Chavangkon Kanjanakuldit (Ken) 6681371

<https://github.com/Ken19149/Graphics/tree/main/week_3/shapes.cpp>

#define GL\_SILENCE\_DEPRECATION

#include <iostream>

#include <vector>

#include <cmath>

#include <set>

#ifdef \_\_APPLE\_\_

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

using namespace std;

int screenWidth = 800;

int screenHeight = 600;

vector<pair<int, int>> lines; // Store pairs of points for lines

vector<pair<pair<int, int>, int>> circles; // Store (center, radius)

vector<pair<int, int>> rectangles;

vector<vector<pair<int, int>>> shapes;

vector<pair<int, int>> temp\_shape;

string drawState = "line"; // l = line, c = circle, r = rectangle, s = shape

bool drawingCircle = false;

int circleCenterX, circleCenterY;

bool drawingShape = false;

void drawDot(int x, int y)

{

glBegin(GL\_POINTS);

glVertex2i(x, y);

glEnd();

}

// my function

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

{

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

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

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

int err = dx - dy;

int n = dy > dx ? dy : dx;

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

{

drawDot(x1, y1);

int e2 = 2 \* err;

if (e2 > -dy)

{

err -= dy;

x1 += sx;

}

if (e2 < dx)

{

err += dx;

y1 += sy;

}

// std::cout << x1 << " " << y1 << std::endl;

}

}

void drawCircleBresenham(int xc, int yc, int r)

{

int x = 0;

int y = r;

int pk = 3 - 2 \* r;

while (x <= y)

{

// Draw all eight symmetric points

drawDot(xc + x, yc + y); // Quadrant1

drawDot(xc + y, yc + x); // ... Quadrant2

drawDot(xc + y, yc - x); // ... Quadrant3

drawDot(xc + x, yc - y); // ... Quadrant4

drawDot(xc - x, yc - y); // ... Quadrant5

drawDot(xc - y, yc - x); // ... Quadrant6

drawDot(xc - y, yc + x); // ... Quadrant7

drawDot(xc - x, yc + y); // ... Quadrant8

x++;

if (pk < 0)

{

// pk ... case #1 <-- HERE

pk = pk + 4 \* x + 6;

}

else

{

y--;

// pk ... case #2 <-- HERE

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

}

}

}

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

{

// Draw all four sides using Bresenham line algorithm

drawLineBresenham(x1, y1, x2, y1); // Top

drawLineBresenham(x2, y1, x2, y2); // Right // <-- HERE

drawLineBresenham(x2, y2, x1, y2); // Bottom // <-- HERE

drawLineBresenham(x1, y2, x1, y1); // Left

// cout << "Rectangle drawn using line algorithm" << endl;

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0, 1.0, 1.0); // <-- HERE

// Draw all lines using Bresenham algorithm

for (size\_t i = 0; i + 1 < lines.size(); i += 2)

{

drawLineBresenham(lines[i].first, lines[i].second,

lines[i + 1].first, lines[i + 1].second);

}

for (const auto &circle : circles) // draw circles

{

drawCircleBresenham(circle.first.first, circle.first.second, circle.second);

}

for (size\_t i = 0; i + 1 < rectangles.size(); i += 2)

{

drawRectangle(rectangles[i].first, rectangles[i].second,

rectangles[i + 1].first, rectangles[i + 1].second);

}

for (size\_t i = 0; i < shapes.size(); i++)

{

for (size\_t j = 0; j + 1 < shapes[i].size(); j++)

{

drawDot(shapes[i][j].first, shapes[i][j].second);

drawLineBresenham(shapes[i][j].first, shapes[i][j].second, shapes[i][j + 1].first, shapes[i][j + 1].second);

}

}

glutSwapBuffers();

}

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

{

y = screenHeight - y;

if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN && drawState != "shape")

{

if (drawState == "line")

{

lines.push\_back({x, y});

cout << "Point at (" << x << ", " << y << ")" << endl;

if (lines.size() % 2 == 0)

{

cout << "Line drawn using Bresenham algorithm" << endl;

}

}

else if (drawState == "circle")

{

if (!drawingCircle)

{

circleCenterX = x;

circleCenterY = y;

drawingCircle = true;

cout << "Circle center at (" << x << ", " << y << ")" << endl;

}

else

{

int radius = (int)sqrt(pow(x - circleCenterX, 2) + pow(y - circleCenterY, 2));

circles.push\_back({{circleCenterX, circleCenterY}, radius});

drawingCircle = false;

cout << "Circle with radius: " << radius << " (Bresenham Algorithm)" << endl;

}

}

else if (drawState == "rectangle")

{

rectangles.push\_back({x, y});

cout << "Rectangle start at (" << x << ", " << y << ")" << endl;

if (rectangles.size() % 2 == 0)

{

cout << "Rectangle completed!" << endl;

}

}

else if (drawState == "shape")

{

temp\_shape.clear();

}

} else if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_UP && drawState == "shape")

{

shapes.push\_back(temp\_shape);

temp\_shape.clear();

std::cout << "success\n";

}

glutPostRedisplay();

}

void mouseMotion(int x, int y)

{

if (drawState == "shape")

{

temp\_shape.push\_back({x, screenHeight - y});

std::cout << x << " " << y << endl;

}

}

void keyboard(unsigned char key, int x, int y)

{

if (key == ' ')

{

lines.clear();

circles.clear();

rectangles.clear();

shapes.clear();

temp\_shape.clear();

cout << "Screen cleared" << endl;

}

else if (key == 'l')

{

drawState = "line";

}

else if (key == 'c')

{

drawState = "circle";

}

else if (key == 'r')

{

drawState = "rectangle";

}

else if (key == 's')

{

drawState = "shape";

}

else if (key == 'q' || key == 'Q')

{

exit(0);

}

glutPostRedisplay();

}

void init()

{

glClearColor(0.0, 0.0, 0.0, 1.0); // <-- HERE

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0, screenWidth, 0, screenHeight);

glPointSize(2.0); // <-- HERE

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(screenWidth, screenHeight);

glutInitWindowPosition(100, 100); // <-- HERE

glutCreateWindow("Midpoint Circle Algorithm");

init();

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutMotionFunc(mouseMotion);

glutKeyboardFunc(keyboard);

cout << "=== Bresenham Line Algorithm ===" << endl;

cout << "Click two points to draw a line" << endl;

cout << "Space - Clear Screen" << endl;

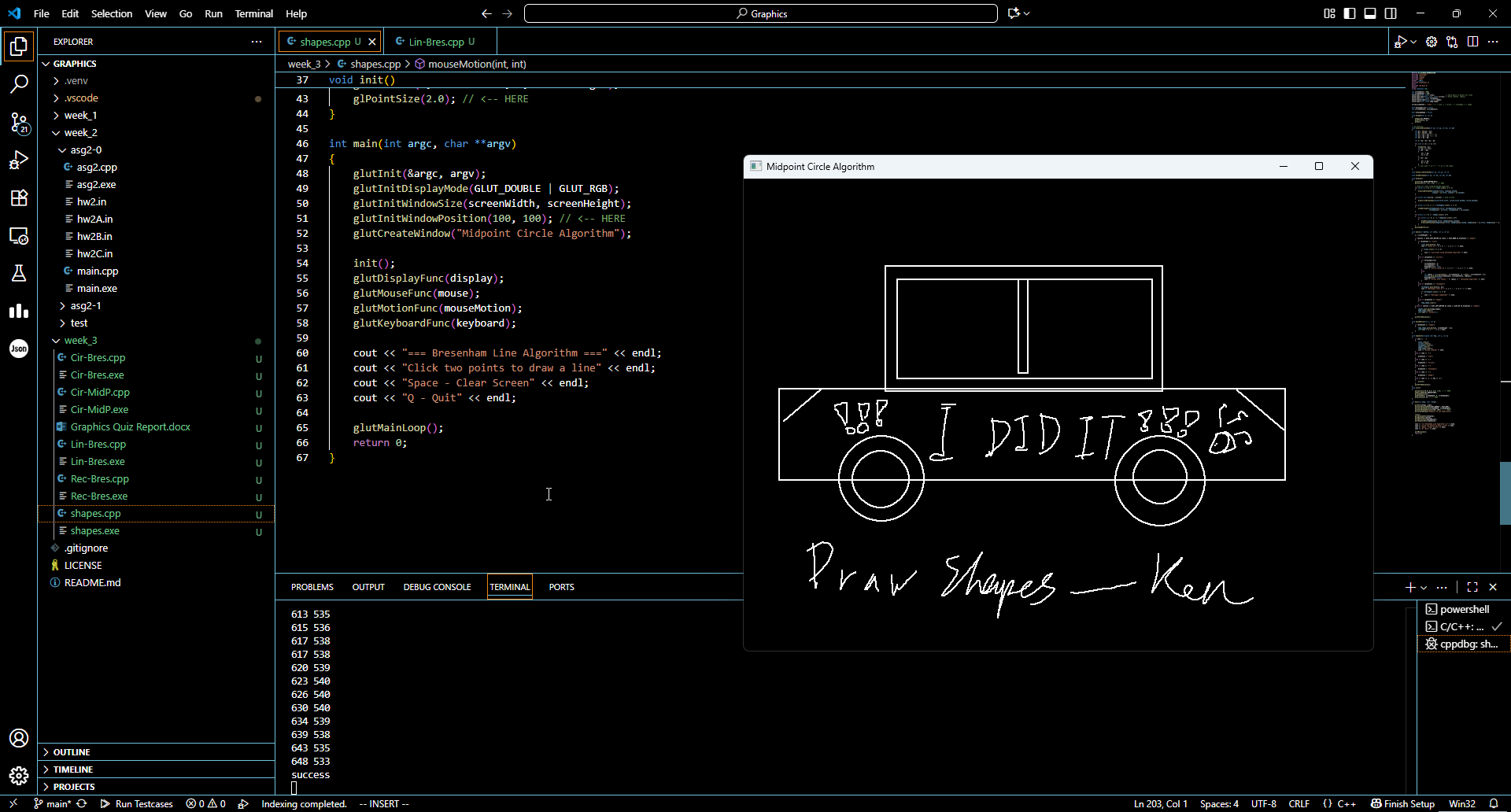
cout << "Q - Quit" << endl;

glutMainLoop();

return 0;

}

Output:



Button: l - lines, c - circles, r - rectangle, s - shapes, space - clear, q - quit

For the draw shapes I store data in a pair of int vectors inside a vector so I can store many lines.

Then I use the line drawing algorithm to trace each point to make it look smoother.