**1. Write and OPENGL app to draw primitives such as a line and unfilled rectangle, triangle, hexagon.**

#include <GL/glut.h>

void drawShapes()

{

glLineWidth(3);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(-0.8, 0.8);

glVertex2f(0.8, 0.5);

glEnd();

glColor3f(0.0, 1.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(-0.9, -0.5);

glVertex2f(-0.5, -0.5);

glVertex2f(-0.5, 0.5);

glVertex2f(-0.9, 0.5);

glEnd();

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(-0.4, -0.5);

glVertex2f(-0.1, 0.5);

glVertex2f(0.2, -0.5);

glEnd();

glColor3f(1.0, 1.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(0.1, 0.0);

glVertex2f(0.4, 0.5);

glVertex2f(0.7, 0.5);

glVertex2f(1.0, 0.0);

glVertex2f(0.7, -0.5);

glVertex2f(0.4, -0.5);

glEnd();

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowSize(800, 600);

glutInitWindowPosition(100, 100);

glutCreateWindow("Utsarga");

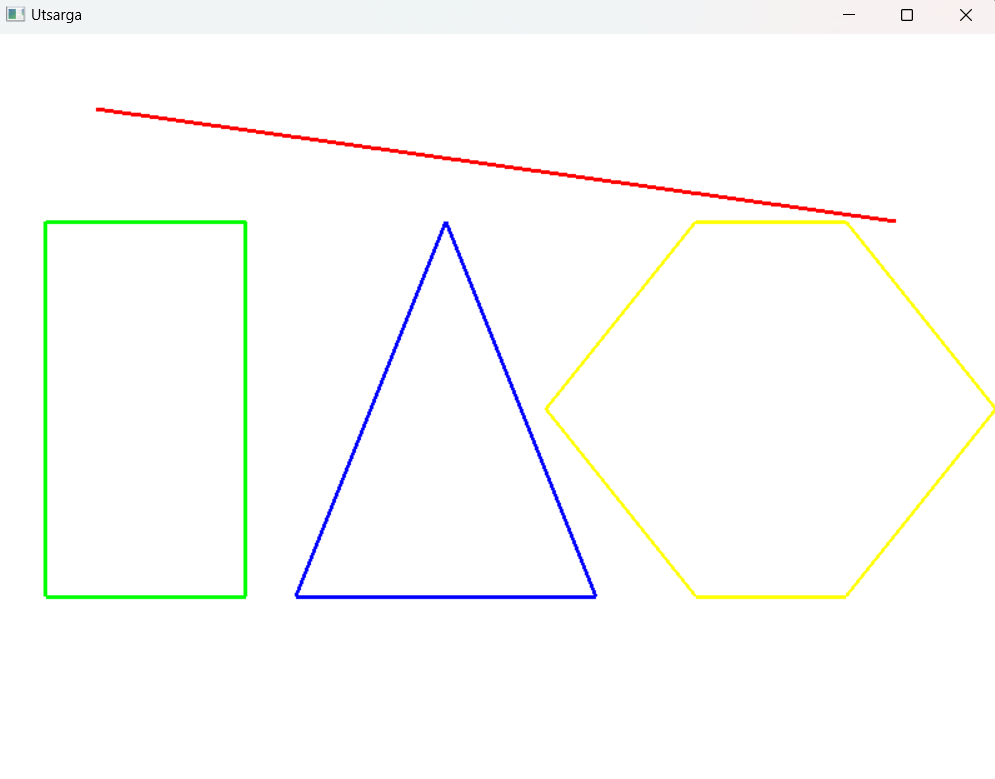
glClearColor(1.0, 1.0, 1.0, 1.0);

glutDisplayFunc(drawShapes);

glutMainLoop();

return 0;

}



**2. Write an OPENGL program app that draws following polygons: red square, blue rectangle, and green pentagon at the same location and style.**

#include<GL/glut.h>

void draw(){

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3d(1,0,0);

glBegin(GL\_POLYGON);

glVertex2d(0,0);

glVertex2d(0,0.4);

glVertex2d(-0.4,0.4);

glVertex2d(-0.4,0);

glEnd();

glFlush();

glColor3d(0,0,1);

glBegin(GL\_TRIANGLES);

glVertex2d(0,0.2);

glVertex2d(-0.2,-0.2);

glVertex2d(0.2,-0.2);

glEnd();

glFlush();

glColor3d(0,1,0);

glBegin(GL\_POLYGON);

glVertex2d(0.1,0);

glVertex2d(0.4,0);

glVertex2d(0.6,-0.3);

glVertex2d(0.4,-0.6);

glVertex2d(0.1,-0.6);

glVertex2d(-0.1,-0.3);

glEnd();

glFlush();

}

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

glutInit(&argC, argv);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100,100);

glutCreateWindow("Utsarga");

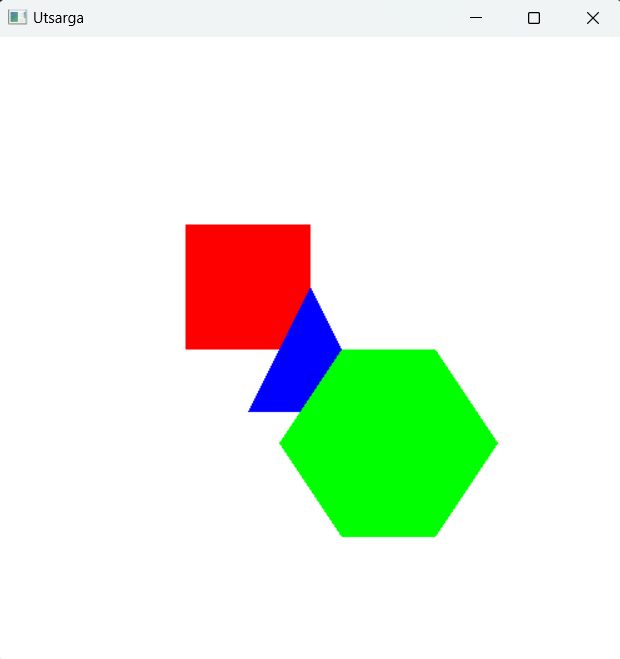
glClearColor(1.0,1.0,1.0,1.0);

glutDisplayFunc(draw);

glutMainLoop();

return 0;

}



**3. Write and OPENGL program to animate bouncing square**

#include <GL/glut.h>

float squareWidth = 100.0f, squarePosX = 0.0f, squarePosY = 0.0f ,squareSpeedX = 5.0f, squareSpeedY = 5.0f ,x\_max = 800.0f,y\_max = 600.0f;

void drawSquare()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0f, 0.0f, 0.0f);

glBegin(GL\_POLYGON);

glVertex2f(squarePosX, squarePosY);

glVertex2f(squarePosX + squareWidth, squarePosY);

glVertex2f(squarePosX + squareWidth, squarePosY + squareWidth);

glVertex2f(squarePosX, squarePosY + squareWidth);

glEnd();

glutSwapBuffers();

}

void update(int value)

{

squarePosX += squareSpeedX;

squarePosY += squareSpeedY;

if (squarePosX + squareWidth > x\_max || squarePosX < 0.0f)

{

squareSpeedX = -squareSpeedX;

}

if (squarePosY + squareWidth > y\_max || squarePosY < 0.0f)

{

squareSpeedY = -squareSpeedY;

}

glutPostRedisplay();

glutTimerFunc(16, update, 0);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(x\_max, y\_max);

glutCreateWindow("Utsarga");

glClearColor(1.0,1.0,1.0,1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0,800,0,600);

glMatrixMode(GL\_MODELVIEW);

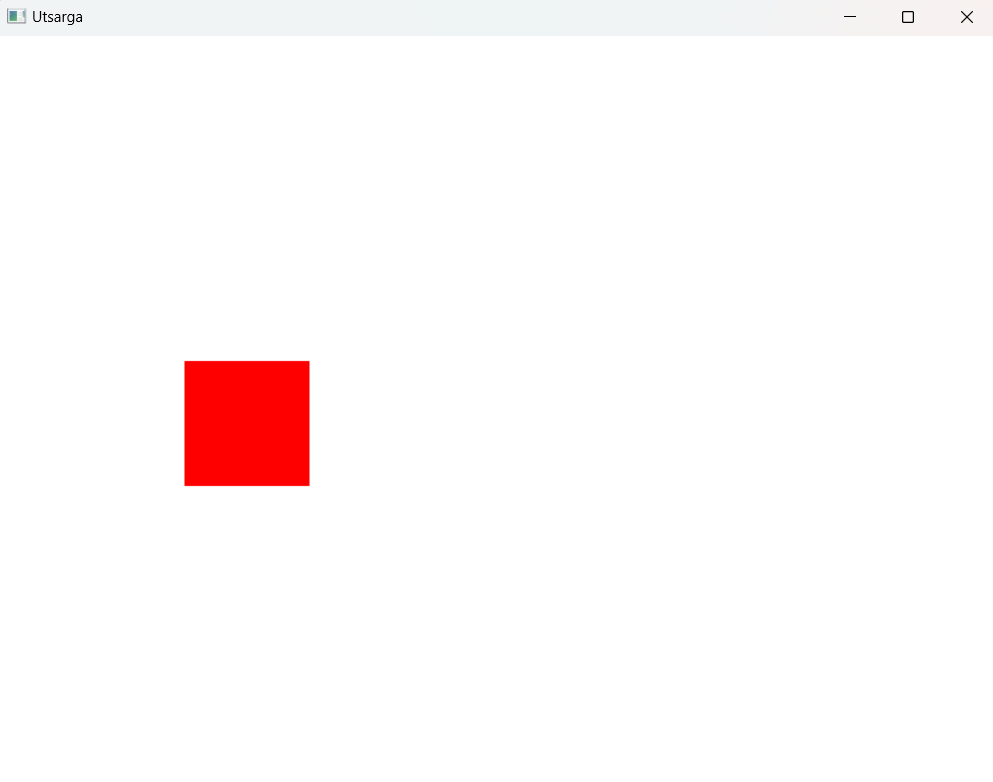
glutDisplayFunc(drawSquare);

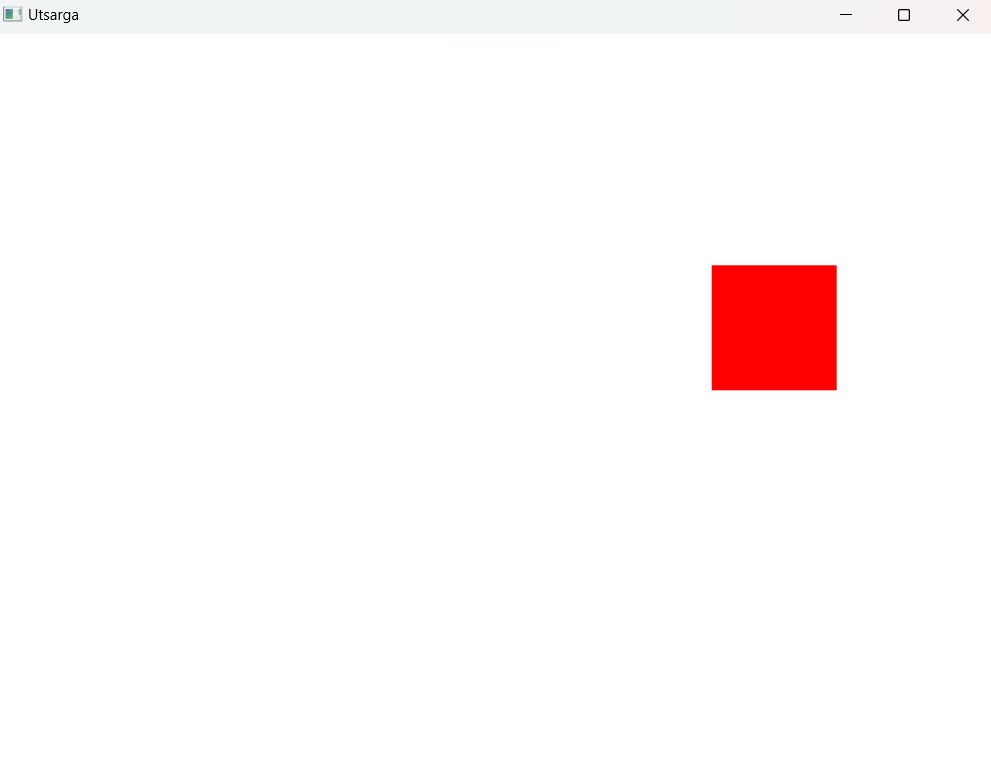
glutTimerFunc(0, update, 0);

glutMainLoop();

return 0;

}





**4. Write a OPENGL app to draw red square at the plate of mouse click.**

Source Code

#include <GL/glut.h>

int windowWidth = 800;

int windowHeight = 600;

int squareSize = 70;

int mouseX = 0;

int mouseY = 0;

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0f, 0.0f, 0.0f); // Red square

glBegin(GL\_POLYGON);

glVertex2f(mouseX, windowHeight - mouseY);

glVertex2f(mouseX + squareSize, windowHeight - mouseY);

glVertex2f(mouseX + squareSize, windowHeight - mouseY + squareSize);

glVertex2f(mouseX, windowHeight - mouseY + squareSize);

glEnd();

glutSwapBuffers();

}

void mouse(int button, int state, int x, int y)

{

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

{

mouseX = x - 2;

mouseY = y - 2;

display();

glutPostRedisplay();

}

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(windowWidth, windowHeight);

glutCreateWindow("Draw Square on Mouse Click-By Atullya");

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, windowWidth, 0, windowHeight);

glMatrixMode(GL\_MODELVIEW);

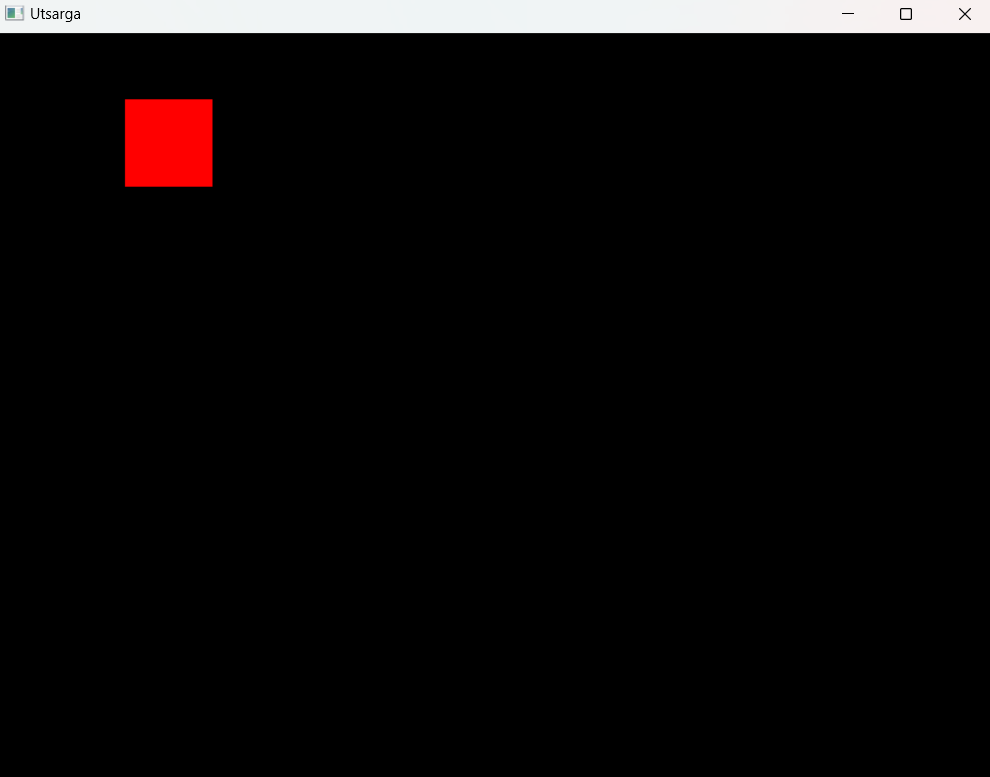
glutDisplayFunc(display);

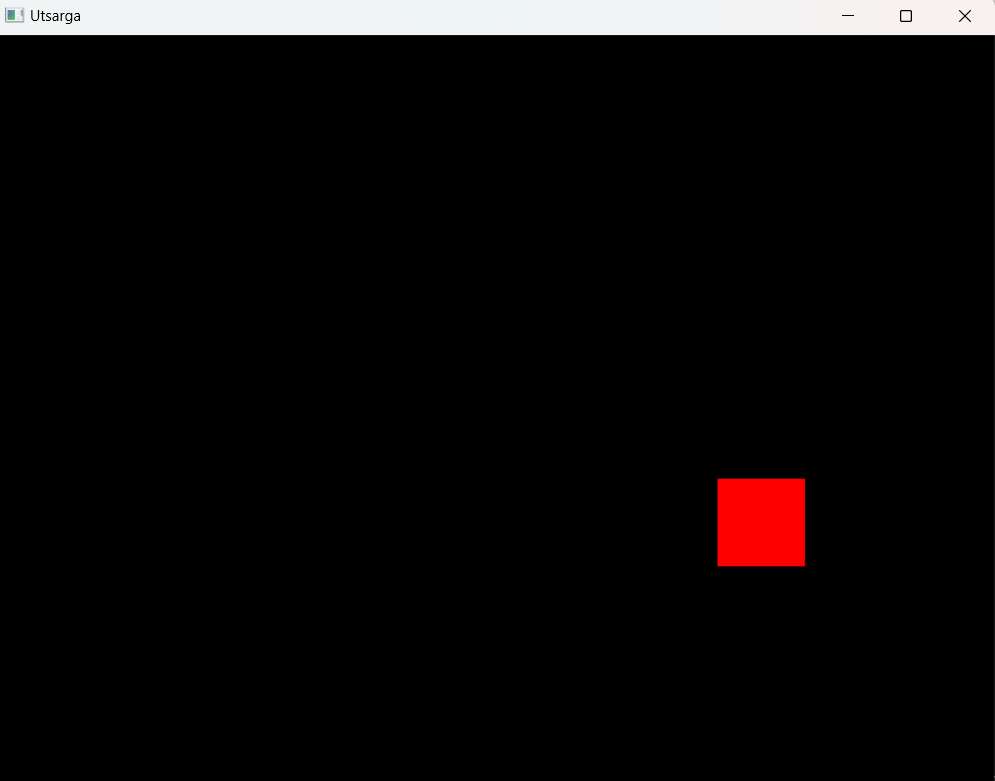
glutMouseFunc(mouse);

glutMainLoop();

return 0;

}





**5. Write an OPENGL app to display following HUT.**

Source Code

#include <GL/glut.h>

void drawHut()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.3490f, 0.5098f, 0.6902f);

glBegin(GL\_TRIANGLES);

glVertex2f(230.0f, 350.0f);

glVertex2f(570.0f, 350.0f);

glVertex2f(400.0f, 500.0f);

glEnd();

glColor3f(0.5019f, 0.8314f, 0.1059f); // Red color

glBegin(GL\_QUADS);

glVertex2f(230.0f, 50.0f);

glVertex2f(230.0f, 350.0f);

glVertex2f(570.0f, 350.0f);

glVertex2f(570.0f, 50.0f);

glEnd();

glColor3f(0.3490f, 0.5098f, 0.6902f);

glBegin(GL\_QUADS);

glVertex2f(350.0f, 60.0f);

glVertex2f(350.0f, 185.0f);

glVertex2f(450.0f, 185.0f);

glVertex2f(450.0f, 60.0f);

glEnd();

glColor3f(0.9412f, 0.0471f, 0.0471f);

glBegin(GL\_QUADS);

glVertex2f(430.0f, 200.0f);

glVertex2f(430.0f, 280.0f);

glVertex2f(510.0f, 280.0f);

glVertex2f(510.0f, 200.0f);

glEnd();

glBegin(GL\_QUADS);

glVertex2f(290.0f, 200.0f);

glVertex2f(290.0f, 280.0f);

glVertex2f(370.0f, 280.0f);

glVertex2f(370.0f, 200.0f);

glEnd();

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 600);

glutCreateWindow("Utsarga");

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, 800, 0, 600);

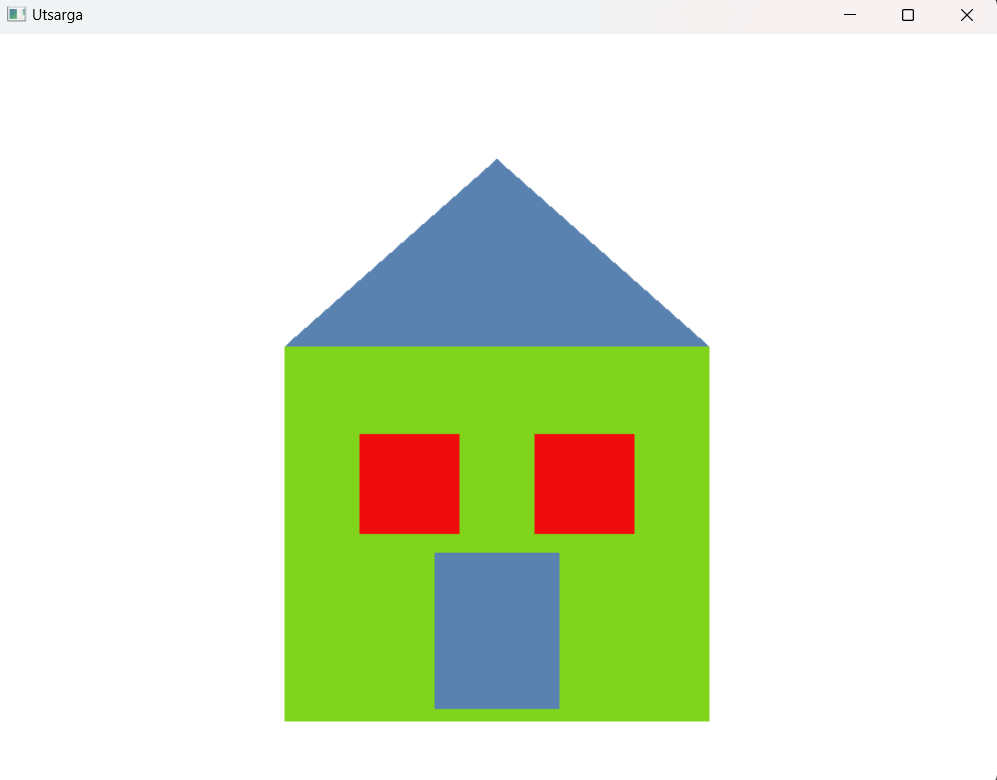
glMatrixMode(GL\_MODELVIEW);

glutDisplayFunc(drawHut);

glutMainLoop();

return 0;

}



**6. Write an OPENGL app to generate 50 random points and plot them.**

Source Code

#include <GL/glut.h>

int windowWidth = 800;

int windowHeight = 600;

void plotPoint()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0f, 0.0f, 0.0f); // Black points

glPointSize(5.0f);

glBegin(GL\_POINTS);

for (int i = 0; i < 50; i++)

{

glVertex2f(rand() % windowWidth, rand() % windowHeight);

}

glEnd();

glFlush();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(windowWidth, windowHeight);

glutCreateWindow("Utsarga");

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, windowWidth, 0, windowHeight);

glMatrixMode(GL\_MODELVIEW);

glutDisplayFunc(plotPoint);

glutMainLoop();

return 0;

}



**8. Implement direct method to draw line given two end points.**

Source Code

#include <GL/glut.h>

#include <iostream>

#include <cmath>

using namespace std;

float xi, yi, xf, yf;

void plotPixel() {

float x, y, dx, dy, m, c, xend;

dx = xf - xi;

dy = yf - yi;

m = dy / dx;

c = yi - m \* xi;

if(xi > xf) {

x = xf;

y = yf;

xend = xi;

} else {

x = xi;

y = yi;

xend = xf;

}

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_POINTS);

while(x <= xend) {

glVertex2f(round(x), round(y));

x += 1;

y = m \* x + c;

}

glEnd();

glFlush();

}

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

cout << "Enter the initial points: ";

cin >> xi >> yi;

cout << "Enter the final points: ";

cin >> xf >> yf; glutInit( & argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowSize(800, 600);

glutInitWindowPosition(100, 100);

glutCreateWindow("Utsarga");

glClearColor(1.0, 1.0, 1.0, 1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

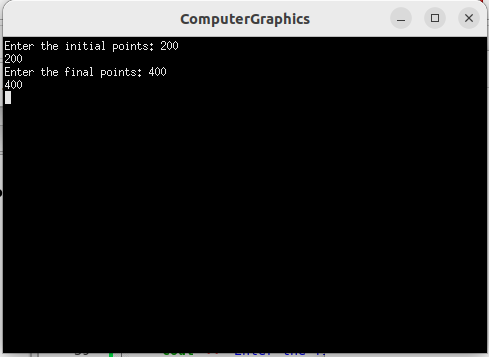
gluOrtho2D(0, 800, 0, 600);

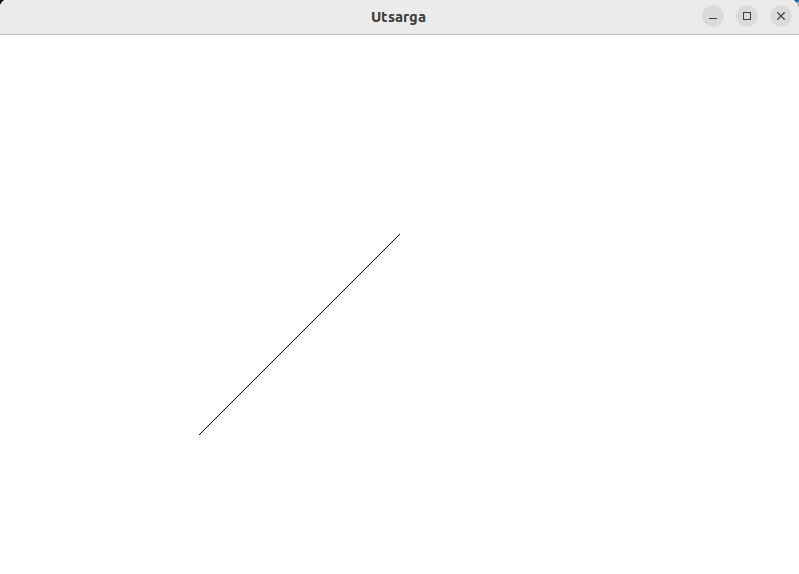
glutDisplayFunc(plotPixel);

glutMainLoop();

return 0;

}





**9. Implement DDA to draw line given two end points of line.**

Source Code

#include <GL/glut.h>

#include <cmath>

#include <iostream>

using namespace std;

float xf, yf, xi, yi;

void drawLine()

{

float steps, dx = xf - xi, dy = yf - yi;

if (abs(dx) < abs(dy))

{

steps = abs(dy);

}

else

{

steps = abs(dx);

}

float xIncrement = dx / steps;

float yIncrement = dy / steps;

float x = xi;

float y = yi;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_POINTS);

glVertex2f(round(x), round(y));

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

{

x += xIncrement;

y += yIncrement;

glVertex2f(round(x), round(y));

}

glEnd();

glFlush();

}

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

{

cout << "Enter the initial points: ";

cin >> xi >> yi;

cout << "Enter the final points: ";

cin >> xf >> yf;

glutInit( & argc, argv);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500, 500);

glutInitWindowPosition(200, 200);

glutCreateWindow("Utsarga");

glClearColor(1.0, 1.0, 1.0, 0.0);

glMatrixMode(GL\_PROJECTION);

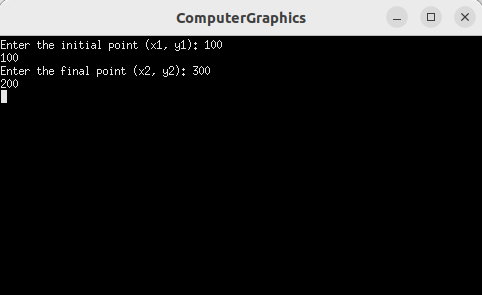
gluOrtho2D(0.0, 500.0, 0.0, 500.0);

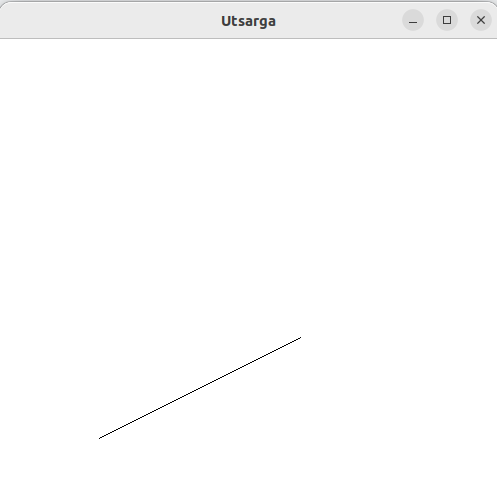
glutDisplayFunc(drawLine);

glutMainLoop();

return 0;

}





**10. Compare time taken to draw given line using direct method, DDA and Bresenham.**

Source Code

#include <GL/glut.h>

#include <iostream>

#include <ctime>

using namespace std;

int x\_max = 700, y\_max = 600;

int x1, y1, x2, y2;

void drawPixel(int x, int y) {

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

// Direct method

void drawDirect() {

glColor3f(1.0f, 0.0f, 0.0f);

int dx = x2 - x1;

int dy = y2 - y1;

int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

float xIncrement = static\_cast<float>(dx) / steps;

float yIncrement = static\_cast<float>(dy) / steps;

float x = x1, y = y1;

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

drawPixel(static\_cast<int>(x + 0.5), static\_cast<int>(y + 0.5));

x += xIncrement;

y += yIncrement;

}

glFlush();

}

// DDA algorithm

void drawDDA() {

glColor3f(0.0f, 0.0f, 1.0f);

int dx = x2 - x1;

int dy = y2 - y1;

int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

float xinc = static\_cast<float>(dx) / steps;

float yinc = static\_cast<float>(dy) / steps;

float x = x1, y = y1;

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

drawPixel(static\_cast<int>(x + 0.5), static\_cast<int>(y + 0.5));

x += xinc;

y += yinc;

}

glFlush();

}

// Bresenham algorithm

void drawBresenham() {

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

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

int x = x1, y = y1;

int sx = (x2 - x1 >= 0) ? 1 : -1;

int sy = (y2 - y1 >= 0) ? 1 : -1;

if (dy <= dx) {

int p = 2 \* dy - dx;

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

drawPixel(x, y);

x += sx;

if (p >= 0) {

y += sy;

p -= 2 \* dx;

}

p += 2 \* dy;

}

} else {

int p = 2 \* dx - dy;

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

drawPixel(x, y);

y += sy;

if (p >= 0) {

x += sx;

p -= 2 \* dy;

}

p += 2 \* dx;

}

}

glFlush();

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glPointSize(2);

// Measure time taken for Direct method

clock\_t start\_time = clock();

drawDirect();

clock\_t end\_time = clock();

double direct\_time = double(end\_time - start\_time) / CLOCKS\_PER\_SEC;

// Measure time taken for DDA algorithm

start\_time = clock();

drawDDA();

end\_time = clock();

double dda\_time = double(end\_time - start\_time) / CLOCKS\_PER\_SEC;

// Measure time taken for Bresenham algorithm

start\_time = clock();

drawBresenham();

end\_time = clock();

double bresenham\_time = double(end\_time - start\_time) / CLOCKS\_PER\_SEC;

cout << "Time taken by Direct method: " << direct\_time << " seconds" << endl;

cout << "Time taken by DDA algorithm: " << dda\_time << " seconds" << endl;

cout << "Time taken by Bresenham algorithm: " << bresenham\_time << " seconds" << endl;

glFlush();

}

void myInit() {

glClearColor(1.0, 1.0, 1.0, 0.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, x\_max, 0.0, y\_max);

}

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

cout << "Enter first point (x1 y1): ";

cin >> x1 >> y1;

cout << "Enter second point (x2 y2): ";

cin >> x2 >> y2;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(x\_max, y\_max);

glutInitWindowPosition(100, 100);

glutCreateWindow("Line Drawing Algorithms Comparison");

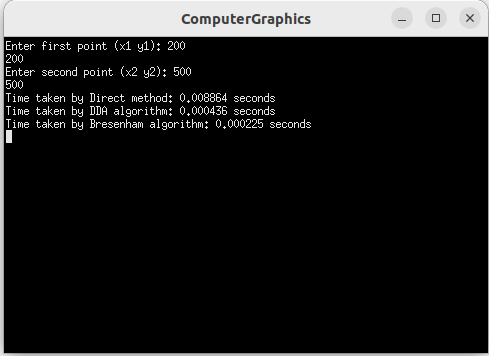
myInit();

glutDisplayFunc(display);

glutMainLoop();

return 0;

}





**11. Implement mid point algorithm to draw circle given its radius and center.**

Source Code

#include <iostream>

#include <GL/glut.h>

using namespace std;

int xi, yi, radius;

void plot(int x, int y)

{

glBegin(GL\_POINTS);

glVertex2i(x + xi, y + yi);

glEnd();

}

void drawCircle()

{

int x = 0;

int y = radius;

float decision = 1 - radius;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 0.0);

glPointSize(1.0);

plot(x, y);

while (x <= y)

{

x++;

if (decision < 0)

{

decision += 2 \* x + 1;

}

else

{

y--;

decision += 2 \* (x - y) + 1;

}

plot(x, y);

plot(x, -y);

plot(-x, y);

plot(-x, -y);

plot(y, x);

plot(-y, x);

plot(y, -x);

plot(-y, -x);

}

glFlush();

}

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

{

cout << "Enter the coordinates of the center (x,y): ";

cin >> xi >> yi;

cout << "Enter the radius of the circle: ";

cin >> radius;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 600);

glutInitWindowPosition(100, 150);

glutCreateWindow("Utsarga");

glClearColor(1.0, 1.0, 1.0, 0.0);

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

glPointSize(4.0);

glMatrixMode(GL\_PROJECTION);

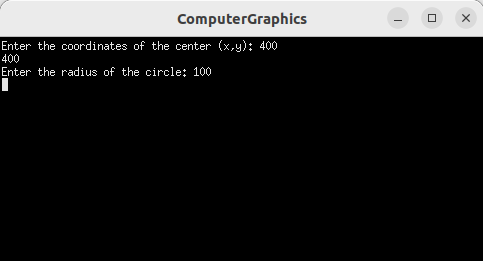
glLoadIdentity();

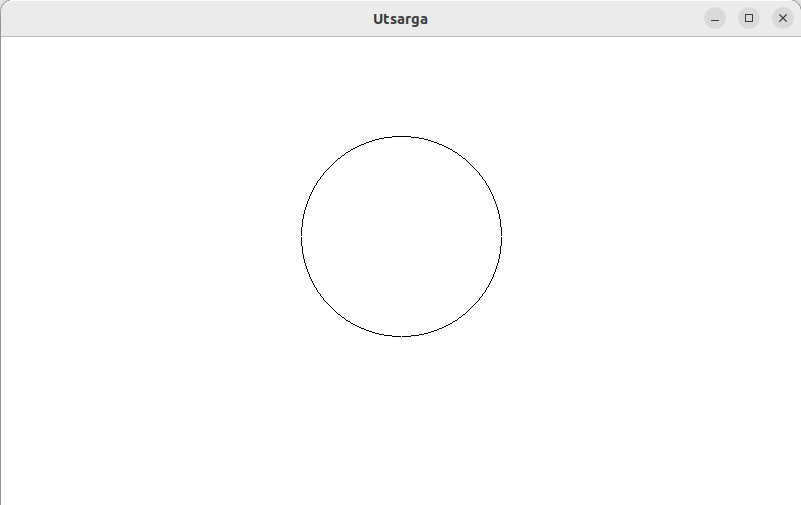
gluOrtho2D(0, 800, 0, 600);

glutDisplayFunc(drawCircle);

glutMainLoop();

}





**12.Implement mid point algorithm to draw ellipse given its primary, secondary axis and center.**

Source Code

#include <iostream>

#include <GL/glut.h>

using namespace std;

int xi, yi, a, b;

void plot(int x, int y)

{

glBegin(GL\_POINTS);

glVertex2i(x + xi, y + yi);

glEnd();

}

void drawEllipse()

{

int x = 0;

int y = b;

float decision = b \* b - a \* a \* b + 0.25 \* a \* a;

int dx = 2 \* b \* b \* x;

int dy = 2 \* a \* a \* y;

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 0.0);

glPointSize(1.0);

plot(x, y);

while (dx < dy)

{

if (decision < 0)

{

x++;

dx += 2 \* b \* b;

decision += dx + b \* b;

}

else

{

x++;

y--;

dx += 2 \* b \* b;

dy -= 2 \* a \* a;

decision += dx - dy + b \* b;

}

plot(x, y);

plot(-x, y);

plot(x, -y);

plot(-x, -y);

}

decision = b \* b \* (x + 0.5) \* (x + 0.5) + a \* a \* (y - 1) \* (y - 1) - a \* a \* b \* b;

while (y >= 0)

{

if (decision > 0)

{

y--;

dy -= 2 \* a \* a;

decision += a \* a - dy;

}

else

{

y--;

x++;

dx += 2 \* b \* b;

dy -= 2 \* a \* a;

decision += dx - dy + a \* a;

}

plot(x, y);

plot(-x, y);

plot(x, -y);

plot(-x, -y);

}

glFlush();

}

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

{

cout << "Enter the coordinates of the center (x,y): ";

cin >> xi >> yi;

cout << "Enter the length of the major axis (a): ";

cin >> a;

cout << "Enter the length of the minor axis (b): ";

cin >> b;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 600);

glutInitWindowPosition(100, 150);

glutCreateWindow("Utsarga");

glClearColor(1.0, 1.0, 1.0, 0.0);

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

glPointSize(4.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

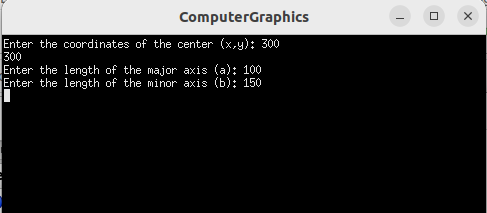
gluOrtho2D(0, 800, 0, 600);

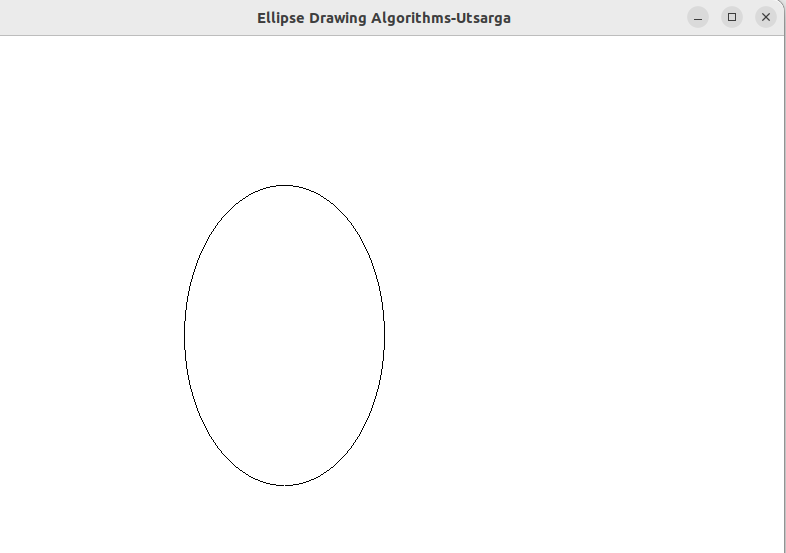
glutDisplayFunc(drawEllipse);

glutMainLoop();

return 0;

}





**13. Write an OPENGL app to rotate a rectangle about origin with one point being (100,100), width 200 and height 50 by 30 degree anticlockwise by using opengl function.**

Source Code

#include <GL/glut.h>

#include <cmath>

float x, y;

float width, height;

int win\_width = 800;

int win\_height = 600;

void drawRectangle(float x, float y, float width, float height) {

glColor3f(0.0f, 0.0f, 1.0f);

glLineWidth(2.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(x, y);

glVertex2f(x + width, y);

glVertex2f(x + width, y + height);

glVertex2f(x, y + height);

glEnd();

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawRectangle(x, y, width, height);

float centerX = x + width / 2;

float centerY = y + height / 2;

glPushMatrix();

glRotatef(30.0, 0.0, 0.0, 1.0);

drawRectangle(x, y, width, height);

glPopMatrix();

glFlush();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-100, win\_width, -100, win\_height);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

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

x = 100;

y = 100;

width = 200;

height = 50;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(win\_width, win\_height);

glutInitWindowPosition(100, 100);

glutCreateWindow("Utsarga");

glClearColor(1.0, 1.0, 1.0, 1.0);

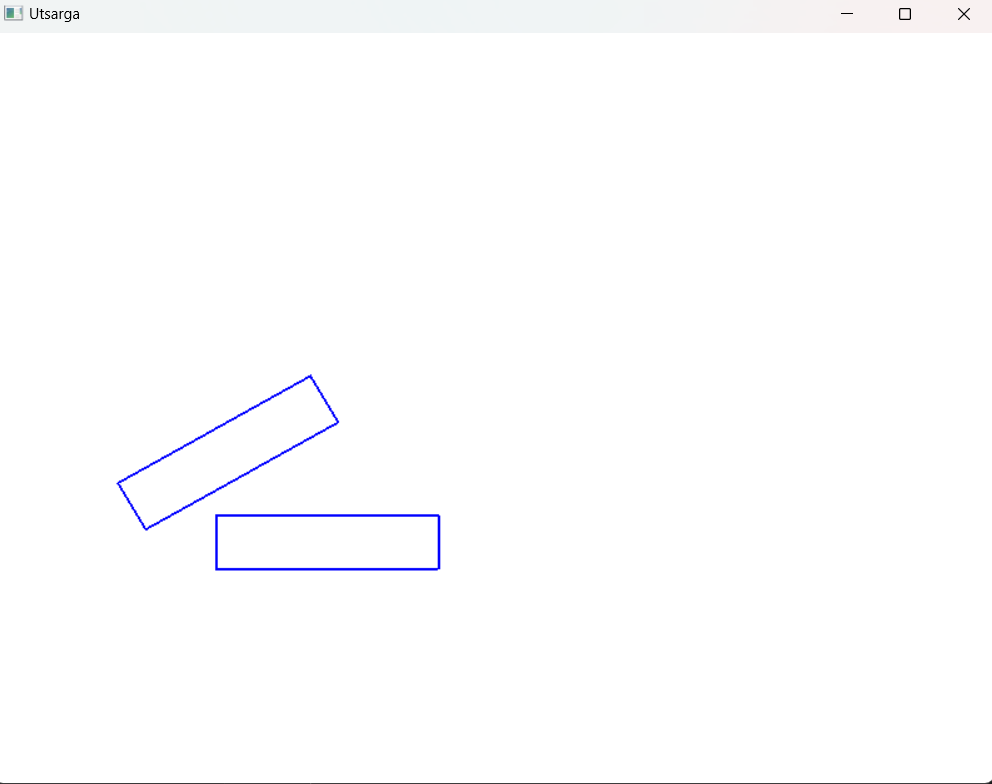
glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}



**14. Write an OPENGL app to rotate a rectangle about origin with one point being (100,100), width 200 and height 50 by 30 degree anticlockwise without using opengl function.**

Source Code

#include <GL/glut.h>

#include <cmath>

#include <iostream>

using namespace std;

float x, y;

float width, height;

int win\_width = 800;

int win\_height = 600;

void rotate()

{

float theta = (30 \* 3.14159265) / 180.0;

float rotationMatrix[3][3] = {cos(theta), -sin(theta), 0, sin(theta), cos(theta), 0, 0, 0, 1};

float originalVertex[3][4] = {{x, x + width, x + width, x}, {y, y, y + height, y + height}, {1, 1, 1, 1}};

float finalVertex[3][4] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

glColor3f(0.0f, 0.0f, 1.0f);

glLineWidth(2.0);

glBegin(GL\_LINE\_LOOP);

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

{

glVertex2f(originalVertex[0][i], originalVertex[1][i]);

}

glEnd();

glFlush();

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

{

for (int j = 0; j < 4; j++)

{

for (int k = 0; k < 3; k++)

{

finalVertex[i][j] += rotationMatrix[i][k] \* originalVertex[k][j];

}

}

}

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP);

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

{

glVertex2f(finalVertex[0][i], finalVertex[1][i]);

}

glEnd();

glFlush();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

rotate();

}

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

{

x = 100;

y = 100;

width = 200;

height = 50;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowSize(win\_width, win\_height);

glutInitWindowPosition(100, 100);

glutCreateWindow("Utsarga");

glClearColor(1.0, 1.0, 1.0, 1.0);

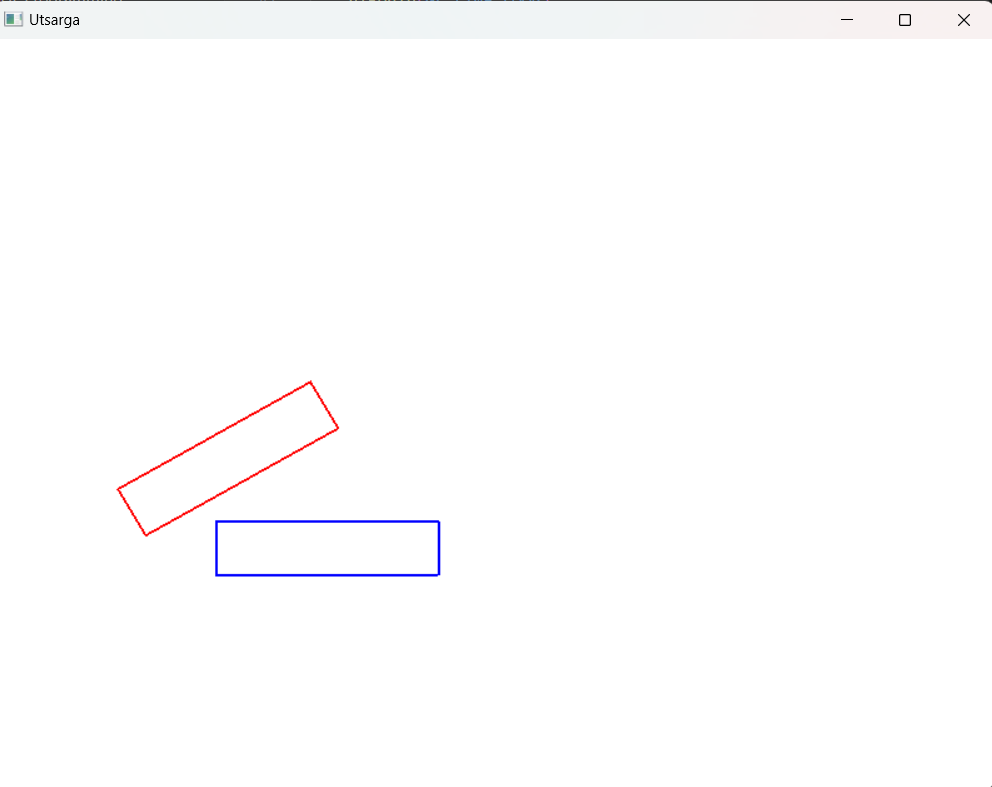
gluOrtho2D(-100, win\_width, -100, win\_height);

glutDisplayFunc(display);

glutMainLoop();

return 0;

}



**15. Use the rectangle in Question no 13. and scale it by (2,2) using opengl function**

Source Code

#include <GL/glut.h>

#include <cmath>

float x, y;

float width, height;

int win\_width = 800;

int win\_height = 600;

void drawRectangle(float x, float y, float width, float height) {

glColor3f(0.0f, 0.0f, 1.0f);

glLineWidth(2.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(x, y);

glVertex2f(x + width, y);

glVertex2f(x + width, y + height);

glVertex2f(x, y + height);

glEnd();

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawRectangle(x, y, width, height);

float centerX = x + width / 2;

float centerY = y + height / 2;

glPushMatrix();

glScalef(2.0, 2.0, 1.0); // Scale by (2, 2)

drawRectangle(x, y, width, height);

glPopMatrix();

glFlush();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-100, win\_width, -100, win\_height);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

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

x = 100;

y = 100;

width = 200;

height = 50;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(win\_width, win\_height);

glutInitWindowPosition(100, 100);

glutCreateWindow("Utsarga");

glClearColor(1.0, 1.0, 1.0, 1.0);

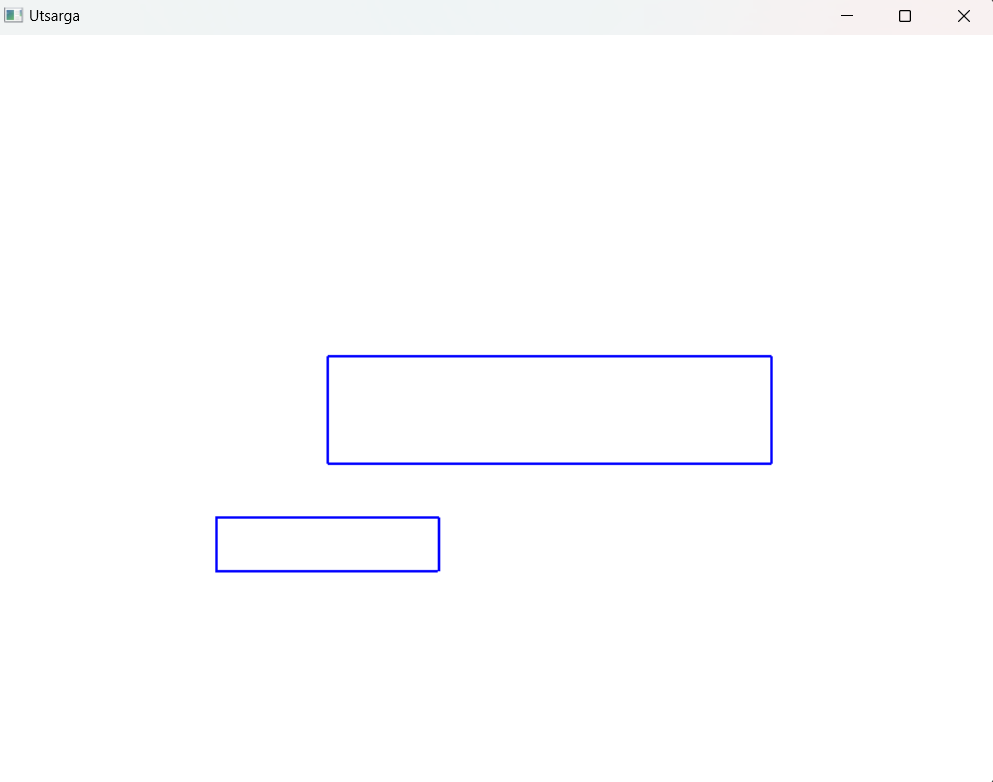
glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}



**16. Do the same thing in Question no 15 without using opengl scaling function**

Source Code

#include <GL/glut.h>

#include <cmath>

using namespace std;

float x, y;

float width, height;

int win\_width = 800;

int win\_height = 600;

void drawRectangle(float rectangle[][4]){

glLineWidth(2.0);

glBegin(GL\_LINE\_LOOP);

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

{

glVertex2f(rectangle[0][i], rectangle[1][i]);

}

glEnd();

glFlush();

}

void scale()

{

float theta = (30 \* 3.14159265) / 180.0;

float sx = 2, sy = 2;

float scaleMatrix[3][3] = {sx,0,0, 0,sy,0, 0, 0, 1};

float originalVertex[3][4] = {{x, x + width, x + width, x}, {y, y, y + height, y + height}, {1, 1, 1, 1}};

float finalVertex[3][4] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

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

drawRectangle(originalVertex);

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

{

for (int j = 0; j < 4; j++)

{

for (int k = 0; k < 3; k++)

{

finalVertex[i][j] += scaleMatrix[i][k] \* originalVertex[k][j];

}

}

}

glColor3f(0.5, 0.5, 0.5);

drawRectangle(finalVertex);

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0, 0.0, 0.0); // Black color

glFlush();

scale();

}

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

{

x = 100;

y = 100;

width = 200;

height = 50;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);

glutInitWindowSize(win\_width, win\_height);

glutInitWindowPosition(100, 100);

glutCreateWindow("Utsarga");

glClearColor(1.0, 1.0, 1.0, 1.0);

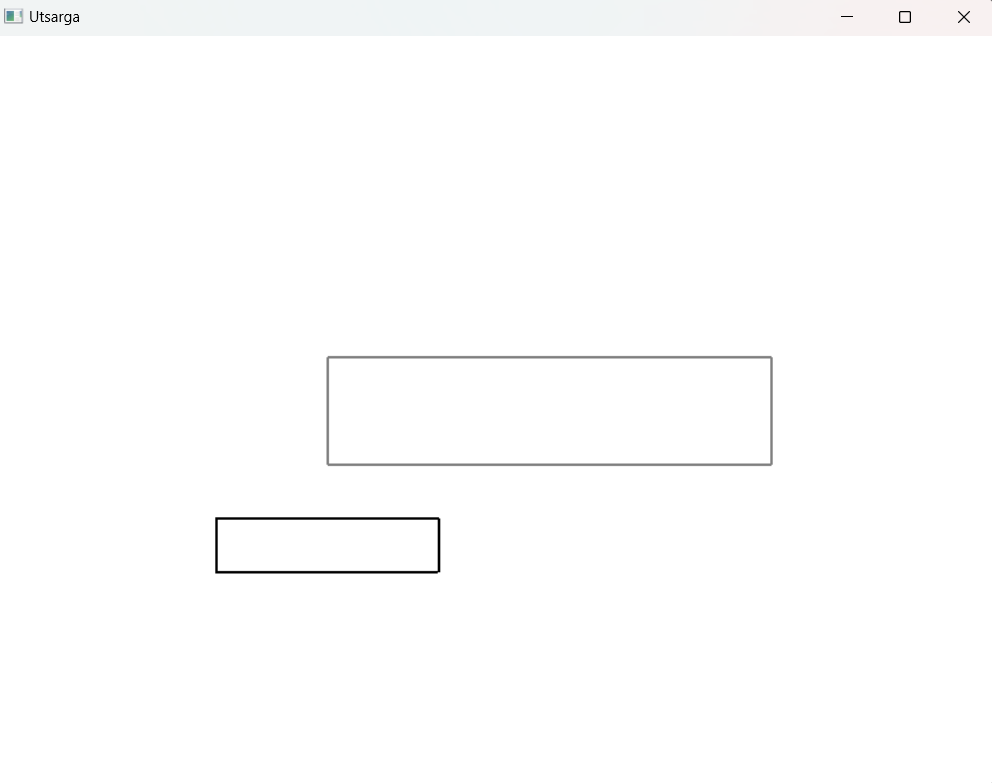
gluOrtho2D(-100, win\_width, -100, win\_height);

glutDisplayFunc(display);

glutMainLoop();

return 0;

}



**17. Use the rectangle in the Question no. 13 to rotate it by 30 degree anticlockwise about a fixed point (150,125) and see how the results is different from Question no 13 by using opengl transformation methods.**

Source Code

#include <GL/glut.h>

#include <cmath>

float x, y;

float width, height;

int win\_width = 800;

int win\_height = 600;

void drawRectangle(float x, float y, float width, float height) {

glColor3f(0.0f, 0.0f, 1.0f);

glLineWidth(2.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(x, y);

glVertex2f(x + width, y);

glVertex2f(x + width, y + height);

glVertex2f(x, y + height);

glEnd();

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawRectangle(x, y, width, height);

float centerX = 150.0f;

float centerY = 125.0f;

glPushMatrix();

glTranslatef(centerX, centerY, 0.0f);

glRotatef(30.0, 0.0, 0.0, 1.0);

glTranslatef(-centerX, -centerY, 0.0f);

drawRectangle(x, y, width, height);

glPopMatrix();

glFlush();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-100, win\_width, -100, win\_height);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

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

x = 100;

y = 100;

width = 200;

height = 50;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(win\_width, win\_height);

glutInitWindowPosition(100, 100);

glutCreateWindow("Utsarga");

glClearColor(1.0, 1.0, 1.0, 1.0);

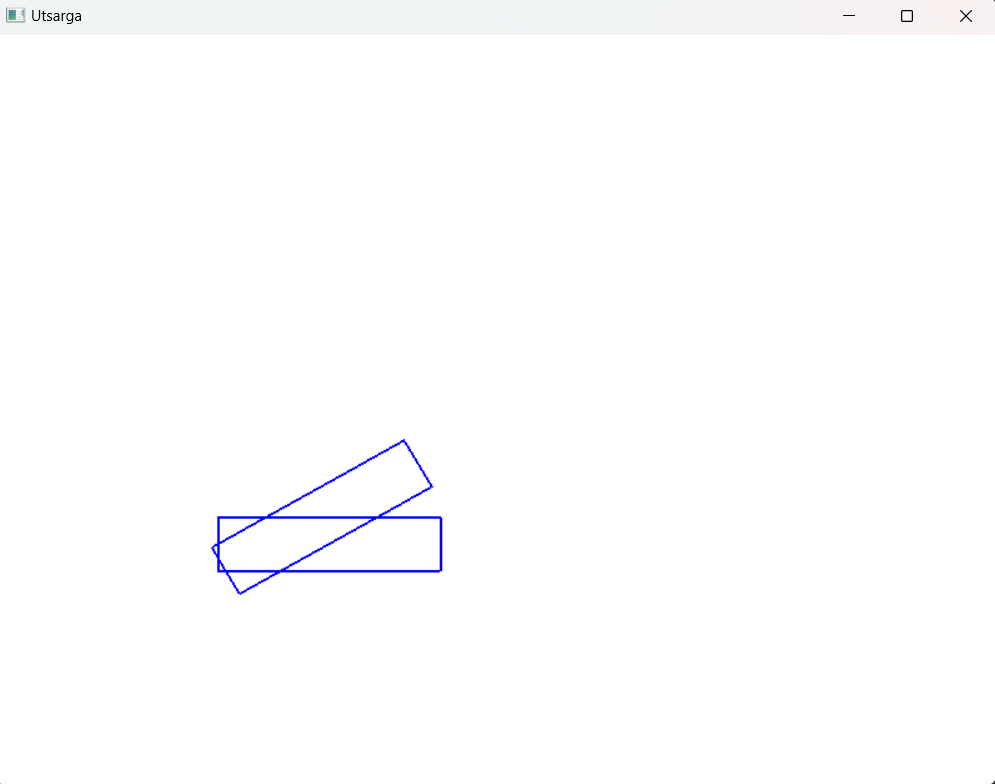
glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}



**18. Do the same thing 17 without using opengl transformation methods and see if the results are same.**

Source Code

#include <GL/glut.h>

#include <cmath>

float x, y;

float width, height;

int win\_width = 800;

int win\_height = 600;

void drawRectangle(float x, float y, float width, float height) {

glColor3f(0.0f, 0.0f, 1.0f);

glLineWidth(2.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(x, y);

glVertex2f(x + width, y);

glVertex2f(x + width, y + height);

glVertex2f(x, y + height);

glEnd();

}

void matrixMultiply(float matrix[3][3], float& x, float& y) {

float tempX = matrix[0][0] \* x + matrix[0][1] \* y + matrix[0][2];

float tempY = matrix[1][0] \* x + matrix[1][1] \* y + matrix[1][2];

x = tempX;

y = tempY;

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

// Draw the original rectangle

drawRectangle(x, y, width, height);

// Calculate the center of rotation

float centerX = 150.0f;

float centerY = 125.0f;

// Create rotation matrix

float angle = 30.0 \* M\_PI / 180.0;

float cosA = cos(angle);

float sinA = sin(angle);

float rotationMatrix[3][3] = {

{cosA, -sinA, centerX \* (1 - cosA) + centerY \* sinA},

{sinA, cosA, centerY \* (1 - cosA) - centerX \* sinA},

{0, 0, 1}

};

// Rotate the vertices of the rectangle using matrix multiplication

float x1 = x;

float y1 = y;

float x2 = x + width;

float y2 = y;

float x3 = x + width;

float y3 = y + height;

float x4 = x;

float y4 = y + height;

matrixMultiply(rotationMatrix, x1, y1);

matrixMultiply(rotationMatrix, x2, y2);

matrixMultiply(rotationMatrix, x3, y3);

matrixMultiply(rotationMatrix, x4, y4);

// Draw the rotated rectangle

glBegin(GL\_LINE\_LOOP);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glVertex2f(x3, y3);

glVertex2f(x4, y4);

glEnd();

glFlush();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-100, win\_width, -100, win\_height);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

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

x = 100;

y = 100;

width = 200;

height = 50;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(win\_width, win\_height);

glutInitWindowPosition(100, 100);

glutCreateWindow("Utsarga");

glClearColor(1.0, 1.0, 1.0, 1.0);

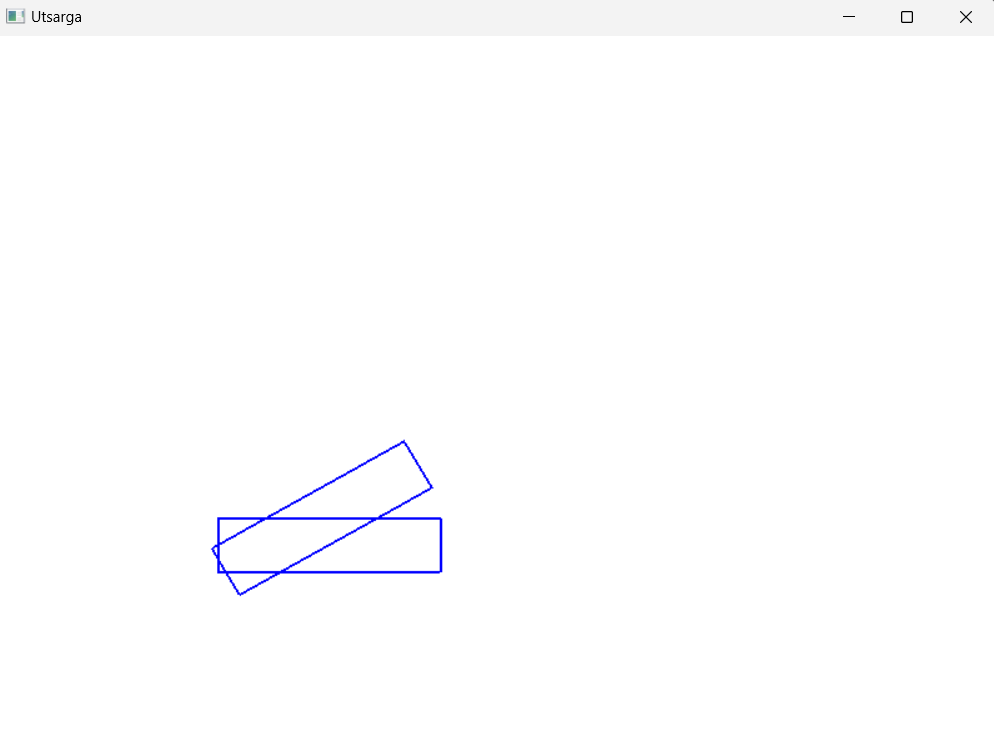
glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}



**19. Write an OPENGL app to illustrate orthogonal projection**

Source Code

#include <GL/glut.h>

#include <iostream>

int windowWidth = 800;

int windowHeight = 600;

void drawScene() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

// Set the background color to white

glClearColor(1.0, 1.0, 1.0, 1.0);

// Clear the color buffer

glClear(GL\_COLOR\_BUFFER\_BIT);

// Set the square color to black

glColor3f(0.0, 0.0, 0.0);

// Draw a square

glBegin(GL\_QUADS);

glVertex2f(-0.5, -0.5);

glVertex2f(0.5, -0.5);

glVertex2f(0.5, 0.5);

glVertex2f(-0.5, 0.5);

glEnd();

glutSwapBuffers();

}

void reshape(int width, int height) {

glViewport(0, 0, width, height);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(windowWidth, windowHeight);

glutCreateWindow("Utsarga");

glutDisplayFunc(drawScene);

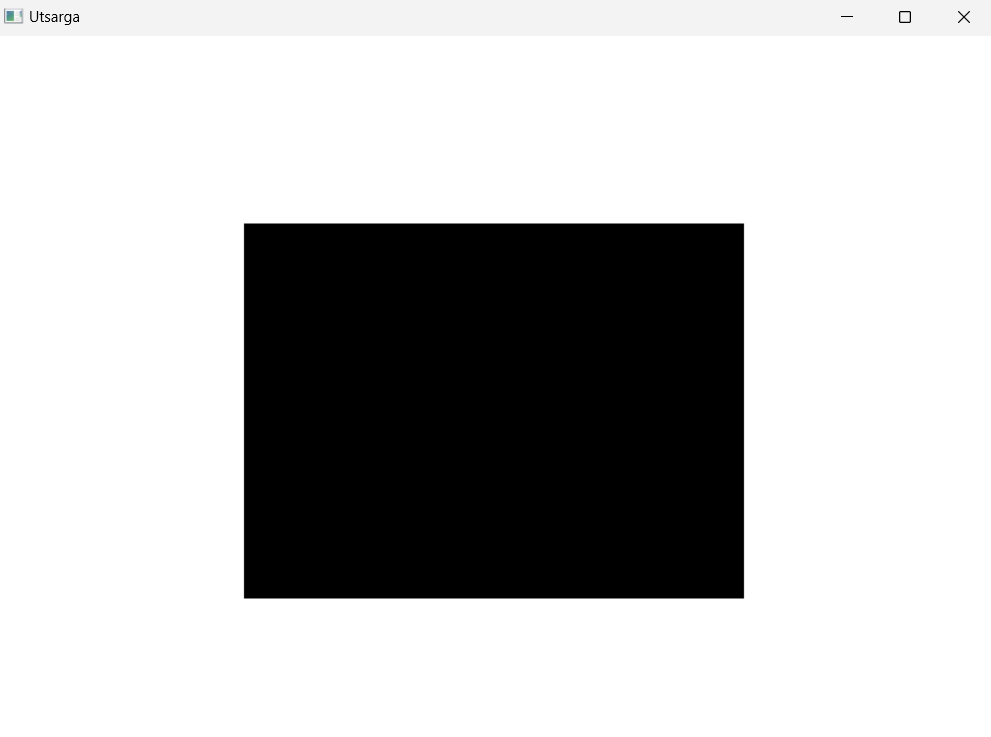
glutReshapeFunc(reshape);

glClearColor(1.0, 1.0, 1.0, 1.0);

glutMainLoop();

return 0;

}



**20. Write an OPENGL app to show clipping using orthogonal projection.**

Source Code

#include <GL/glut.h>

int win\_width = 800;

int win\_height = 600;

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

// Set up the clipping region using glOrtho

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-0.4, 0.4, -0.4, 0.4, -1.0, 1.0);

// Draw the original non-filled triangle

glColor3f(0.0f, 0.0f, 1.0f);

glBegin(GL\_LINE\_LOOP);

glVertex2f(-0.5f, -0.4f);

glVertex2f(0.7f, -0.2f);

glVertex2f(0.2f, 0.7f);

glEnd();

glutSwapBuffers();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-1.0, 1.0, -1.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(win\_width, win\_height);

glutInitWindowPosition(100, 100);

glutCreateWindow("Utsarga");

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

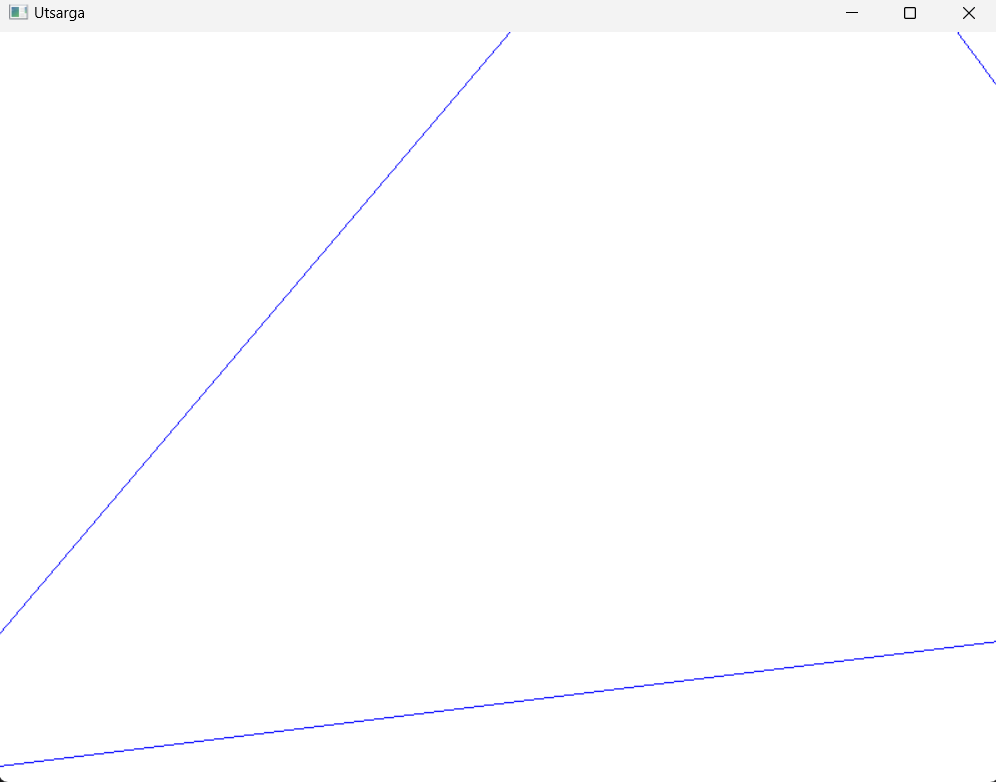
glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}



**21. Write an OPENGL app to show rectangle in two different viewports.**

Source Code

#include <GL/glut.h>

int win\_width = 800;

int win\_height = 600;

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glViewport(0, 0, win\_width / 2, win\_height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glColor3f(0.0f, 0.0f, 1.0f);

glBegin(GL\_POLYGON);

glVertex2f(-0.8f, -0.8f);

glVertex2f(-0.2f, -0.8f);

glVertex2f(-0.2f, 0.8f);

glVertex2f(-0.8f, 0.8f);

glEnd();

glViewport(win\_width / 2, 0, win\_width / 2, win\_height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glColor3f(1.0f, 0.0f, 0.0f);

glBegin(GL\_POLYGON);

glVertex2f(0.2f, -0.8f);

glVertex2f(0.8f, -0.8f);

glVertex2f(0.8f, 0.8f);

glVertex2f(0.2f, 0.8f);

glEnd();

glutSwapBuffers();

}

void reshape(int w, int h) {

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-1.0, 1.0, -1.0, 1.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(win\_width, win\_height);

glutInitWindowPosition(100, 100);

glutCreateWindow("Utsarga");

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

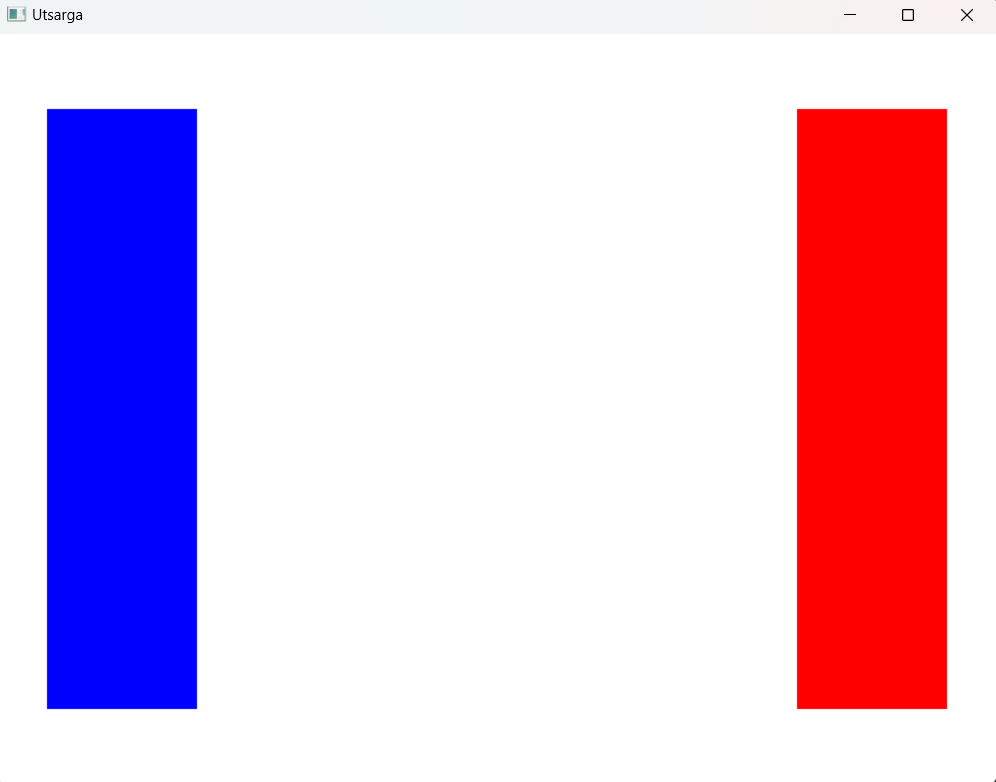
glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMainLoop();

return 0;

}



**22. Implement sutherland cohen clipping algorithm to clip the given line.**

Source Code

#include <GL/glut.h>

#include <iostream>

int wx\_max = 100, wy\_max = 150, wx\_min = 50, wy\_min = 50;

int x\_1, x\_2, y\_1, y\_2; // Variables to store user input

int getCode(int x, int y)

{

int code = 0;

if (x < wx\_min)

code |= 1;

else if (x > wx\_max)

code |= 2;

if (y < wy\_min)

code |= 4;

else if (y > wy\_max)

code |= 8;

return code;

}

void drawline(int x1, int y1, int x2, int y2)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0.0f, 0.0f, 0.0f); // black

glLineWidth(2.0);

// Draw the window

glBegin(GL\_LINE\_LOOP);

glVertex2i(wx\_min, wy\_min);

glVertex2i(wx\_max, wy\_min);

glVertex2i(wx\_max, wy\_max);

glVertex2i(wx\_min, wy\_max);

glEnd();

// Draw the line

glColor3f(0.0f, 0.0f, 1.0f); // blue

glBegin(GL\_LINES);

glVertex2i(x1, y1);

glVertex2i(x2, y2);

glEnd();

glFlush();

}

void cohen\_sutherland()

{

// ... (unchanged)

}

void timer(int value)

{

cohen\_sutherland();

glutPostRedisplay();

glutTimerFunc(1000, timer, 0);

}

void display()

{

drawline(x\_1, y\_1, x\_2, y\_2);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 600);

glutCreateWindow("Utsarga");

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-20, 200, -20, 200);

glMatrixMode(GL\_MODELVIEW);

glutDisplayFunc(display);

// User input

std::cout << "Enter x1 y1 x2 y2: ";

std::cin >> x\_1 >> y\_1 >> x\_2 >> y\_2;

glutTimerFunc(1000, timer, 0); // Start the timer

glutMainLoop();

return 0;

}

