Graphics Assignment No: 3

Shikhar Jaiswal
1601CS44

February 10, 2019

1 Assignment Description

Implement Scan-Line Polygon Filling algorithm to draw and fill the provided figure using its coordinates.

2 Procedure

Installation

Install the following packages from the Ubuntu repository:
• freeglut3-dev
• mesa-common-dev

```
sudo apt-get install freeglut3 freeglut3-dev mesa-common-dev
```

Check your $/usr/include/GL$ folder to verify the installation of the OpenGL headers that you intend to use.

Compiling and Linking

We will have to use the $-lglut$ linker option with $gcc/g++$ to compile a program with *glut* library.

For example, to compile the program, use the following to get the binary executable code:

```
g++ polygon_fill.cpp -lGL -lGLU -lglut -o polygon_fill
```

3 Discussion

The primary objective of the assignment is to implement an algorithm capable of filling up the space inside of a polygon with colour. For this purpose, Scan Line Polygon Filling algorithm is implemented. The algorithm works by tracing individual pixels between two edges of the polygon for every scan line, and consequently colouring all the pixels between the two determined pixels. For a detailed explanation, please refer the code.

Polygon Coordinates - Scaled By 32

```
2 3 -> 64 96
7 1 -> 224 32
13 6 -> 416 192
13 11 -> 424 352
7 7 -> 224 224
2 10 -> 64 320
2 3 -> 64 96
```

OpenGL Code

```
#include <iostream>
#include <fstream>
#include <tuple>
#include <vector>
#include <algorithm>
#include <GL/glut.h>

using namespace std;

// Global variables for window size, edge table
// and active edge list.
int max_height = 800, max_width = 600;
typedef tuple<int, double, double> bucket;
vector<vector<bucket>> edge_table(max_height);
vector<bucket> active_edge_list;

// Function for printing a particular set of tuples.
void print_tuples(vector<bucket> &v)
{
    if (v.size())
    {
        cout << "Count: " << v.size() << endl;
    }
    for (auto i : v)
    {
        cout << "y: " << get<0>(i) << ", x (intersection): "
             << get<1>(i) << ", dx/dy: " << get<2>(i) << endl;
    }
}

// Function for printing the whole edge list.
void print_table()
{
    int count = 0;
    cout << endl << "Printing Edge Table:" << endl;
    for (auto &i : edge_table)
    {
        if (i.size())
        {
            cout << "Scanline: " << count << endl;
        }
        print_tuples(i);
        count++;
    }
}

// Function for conditional sorting of tuples.
bool sorter(const tuple<int, double, double> &a,
            const tuple<int, double, double> &b)
{
    return (get<1>(a) < get<1>(b));
}

// Function for inserting a tuple in a vector.
void add_edge(vector<bucket> &v, int y, int x, double dx_dy)
{
    v.push_back(make_tuple(y, static_cast<double>(x), dx_dy));
    sort(v.begin(), v.end(), sorter);
}

// Function for initializing the entire edge list
// from the given list of coordinates.
void store_edge_table(int x1, int y1, int x2, int y2)
{
    double dx_dy;

    if (y2 == y1)
    {
        return ;
    } else if (x2 == x1)
    {
        dx_dy = 0.0;
    } else
    {
        dx_dy = (static_cast<double>(x2 - x1)) / (y2 - y1);
        cout << "Slope string for " << x1 << ", " << y1
             << " and " << x2 << ", " << y2 << ": " << dx_dy
             << endl;
    }

    if (y1 > y2)
    {
        add_edge(edge_table[y2], y1, x2, dx_dy);
    } else
    {
        add_edge(edge_table[y1], y2, x1, dx_dy);
    }
}

// Function for removing a tuple from the active edge list.
void remove_edge(vector<bucket> &v, int y)
{
    v.erase(remove_if(v.begin(), v.end(),
            [&](const bucket b)->bool{ return get<0>(b) == y; }),
            v.end());
}

// Function for updating the intersection coordinate
// with respect to scan line.
void update_active_edge_list()
{
    for (auto &i : active_edge_list)
    {
        get<1>(i) += get<2>(i);
    }
}

// Function for filling the individual pixels between two
// intersection coordinates of the edges of the polygon
// with respect the scan line.
void scanline_fill()
{
    for (int i = 0; i < max_height; i++)
    {
        for (auto j : edge_table[i])
        {
```

```
        add_edge(active_edge_list, get<0>(j),
                 get<1>(j), get<2>(j));
    }
    print_tuples(active_edge_list);
    remove_edge(active_edge_list, i);
    sort(active_edge_list.begin(), active_edge_list.end(),
         sorter);
    print_tuples(active_edge_list);

    bool flag = false;
    int count = 0;
    int x1 = 0, x2 = 0, y1 = 0, y2 = 0;

    for (auto j : active_edge_list)
    {
        if (count % 2 == 0)
        {
            x1 = static_cast<int>(get<1>(j));
            y1 = get<0>(j);

            if (x1 == x2)
            {
                if (((x1 == y1) and (x2 != y2)) or
                    ((x1 != y1) and (x2 == y2)))
                {
                    x2 = x1;
                    y2 = y1;
                } else
                {
                    count++;
                }
            } else
            {
                count++;
            }
        } else
        {
            x2 = static_cast<int>(get<1>(j));
            y2 = get<0>(j);
            flag = false;
            if (x1 == x2)
            {
                if (((x1 == y1) and (x2 != y2)) or
                    ((x1 != y1) and (x2 == y2)))
                {
                    x1 = x2;
                    y1 = y2;
                } else
                {
                    count++;
                    flag = true;
                }
            } else
            {
                count++;
                flag = true;
            }

            if (flag)
            {
                glColor3ub(255, 255, 255);
                glBegin(GL_LINES);
                    glVertex2i(x1, i);
                    glVertex2i(x2, i);
                glEnd();
                glFlush();
            }
        }
    }
    update_active_edge_list();
  }
  cout << endl << "Scanline Filling Complete" << endl;
}

// Function to read the file with polygon coordinates
// and insert them into the edge table, and drawing the polygon.
void draw_polygon(void)
{
    ifstream polygon;
    polygon.open("polygon.txt");

    if (not polygon.is_open())
    {
        cerr << "The File Cannot Be Read!" << endl;
        return ;
    }
    // Colour fill.
    glColor3ub(255, 255, 255);
    // Set point sizes.
    glPointSize(2.0);

    int x1, x2, y1, y2;
```

```cpp
    polygon >> x1 >> y1;

    cout << "Storing Edges In Table:" << endl;
    while (polygon >> x2 >> y2)
    {
        glBegin(GL_LINES);
            glVertex2i(x1, y1);
            glVertex2i(x2, y2);
        glEnd();
        store_edge_table(x1, y1, x2, y2);
        glFlush();
        x1 = x2;
        y1 = y2;
    }

    polygon.close();
}

// Function to display the drawn and filled polygon.
void display()
{
    draw_polygon();
    print_table();
    scanline_fill();
}

int main(int argc, char **argv)
{
    // Initialize to the command-line arguments.
    glutInit(&argc, argv);
    // Setup the colour depth of the window buffers.
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

    // Assign the position, size and name to the window.
    glutInitWindowPosition(100, 150);
    glutInitWindowSize(max_height, max_width);
    glutCreateWindow("Scan Line Polygon Filling Algorithm");

    // Setup a black background.
    glClearColor(0.0, 0.0, 0.0, 0.0);
    // Setup viewing projection.
    glMatrixMode(GL_PROJECTION);
    // Initialize identity matrix.
    glLoadIdentity();
    // Setup a viewport.
    gluOrtho2D(-150.0, 650.0, -150.0, 450.0);
    // Set the display area colour set using glClearColor().
    glClear(GL_COLOR_BUFFER_BIT);

    // Pass the display function to generate the display.
    glutDisplayFunc(display);
    // Hand over the execution to the glut library.
    glutMainLoop();

    return 0;
}
```

Python Code

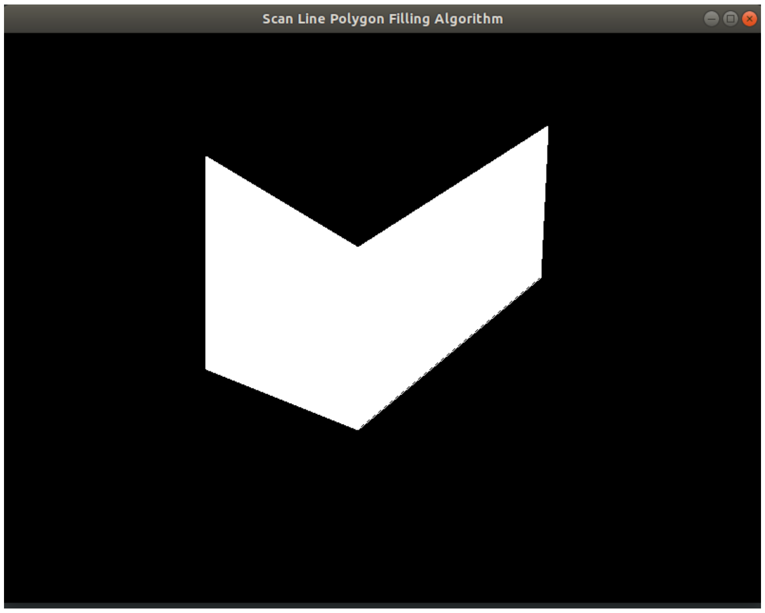
```
# Using turtle graphics library.
import turtle

# Set the window, edge table and active edge list sizes.
max_height = 1600
max_width = 1200
edge_table = [[] for i in range(max_height)]
active_edge_list = []

# Function for printing the tuples.
def print_tuples(buckets):
    if len(buckets) > 0:
        print("Count: " + str(len(buckets)))

    for i in buckets:
        print("y: " + str(i[0]) + ", x (intersection): " +
            str(i[1]) + ", dx/dy: " + str(i[2]))

# Function for printing the edge tables.
def print_table():
    count = 0
    print("Printing Edge Table:")
    for i in edge_table:
        if len(i) > 0:
            print("Scanline: " + str(count))
        print_tuples(i)
        count += 1

# Function for adding an edge to the list of buckets.
def add_edge(buckets, y, x, dx_dy):
    buckets.append([y, x, dx_dy])
    buckets.sort(key = lambda k : k[1])
```

```
# Function for removing and edge from the active edge list.
def remove_edge(buckets, y):
    i = 0
    while i < len(buckets):
        if buckets[i][0] == y:
            del buckets[i]
        else:
            i += 1

# Function for updating the active edge list.
def update_active_edge_list():
    for i in active_edge_list:
        i[1] += i[2]

# Function for storing the edges in the edge list.
def store_edge_table(x1, y1, x2, y2):
    if y1 == y2:
        return
    elif x2 == x1:
        dx_dy = 0.0
    else:
        dx_dy = (x2 - x1) / (y2 - y1)
        print("Slope string for " + str(x1) + ", " + str(y1) +
        " and " + str(x2) + ", " + str(y2) + ": " + str(dx_dy))

    if y1 > y2:
        add_edge(edge_table[y2], y1, x2, dx_dy)
    else:
        add_edge(edge_table[y1], y2, x1, dx_dy)

# Function for filling the individual pixels between two
# intersection coordinates of the edges of the polygon
# with respect the scan line.
def scanline_fill():
    for i in range(max_height):
        for j in edge_table[i]:
            add_edge(active_edge_list, j[0], j[1], j[2])
        print_tuples(active_edge_list)
        remove_edge(active_edge_list, i)
        active_edge_list.sort(key = lambda k : k[1])
        print_tuples(active_edge_list)

        count = 0
        x1 = 0
        x2 = 0
        y1 = 0
        y2 = 0
        flag = False

        for j in active_edge_list:
            if count % 2 == 0:
                x1 = int(j[1])
                y1 = int(j[0])
                if x1 == x2:
                    if (((x1 == y1) and (x2 != y2)) or
                        ((x1 != y1) and (x2 == y2))):
                        x2 = x1
                        y2 = y1
                    else:
                        count += 1
                else:
                    count += 1
            else:
                x2 = int(j[1])
                y2 = int(j[0])
                flag = False

                if x1 == x2:
                    if (((x1 == y1) and (x2 != y2)) or
                        ((x1 != y1) and (x2 == y2))):
                        x1 = x2
                        y1 = y2
                    else:
                        flag = True
                        count += 1
                else:
                    flag = True
                    count += 1

                if flag == True:
                    turtle.pencolor("white")
                    turtle.goto(x1, i)
                    turtle.pendown()
                    turtle.goto(x2, i)
                    turtle.penup()

        update_active_edge_list()
    print("Scanline Filling Complete")

# Function to read the file with polygon coordinates
# and insert them into the edge table, and drawing the polygon.
```

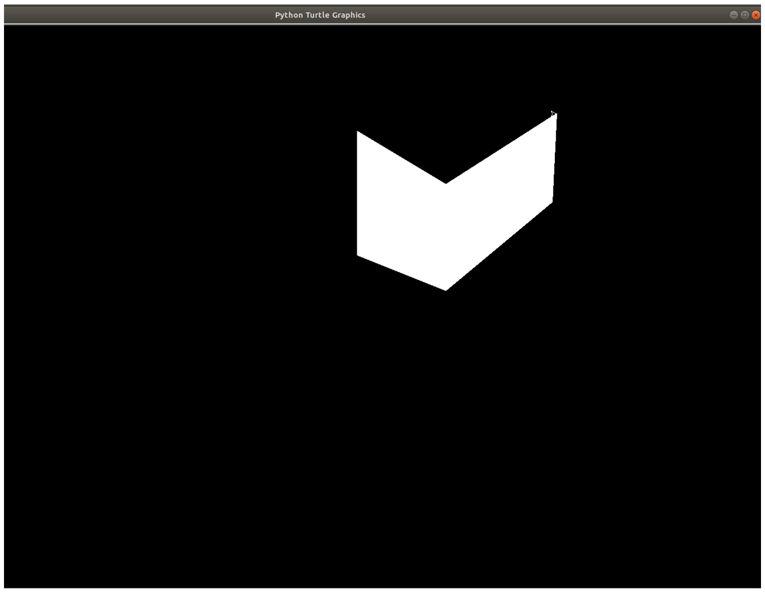
```python
def draw_polygon():
    with open('polygon.txt') as f:
        x1, y1 = [int(x) for x in next(f).split()]
        print("Storing Edges In Table:")
        for line in f:
            x2, y2 = [int(x) for x in line.split()]
            turtle.penup()
            turtle.pencolor("white")
            turtle.goto(x1, y1)
            turtle.pendown()
            turtle.goto(x2, y2)
            turtle.penup()
            store_edge_table(x1, y1, x2, y2)
            x1 = x2
            y1 = y2

# Function to display the drawn and filled polygon.
def display():
    draw_polygon()
    print_table()
    scanline_fill()

# Initial input.
print("Scan Line Polygon Filling Algorithm\n")

# Initialization and background colour.
turtle.setup(max_height, max_width)
turtle.bgcolor("black")

# Set the fill colour to black.
turtle.fillcolor("black")
# Initiate the algorithm.
display()

# Exit on click.
turtle.exitonclick()
```

4 Result

OpenGL



Python



5 References

[1] How to install OpenGL/GLUT libraries

[2] An Introduction to OpenGL Programming

[3] Turtle Graphics - The Python Standard Library

[4] Hackernoon - Scan Line Polygon Filling Algorithm