Chittagong University of Engineering & Technology

Assignment 6

Title: Draw a shape using C curve.

Name: Syed Tajir Hasnain

Id: 1604081

Session: 2020-21

Course code: 458

Course Title: Computer Graphics

Submitted to: Ms. Sabiha Anan,

Lecturer, Department of Computer Science & Engineering, CUET

GitHub Repository:

https://github.com/tajirhas9/opengl-practice/tree/main/assignment_6

Source Code:

- Files Structure
  - src/
    - main.cpp
      - Contains the main program
    - glib.h
      - Contains all the GLUT drawing utilities and algorithms.
      - Algorithms:
        - DDA Line Drawing Algorithm
        - Bresenham's Line Drawing Algorithm
        - Bresenham's Circle Drawing Algorithm
        - Midpoint Circle Algorithm
        - C Curve Algorithm
    - geometry.h
      - Contains Point class that defines the cartesian (x,y) points and their input and output definition.

geometry.h

```
#include <iostream>

namespace geo
{

	const double eps = 0.000000001;

	class Point
	{
	   public:
		Point() : x(0), y(0) {}
		Point(double x, double y) : x(x), y(y) {}
		double x, y;

		friend std::istream &operator>>(std::istream &input, Point &p)
		{
			input >> p.x >> p.y;
			return input;
		}

		friend std::ostream &operator<<(std::ostream &output, Point &p)
		{
			output << "(" << p.x << ", " << p.y << ")";
			return output;
		}
	};
}
```

`glib.h`

```
/**
* @author:                    Syed Tajir Hasnain
* @date:                      15/09/2021
* @project_details:           A GLUT utils header file
* @supported_operations:      1. initializes GLUT
*                             2. draw line with DDA algorithm
*                             3. draw line with Bresenham's Line Algorithm
*                             4. draw circle with Bresenham's Circle Algorithm
*                             5. draw circle with Midpoint Circle Algorithm
*/

#include <GL/glut.h>
#include <cmath>
#include <stdio.h>
#include <iostream>
#include "geometry.h"

namespace glib
{
	/**
	*
	* @utility:     Initializes GLUT library
	* @params:   takes the params supplied in the main() function
	*
	*/
	void init(int argc, char **argv)
	{
		glutInit(&argc, argv);
		glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
		glutInitWindowSize(500, 500);
		glutInitWindowPosition(100, 100);
		glutCreateWindow("");
		glClear(GL_COLOR_BUFFER_BIT);
		glClearColor(0, 0, 0, 0);
		glMatrixMode(GL_PROJECTION);
		glLoadIdentity();
		gluOrtho2D(-100, 100, -100, 100);
	}
	/**
	* @utility: takes the drawing callback and executes it
	*/
	void display(void (*callback)(void))
	{
		glutDisplayFunc(callback);
		glutMainLoop();
	}

	/**
```

```
* @algorithm: DDA Line Drawing Algorithm
*/
void drawLineDDA(geo::Point start, geo::Point end)
{
	glBegin(GL_LINES);
	std::cout << "drawing line from " << start << " ";
	std::cout << "to " << end << std::endl
	<< std::endl;

	double x, y, m, dx, dy;

	dx = end.x - start.x;
	dy = end.y - start.y;

	if (std::abs(dy) <= std::abs(dx))
	{
		x = start.x, y = start.y, dx = 1;
		m = dy / dx;
		while (std::abs(x - end.x) > geo::eps)
		{

			std::cout << "[plotting] (x,y): (" << x << "," << y << ")" <<
			std::endl;
			glVertex2d(x, y);
			x += (end.x - start.x >= 0 ? 1 : -1), y += m;
		}
	}
	else
	{
		x = start.x, y = start.y, dy = 1;
		m = dy / dx;
		while (std::abs(y - end.y) > geo::eps)
		{
			std::cout << "[plotting] (x,y): (" << x << "," << y << ")" <<
			std::endl;
			glVertex2d(x, y);
			x += 1 / m, y += (end.y - start.y >= 0 ? 1 : -1);
		}
	}
	glEnd();
	std::cout << "\n\n";
}

void drawLineBresenham(geo::Point start, geo::Point end) {

	// meeting assumption start < end

	if(start.x > end.x) {
		std::swap(start, end);
	}
```

```
std::cout << "drawing line from " << start << " ";
std::cout << "to " << end << std::endl
<< std::endl;

double dx,dy,inc1,inc2,d,x,y,xEnd, yEnd;
dx = end.x - start.x;
dy = end.y - start.y;

if(std::abs(dy) < std::abs(dx))
{
        inc1 = 2 * dy;
        inc2 = 2 * (dy - dx);
        d = inc1 - dx;

        if(dx <= 0) {
                x = end.x, y = end.y;
                xEnd = start.x;
        } else if(dx > 0) {
                x = start.x , y = start.y;
                xEnd = end.x;
        }
        std::cout << "[initial] (x,y): (" << x << "," << y << ")" << std::endl;
        glBegin(GL_LINES);
        for ( ; x <= xEnd;x++)
        {
                std::cout << "[plotting] (x,y): (" << x << "," << y << ")" <<
                std::endl;
                glVertex2d(x, y);

                if (d < 0) {
                        d += inc1;
                } else {
                        d += inc2;
                        y++;
                }
        }
        glEnd();
}
else
{
        inc1 = 2 * dx;
        inc2 = 2 * (dx - dy);
        d = inc1 - dy;

        if(dy <= 0) {
                x = end.x, y = end.y;
                yEnd = start.y;
        } else if(dy > 0) {
                x = start.x , y = start.y;
                yEnd = end.y;
        }
```

```
                std::cout << "[initial] (x,y): (" << x << "," << y << ")" << std::endl;
                glBegin(GL_LINES);
                for ( ; y <= yEnd;y++) {
                        std::cout << "[plotting] (x,y): (" << x << "," << y << ")" <<
                        std::endl;
                        glVertex2d(x, y);

                        if (d < 0) {
                                d += inc1;
                        } else {
                                d += inc2;
                        }
                }
                glEnd();
        }
        std::cout << "\n\n";
}

/**
* @algorithm: Breseham's Circle Algorithm
*/
void drawCircleBresenham(geo::Point c, double r)
{
        double x, y, d;
        x = 0, y = r;
        d = 3 - 2 * r;

        glBegin(GL_POINTS);
        while (std::abs(y - x) > geo::eps)
        {
                glVertex2i(c.x + x, c.y + y);
                glVertex2i(c.x + y, c.y + x);
                glVertex2i(c.x - y, c.y + x);
                glVertex2i(c.x - x, c.y + y);
                glVertex2i(c.x - x, c.y - y);
                glVertex2i(c.x - y, c.y - x);
                glVertex2i(c.x + y, c.y - x);
                glVertex2i(c.x + x, c.y - y);

                if (d < 0)
                {
                        d += 4 * x + 6;
                        x++;
                }
                else
                {
                        d += 4 * (x - y) + 10;
                        x++, y--;
                }
        }
        glEnd();
```

```
}

void drawCircleMidPoint(geo:: Point c, double r) {
	double x,y,p;

	x = 0, y = r;
	p = 1 - r;

	glBegin(GL_POINTS);

	while(x <= y) {
		glVertex2i(c.x + x, c.y + y);
		glVertex2i(c.x + y, c.y + x);
		glVertex2i(c.x - x, c.y + y);
		glVertex2i(c.x - y, c.y + x);
		glVertex2i(c.x + x, c.y - y);
		glVertex2i(c.x + y, c.y - x);
		glVertex2i(c.x - x, c.y - y);
		glVertex2i(c.x - y, c.y - x);
		if(p < 0) {
			p += 2 * x + 3;
			x++;
		} else {
			p += 2 * (x-y)+5;
			x++, y--;
		}
	}

	glEnd();
}

void drawEllipseMidpoint(geo::Point c, double a, double b)
{
	int x = 0, p, aa, bb, aa2, bb2, fx = 0, fy, y = b;

	aa = a * a;
	bb = b * b;
	aa2 = 2 * aa;
	bb2 = 2 * bb;
	fy = aa2 * b;

	p = bb - (aa * b) + (0.25 * aa);
	while (fx < fy)
	{
		glBegin(GL_POINTS);
		glVertex2i(c.x + x, c.y + y);
		glVertex2i(c.x - x, c.y + y);
		glVertex2i(c.x - x, c.y - y);
		glVertex2i(c.x + x, c.y - y);
		glEnd();
		x++;
```

```
                fx += bb2;
                if (p < 0)
                {
                        p += fx + bb;
                }
                else
                {
                        y--;
                        fy -= aa2;
                        p += fx + bb - fy;
                }
        }
        p = (bb * (x + 0.5) * (x + 0.5)) + (aa * (y - 1) * (y - 1)) - (aa * bb);
        while (y > 0)
        {
                glBegin(GL_POINTS);
                glVertex2i(c.x + x, c.y + y);
                glVertex2i(c.x - x, c.y + y);
                glVertex2i(c.x - x, c.y - y);
                glVertex2i(c.x + x, c.y - y);
                glEnd();
                y--;
                fy -= aa2;
                if (p >= 0)
                {
                        p -= fy + aa;
                }
                else
                {
                        x++;
                        fx += bb2;
                        p += fx - fy + aa;
                }
        }
}

// c curve
void c_curve(geo::Point a, double length, double alpha, int n, geo::Point &_a)
{
        std::cout << a << " " << length << " " << alpha << " " << n << " " << _a
        << std::endl;
        if(n>0)
        {
                int extra = 36;
                length = length/sqrt(2.0);
                c_curve(a, length, alpha+extra, n-1, _a);
                a.x += length*cos((alpha+extra) * M_PI/180);
                a.y += length*sin((alpha+extra) * M_PI/180);
                c_curve(a, length, alpha-extra, n-1, _a);
        }
```

```
        else
        {
                glBegin(GL_LINES);
                glVertex2d(a.x, a.y);
                _a.x = a.x + (length*cos(alpha * M_PI/180));
                _a.y = a.y + (length*sin(alpha * M_PI/180));
                glVertex2d(a.x + (length*cos(alpha * M_PI/180)), a.y+
                (length*sin(alpha * M_PI/180)));
                glEnd();
                glFlush();
        }
}


        // make sure to flush everytime
        inline void close()
        {
                glFlush();
        }
}
```

`main.cpp`

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

void input(geo::Point &a, double &alpha, double &length)
{
        cout << "Enter starting co-ordinate: ";
        cin >> a;
        cout << "Enter rotation and length : ";
        cin >> alpha >> length;
}

void drawShape(void)
{
        geo::Point a, _a;
        double alpha, length;

        // Draw the rectangle
        input(a, alpha, length);
        cout << a << " " << alpha << " " << length << endl;
        int cnt = 0;
        glib::c_curve(a, length, alpha, 1, _a);
        while (cnt != 4)
        {
        alpha += 72;
        glib::c_curve(_a, length, alpha, 1, _a);
        cnt++;
        }

        glib::close();
}

int main(int argc, char **argv)
{
        glib::init(argc, argv);
        glib::display(drawShape);

        return 0;
}
```

Sample input:

Enter starting co-ordinate: -25 -25

Enter rotation and length : 0 50

Sample Output: