.6);

        glVertex2f(0.3 + 0.5, -0.1 - 0.6);
        glVertex2f(0.3 + 0.5, 0.2 - 0.6);

    glEnd();

    // 10) POLYGON

    glColor3f(0.6,0.1,0.5);
    glBegin(GL_POLYGON);

        glVertex2f(-0.1 + 0.5, -0.1 + 0.6);
        glVertex2f(-0.1 + 0.5, 0.1 + 0.6);
        glVertex2f(0.0 + 0.5, 0.2 + 0.6);
        glVertex2f(0.1 + 0.5, 0.1 + 0.6);
        glVertex2f(0.1 + 0.5, -0.1 + 0.6);

    glEnd();

    glFlush();
}
int main(int argc,char* argv[])
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(640,480);
    glutCreateWindow("check");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 1;
}
```

B) Checkerboard:

```c
#include <GL/glut.h>
void initGL() {

    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);

}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    GLint x;
    GLint y;
    GLint colorCode = 1;

    for(y = 50; y <= 350; y += 50) {

        for(x = 50; x <= 350; x += 50) {
            if (colorCode == 1) {
                glColor3f(0.0, 0.0, 0.0);
                colorCode = 0;
            }
            else {
                glColor3f(1.0, 1.0, 1.0);
                colorCode = 1;
            }
            glBegin(GL_QUADS);
            glVertex2i(x, y);
            glVertex2i(x, y + 50);
            glVertex2i(x + 50, y + 50);
            glVertex2i(x + 50, y);
            glEnd();

        }

    }

    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Understandable Checkers");
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(50, 50);
```

```
    glutDisplayFunc(display);
    initGL();
    glutMainLoop();
    return 0;
}
```

C) House:

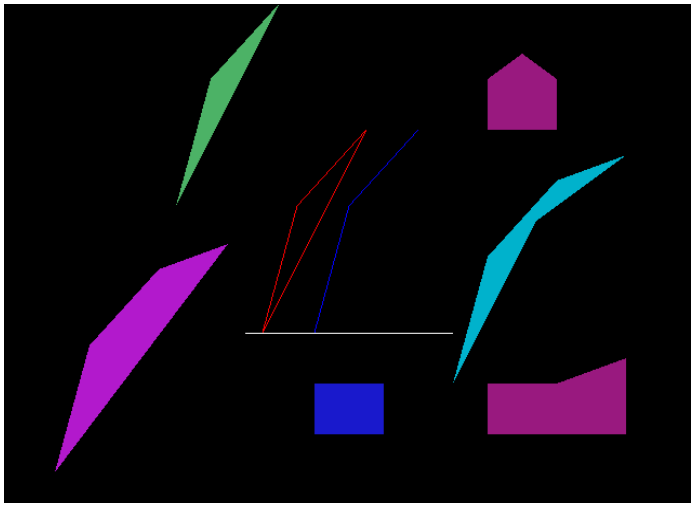
```
#include<GL/glut.h>
#include<stdio.h>
void myInit()
{
   glClearColor(0.0,0.0,0.0,1.0);
   gluOrtho2D(-1.0,1.0,-1.0,1.0);
}
void myDisplay()
{
   glClear(GL_COLOR_BUFFER_BIT);
   // glColor3f(1.0,0.0,0.0);
   // glBegin(GL_LINES);
   //     glVertex2f(-0.7,-0.7);
   //     glVertex2f(-0.1,-0.7);

   //     glVertex2f(-0.7,-0.7);
   //     glVertex2f(-0.7,-0.1);

   //     glVertex2f(-0.1,-0.7);
   //     glVertex2f(-0.1,-0.1);

   //     glVertex2f(-0.7,-0.1);
   //     glVertex2f(-0.1,-0.1);

   // glEnd();

   // glColor3f(0.0,1.0,0.0);
   // glBegin(GL_LINES);

   //     glVertex2f(-0.7,-0.1);
   //     glVertex2f(-0.1,-0.1);

   // glEnd();

   glColor3f(0.0,1.0,0.0);
   glBegin(GL_QUADS);

       glVertex2f(-0.7, -0.7);
       glVertex2f(-0.7, 0.3);
       glVertex2f(0.7, 0.3);
```

```
    glVertex2f(0.7, -0.7);

glEnd();

glColor3f(1.0,0.0,0.0);
glBegin(GL_TRIANGLES);

    glVertex2f(-0.7, 0.3);
    glVertex2f(0.7, 0.3);
    glVertex2f(0.0, 0.75);

glEnd();

// door
glColor3f(0.0,0.0,1.0);
glBegin(GL_QUADS);

    glVertex2f(-0.15, -0.7);
    glVertex2f(-0.15, -0.25);
    glVertex2f(0.15, -0.25);
    glVertex2f(0.15, -0.7);

glEnd();

// left window
glColor3f(0.0,0.5,0.5);
glBegin(GL_QUADS);

    glVertex2f(-0.3, -0.6);
    glVertex2f(-0.3, -0.4);
    glVertex2f(-0.5, -0.4);
    glVertex2f(-0.5, -0.6);

glEnd();

// right window
glColor3f(0.0,0.5,0.5);
glBegin(GL_QUADS);

    glVertex2f(0.3, -0.6);
    glVertex2f(0.3, -0.4);
    glVertex2f(0.5, -0.4);
    glVertex2f(0.5, -0.6);

glEnd();

glColor3f(0.0, 0.0, 0.0);
glBegin(GL_LINES);
```

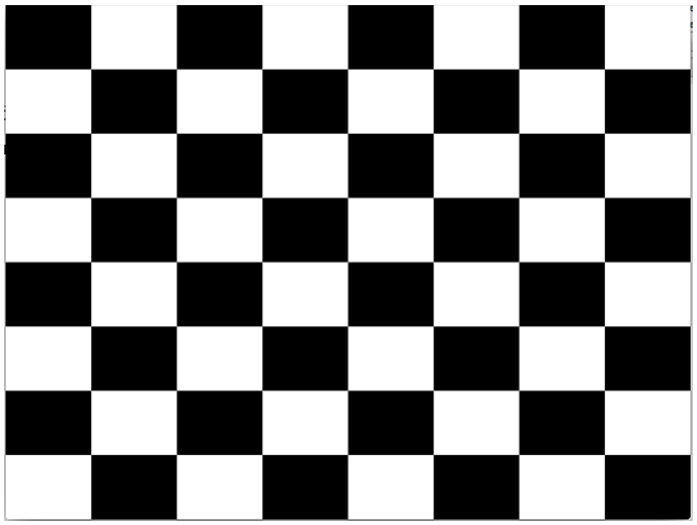
```
        glVertex2f(-0.7, -0.15);
        glVertex2f(0.7, -0.15);

    glEnd();

    // upper window

    glColor3f(0.5,0.5,0.0);
    glBegin(GL_QUADS);

        glVertex2f(-0.6, -0.05);
        glVertex2f(-0.6, 0.15);
        glVertex2f(0.6, 0.15);
        glVertex2f(0.6, -0.05);

    glEnd();

    // glBegin(GL_POINTS);
    //     glVertex2f(-0.1,-0.7);
    // glEnd();

    glFlush();
}
int main(int argc,char* argv[])
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(640,480);
    glutCreateWindow("check");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 1;
}
```

OUTPUTS:

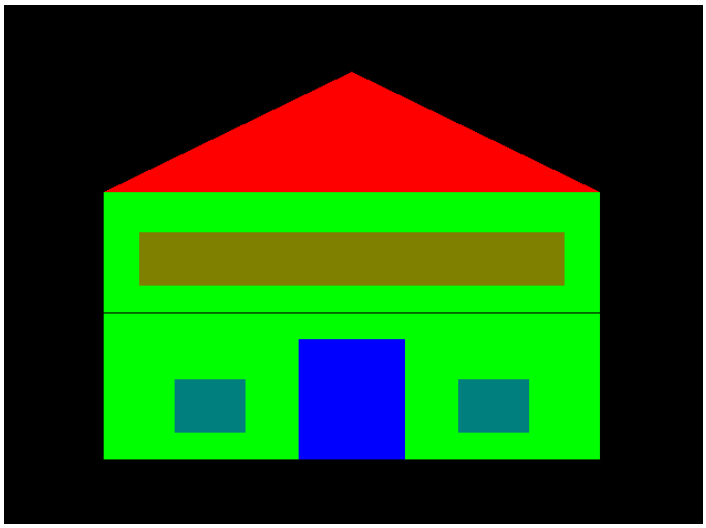
A)



B)



C)



UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No. 2 DDA Line Drawing Algorithm in C++ using OpenGL

Date: 26/7/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using DDA line drawing algorithm.

Case 1: +ve slope Left to Right line

Case 2: +ve slope Right to Left line

Case 3: -ve slope Left to Right line

Case 4: -ve slope Right to Left line Each case has two subdivisions (i) |m|<= 1 (ii) |m|>1

Note that all four cases of line drawing must be given as test cases.

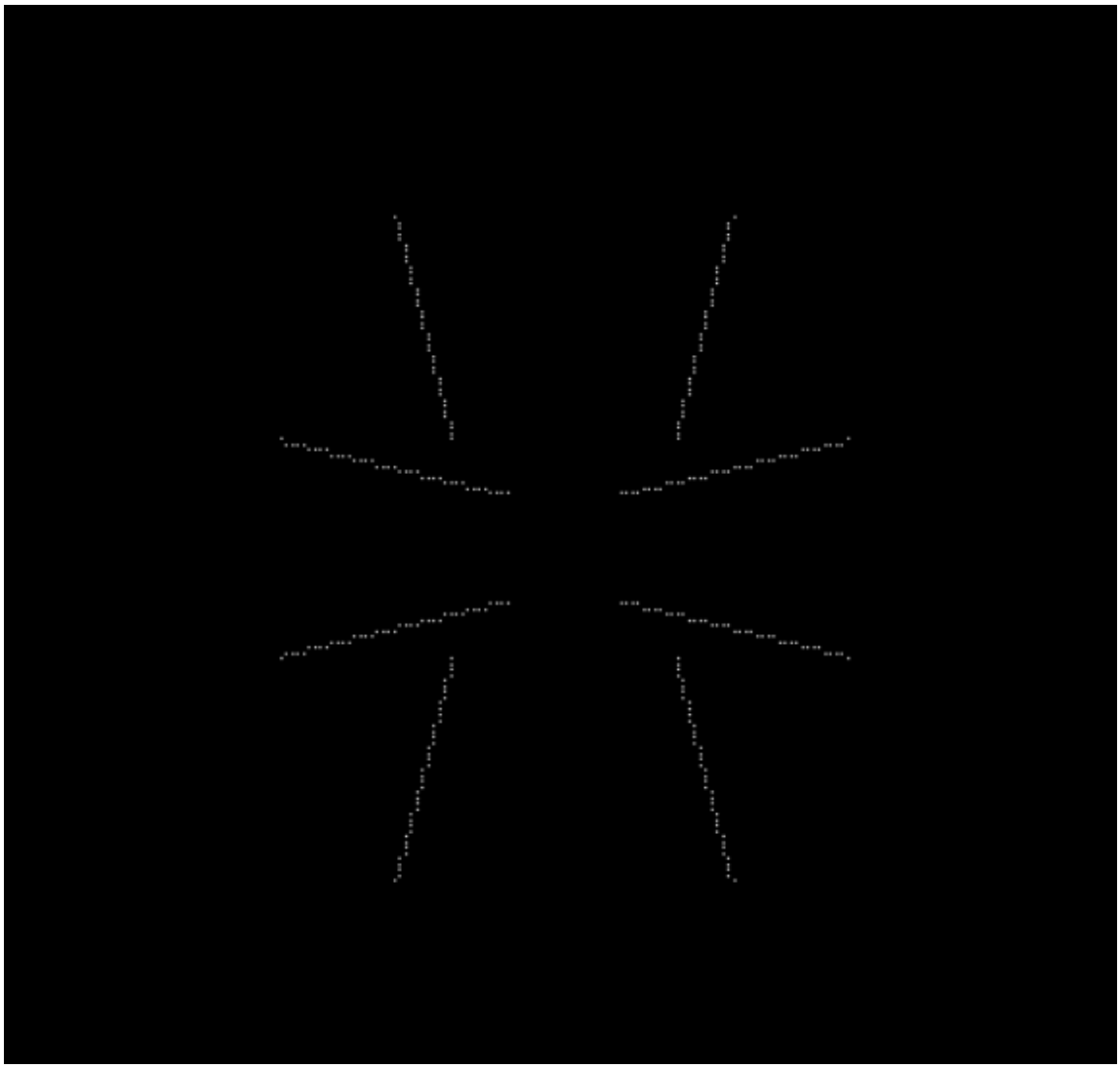
Code:

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

using namespace std;

float x1_arr, y1_arr, x2_arr, y2_arr;

void myInit()
{
glClearColor(1.0,1.0,1.0,0.0);
gluOrtho2D(-100,100,-100,100);
}

void myDisplay()
{
glClear(GL_COLOR_BUFFER_BIT);
float dy,dx,step,x,y,k,Xin,Yin;
float x1, y1, x2, y2;


x1 = x1_arr;
y1 = y1_arr;
x2 = x2_arr;
y2 = y2_arr;
```

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

if(abs(dx)> abs(dy))
{
step = abs(dx);
}
else
step = abs(dy);

Xin = dx/step;
Yin = dy/step;

x= x1;
y=y1;

glColor3f(0.0,0.0,0.0);
glBegin(GL_POINTS);
glVertex2f(x,y);
glEnd();

for (k=1 ;k<=step;k++)
{
x= x + Xin;
y= y + Yin;

glColor3f(0.0,0.0,0.0);
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}


glFlush();

}

int main(int argc,char* argv[])
{
int temp1,temp2,temp3,temp4;
cin>>temp1;
cin>>temp2;
cin>>temp3;
cin>>temp4;
x1_arr = temp1;
```

```
y1_arr = temp2;
x2_arr = temp3;
y2_arr = temp4;


glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100,100);
glutCreateWindow("check");
glutDisplayFunc(myDisplay);
myInit();
glutMainLoop();
return 1;
}
```

Output:



UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No. 3 Bresenham's Line Drawing Algorithm in C++ using OpenGL

Date: 2/8/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using Bresenham's line drawing algorithm.

Case 1: +ve slope Left to Right line

Case 2: +ve slope Right to Left line

Case 3: -ve slope Left to Right line

Case 4: -ve slope Right to Left line

Each case has two subdivisions (i) |m|<= 1 (ii) |m|>1

Note that all four cases of line drawing must be given as test cases.

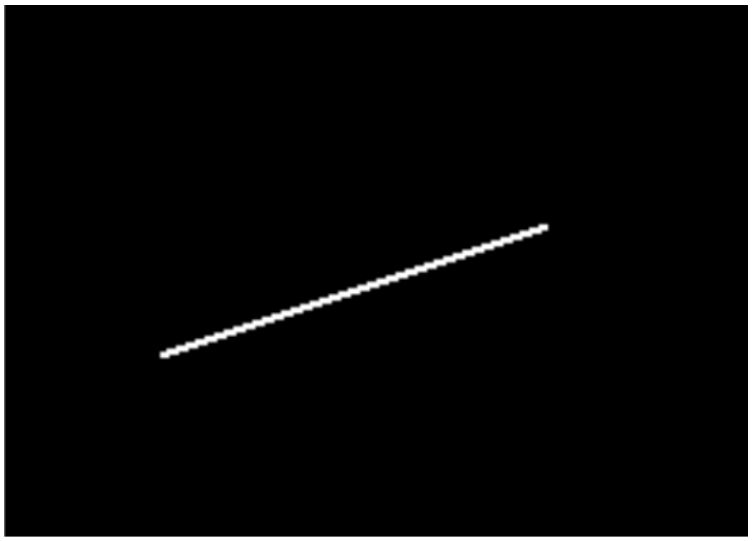
Code:

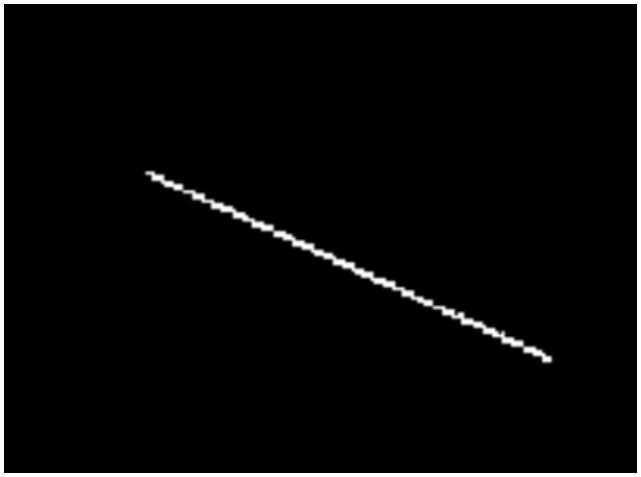
```
#include<GL/glut.h>
#include<stdio.h>
#include<iostream>

using namespace std;

float x1_arr[8], y1_arr[8], x2_arr[8], y2_arr[8];

void myInit()
{
glClearColor(0.0,0.0,0.0,1.0);
gluOrtho2D(-20,100,-20,100);
}

void myDisplay()
{
glClear(GL_COLOR_BUFFER_BIT);
float dy,dx,step,x,y,k,Xin,Yin,p;
float x1, y1, x2, y2;

for (int i = 0; i < 3; i++) {

x1 = x1_arr[i];
y1 = y1_arr[i];
x2 = x2_arr[i];
```

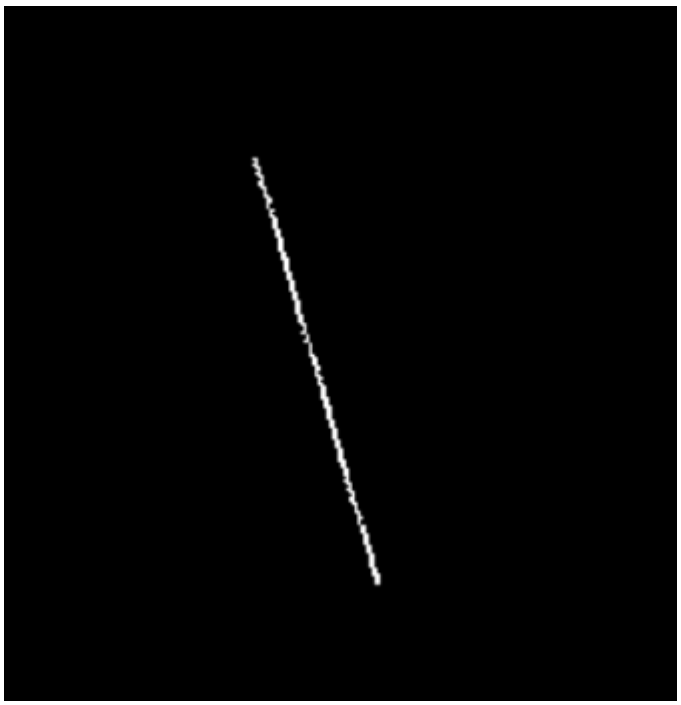
```
y2 = y2_arr[i];

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

        p=2dy-dx;

x= x1;
y=y1;

        while(x<=x2 && y <= y2)
        {
            if(p>=0)
            {
                glColor3f(1.0,1.0,1.0);
                glBegin(GL_POINTS);
                    glVertex2i(x,y);
                glEnd();

                y=y+1;
                p=p+2dy-2dx;
            }
            else
            {
                glColor3f(1.0,1.0,1.0);
                glBegin(GL_POINTS);
                    glVertex2i(x,y);
                glEnd();

                p=p+2dy;
                x=x+1;
            }

        }

        glColor3f(1.0,1.0,1.0);
        glBegin(GL_POINTS);
            glVertex2i(x,y);
        glEnd();

}

glFlush();

}

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

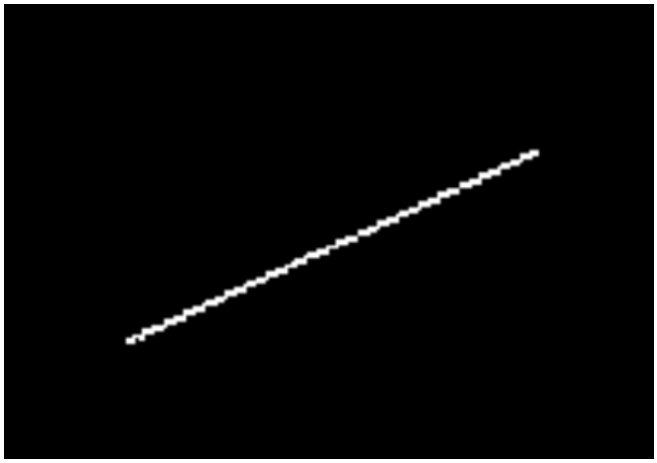
```
x1_arr[0] = 10;
y1_arr[0] = 10;
x2_arr[0] = 80;
y2_arr[0] = 10;

x1_arr[1] = 10;
y1_arr[1] = 70;
x2_arr[1] = 80;
y2_arr[1] = 70;

    x1_arr[2] = 45;
y1_arr[2] = 10;
x2_arr[2] = 45;
y2_arr[2] = 70;

glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100,100);
glutCreateWindow("check");
glutDisplayFunc(myDisplay);
myInit();
glutMainLoop();
return 1;
}
```

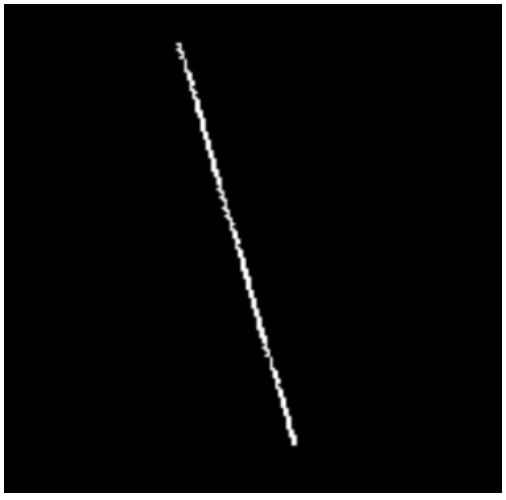
Output:











UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No. 4 Midpoint Circle Drawing Algorithm in C++ using OpenGL

Date: 9/8/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

a) To plot points that make up the circle with center (xc,yc) and radius r using Midpoint circle drawing algorithm. Give atleast 2 test cases. Case 1: With center (0,0) Case 2: With center (xc,yc)

b) To draw any object using line and circle drawing algorithms

Code:

a)

```
#include <stdlib.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

int xc, yc, r;

void myInit()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.4, 0.4, 0.9);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glPointSize(2);
    gluOrtho2D(-250.0, 250.0, -250.0, 250.0);
}

void plotAll(int x, int y, int xc, int yc)
{
    glVertex2d(x + xc, y + yc);
    glVertex2d(x + xc, -y + yc);
    glVertex2d(-x + xc, y + yc);
    glVertex2d(-x + xc, -y + yc);

    glVertex2d(y + xc, x + yc);
    glVertex2d(y + xc, -x + yc);
    glVertex2d(-y + xc, x + yc);
    glVertex2d(-y + xc, -x + yc);
}

void circle()
{
    int x = r, y = 0, pk = 1 - r;
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS);
    plotAll(x, y, xc, yc);
    while (x > y)
    {
        y++;
```

```
        if (pk < 0)
        {
            pk += (2 * y) + 1;
        }
        else
        {
            x--;
            pk += (2 * y) - (2 * x) + 1;
        }
        plotAll(x, y, xc, yc);
    }
    glEnd();
    glFlush();
}

int main(int argc, char* argv[])
{
    cout << "Enter coordinates of line center of circle and radius xc,yc,r: ";
    cin >> xc >> yc >> r;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Expt 04a - Mid Point Circle Algorithm");
    glutDisplayFunc(circle);
    myInit();
    glutMainLoop();
    return 1;
}
```

b)

```
#include <stdlib.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;

int xc, yc, r;

void myInit()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.4, 0.4, 0.9);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glPointSize(2);
    gluOrtho2D(-250.0, 250.0, -250.0, 250.0);
}

void plotAll(int x, int y, int xc, int yc)
{
    glVertex2d(x + xc, y + yc);
    glVertex2d(x + xc, -y + yc);
    glVertex2d(-x + xc, y + yc);
    glVertex2d(-x + xc, -y + yc);

    glVertex2d(y + xc, x + yc);
    glVertex2d(y + xc, -x + yc);
    glVertex2d(-y + xc, x + yc);
    glVertex2d(-y + xc, -x + yc);
}
```

```
void circle()
{
    int x = r, y = 0, pk = 1 - r;
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS);
    plotAll(x, y, xc, yc);
    while (x > y)
    {
        y++;

        if (pk < 0)
        {
            pk += (2 * y) + 1;
        }
        else
        {
            x--;
            pk += (2 * y) - (2 * x) + 1;
        }
        plotAll(x, y, xc, yc);
    }

    glVertex2d(100, 200);
    glVertex2d(50, 200);
    glVertex2d(20, 100);
    glVertex2d(40, 100);

    glEnd();
    glFlush();
}

int main(int argc, char* argv[])
{
    cout << "Enter coordinates of line center of circle and radius xc,yc,r: ";
    cin >> xc >> yc >> r;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Expt 04a - Mid Point Circle Algorithm");
    glutDisplayFunc(circle);
    myInit();
    glutMainLoop();
    return 1;
}
```

Output:

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No. 5 2D Transformations in C++ using OpenGL

Date: 16/8/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

To apply the following 2D transformations on objects and to render the final output along with the original object.

1) Translation

2) Rotation a) about origin b) with respect to a fixed point (xr,yr)

3) Scaling with respect to a) origin - Uniform Vs Differential Scaling b) fixed point (xf,yf)

4) Reflection with respect to a) x-axis b) y-axis c) origin d) the line x=y \

5) Shearing a) x-direction shear b) y-direction shear

Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants. (Use LINES primitive to draw x and y axis.

Code:

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <vector>
#include <gl/glut.h>
using namespace std;

int pntX1, pntY1, op = 0, edges; vector<int> pntX;
vector<int> pntY;
int transX, transY, lineX1, lineX2, lineY1, lineY2; double scaleX,
scaleY;
double angle, angleRad; char reflectionAxis;

int shearingX, shearingY;


double round(double d)
{
	return floor(d + 0.5);
}

void drawPolygon()
{
```

```
	glBegin(GL_POLYGON); glColor3f(0.48, 0, 0.7);
	for (int i = 0; i < edges; i++)
	{
		glVertex2i(pntX[i], pntY[i]);
	}
	glEnd();

	glBegin(GL_LINES); glVertex2d(lineX1, lineY1);
glVertex2d(lineX2, lineY2); glEnd();
}


void translate(int x, int y)
{
	glBegin(GL_POLYGON);

	glColor3f(0.08, 0.67, 0);
	for (int i = 0; i < edges; i++)
	{
		glVertex2i(pntX[i] + x, pntY[i] + y);
	}
	glEnd(); glBegin(GL_LINES);
	glVertex2d(lineX1 + x, lineY1 + y); glVertex2d(lineX2 + x,
lineY2 + y); glEnd();
}

void scale(double x, double y)
{
	glBegin(GL_POLYGON); glColor3f(0.08, 0.67, 0);
	for (int i = 0; i < edges; i++)
	{
		glVertex2i(round(pntX[i] * x) + 300, round(pntY[i] * y));
	}
	glEnd(); glBegin(GL_LINES);
	glVertex2d(round(lineX1 * x), round(lineY1 * y));
glVertex2d(round(lineX2 * x), round(lineY2 * y)); glEnd();
}

void rotate(double theta)
{
	glBegin(GL_POLYGON); glColor3f(0.08, 0.67, 0);
	for (int i = 0; i < edges; i++)
	{
		glVertex2i(round((pntX[i] * cos(theta)) - (pntY[i] *
sin(theta))), round((pntX[i] * sin(theta)) + (pntY[i] *
cos(theta))));
	}
	glEnd();
```

```
	glBegin(GL_LINES);
	glVertex2d(round((lineX1 * cos(theta)) - (lineY1 *
sin(theta))), round((lineX1 * sin(theta)) + (lineY1 * cos(theta))));
	glVertex2d(round((lineX2 * cos(theta)) - (lineY2 *
sin(theta))), round((lineX2 * sin(theta)) + (lineY2 * cos(theta))));

	glEnd();
}

void reflection(int option)
{
	if (option == 4) {
		glBegin(GL_LINES); glVertex2i(-640, -640);
		glVertex2i(640, 640); glEnd();
	}
	glBegin(GL_POLYGON); glColor3f(0.02, 0.72, 0.09);

	//X axis reflection

	if (option == 1)
	{
		for (int i = 0; i < edges; i++)
		{
			glVertex2i(round(pntX[i]), round(pntY[i] * -1));

		}
	}//Y axis reflection
	else if (option == 2)
	{
		for (int i = 0; i < edges; i++)
		{
			glVertex2i(round(pntX[i] * -1), round(pntY[i]));
		}
	}//origin reflection
	else if (option == 3) {
		for (int i = 0; i < edges; i++) {
			glVertex2i(round(pntX[i] * -1), round(pntY[i]) * -
1);
		}
	}//Y=X reflection
	else if (option == 4) {
		for (int i = 0; i < edges; i++) {
			glVertex2i(round(pntY[i]), round(pntX[i]));
		}
	}
	glEnd();
}

void shearing(int option)
```

```
{
	glBegin(GL_POLYGON); glColor3f(0.02, 0.72, 0.09);

	//translating the transformed polygon so that it doesn't
overlap on the original polygon
	if (option == 5)
	{
		glVertex2i(pntX[0] + 200, pntY[0]);

		glVertex2i(pntX[1] + shearingX + 200, pntY[1]);
glVertex2i(pntX[2] + shearingX + 200, pntY[2]);

		glVertex2i(pntX[3] + 200, pntY[3]);

	}
	else if (option == 6)
	{
		glVertex2i(pntX[0] + 200, pntY[0]); glVertex2i(pntX[1] +
200, pntY[1]);

		glVertex2i(pntX[2] + 200, pntY[2] + shearingY);
glVertex2i(pntX[3] + 200, 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)
{
	while (true) {
		glClear(GL_COLOR_BUFFER_BIT); glColor3f(0.0, 0.0, 0.0);
drawPolygon();
		cout << "1. Translation\n"; cout << "2. Scaling\n"; cout
<< "3. Rotation\n"; cout << "4. Exit\n";
		cout << "Enter your choice : "; cin >> op;
		if (op == 4) {
			break;
		}

		if (op == 1)
		{
```

```
                transX >> transY;

            }

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

            else if (op == 2)
            {


                >> scaleY;

            }

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

            else if (op == 3)
            {
                cout << "Enter the angle for rotation: "; cin >>
angle; angleRad = angle * 3.1416 / 180;
                rotate(angleRad);
            }
            glFlush();
      }

}

void main(int argc, char** argv)
{
      cout << "\nFor Polygon:\n" << endl;
      cout << "Enter no of edges: "; cin >> edges; cout << "\nEnter
Polygon Coordinates : \n";

      for (int i = 0; i < edges; i++) {
            cout << "Vertex " << i + 1 << " : "; cin >> pntX1 >>
pntY1; pntX.push_back(pntX1);
            pntY.push_back(pntY1);
      }

      cout << "\nEnter Line Coordinates : \n";
      cout << "Point 1 : "; cin >> lineX1 >> lineY1; cout << "Point
2 : "; cin >> lineX2 >> lineY2;

      glutInit(&argc, argv); glutInitDisplayMode(GLUT_SINGLE |
GLUT_RGB); glutInitWindowSize(640, 480);
```

```
	glutInitWindowPosition(100, 150);
glutCreateWindow("Transformations"); glutDisplayFunc(myDisplay);
myInit();
	glutMainLoop();

}
```

Output:

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No.6 2D Composite Transformations and Windowing in C++ using OpenGL

Date: 30/8/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

a) To compute the composite transformation matrix for any 2 transformations given as input by the user and applying it on the object. The transformation can be any combination of the following.

1) Translation

2) Rotation

3) Scaling

4) Reflection

5) Shearing Display the original and the transformed object. Calculate the final transformation matrix by multiplying the two individual transformation matrices and then apply it to the object.

Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants. (Use LINES primitive to draw x and y axis)

b) Create a window with any 2D object and a different sized viewport. Apply window to viewport transformation on the object. Display both window and viewport.

Code:

```
#include<GL/glut.h>
#include<iostream>
#include<cmath>
using namespace std;
int flag;
double a[3][1], b[3][1], c[3][1];
double a1[3][1], b1[3][1], c1[3][1];
double t1[3][3], t2[3][3];

void myInit() {
    glClearColor(1.0,1.0,1.0,0.0);
    glColor3f(0.0f,0.0f,0.0f);
    glPointSize(2);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,1920.0,0.0,1080.0);
```

```
}
void transformObject(){
    double m[3][3];
    memset(m, 0, sizeof(m));
    for(int i=0; i<3; i++){
    for(int j=0; j<3; j++)cout<<t1[i][j]<<' ';
        cout<<'\n';
    }
    cout<<'\n';
    for(int i=0; i<3; i++){
        for(int j=0; j<3; j++)cout<<t2[i][j]<<' ';
        cout<<'\n';
    }
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            for(int k=0;k<3;k++){
                m[i][j] += t2[i][k]*t1[k][j];
            }
        }
    }
    memset(a1, 0, sizeof(a1));
    memset(b1, 0, sizeof(b1));
    memset(c1, 0, sizeof(c1));
    for(int i=0;i<3;i++){
        for(int j=0;j<1;j++){
            for(int k=0;k<3;k++){
                a1[i][j]+= m[i][k]*a[k][j];
                b1[i][j]+= m[i][k]*b[k][j];
                c1[i][j]+= m[i][k]*c[k][j];
                }
            }
        }
    }
void assignTM(double tm[3][3]){
    if(flag == 0){
        for(int i=0;i<3;i++)for(int j=0;j<3;j++)t1[i][j] = tm[i][j];
        flag = 1;
    }
    for(int i=0;i<3;i++)for(int j=0;j<3;j++)t2[i][j] = tm[i][j];
}
void drawTriangle(double a[3][1], double b[3][1], double c[3][1], int s=0) {
    a[0][0] += 960;
    b[0][0] += 960;
    c[0][0] += 960;
    a[1][0] += 540;
    b[1][0] += 540;
    c[1][0] += 540;
    glBegin(GL_TRIANGLES);
```

```
    if(s)
        glColor4f(0.1f, 0.3f, 0.9f, 0.5f);
    else
        glColor4f(0.9f, 0.2f, 0.1f, 0.5f);

    glVertex2d(a[0][0],a[1][0]);
    glVertex2d(b[0][0],b[1][0]);
    glVertex2d(c[0][0],c[1][0]);
    glEnd();
}
void translateObject(double tx, double ty){
    double tm[3][3];
    memset(tm, 0, sizeof(tm));
    tm[0][0] = tm[1][1] = tm[2][2] = 1;
    tm[0][2] = tx;
    tm[1][2] = ty;
    assignTM(tm);
}
void rotateObject(double deg, int ch){
    int tx = 0, ty = 0;
    if(ch == 2){
        cout<<"Enter pivot: ";
        cin>>tx>>ty;
    }
    deg = (3.14)*deg/180;
    double tm[3][3];
    memset(tm, 0, sizeof(tm));
    tm[2][2] = 1;
    tm[0][0] = cos(deg);tm[1][1] = tm[0][0];
    tm[1][0] = sin(deg);tm[0][1] = -tm[1][0];
    tm[0][2] = tx*(double)(1.0-cos(deg)) + ty*sin(deg);
    tm[1][2] = ty*(double)(1.0-cos(deg)) - tx*sin(deg);
    assignTM(tm);
}
void scaleObject(double sx, double sy, int ch){
    int tx = 0, ty = 0;
    if(ch == 2){
        cout<<"Enter pivot: ";
        cin>>tx>>ty;
    }
    double tm[3][3];
    memset(tm, 0, sizeof(tm));
    tm[2][2] = 1;
    tm[0][0] = sx;tm[1][1] = sy;
    tm[0][2] = tx*(double)(1.0-sx);
    tm[1][2] = ty*(double)(1.0-sy);
    assignTM(tm);
}
```

```
void reflectObject(int ch){
    double tm[3][3];
    memset(tm, 0, sizeof(tm));
    tm[2][2] = 1;
    switch(ch){
        case 1: tm[0][0] = 1, tm[1][1] = -1;
        break;
        case 2: tm[0][0] = -1, tm[1][1] = 1;
        break;
        case 3: tm[0][0] = -1, tm[1][1] = -1;
        break;
        case 4: tm[0][1] = 1, tm[1][0] = 1;
        break;
    }
    assignTM(tm);
}
void shearObject(double s, int x, int ch){
    double tm[3][3];
    memset(tm, 0, sizeof(tm));
    tm[0][0] = tm[1][1] = tm[2][2] = 1;
    double ref;
    if(ch==2){
        cout<<"Enter reference line: ";
        cin>>ref;
    }
    if(x)tm[0][1] = s, tm[0][2] = -s*ref;
    else tm[1][0] = s, tm[1][2] = -s*ref;
    assignTM(tm);
}
void chooseTransformation(int choice) {
    int ch;
    switch(choice){
        case 1: double tx, ty;
            cout<<"Enter translation vector: ";
            cin>>tx>>ty;
            translateObject(tx, ty);
            break;
        case 2:
            double deg;cout<<"Enter angle: ";cin>>deg;
            cout<<"Options:\n\t1. Origin\n\t2. Pivot\n\tChoice: ";
            cin>>ch;
            rotateObject(deg, ch);
            break;
        case 3:
            double sx, sy;cout<<"Enter scaling factor: ";cin>>sx>>sy;
            cout<<"Options:\n\t1. Origin\n\t2. Pivot\n\tChoice: ";
            cin>>ch;
            scaleObject(sx, sy, ch);
```

```
            break;
        case 4:
            cout<<"Options:\n\t1. X-axis\n\t2. Y-axis\n\t3. Origin\n\t4.
X=Y\n\tChoice: ";
            cin>>ch;
            reflectObject(ch);
            break;
        case 5:
            double s;cout<<"Enter shear parameter: ";cin>>s;
            int x;cout<<"Enter axis:(1-x, 0-y) ";cin>>x;
            cout<<"Options:\n\t1. Origin\n\t2. Different reference
line\n\tChoice: ";
            cin>>ch;
            shearObject(s, x, ch);
    }
}
void myDisplay(){
    glColor3f(1, 1, 1);
    glBegin(GL_LINES);
    glVertex2d(0,540);
    glVertex2d(1920,540);
    glVertex2d(960,0);
    glVertex2d(960,1080);
    glEnd();
    drawTriangle(a1, b1, c1, 1);
    drawTriangle(a, b, c);
    glFlush();
}
int main(int argc, char** argv) {
    memset(a, 0, sizeof(a));
    memset(b, 0, sizeof(b));
    memset(c, 0, sizeof(c));
    memset(t1, 0, sizeof(t1));
    memset(t2, 0, sizeof(t2));
    cout<<"Enter coordinates of triangle: ";
    cin>>a[0][0]>>a[1][0]>>b[0][0]>>b[1][0]>>c[0][0]>>c[1][0];
    int choice;
    a[2][0] = b[2][0] = c[2][0] = 1;
    cout<<"Select two transformations\n\n";
    cout<<"Transformation 1:\n\t1. Translation\n\t2. Rotation\n\t3.
Scaling\n\t4.Reflection\n\t5. Shearing\n\tChoice: ";
    cin>>choice;
    chooseTransformation(choice);
    cout<<"\nTransformation 2:\n\t1. Translation\n\t2. Rotation\n\t3.
Scaling\n\t4.Reflection\n\t5. Shearing\n\tChoice: ";
    cin>>choice;
    chooseTransformation(choice);
    transformObject();
```

```
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Composite Transformations");
    myInit();
    glutDisplayFunc(myDisplay);
    glutMainLoop();
    return 0;
}
```

Outputs:




UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No.7 Cohen Sutherland Line clipping in C++ using OpenGL

Date: 6/9/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

Apply Cohen Sutherland line clipping on a line (x1,y1) (x2,y2) with respect to a clipping window (XWmin,YWmin) (XWmax,YWmax).

After clipping with respect to an edge, display the line segment with the calculated intermediate intersection points and the vertex list.

Input: The clipping window co-ordinates and the line endpoints

Note: The output should show the clipping window and the line to be clipped in different colors. You can show the intermediate steps using time delay.

Code:

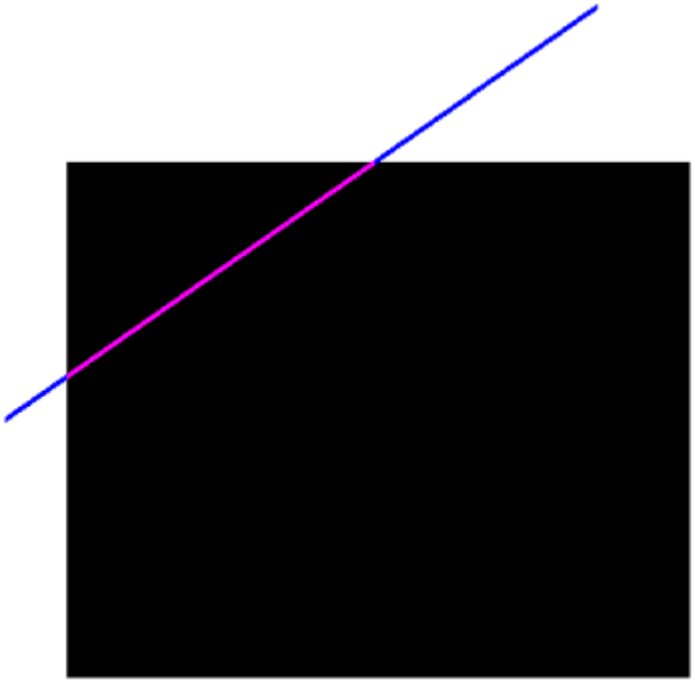
```
#include <bits/stdc++.h>
#include <GL/glut.h>
using namespace std;
using ld = long double;
const int WINDOW_WIDTH = 850;
const int WINDOW_HEIGHT = 700;
void myInit();
void myDisplay();
void printCohen();
const ld PADDING = 250;
const ld STEP = 1;
const ld SCALE = 5;
const int xmin = -50;
const int xmax = 50;
const int ymin = -50;
const int ymax = 50;
double x1 = -54.0;
double y11 = 60.0;
double x2 = 40.0;
double y2 = 55.0;
int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
printf("COHEN SUTHERLAND CLIPPING ALGORITHM\n Window:xmin,ymin =
(-50,-50) xmax,ymax = (50,50)\n");
printf("Enter the value of x1 coord:");
```

```
scanf("%lf",&x1);
printf("Enter the value of y1 coord : ");
scanf("%lf",&y11);
printf("Enter the value x2 coord : ");
scanf("%lf",&x2);
printf("Enter the value y2 coord : ");
scanf("%lf",&y2);
glutCreateWindow("Cohen's Algorithm");
glutDisplayFunc(myDisplay);
myInit();
glutMainLoop();
return 1;
}
void myInit()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPointSize(15.0);
    glLineWidth(4);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-100.0, 100, -100, 100);
}
void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0f, 0.0f, 0.0f);
    glRectf(xmin, ymin, xmax, ymax);
    glColor3f(0, 0, 1);
    glBegin(GL_LINES);
    glVertex2f(x1, y11);
    glVertex2f(x2, y2);
    glEnd();
    printCohen();
    glFlush();
}
char sign(int x)
{
    if (x <= 0)
        return '0';
    else
        return '1';
}
string codeGen(double x, double y)
{
    string code = "";
    code += sign(y - ymax);
    code += sign(ymin - y);
```

```
    code += sign(x - xmax);
    code += sign(xmin - x);
    return code;
}
pair<double, double> CohenPoint(int x, int y, string code, double m)
{
    pair<double, double> point;
    point.first = x;
    point.second = y;
    double xclip, yclip;
    if (code[0] == '1')
    {
        xclip = x + (ymax - y) / m;
        if (xclip <= xmax && xclip >= xmin)
        {
            point.first = xclip;
            point.second = ymax;
        }
    }
    if (code[1] == '1')
    {
        xclip = x + (ymin - y) / m;
        if (xclip <= xmax && xclip >= xmin)
        {
            point.first = xclip;
            point.second = ymin;
        }
    }
    if (code[2] == '1')
    {
        yclip = y + m * (xmax - x);
        if (yclip <= ymax && yclip >= ymin)
        {
            point.first = xmax;
            point.second = yclip;
        }
    }
    if (code[3] == '1')
    {
        yclip = y + m * (xmin - x);
        if (yclip <= ymax && yclip >= ymin)
        {
            point.second = yclip;
            point.first = xmin;
        }
    }
    cout << point.first << " " << point.second << " ";
    return point;
```

```
}
void printCohen()
{
    string code1, code2;
    pair<double, double> p1, p2;
    code1 = codeGen(x1, y11);
    code2 = codeGen(x2, y2);
    cout << code1 << " " << code2;
    int code_int1, code_int2;
    code_int1 = stoi(code1, 0, 2);
    code_int2 = stoi(code2, 0, 2);
    cout << code_int1 << " " << code_int2;
    double m;
    m = (y2 - y11) * 1.0 / (x2 - x1);
    if ((code_int1 | code_int2) == 0)
    {
        cout << "Line Inside";
        glColor3f(0, 1, 1);
        glBegin(GL_LINES);
        glVertex2f(x1, y11);
        glVertex2f(x2, y2);
        glEnd();
    }
    else if ((code_int1 & code_int2) != 0)
    {
        cout << "Line Outside";
    }
    else
    {
        cout << "Line to be clipped\n";
        p1 = CohenPoint(x1, y11, code1, m);
        p2 = CohenPoint(x2, y2, code2, m);
        glColor3f(1, 0, 1);
        glBegin(GL_LINES);
        glVertex2f(p1.first, p1.second);
        glVertex2f(p2.first, p2.second);
        cout << "Line after Clipping: "
             << "p1 :(" << p1.first << ", " << p1.second << ")p2 : (" <<
p2.first << ", " << p2.second << ")\n ";
        glEnd();
    }
}
```

Output:



UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No.8 3D Transformations

Date: 20/9/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

Perform the following basic 3D Transformations on any 3D Object.

1) Translation

2) Rotation

3) Scaling

Use only homogeneous coordinate representation and matrix multiplication to perform transformations.

Set the camera to any position on the 3D space. Have (0,0,0) at the center of the screen.

Draw X , Y and Z axis.

Code:

```
#include<GL/glut.h>
#include<iostream>
#include<math.h>

using namespace std;
#define PI 3.14159265358979323846264338327950288419716939937510582
typedef struct Point {
	double x, y, z, h;
}Point;
typedef struct Face {
	Point v[4];
}Face;
typedef struct Cuboid {
	Point v[8];
	Face faces[6];
}Cuboid;
int assignList[6][4] = { {0, 1, 3, 2}, {0, 4, 5, 1}, {0, 4, 6, 2},
{4, 5, 7, 6}, {2, 6, 7, 3}, {1, 5, 7, 3}
};
float colors[6][3] = { {0, 1, 0}, {1, 0, 0}, {0, 0, 1},
{1, 1, 0}, {0, 1, 1}, {1, 0, 1}
};
double tMat[4][4];
bool tflag = false;
Cuboid cuboid, tcuboid;
```

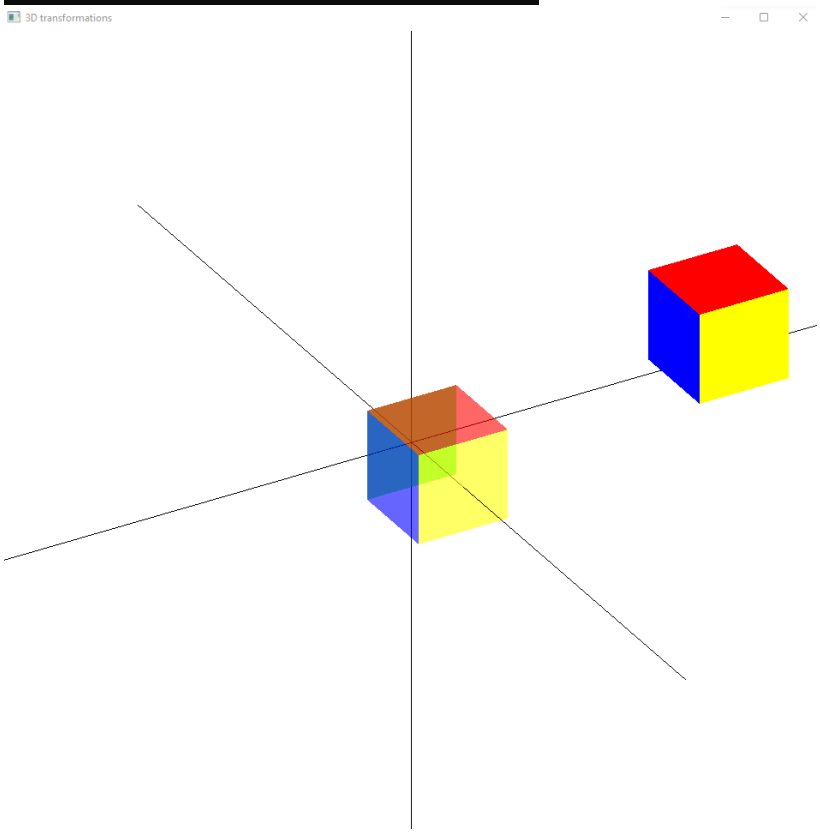
```
Cuboid initCuboid() {
	Cuboid cuboid;
	double po[8][3] = { {-25, 25, 0}, {25, 25, 0}, {-25, -25, 0},
{25,
	-25, 0},
	{-25, 25, 50}, {25, 25, 50}, {-25, -25, 50}, {25,
	-25, 50}
	};
	for (int i = 0; i < 8; i++) {
		cuboid.v[i].x = po[i][0];
		cuboid.v[i].y = po[i][1];
		cuboid.v[i].z = po[i][2];
		cuboid.v[i].h = 1;
	}
	for (int i = 0; i < 6; i++) {
		for (int j = 0; j < 4; j++) {
			cuboid.faces[i].v[j] = cuboid.v[assignList[i][j]];
		}
	}
	return cuboid;
}
void myInit() {
	glClearColor(1.0, 1.0, 1.0, 0.0);
	glColor3f(0, 0, 0);
	glEnable(GL_BLEND);
	glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
	glLoadIdentity();
	glOrtho(-200, 200, -200, 200, -200, 200);
	glEnable(GL_DEPTH_TEST);
}
void disp() {
	glRotatef(30, 1, 0, 0);
	glRotatef(30, 0, 1, 0);
}
void transformCuboid() {
	tflag = true;
	for (int i = 0; i < 8; i++) {
		cout << cuboid.v[i].x << " " << cuboid.v[i].y << "
"<<cuboid.v[i].z<<"\n";
	}
	cout << "\n\n";
	for (int p = 0; p < 8; p++) {
		double pnt[4][1], pnt1[4][1];
		pnt[0][0] = cuboid.v[p].x;
		pnt[1][0] = cuboid.v[p].y;
		pnt[2][0] = cuboid.v[p].z;
		pnt[3][0] = cuboid.v[p].h;
		memset(pnt1, 0, sizeof(pnt1));
		for (int i = 0; i < 4; i++) {
```

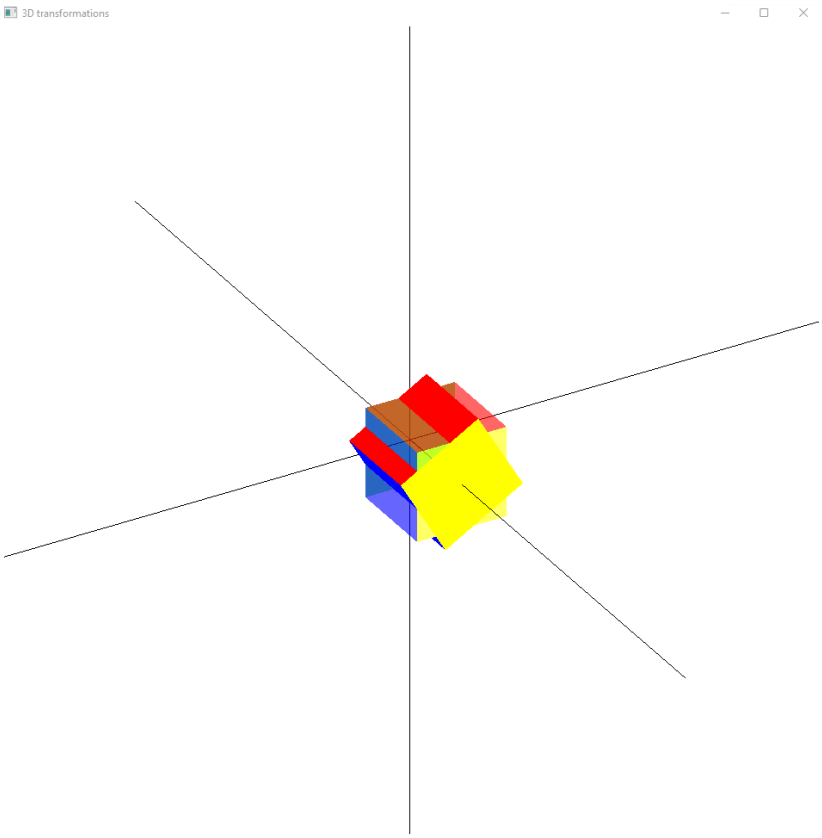
```
                for (int j = 0; j < 1; j++) {
                    for (int k = 0; k < 4; k++) {
                        pnt1[i][j] += tMat[i][k] * pnt[k][j];
                    }
                }
            }
            tcuboid.v[p].x = pnt1[0][0];
            tcuboid.v[p].y = pnt1[1][0];
            tcuboid.v[p].z = pnt1[2][0];
            tcuboid.v[p].h = pnt1[3][0];
        }
        for (int i = 0; i < 6; i++) {
            for (int j = 0; j < 4; j++) {
                tcuboid.faces[i].v[j] =
                    tcuboid.v[assignList[i][j]];
            }
        }
        for (int i = 0; i < 8; i++) {
            cout << tcuboid.v[i].x << " " << tcuboid.v[i].y << "
"<<tcuboid.v[i].z<<"\n";
        }
        glutPostRedisplay();
}
void getTransformMatrix() {
        memset(tMat, 0, sizeof(tMat));
        tMat[0][0] = tMat[1][1] = tMat[2][2] = tMat[3][3] = 1;
        int ch;
        cout <<
"Menu:\n\t1.Translation\n\t2.Rotation\n\t3.Scaling\n\tChoice:";
        cin >> ch;
        switch (ch) {
        case 1:
            cout << "Enter translation parameters: ";
            cin >> tMat[0][3] >> tMat[1][3] >> tMat[2][3];
            break;
        case 2:
            cout << "Enter degree of rotation: ";
            double deg;
            cin >> deg;
            deg = deg * PI / 180;
            tMat[0][0] = cos(deg);
            tMat[0][1] = -sin(deg);
            tMat[1][1] = tMat[0][0];
            tMat[1][0] = -tMat[0][1];
            break;
        case 3:
            cout << "Enter scaling parameters: ";
            cin >> tMat[0][0] >> tMat[1][1] >> tMat[2][2];
            break;
```

```
	default: cout << "Incorrect choice\n";
	}
	transformCuboid();
}
void displayCuboid(Cuboid cuboid, double alpha = 0.6) {
	for (int i = 0; i < 6; i++) {
		glColor4f(colors[i][0], colors[i][1], colors[i][2],
alpha);
		glBegin(GL_POLYGON);
		for (int j = 0; j < 4; j++) {
			glVertex3d(cuboid.faces[i].v[j].x,
cuboid.faces[i].v[j].y, cuboid.faces[i].v
				[j].z);
		}
		glEnd();
	}
}
void myDisplay() {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glColor4f(0, 0, 0, 1);
	glBegin(GL_LINES);
	glVertex3d(300, 0, 0);
	glVertex3d(-300, 0, 0);
	glEnd();
	glBegin(GL_LINES);
	glVertex3d(0, 300, 0);
	glVertex3d(0, -300, 0);
	glEnd();
	glBegin(GL_LINES);
	glVertex3d(0, 0, 300);
	glVertex3d(0, 0, -300);
	glEnd();
	displayCuboid(cuboid);
	if (tflag)displayCuboid(tcuboid, 1);
	glFlush();
	getTransformMatrix();
}
int main(int argc, char** argv) {
	cuboid = initCuboid();
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(960, 960);
	glutInitWindowPosition(0, 0);
	glutCreateWindow("3D transformations");
	myInit();
	disp();
	glutDisplayFunc(myDisplay);
	glutMainLoop();
}
```

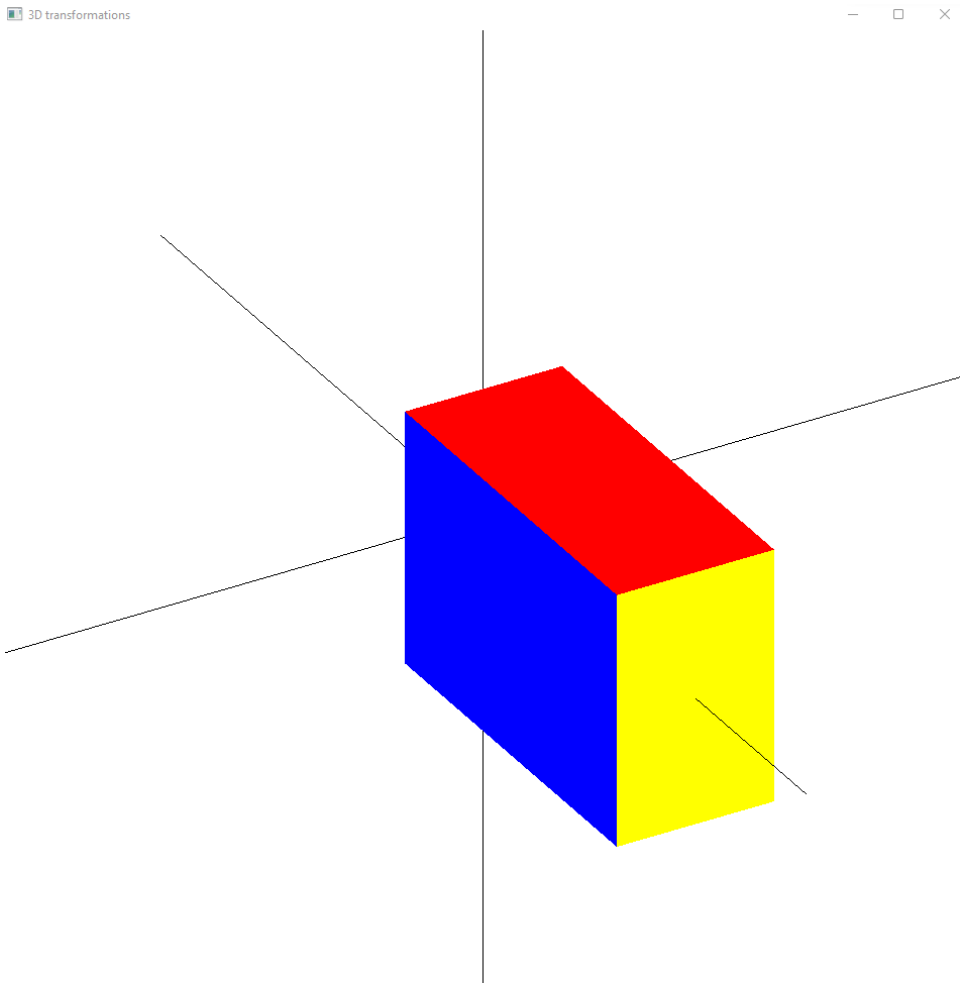
Outputs:

```
D:\vsRepos\testglut\Debug\testglut.exe
125 125 150
75 75 150
125 75 150
Menu:
        1.Translation
        2.Rotation
        3.Scaling
        Choice:2
Enter degree of rotation: 30
-25 25 0
25 25 0
-25 -25 0
25 -25 0
-25 25 50
25 25 50
-25 -25 50
25 -25 50


-34.1506 9.15064 0
9.15064 34.1506 0
-9.15064 -34.1506 0
34.1506 -9.15064 0
-34.1506 9.15064 50
9.15064 34.1506 50
-9.15064 -34.1506 50
34.1506 -9.15064 50
```






```
Menu:
        1.Translation
        2.Rotation
        3.Scaling
        Choice:3
Enter scaling parameters: 1.5 2.4 3.5
-25 25 0
25 25 0
-25 -25 0
25 -25 0
-25 25 50
25 25 50
-25 -25 50
25 -25 50


-37.5 60 0
37.5 60 0
-37.5 -60 0
37.5 -60 0
-37.5 60 175
37.5 60 175
-37.5 -60 175
37.5 -60 175
```


3D transformations

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No.9 Parallel and Perspective projections

Date: 27/9/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

Write a menu driven program to perform Orthographic parallel projection and Perspective projection on any 3D object. Set the camera to any position on the 3D space.

Have (0,0,0) at the center of the screen.

Draw X, Y and Z axis. You can use gluPerspective() to perform perspective projection.

Use keyboard functions to rotate and show different views of the object. [Can use built-in functions for 3D transformations].

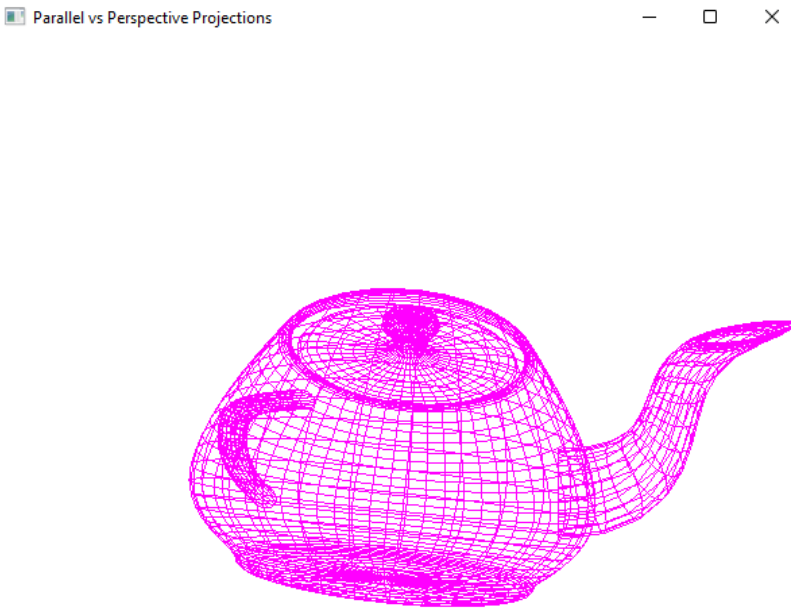
Code:

```
#include<iostream>
#include<math.h>
#include <GL/glut.h>

using namespace std;

int x = 0, y = 0;
int projType = 0;

void init() {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glEnable(GL_DEPTH_TEST);
}

void disp(int projType) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (projType == 1)
        gluPerspective(100, 1, 0.1, 100);
    else
        glOrtho(-2, 2, -2, 2, -2, 2);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0, 0, 1, 0, 0, 0, 0, 1, 0);
}

void display() {
    disp(projType);
    glRotatef(x, 0, 1, 0);
```

```
    glRotatef(y, 1, 0, 0);
    glColor3f(59.0, 0.0, 1.0);
    glutWireTeapot(0.5);
    glPopMatrix();
    glFlush();
}

void percieveKeyInterrupt(int key, int a, int b) {
    switch (key) {
    case GLUT_KEY_RIGHT: {
        x++;
        break;
    }

    case GLUT_KEY_LEFT: {
        x--;
        break;
    }

    case GLUT_KEY_UP: {
        y++;
        break;
    }

    case GLUT_KEY_DOWN: {
        y--;
        break;
    }

    }
    glutPostRedisplay();
}

void changeProjection(unsigned char c, int a, int b) {
    if (c == ' ') {
        projType ^= 1;
    }

    glutPostRedisplay();
}

int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(600, 600);
    glutCreateWindow("Parallel vs Perspective Projections");
    init();
    glutDisplayFunc(display);
    glutSpecialFunc(percieveKeyInterrupt);
```

```
    glutKeyboardFunc(changeProjection);
    glutMainLoop();
    return 0;
}
```

Outputs:



UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No.10 Creating a 3D Scene in C++ using OpenGL

Date: 4/10/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

Write a C++ program using Opengl to draw atleast four 3D objects.

Apply lighting and texture and render the scene.

Apply transformations to create a simple 3D animation.

[Use built-in transformation functions]

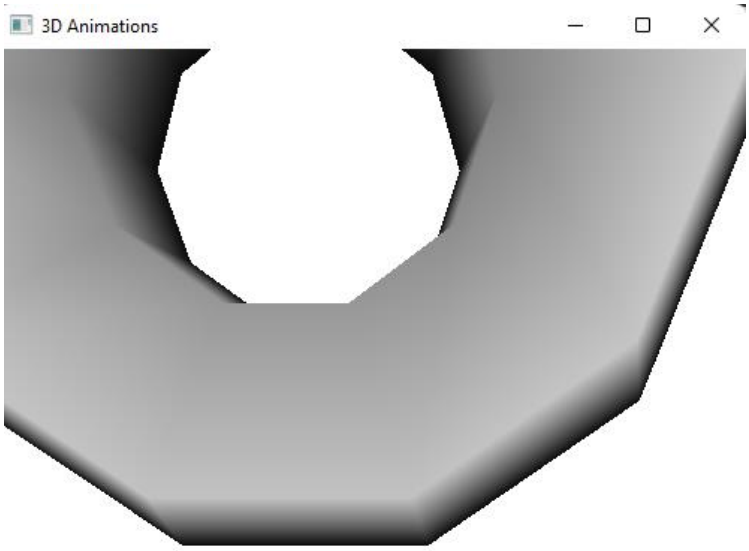
OpenGL Functions to use: glShadeModel() glMaterialfv() glLightfv() glEnable() glGenTextures() glTexEnvf() glBindTexture() glTexParameteri() glTexCoord2f()

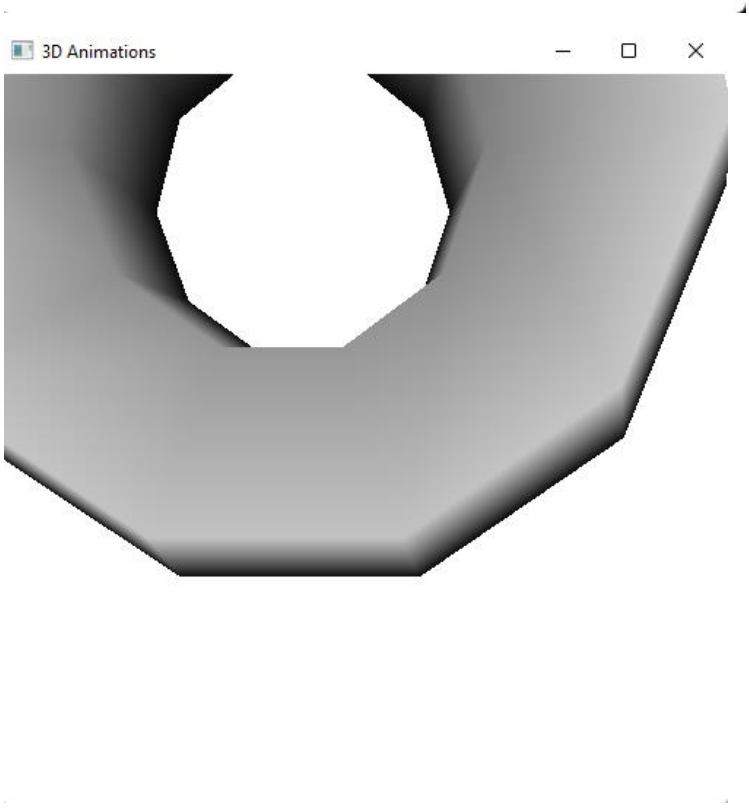
Code:

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

int increment = 1;
void initialize(void)
{
	glClearColor(1.0, 1.0, 1.0, 0.0);
	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	glEnable(GL_DEPTH_TEST);
}

void draw_objects(int state)
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	if (state == 0) increment = 1;
	else if (state == 10) increment = -1;

	glLoadIdentity();
	gluLookAt(0.0, 1.0, 7.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
	glMatrixMode(GL_MODELVIEW);

	glPushMatrix();
	GLfloat ball_color[] = {
		0.59,
		0.1,
```

```
            0.55,
            1.0
      };

      double scale = 0.15;
      glTranslatef(-1, 2.5 + scale * state, 0);
      /*glutSolidSphere(0.5, 10, 10);*/
      /*glutSolidCube(1.0);*/
      /*glutSolidTeapot(0.7);*/
      glutSolidTorus(2.0, 4.0, 5, 10);
      glPopMatrix();
      glutSwapBuffers();

      glutTimerFunc(1000 / 60, draw_objects, state + increment);
}

void threedanim() {
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glutTimerFunc(1000 / 60, draw_objects, 0);
}

void reshape(int width, int height)
{
      glViewport(0, 0, (GLsizei)width, (GLsizei)height);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(75, 1, 1, 20);
      glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv)
{
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
      glutInitWindowSize(500, 500);
      glutCreateWindow("3D Animations");
      initialize();
      glutDisplayFunc(threedanim);
      glutReshapeFunc(reshape);
      glutMainLoop();
      return 0;
}
```

Output:


3D Animations


3D Animations

UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No.11 Image Editing and Manipulation

Date: 11/10/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

a) Using GIMP, include an image and apply filters, noise and masks.

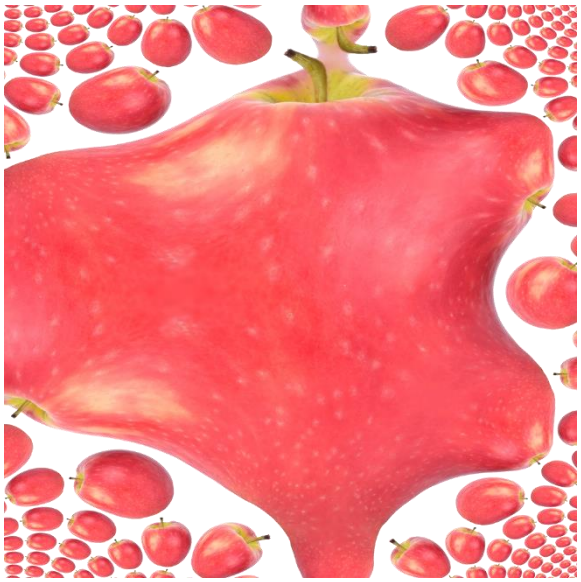
b) Using GIMP, create a GIF animated image.

Output Screenshots:

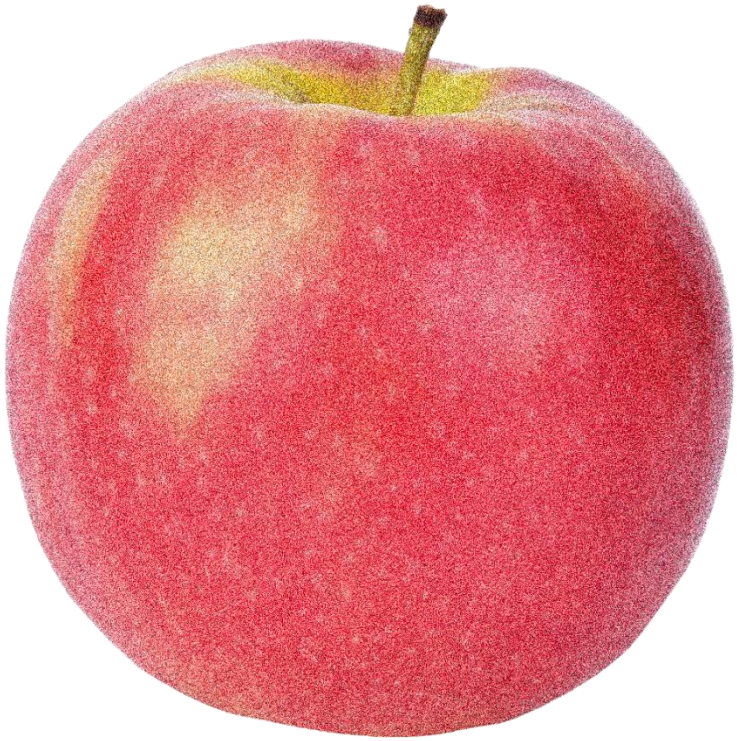
Answer A):

Original image:

With Filter applied:

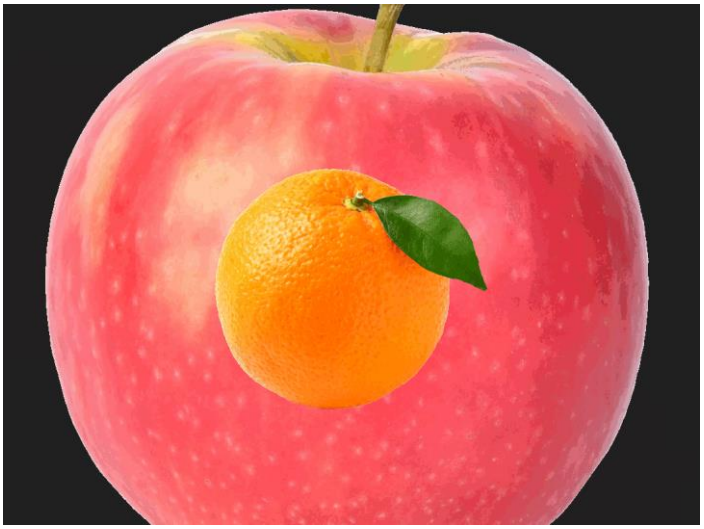


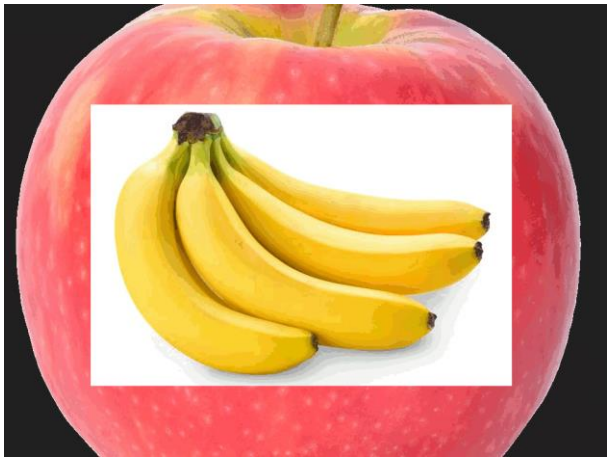
With noise applied:

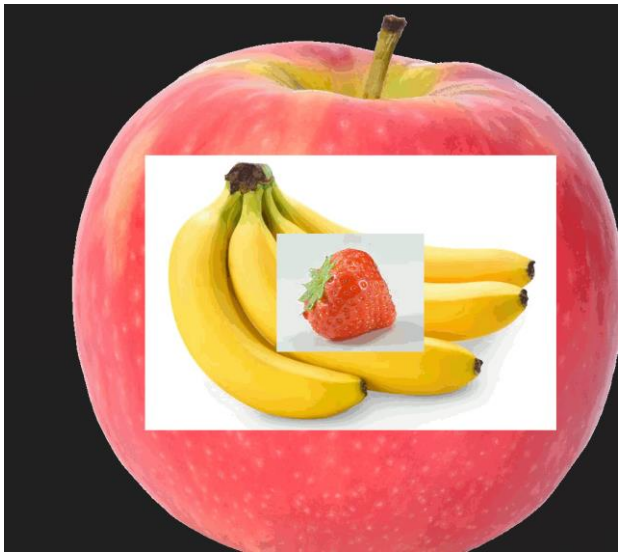


With mask applied:

Answer B) GIF:







UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No.12 Creating 2D animation

Date: 18/10/21 **Name:** Srinath S

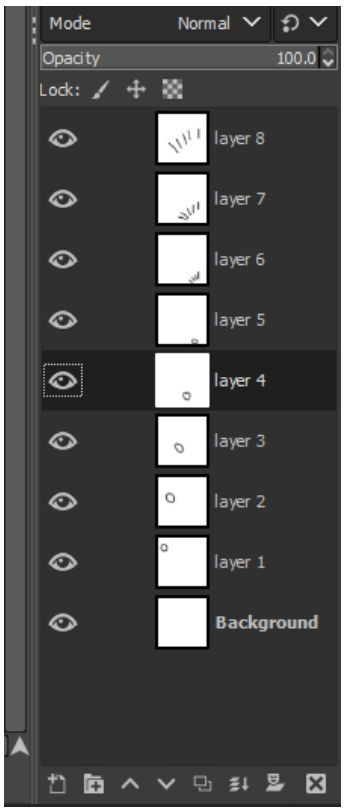
Class: CSE-C **Roll:** 185001205

Question:

Using GIMP, include layers and create a simple animation of your choice.

Output Screenshots:

Layers:



Animation: