МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Отчет по лабораторной работе №2

По дисциплине «Компьютерная геометрия и графика»

Изучение алгоритмов отрисовки растровых линий с применением сглаживания и гамма-коррекции

Выполнил студент группы №M3101:

Пантелеев Ярослав Кириллович

Преподаватель:

Скаков Павел Сергеевич

САНКТ-ПЕТЕРБУРГ

2020

Цель работы:

Изучить алгоритмы и реализовать программу, рисующую линию на изображении в формате PGM (P5) с учетом гамма-коррекции sRGB.

Описание работы:

Программа должна поддерживать серые изображения (PNM P5), самостоятельно определяя формат по содержимому, быть написана на языке C/C++ и не использовать внешние библиотеки.

Аргументы программе передаются через командную строку:

program.exe <имя_входного_файла> <имя_выходного_файла> <яркость_линии> <толщина_линии> <x_начальный> <y_начальный> <x_конечный> <y_конечный> <гамма>

где:

- <яркость_линии>: целое число 0..255;
- <толщина_линии>: положительное дробное число;
- <x, y>: координаты внутри изображения, (0;0) соответствует левому верхнему углу, дробные числа (целые значения соответствуют центру пикселей).
- <гамма>: (optional)положительное вещественное число: гамма-коррекция с введенным значением в качестве гаммы. При его отсутствии используется sRGB.

Теоретическая часть:

Алгоритм Брезенхэма:

Алгоритм заключается в том, что между начальной и конечной точкой проводится векторная линия. Затем цикл проходится по координате с большей проекцией, смещая вторую координату в зависимости от ошибки. Ошибкой в данном алгоритме является расстояние от центра рассматриваемого пикселя до линии. Алгоритм не поддерживает сглаживание.

Алгоритм Ву

Как и в алгоритме Брезенхэма между начальной и конечной точкой проводится векторная линия. Алгоритм Ву, однако, заключается в прохождении циклом по координате с большей проекцией, при этом на каждом шаге рисуется два пикселя, прозрачность которых зависит от

удаленности их центров от векторной линии. Таким образом в сумме яркость этих пикселей даст 1, что позволяет создать видимость толщины 1, при этом сгладив её.

Алгоритм Мёрфи

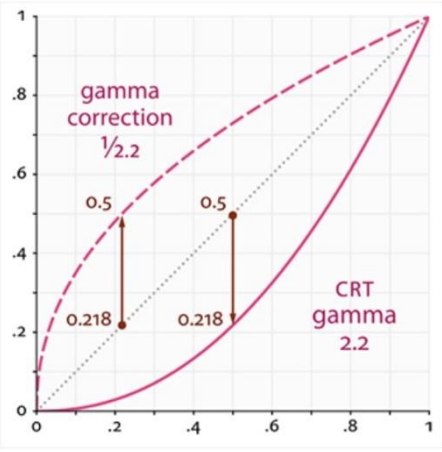
Алгоритм Алана Мёрфи представляет собой модификацию алгоритма Брезенхеэма для рисования толстых линий. Идея алгоритма заключается в рисовании перпендикулярных линий из каждой точки линии Брезенхэма с применением коррекции толщины.

Примененный в работе алгоритм

В силу того, что по заданию необходимо нарисовать и толстые и сглаженные линии, после нескольких неуспешных попыток реализации алгоритмов Брезенхэма и Мёрфи, было принято решение взять придуманное для Мёрфи сглаживание, идея которого будет описана ниже, и применить его для всех пикселей на части картинки, в которую с вероятностью 100% попадает линия, используя вложенный цикл. Идея же сглаживания заключается в следующем: в силу того, что в действительности мы лишь растеризуем толстую линию (векторный прямоугольник), непрозрачность пикселя можно считать, как его площадь, покрытую тем самым прямоугольником. Таким образом для пикселей внутри линии она будет равна 1, а для пикселей на краях - отношению пересечения двух прямоугольников (пикселя и линии соответственно) к площади пикселя.

Гамма-коррекция

Гамма-коррекция (иногда — гамма) — предыскажения яркости чёрно-белого или цветоделённых составляющих цветного изображения при его записи. В качестве передаточной функции при гамма-коррекции чаще всего используется степенная в виде $V_{out} = V_{in}^{\gamma}$. Идея гамма-коррекции заключается в том, чтобы применить инверсию гаммы монитора к окончательному цвету перед выводом на монитор (записью в файл), чтобы картинка представлялась пользователю в корректном виде.



Здесь:

- Gamma correction – записанные в файле данные
- CRT gamma – гамма монитора – искажения им накладываемые
- Пунктирная линия – то, что увидит пользователь

sRGB является стандартом представления цветового спектра с использованием модели RGB. sRGB создан для унификации использования модели RGB в мониторах, принтерах и Интернет-сайтах.

sRGB использует основные цвета, описанные стандартом BT.709, аналогично студийным мониторам и HD-телевидению, а также гамма-коррекцию, аналогично мониторам с электронно-лучевой трубкой. Такая спецификация позволила sRGB в точности отображаться на обычных CRT-мониторах и телевизорах, что стало в своё время основным фактором, повлиявшим на принятие sRGB в качестве стандарта.

Экспериментальная часть

Язык выполнения работы: C++17, компилятор Microsoft Visual C++.

Этапы работы программы:

1) Чтение файла картинки (хедера и информации о цвете каждого пикселя) с обработкой ошибок чтения
2) Нахождение четырех граничных координат линии, которую будем рисовать (находим «среднюю линию» прямоугольника данной линии и проводим с каждого конца по два перпендикуляра к этой линии в разные стороны длиной thickness/2)
3) Проходим по всем пикселям изображения и проверяем принадлежность его линии в процентах – этот коэффициент и является коэффициентом яркости цвета, который будет наложен поверх цвета данного пикселя

Вывод:

В ходе лабораторной работы был реализован собственный алгоритм растрирования линий произвольной толщины, соответствующий всему необходимому функционалу, указанному в техническом задании к лабораторной работе. Выполнение данной лабораторной работы позволило узнать об алгоритмах рисования прямых линий со сглаживанием.

Листинг исходного кода:

main.cpp

```
#include <string>
#include "Image.h"


int main(int argc, char *argv[]) {
    if(argc != 9 && argc != 10) {
        std::cerr << "Invalid command line arguments!" << "\n";
        return 1;
    }
    const std::string fin = std::string(argv[1]);
    const std::string fout = std::string(argv[2]);
    double thickness, x_s, y_s, x_f, y_f, gamma = 2.4;
    bool SRGB = true;
    int brightness;
    try {
        brightness = atoi(argv[3]);
        thickness = std::stod(argv[4]);
        x_s = std::stod(argv[5]);
        y_s = std::stod(argv[6]);
        x_f = std::stod(argv[7]);
        y_f = std::stod(argv[8]);
    }
    catch (const std::exception& error) {
        std::cerr << error.what() << "\n";
        return 1;
    }
    if(argc == 10) {
        try {
            gamma = std::stod(argv[9]);
            SRGB = false;
        }
        catch (const std::exception& error) {
            std::cerr << error.what() << std::endl;
            return 1;
        }
    }


    Image* img;
    try {
        img = new Image(fin);
    }
    catch (const std::exception& error) {
        std::cerr << error.what() << "\n";
        return 1;
    }


    img->draw_line(Point{ x_s, y_s }, Point{ x_f, y_f }, thickness, brightness, gamma, SRGB);
    try {
        img->write(fout);
    }
    catch (const std::exception& error) {
        std::cerr << error.what() << "\n";
        delete img;
        return 1;
    }
    delete img;
    return 0;
}
```

Image.h

```
#pragma once
#include <vector>
#include <string>

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

        class Image{
        public:
            Image(std::string);
            void write(std::string);
            void draw_line(Point, Point, double, int, double, bool);
        private:
            int width;
            int height;
            int color_depth;
            std::vector<std::vector<unsigned char>> image;
            void plot(int, int, double, int, double, bool);
        };
```

Image.cpp

```
#include "Image.h"
#include <cmath>
#include <algorithm>
#include <fstream>

double sqr(double a) {
    return a * a;
}
double distance(Point p1, Point p2) {
            return sqrt(sqr(p1.x - p2.x) + sqr(p1.y - p2.y));
        }
        double triangle_square(Point A, Point B, Point C) {
            double a = distance(A, B);
            double b = distance(B, C);
            double c = distance(C, A);
            double p = (a + b + c) / 2;
            return sqrt(p * (p - a) * (p - b) * (p - c));
        }
        bool point_in_rectangle(Point A, Point B, Point C, Point D, Point P) {
            const double rectangle_h = distance(A, B);
            const double rectangle_w = distance(A, C);
            const double rectangle_square = rectangle_h * rectangle_w;
            const double possible_size = triangle_square(P, A, B) + triangle_square(P, A,
C) + triangle_square(P, C, D) + triangle_square(P, D, B);
            return fabs(rectangle_square - possible_size) < 1e-5;
        }
        double intersection_square(Point A, Point B, Point C, Point D, double X, double Y) {
            if (point_in_rectangle(A, B, C, D, Point{ X, Y }) &&
                point_in_rectangle(A, B, C, D, Point{ X + 1.0, Y * 1.0 }) &&
                point_in_rectangle(A, B, C, D, Point{ X * 1.0, Y + 1.0 }) &&
                point_in_rectangle(A, B, C, D, Point{ X + 1.0, Y + 1.0 })) {
                return 1.0;
            }
            int ins = 0;
            for(double i = X + 0.1; i + 0.1 <= X + 1; i += 0.1)
                for(double j = Y + 0.1; j + 0.1 <= Y + 1; j += 0.1)
                    if(point_in_rectangle(A, B, C, D, Point{i, j}))
```

```
37.                         ins++;
38.             return ins * 1.0 / 100;
39.         }
40.
41.         Point::Point(double x, double y) {
42.             this->x = x;
43.             this->y = y;
44.         }
45.
46.         Image::Image(std::string filename) {
47.             std::ifstream fin(filename, std::ios::binary);
48.             if(!fin.is_open())
49.                 throw std::runtime_error("failed to open file");
50.
51.             char ch[2];
52.             fin >> ch[0] >> ch[1];
53.             if(ch[0] != 'P' || ch[1] != '5')
54.                 throw std::runtime_error("expected P5 format");
55.             fin >> this->width >> this->height >> this->color_depth;
56.             this->image.assign(this->height, std::vector<unsigned char>(this->width));
57.             char pixel;
58.             fin.read(&pixel, 1);
59.             for (int i = 0; i < this->height; i++) {
60.                 for (int j = 0; j < this->width; j++) {
61.                     fin.read(&pixel, sizeof(unsigned char));
62.                     this->image[i][j] = pixel;
63.                 }
64.             }
65.             fin.close();
66.         }
67.
68.         void Image::write(std::string filename) {
69.             std::ofstream fout(filename, std::ios::binary);
70.             if(!fout.is_open()) {
71.                 throw std::runtime_error("cannot open output file");
72.             }
73.             fout << "P5\n" << width << ' ' << height << '\n' << color_depth << '\n';
74.             for(int i = 0; i < height; i ++)
75.                 for(int j = 0; j < width; j ++)
76.                     fout << static_cast<unsigned char>(image[i][j]);
77.             fout.flush();
78.             fout.close();
79.         }
80.
81.         void Image::plot(int x, int y, double brightness, int color, double gamma, bool SRGB) {
82.             if(x < 0 || x >= this->width || y < 0 || y >= this->height || brightness < 0) {
83.                 return;
84.             }
85.             double old = static_cast<double>(this->image[y][x]) / this->color_depth;
86.             double corrected_color = color * 1.0 / 255;
87.             if (SRGB) {
88.                 old = (old < 0.04045 ? old / 12.92 : pow((old + 0.055) / 1.055, gamma));
89.                 corrected_color = (corrected_color < 0.04045 ? corrected_color / 12.92 : pow((co
rrected_color + 0.055) / 1.055, gamma));
90.             }
91.             else {
92.                 old = pow(old, gamma);
93.                 corrected_color = pow(corrected_color, gamma);
94.             }
95.             old *= (1.0 - brightness);
96.             old += brightness * corrected_color;
97.             if(SRGB)
98.                 old = (old <= 0.0031308 ? old * 12.92 : pow(old,  1.0/gamma)*1.055 - 0.055);
99.             else
100.                old = pow(old, 1.0 / gamma);
101.            if(old >= 0.9999) old = 1.0;
102.            this->image[y][x] = static_cast<unsigned char>(this->color_depth * old);
103.        }
```

```
104.
105.       void Image::draw_line(Point start, Point
end, double thickness, int color, double gamma, bool SRGB) {
106.           if(start.x == end.x) {
107.               start.x += 0.5;
108.               end.x += 0.5;
109.           }
110.           if(start.y == end.y) {
111.               start.y += 0.5;
112.               end.y += 0.5;
113.           }
114.           const Point line_vector = {end.x - start.x, end.y - start.y};
115.           Point thickness_vector = {1.0, 0.0};
116.           if(line_vector.x != 0)
117.               thickness_vector = {-line_vector.y / line_vector.x, 1.0};
118.           const double thickness_coef = sqrt((thickness*thickness / 4) / (thickness_vector.x*t
hickness_vector.x + thickness_vector.y*thickness_vector.y));
119.           Point A = {start.x + thickness_coef*thickness_vector.x,
start.y + thickness_coef*thickness_vector.y};
120.           Point B = {start.x - thickness_coef*thickness_vector.x,
start.y - thickness_coef*thickness_vector.y};
121.           Point C = {end.x + thickness_coef*thickness_vector.x,
end.y + thickness_coef*thickness_vector.y};
122.           Point D = {end.x - thickness_coef*thickness_vector.x,
end.y - thickness_coef*thickness_vector.y};
123.           for(int i = std::max(0, int(std::min(std::min(A.x, B.x), std::min(C.x,
D.x))) - 3); i < std::min(width, int(std::max(std::max(A.x, B.x), std::max(C.x, D.x))) + 3); i++) {
124.               for (int j = std::max(0, int(std::min(std::min(A.y, B.y), std::min(C.y,
D.y))) - 3); j < std::min(height, int(std::max(std::max(A.y, B.y), std::max(C.y, D.y))) + 3); j++)
125.                   plot(i, j, intersection_square(A, B, C, D, i, j), color, gamma, SRGB);
126.           }
127.       }
```