Chittagong University of Engineering & Technology

Assignment 4

Title: Draw a shape using DDA and Bresenham's circle Algorithm.

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:

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

    - geometry.h

      - Contains Point class that defines the cartesian (x,y) points and their input and output definition.

`geometry.h`

```cpp
#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`

```cpp
/**
* @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();
    }

    /**
```

```cpp
 * @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);
        }
```

```cpp
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;
        }
```

```cpp
                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();
```

```cpp
        }

        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();
        }

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

`main.cpp`

```cpp
/**
 * @author:              Syed Tajir Hasnain
 * @date:                15/09/2021
 * @assignment_no:       4
 * @assignment_title:    Draw a shape using DDA and Bresenham's circle
Algorithm.
 *
 * @ps:                  1. `class Point` is declared in the "geometry.h" header
file under `geo` namespace
 *                       2. DDA algorithm is implemented inside "glib.h" header
file under `glib` namespace
 *
 * @procedure:           1. initialize glut with glib::init() function
 *                       2. input the 4 points [a,b,c,d] of the rectangle.
 *                       3. draw the line from a to b
 *                       4. draw line from b to c
 *                       5. draw line from c to d
 *                       6. draw line from d to a
 *                       7. end drawing and flush.
 */

#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "glib.h"

void inputRectangle(geo::Point &a,
                    geo::Point &b,
                    geo::Point &c,
                    geo::Point &d)
{

    std::cout << "point a: ";
    std::cin >> a;

    std::cout << "point b: ";
    std::cin >> b;

    std::cout << "point c: ";
    std::cin >> c;

    std::cout << "point d: ";
    std::cin >> d;
}

void inputCircle(geo::Point &c, double &r)
```

```cpp
{
    std::cout << "center: ";
    std::cin >> c;

    std::cout << "radius: ";
    std::cin >> r;
}

void drawShape(void)
{
    geo::Point a, b, c, d, center;
    double radius;

    // Draw the rectangle
    inputRectangle(a, b, c, d);
    glib::drawLineBresenham(a, b);
    glib::drawLineBresenham(b, c);
    glib::drawLineBresenham(c, d);
    glib::drawLineBresenham(d, a);

    // Draw the circle
    inputCircle(center, radius);
    glib::drawCircleMidPoint(center, radius);

    glib::close();
}

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

    glib::display(drawShape);

    return 0;
}
```

Sample input:

point a: -50 -50

point b: 50 -50

point c: 50 50

point d: -50 50

Sample Output: