//DDA Line

#include <GL/glut.h>

#include <iostream>

using namespace std;

int lineType = 0;

bool triangleDrawn = false;

// Initialize OpenGL

void init() {

glClearColor(1, 1, 1, 1); // White background

gluOrtho2D(0, 800, 0, 800); // 2D projection

glPointSize(2); // Pixel size

}

// Draw a single pixel

void drawPixel(float x, float y) {

glBegin(GL\_POINTS);

glVertex2f(x, y);

glEnd();

}

// DDA Algorithm with Line Styles

void drawLineDDA(int x1, int y1, int x2, int y2, int type) {

float dx = x2 - x1;

float dy = y2 - y1;

int steps = max(abs(dx), abs(dy));

float xInc = dx / steps;

float yInc = dy / steps;

float x = x1, y = y1;

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

bool draw = false;

if (type == 0) draw = true; // Simple

else if (type == 1 && i % 5 == 0) draw = true; // Dotted

else if (type == 2 && (i % 10 < 5)) draw = true; // Dashed

else if (type == 3) { // Solid line

glLineWidth(4);

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

return;

}

if (draw)

drawPixel(x, y);

x += xInc;

y += yInc;

}

}

// Draw Axis

void drawAxis() {

glColor3f(0, 0, 0);

drawLineDDA(0, 400, 800, 400, 0);

drawLineDDA(400, 0, 400, 800, 0);

}

// Draw Triangle

void drawTriangle() {

glColor3f(0, 0, 1); // Blue triangle

drawLineDDA(400, 500, 500, 300, lineType);

drawLineDDA(500, 300, 300, 300, lineType);

drawLineDDA(300, 300, 400, 500, lineType);

}

// Display callback

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawAxis();

if (triangleDrawn)

drawTriangle();

glFlush();

}

// Menu callback

void menu(int option) {

lineType = option;

glutPostRedisplay();

}

// Mouse callback

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

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

triangleDrawn = true;

glutPostRedisplay();

}

}

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

glutInit(&argc, argv);

glutInitWindowSize(800, 800);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutCreateWindow("DDA Triangle Drawing");

init();

glutDisplayFunc(display);

glutMouseFunc(mouse);

// Right-click menu

glutCreateMenu(menu);

glutAddMenuEntry("Simple Line", 0);

glutAddMenuEntry("Dotted Line", 1);

glutAddMenuEntry("Dashed Line", 2);

glutAddMenuEntry("Solid Line", 3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

return 0;

}

//Bresenham’s Circle

#include<GL/glut.h>

#include <cstdlib>

#include<iostream>

using namespace std;

int r,x,y,x1,y1;

bool first=true;

void myInit(){

glClearColor(1.0,1.0,1.0,1.0);

glPointSize(3);

glViewport(0,0,800,800);

gluOrtho2D(0,800,0,800);

}

void line(int x1,int y1,int x2,int y2) {

glLineWidth(2);

glBegin(GL\_LINE\_LOOP);

glVertex2i(x1,y1);

glVertex2i(x2,y2);

glEnd();

glFlush();

}

void axis(){

glColor3f(0.0,0.0,1.0);

line(0,400,800,400);

line(400,0,400,800);

for(x=0;x<=800;x+=15) {

line(x,390,x,410);

}

for(y=0;y<=800;y+=15) {

line(390,y,410,y);

}

}

void draw\_circ(int x, int y, int x1, int y1) {

glBegin(GL\_POINTS);

glVertex2i(x1+x,y1+y);

glVertex2i(x1-x,y1+y);

glVertex2i(x1+x,y1-y);

glVertex2i(x1-x,y1-y);

glVertex2i(x1+y,y1+x);

glVertex2i(x1-y,y1+x);

glVertex2i(x1+y,y1-x);

glVertex2i(x1-y,y1-x);

glEnd();

glFlush();

}

void bres\_circ(int r,int x1,int y1){

int d;

d=3-2\*r;

x=0;

y=r;

do {

draw\_circ(x,y,x1,y1);

if(d<0) {

x++;

d=d+4\*x+6;

} else {

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

x++;

y--;

}

} while (x<=y);

int line\_x1 = 490;

int line\_x2 = 565;

int line\_y = 470;

line(line\_x1, line\_y, line\_x2, line\_y);

}

void myDisplay(){

glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

}

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

{

if(btn==GLUT\_LEFT\_BUTTON)

{

if(state==GLUT\_DOWN)

{

if(first){

axis();

glFlush();

first=false;

}

else{

axis();

glColor3f(0.0,1.0,0.0);

bres\_circ(100,130+400,120+400);

bres\_circ(20,485,540);

bres\_circ(20,565,540);

glFlush();

}

}

}

}

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

{

cout<<"Implementing Bresenham Circle to draw a Face";

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800,800);

glutInitWindowPosition(500,500);

glutCreateWindow("Bresenham Circle");

myInit();

glutDisplayFunc(myDisplay);

glutMouseFunc(mouse);

glutMainLoop();

return 1;

}

//2D Transformation

#include <iostream>

#include <cmath>

#include <GL/glut.h>

using namespace std;

int x1, x2, x3, x4, x5, x6, yy1, y2, y3, y4, y5, y6, nx1, nx2, nx3, nx4 , nx5 ,nx6 ,ny1, ny2, ny3, ny4, ny5, ny6, c;

float sx, sy, shx, shy, xt, yt, r, t;

void line(int x1, int y1, int x2, int y2){

glLineWidth(2);

glBegin(GL\_LINE\_LOOP);

glVertex2i(x1,y1);

glVertex2i(x2,y2);

glEnd();

glFlush();

}

void axis(){

glColor3f(0.0,0.0,0.0);

line(0,400,800,400);

line(400,0,400,800);

for(int x=0;x<=800;x+=20){

line(x,395,x,405);

}

for(int y=0;y<=800;y+=20){

line(395,y,405,y);

}

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

axis();

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

// glBegin(GL\_LINES);

// glVertex2i(-500, 0);

// glVertex2i(500, 0);

// glVertex2i(0, -500);

// glVertex2i(0, 500);

// glEnd();

glColor3f(0.0, 1.0, 0.0); // Green color for original Hexagon

glBegin(GL\_LINES);

glVertex2f(x1+400, yy1+400);

glVertex2f(x2+400, y2+400);

glVertex2f(x2+400, y2+400);

glVertex2f(x3+400, y3+400);

glVertex2f(x3+400, y3+400);

glVertex2f(x4+400, y4+400);

glVertex2f(x4+400, y4+400);

glVertex2f(x5+400, y5+400);

glVertex2f(x5+400, y5+400);

glVertex2f(x6+400, y6+400);

glVertex2f(x6+400, y6+400);

glVertex2f(x1+400, yy1+400);

glEnd();

glColor3f(1.0, 0.0, 0.0); // Red color for transformed Hexagon

glBegin(GL\_LINES);

glVertex2f(nx1+400, ny1+400);

glVertex2f(nx2+400, ny2+400);

glVertex2f(nx2+400, ny2+400);

glVertex2f(nx3+400, ny3+400);

glVertex2f(nx3+400, ny3+400);

glVertex2f(nx4+400, ny4+400);

glVertex2f(nx4+400, ny4+400);

glVertex2f(nx5+400, ny5+400);

glVertex2f(nx5+400, ny5+400);

glVertex2f(nx6+400, ny6+400);

glVertex2f(nx6+400, ny6+400);

glVertex2f(nx1+400, ny1+400);

glEnd();

glFlush();

}

void myinit() {

glClearColor(1.0, 1.0, 1.0, 1.0); // Set clear color to white

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0,800,0,800); // Set the coordinate system

}

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

cout << "Enter the coordinates of the Hexagon (x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6): ";

cin >> x1 >> yy1 >> x2 >> y2 >> x3 >> y3 >> x4 >> y4 >> x5 >> y5 >> x6 >> y6;

while (1) {

cout << "\n1.Translation\n2.Rotation\n3.Scaling\n4.Shear\n5.Exit\nEnter your choice: ";

cin >> c;

switch (c) {

case 1:

cout << "\nEnter the translation factor x: ";

cin >> xt;

cout << "\nEnter the translation factor y: ";

cin >> yt;

nx1 = x1 + xt;

ny1 = yy1 + yt;

nx2 = x2 + xt;

ny2 = y2 + yt;

nx3 = x3 + xt;

ny3 = y3 + yt;

nx4 = x4 + xt;

ny4 = y4 + yt;

nx5 = x5 +xt;

ny5 = y5 +yt;

nx6 = x6 +xt;

ny6 = y6 +yt;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 800);

glutInitWindowPosition(0, 0);

glutCreateWindow("Translation");

glutDisplayFunc(display);

myinit();

glutMainLoop(); // Trigger a display callback

break;

case 2:

cout << "\nEnter the angle of rotation: ";

cin >> r;

t = r \* 3.14 / 180; // Convert to radians

nx1 = (x1 \* cos(t) - yy1 \* sin(t));

ny1 = (x1 \* sin(t) + yy1 \* cos(t));

nx2 = (x2 \* cos(t) - y2 \* sin(t));

ny2 = (x2 \* sin(t) + y2 \* cos(t));

nx3 = (x3 \* cos(t) - y3 \* sin(t));

ny3 = (x3 \* sin(t) + y3 \* cos(t));

nx4 = (x4 \* cos(t) - y4 \* sin(t));

ny4 = (x4 \* sin(t) + y4 \* cos(t));

nx5 = (x5 \* cos(t) - y5 \* sin(t));

ny5 = (x5 \* sin(t) + y5 \* cos(t));

nx6 = (x6 \* cos(t) - y6 \* sin(t));

ny6 = (x6 \* sin(t) + y6 \* cos(t));

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 800);

glutInitWindowPosition(100, 100);

glutCreateWindow("Rotation");

glutDisplayFunc(display);

myinit();

glutMainLoop(); // Trigger a display callback

break;

case 3:

cout << "\nEnter the scaling factor x: ";

cin >> sx;

cout << "\nEnter the scaling factor y: ";

cin >> sy;

nx1 = x1 \* sx;

ny1 = yy1 \* sy;

nx2 = x2 \* sx;

ny2 = y2 \* sy;

nx3 = x3 \* sx;

ny3 = y3 \* sy;

nx4 = x4 \* sx;

ny4 = y4 \* sy;

nx5 = x5 \* sx;

ny5 = y5 \* sy;

nx6 = x6 \* sx;

ny6 = y6 \* sy;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 800);

glutInitWindowPosition(100, 100);

glutCreateWindow("Scaling");

glutDisplayFunc(display);

myinit();

glutMainLoop(); // Trigger a display callback

break;

case 4:

cout<<"\nEnter the shear factor x : ";

cin>>shx;

cout<<"\nEnter the shear factor y : ";

cin>>shy;

nx1=(x1+shx\*yy1);

nx2=(x2+shx\*y2);

nx3=(x3+shx\*y3);

nx4=(x4+shx\*y4);

nx5=(x5+shx\*y5);

nx6=(x6+shx\*y6);

ny1=(yy1+shy\*x1);

ny2=(y2+shy\*x2);

ny3=(y3+shy\*x3);

ny4=(y4+shy\*x4);

ny5=(y5+shy\*x5);

ny6=(y6+shy\*x6);

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(800, 800);

glutInitWindowPosition(100, 100);

glutCreateWindow("Shear");

glutDisplayFunc(display);

myinit();

glutMainLoop(); // Trigger a display callback

break;

case 5:

exit(0);

default:

cout << "Invalid choice!" << endl;

}

}

return 0;

}

//Polygon Filling

#include <GL/glut.h>

#include <iostream>

#include <unistd.h> // For sleep function

using namespace std;

int width = 800, height = 600;

float floodFillColor[3] = {1.0, 0.0, 0.0}; // Red Color

float boundaryFillColor[3] = {0.0, 0.0, 1.0}; // Blue Color

float borderColor[3] = {0.0, 0.0, 0.0}; // Black Border

float bgColor[3] = {1.0, 1.0, 1.0}; // White background

float titleColor[3] = {0.0, 0.0, 0.0}; // Blue for title

void setPixel(int x, int y, float color[3]) {

glBegin(GL\_POINTS);

glColor3fv(color);

glVertex2i(x, y);

glEnd();

glFlush();

usleep(1000); // Delay for visualization

}

void getPixelColor(int x, int y, float color[3]) {

glReadPixels(x, y, 1, 1, GL\_RGB, GL\_FLOAT, color);

}

void floodFill(int x, int y, float oldColor[3], float newColor[3]) {

float color[3];

getPixelColor(x, y, color);

if ((color[0] == oldColor[0] && color[1] == oldColor[1] && color[2] == oldColor[2]) &&

(color[0] != newColor[0] || color[1] != newColor[1] || color[2] != newColor[2])) {

setPixel(x, y, newColor);

floodFill(x + 1, y, oldColor, newColor);

floodFill(x - 1, y, oldColor, newColor);

floodFill(x, y + 1, oldColor, newColor);

floodFill(x, y - 1, oldColor, newColor);

}

}

void boundaryFill(int x, int y, float boundaryColor[3], float newColor[3]) {

float color[3];

getPixelColor(x, y, color);

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

(color[0] != newColor[0] || color[1] != newColor[1] || color[2] != newColor[2])) {

setPixel(x, y, newColor);

boundaryFill(x + 1, y, boundaryColor, newColor);

boundaryFill(x - 1, y, boundaryColor, newColor);

boundaryFill(x, y + 1, boundaryColor, newColor);

boundaryFill(x, y - 1, boundaryColor, newColor);

}

}

void drawTitle() {

// Draw "Flood Fill" text outline (left side)

glColor3fv(borderColor);

glBegin(GL\_LINE\_LOOP);

glVertex2i(50, 500);

glVertex2i(250, 500);

glVertex2i(250, 550);

glVertex2i(50, 550);

glEnd();

// Draw text inside using for loop instead of for-each

glColor3fv(titleColor);

glRasterPos2i(70, 520);

string text1 = "Flood Fill (Left Click)";

for (int i = 0; i < text1.length(); i++) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, text1[i]);

}

// Draw "Boundary Fill" text outline (right side)

glColor3fv(borderColor);

glBegin(GL\_LINE\_LOOP);

glVertex2i(350, 500);

glVertex2i(600, 500);

glVertex2i(600, 550);

glVertex2i(350, 550);

glEnd();

// Draw text inside using for loop instead of for-each

glColor3fv(titleColor);

glRasterPos2i(370, 520);

string text2 = "Boundary Fill (Right Click)";

for (int i = 0; i < text2.length(); i++) {

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, text2[i]);

}

glFlush();

}

void drawShape() {

// Draw a square

glColor3fv(borderColor);

glBegin(GL\_LINE\_LOOP);

glVertex2i(200, 200);

glVertex2i(300, 200);

glVertex2i(300, 300);

glVertex2i(200, 300);

glEnd();

// Draw a triangle

glBegin(GL\_LINE\_LOOP);

glVertex2i(400, 200);

glVertex2i(500, 200);

glVertex2i(450, 300);

glEnd();

glFlush();

}

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT);

drawTitle();

drawShape();

}

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

y = height - y; // Flip y coordinate to match OpenGL's coordinate system

// Check if click is within the title boxes

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

if (x >= 50 && x <= 250 && y >= 500 && y <= 550) {

// Flood fill the "Flood Fill" title

floodFill(100, 525, bgColor, floodFillColor);

} else {

// Flood fill the shape

float oldColor[3];

getPixelColor(x, y, oldColor);

if (oldColor[0] == bgColor[0] && oldColor[1] == bgColor[1] && oldColor[2] == bgColor[2]) {

floodFill(x, y, oldColor, floodFillColor);

}

}

}

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

if (x >= 350 && x <= 550 && y >= 500 && y <= 550) {

// Boundary fill the "Boundary Fill" title

boundaryFill(400, 525, borderColor, boundaryFillColor);

} else {

// Boundary fill the shape

boundaryFill(x, y, borderColor, boundaryFillColor);

}

}

}

void init() {

glClearColor(bgColor[0], bgColor[1], bgColor[2], 1.0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, width, 0, height);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(width, height);

glutCreateWindow("Flood Fill and Boundary Fill Demo");

glutDisplayFunc(display);

glutMouseFunc(mouse);

init();

glutMainLoop();

return 0;

}

//Moving Car

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Moving Car Animation</title>

    <!-- <link rel="stylesheet" href="styles.css"> -->

    <style>

      body {

        display: flex;

        justify-content: center;

        align-items: center;

        height: 100vh;

        margin: 0;

        background-color: #87ceeb; /\* Sky blue background \*/

      }

      .road {

        position: relative;

        width: 100%;

        height: 200px;

        background-color: #333; /\* Road color \*/

        overflow: hidden;

      }

      .car {

        position: absolute;

        bottom: 20px; /\* Positioning the car above the road \*/

        left: -150px; /\* Start position off-screen \*/

        width: 150px;

        height: 60px;

        animation: moveCar 5s linear infinite;

      }

      .car-body {

        position: relative;

        width: 100%;

        height: 100%;

        background-color: red; /\* Car color \*/

        border-radius: 10px 10px 5px 5px;

      }

      .roof {

        position: absolute;

        top: -20px;

        left: 20px;

        width: 110px;

        height: 40px;

        background-color: red;

        border-radius: 10px;

      }

      .window {

        position: absolute;

        top: -15px;

        width: 40px;

        height: 25px;

        background-color: lightblue;

        border-radius: 5px;

      }

      .window-front {

        left: 25px;

      }

      .window-back {

        right: 25px;

      }

      .wheel {

        position: absolute;

        bottom: -15px;

        width: 30px;

        height: 30px;

        background-color: black;

        border-radius: 50%;

      }

      .wheel-front {

        left: 20px;

      }

      .wheel-back {

        right: 20px;

      }

      /\* Animation \*/

      @keyframes *moveCar* {

        0% {

          left: -150px;

        }

        100% {

          left: 100%;

        }

      }

    </style>

  </head>

  <body>

    <div class="road">

      <div class="car">

        <div class="car-body">

          <div class="roof"></div>

          <div class="window window-front"></div>

          <div class="window window-back"></div>

        </div>

        <div class="wheel wheel-front"></div>

        <div class="wheel wheel-back"></div>

      </div>

    </div>

  </body>

</html>

//Bouncing Square

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Bouncing Square</title>

    <style>

      canvas {

        background-color: brown;

        border: 1px solid black;

      }

    </style>

  </head>

  <body>

    <canvas id="myCanvas" width="400" height="400"></canvas>

    <script>

      var canvas = document.getElementById("myCanvas");

      var ctx = canvas.getContext("2d");

      var x = canvas.width / 2;

      var y = canvas.height / 2;

      var dx = 2;

      var dy = -2;

      var squareSize = 50;

      function drawSquare() {

        ctx.clearRect(0, 0, canvas.width, canvas.height);

        ctx.fillStyle = "#0095DD";

        ctx.fillRect(x, y, squareSize, squareSize);

      }

      function updateSquarePosition() {

        if (x + dx > canvas.width - squareSize || x + dx < 0) {

          dx = -dx;

        }

        if (y + dy > canvas.height - squareSize || y + dy < 0) {

          dy = -dy;

        }

        x += dx;

        y += dy;

      }

      function animate() {

        updateSquarePosition();

        drawSquare();

        requestAnimationFrame(animate);

      }

      animate();

    </script>

  </body>

</html>

//Coffee Machine

Index.html

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Coffee Machine Animation</title>

    <link rel="stylesheet" href="style.css" />

  </head>

  <body>

    <div class="coffee-machine">

      <div class="machine-body">

        <div class="cup">

          <div class="coffee"></div>

        </div>

        <button id="brew-button">Brew Coffee</button>

      </div>

    </div>

    <script src="script.js"></script>

  </body>

</html>

Style.css

body {

    display: flex;

    justify-content: center;

    align-items: center;

    height: 100vh;

    background-color: #f3f3f3;

    margin: 0;

}

.coffee-machine {

    display: flex;

    flex-direction: column;

    align-items: center;

}

.machine-body {

    width: 200px;

    height: 300px;

    background-color: #6b4f4f;

    border-radius: 10px;

    position: relative;

    display: flex;

    flex-direction: column;

    justify-content: flex-end;

    align-items: center;

    padding-bottom: 20px;

}

.cup {

    width: 100px;

    height: 120px;

    background-color: #fff;

    border: 2px solid #6b4f4f;

    border-radius: 0 0 50px 50px;

    overflow: hidden;

    position: relative;

}

.coffee {

    width: 100%;

    height: 0;

    background-color: #3e2723;

    position: absolute;

    bottom: 0;

    transition: height 2s ease;

}

#brew-button {

    margin-top: 20px;

    padding: 10px 20px;

    background-color: #6b4f4f;

    color: #fff;

    border: none;

    border-radius: 5px;

    cursor: pointer;

    transition: background-color 0.3s;

}

#brew-button:hover {

    background-color: #5a3e3e;

}

Script.js

document.getElementById("brew-button").addEventListener("click", function () {

  const coffee = document.querySelector(".coffee");

  coffee.style.height = "100%";

  setTimeout(() => {

    coffee.style.height = "0";

  }, 3000); // Adjust the time to keep the coffee in the cup for a certain duration

});