**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное

учреждение высшего образования

**Санкт-Петербургский исследовательский университет**

**Информационных технологий, механики и оптики**

Факультет информационных технологий и программирования.

Дисциплина: компьютерная геометрия и графика.

**Отчет**

по лабораторной работе №2

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

Выполнил: студент гр. M32342

Ларионов А.Н.

Преподаватель: Скаков П.С.

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 быть написана на C/C++ и не использовать внешние библиотеки.

Аргументы 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.

**Частичное решение**: <толщина линии>=1, <гамма>=2.0, координаты начала и конца – целые числа, чёрный фон вместо данных исходного файла (размеры берутся из исходного файла).

**Полное решение**: всё работает (гамма + sRGB, толщина не только равная 1, фон из входного изображения) + корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок.

Если программе передано значение, которое не поддерживается – следует сообщить об ошибке.

Коды возврата:

0 - ошибок нет

1 - произошла ошибка

В поток вывода ничего не выводится (printf, cout).

Сообщения об ошибках выводятся в поток вывода ошибок:

С: fprintf(stderr, "Error\n");

C++: std::cerr

Следующие параметры гарантировано не будут выходить за обусловленные значения:

* *<яркость\_линии>* = целое число 0..255;
* *<толщина\_линии>* = положительное вещественное число;
* width и height в файле - положительные целые значения;
* яркостных данных в файле ровно width \* height;
* *<x\_начальный> <x\_конечный>* = 0..width;
* *<y\_начальный> <y\_конечный>* = 0..height;

**Теоретическая часть**

Для рисования прямых линий со сглаживанием за основу был взят Алгоритм Ву. *Алгоритм Ву* — это алгоритм разложения отрезка в растр со сглаживанием. В этом алгоритме на каждом шаге устанавливается не одна, а две точки. Например, если основной осью является Х, то рассматриваются точки с координатами (х, у) и (х, у+1). В зависимости от величины ошибки, которая показывает, как далеко ушли пиксели от идеальной линии по неосновной оси, распределяется интенсивность между этими двумя точками. Чем больше удалена точка от идеальной линии, тем меньше её интенсивность. Значения интенсивности двух пикселей всегда дают в сумме единицу, то есть это интенсивность одного пикселя, в точности попавшего на идеальную линию. Такое распределение придаст линии одинаковую интенсивность на всём её протяжении, создавая при этом иллюзию, что точки расположены вдоль линии не по две, а по одной.

Помимо отображения самой линии, в лабораторной работе было необходимо применить к ней гамма-коррекцию. Идея гамма-коррекции заключается в том, чтобы применить инверсию гаммы монитора к окончательному цвету перед выводом на монитор (записью в файл).

*Формулы гамма-коррекции:*

/var/folders/ck/6fyc4lyx3z58b08xghqprf0c0000gn/T/com.microsoft.Word/WebArchiveCopyPasteTempFiles/form_gam.png?resize=146%2C40

/var/folders/ck/6fyc4lyx3z58b08xghqprf0c0000gn/T/com.microsoft.Word/WebArchiveCopyPasteTempFiles/form_gamcor.png?resize=158%2C47

Помимо обычных формул гамма-коррекции необходимо было отдельно обработать sRGB.   
*sRGB* является стандартом представления цветового спектра с использованием модели RGB и создан для унификации использования модели RGB в мониторах, принтерах и интернет-сайтах. В отличие от большинства других цветовых пространств RGB, гамма в sRGB не может быть выражена одним числовым значением, так как функция коррекции состоит из линейной части около чёрного цвета, где гамма равна 1.0, и нелинейной части до значения 2.4 включительно.

*Формулы для sRGB:*

**Экспериментальная часть**

Используемый язык программирования: C++11

Рассмотрим случай с линией, имеющей ширину <= 1. Для реализации такого случая воспользуемся обычным алгоритмом Ву. При рисовании линии будем проверять, чтобы обрабатываемые точки не выходили за пределы границы изображения, а так же для корректного отображения линии с толщиной меньше 1 будем умножать яркость пикселя на толщину линии.

Перед началом работы с фоном будем проводить гамма-коррекцию, а перед отрисовкой будем проводить обратную гамма-коррекцию, а также альфа-смешивание по формуле *яркость фона + (яркость пикселя – яркость фона) \* непрозрачность пикселя.*

Для случая с толщиной > 1 необходимо преобразовать нашу линию в прямоугольник, у которого одна сторона будет равна длине нашей линии, а вторая ширине. Чтобы получить такой прямоугольник, проведем из начальной и конечной точки перпендикуляры линии длиной *width* так, чтобы начальная и конечная точка находились на середине полученных перпендикуляров. Получив прямоугольник, закрасим точки находящиеся внутри, но не попадающие на границы. После этого применим алгоритм Ву к каждой стороне нашего прямоугольника.

**Выводы**

В ходе выполнения работы была получена программа, реализующая рисование сглаженных линий по заданным координатам заданной толщины с применением гамма-коррекции или sRGB.

**Листинг**

hw2.cpp

#include <cmath>

#include <algorithm>

#include <iostream>

using namespace std;

typedef unsigned char uchar;

FILE \*file;

uchar\* mark;

uchar\* bytes;

bool steep = false;

int width, height;

int cnt1, cnt2, flasm = 0;

double gc, rangle = 0;

double brightness, thickness;

struct Point {

double x, y;

Point(double a, double b) {

x = a;

y = b;

}

Point() {

x = 0;

y = 0;

}

};

struct Line {

double a, b, c;

Line(double x0, double y0, double x, double y) {

a = y - y0;

b = x0 - x;

c = -a \* x - b \* y;

}

Line() {

a = 0;

b = 0;

c = 0;

}

};

Line ln[4];

bool operator <(Point a, Point b) {

return a.x < b.x || (a.x == b.x && a.y < b.y);

}

double from\_gc(double u) {

return 255.0 \* pow(u / 255.0, gc);

}

double to\_gc(double u) {

return 255.0 \* pow(u / 255.0, 1.0 / gc);

}

double from\_sRGB(double u) {

if (u / 255.0 <= 0.04045) {

return u / 12.92;

} else {

return 255.0 \* pow(((u / 255.0) + 0.055) / 1.055, 2.4);

}

}

double to\_sRGB(double u) {

if (u / 255.0 <= 0.0031308) {

return u \* 12.92;

} else {

return 255.0 \* (1.055 \* pow((u / 255.0), 0.416) - 0.055);

}

}

void inlin(Line l, Point p) {

double tmp = l.a \* p.x + l.b \* p.y + l.c;

if (tmp > 0.0) {

cnt1 += 1;

}

if (tmp < 0.0) {

cnt2 += 1;

}

}

bool check(Point p) {

cnt1 = 0, cnt2 = 0;

for (const auto & i : ln) {

inlin(i, p);

}

return cnt1 == 4 || cnt2 == 4;

}

void draw(int x, int y, double intensity, double rng) {

if (!steep) {

swap(x, y);

}

if ((x >= height) || (y >= width) || (x < 0) || (y < 0)) {

return;

}

if (rng == 45.0 && mark[x \* width + y] == 13) {

return;

}

if (intensity == 0.0) {

return;

}

if (thickness < 1.0) {

intensity \*= thickness;

}

double tmp = 0.0;

if (intensity != 1) {

tmp = (int)bytes[x \* width + y];

if (gc != -1) {

tmp = from\_gc(tmp);

} else {

tmp = from\_sRGB(tmp);

}

}

if (gc != -1) {

bytes[x \* width + y] = uchar(to\_gc(tmp + ((brightness - tmp) \* intensity)));

} else {

bytes[x \* width + y] = uchar(to\_sRGB(tmp + ((brightness - tmp) \* intensity)));

}

mark[x \* width + y] = 13;

}

int get\_int(double x) {

return static\_cast<int>(x);

}

double fpart(double x) {

return x - floor(x);

}

double rfpart(double x) {

return 1.0 - fpart(x);

}

void ralgo(double x0, double y0, double x, double y) {

double dx = x - x0;

double dy = y - y0;

double grad;

if (dx == 0.0) {

grad = 1.0;

} else {

grad = dy / dx;

}

double intery = y0;

int xf = get\_int(round(x));

int xs = get\_int(round(x0));

for (int x1 = xs; x1 <= xf; x1++) {

draw(x1, round(intery), 1 - fpart(intery), 0);

intery += grad;

}

}

void algo(double x0, double y0, double x, double y) {

steep = fabs(y - y0) > fabs(x - x0);

if (steep) {

swap(x, y);

swap(y0, x0);

}

if (x0 > x) {

swap(x, x0);

swap(y, y0);

}

if (rangle == 45 && thickness > 1.0) {

ralgo(x0, y0, x, y);

return;

}

double dx = x - x0;

double dy = y - y0;

double grad;

if (dx == 0.0) {

grad = 1.0;

} else {

grad = dy / dx;

}

double xx = round(x0);

double xgap = 1;

double intery = y0 + grad \* (xx - x0);

int xs = static\_cast<int>(xx);

if (thickness > 1.0) {

xgap = rfpart(x0 + 0.5);

}

xx = round(x);

int xf = (int)xx;

if (y0 == y) {

intery = y0;

if (flasm == 2) {

xs += 1;

xf -= 1;

}

for (int x1 = xs; x1 <= xf; x1++) {

draw(x1, get\_int(intery), 1 - fpart(intery), 0);

draw(x1, get\_int(intery) + 1, fpart(intery), 0);

}

flasm = min(flasm + 1, 2);

return;

}

draw(xs, get\_int(intery), rfpart(intery) \* xgap, 0);

draw(xs, get\_int(intery) + 1, fpart(intery) \* xgap, 0);

intery += grad;

xgap = 1;

if (thickness > 1.0) {

xgap = 1 - rfpart(x + 0.5);

}

double intr = y + grad \* (xx - x);

draw(xf, get\_int(intr), rfpart(intr) \* xgap, 0);

draw(xf, get\_int(intr) + 1, fpart(intr) \* xgap, 0);

for (int x1 = xs + 1; x1 <= xf - 1; x1++) {

draw(x1, get\_int(intery), 1 - fpart(intery), 0);

draw(x1, get\_int(intery) + 1, fpart(intery), 0);

intery += grad;

}

}

double dist(Point a, Point b) {

return sqrt((a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y));

}

int show\_error\_and\_exit(string message) {

std::cerr << message;

return 1;

}

int clear\_data(int cur\_type) {

delete[] bytes;

delete[] mark;

return cur\_type;

}

int main(int argc, char \*argv[]) {

int type, flag;

if (argc < 9 || argc > 10) {

show\_error\_and\_exit("Wrong number of arguments\n");

}

file = fopen(argv[1], "rb");

if (file == nullptr) {

show\_error\_and\_exit("Can’t open input file\n");

}

brightness = atoi(argv[3]);

thickness = atof(argv[4]);

double x0 = atof(argv[5]);

double y0 = atof(argv[6]);

double x = atof(argv[7]);

double y = atof(argv[8]);

if (argc == 10) {

gc = atof(argv[9]);

brightness = from\_gc(brightness);

} else {

gc = -1;

brightness = from\_sRGB(brightness);

}

fscanf(file, "P%d%d%d%d\n", &type, &width, &height, &flag);

if (type != 5) {

fclose(file);

show\_error\_and\_exit("Wrong format of input file\n");

}

bytes = new uchar[height \* width + 100];

mark = new uchar[height \* width + 100];

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

fread(&bytes[i \* width + j], sizeof(uchar), 1, file);

}

}

file = fopen(argv[2], "wb");

if (file == nullptr) {

std::cerr << "Can't open output file\n";

clear\_data(1);

}

for (int i = 0; i < height \* width; i++) {

mark[i] = 12;

}

if (thickness > 1) {

Point p[4];

double angle = atan2(y - y0, x - x0);

rangle = fabs(angle \* 180.0 / M\_PI);

if (rangle >= 90.0) {

rangle = 180.0 - rangle;

}

p[0].x = x0 + (thickness / 2.0) \* cos(angle + M\_PI / 2.0);

p[0].y = y0 + (thickness / 2.0) \* sin(angle + M\_PI / 2.0);

p[1].x = x0 + (thickness / 2.0) \* cos(angle - M\_PI / 2.0);

p[1].y = y0 + (thickness / 2.0) \* sin(angle - M\_PI / 2.0);

p[2].x = x + (thickness / 2.0) \* cos(angle - M\_PI / 2.0);

p[2].y = y + (thickness / 2.0) \* sin(angle - M\_PI / 2.0);

p[3].x = x + (thickness / 2.0) \* cos(angle + M\_PI / 2.0);

p[3].y = y + (thickness / 2.0) \* sin(angle + M\_PI / 2.0);

sort(p, p + 4);

ln[0] = Line(p[0].x, p[0].y, p[1].x, p[1].y);

ln[1] = Line(p[3].x, p[3].y, p[2].x, p[2].y);

ln[2] = Line(p[2].x, p[2].y, p[0].x, p[0].y);

ln[3] = Line(p[1].x, p[1].y, p[3].x, p[3].y);

bool skp = false;

if (dist(p[2], p[3]) < dist(p[1], p[3])) {

algo(p[1].x, p[1].y, p[3].x, p[3].y);

algo(p[2].x, p[2].y, p[0].x, p[0].y);

skp = true;

}

algo(p[2].x, p[2].y, p[3].x, p[3].y);

algo(p[0].x, p[0].y, p[1].x, p[1].y);

if (!skp) {

algo(p[1].x, p[1].y, p[3].x, p[3].y);

algo(p[2].x, p[2].y, p[0].x, p[0].y);

}

steep = false;

for (int i = 0; i < width; i++) {

for (int j = 0; j < height; j++) {

if (check(Point(i, j))) {

draw(i, j, 1, rangle);

}

}

}

if (rangle == 45.0 && ((round(x0) == 0 && round(y0) == 0) || (round(x) == 0 && round(y) == 0))) {

if (thickness > 1) {

draw(0, 0, 1, 0);

}

}

} else {

algo(x0, y0, x, y);

}

fprintf(file, "P%d\n%d %d\n%d\n", type, width, height, flag);

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

fwrite(&bytes[i \* width + j], sizeof(uchar), 1, file);

}

}

fclose(file);

clear\_data(0);

}