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 №M3234:**

***Ота Никита Терентьевич***

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

Аргументы передаются через командную строку:

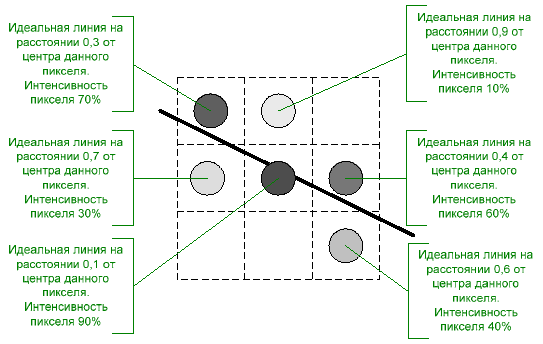
program.exe <имя\_входного\_файла> <имя\_выходного\_файла> <яркость\_линии> <толщина\_линии> <x\_начальный> <y\_начальный> <x\_конечный> <y\_конечный> <гамма>, где

* <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.

**Теоретическая часть:**

**Алгоритм Ву:**

Алгоритм, рисующий линию между начальной и конечной точками. На каждом шаге рисуется два пикселя, прозрачность которых зависит от удалённости цетров от линии. Горизонтальные и вертикальные линии не требуют никакого сглаживания, поэтому их рисование выполняется отдельно. Для остальных линий алгоритм Ву проходит их вдоль основной оси, подбирая координаты по неосновной оси.

Картинка, показывающая распределение интенсивности (из русской википедии, [ссылка](https://ru.wikipedia.org/wiki/Алгоритм_Ву))

**Гамма-коррекция:**

Гамма-коррекция — предыскажения яркости чёрно-белого или цветоделённых составляющих цветного изображения при его записи. В качестве передаточной функции при гамма-коррекции чаще всего используется степенная. 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.

|  |  |
| --- | --- |
|  | * Серая линия соответствует значениям цвета в линейном пространстве; * Сплошная красная линия представляет собой цветовое пространство отображаемое монитором. * Важное примечание: дальше разговор будет вестись про яркость в диапазоне [0..1], а не [0..255], как мы привыкли (т.е. используются нормированные значения яркости) |

**sRGB:**

sRGB является стандартом представления цветового спектра с использованием модели RGB. sRGB создан для унификации использования модели RGB в мониторах, принтерах и Интернет-сайтах. В отличии 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. Приблизительно можно считать, что гамма равна 2.2. Гамма может изменяться от 1.0 до 2.3. 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.

**Практическая часть:**

Язык, на котором, написана программа, решающая поставленную задачу: C++.  
Ключи компиляции: -std=c++11 -fsanitize=address -fno-stack-limit.

Компилятор: g++.

План работы программы:

1. Чтение всех данных (переданных через аргументы запуска и из входного файла-изображения, переданного в качестве второго аргумента) с обработкой ошибок чтения данных.
2. Подготовка к покраске изображения, черчение линии на изображении посредством разделения её толщины на половину и рисования двух линий (из алгоритма Ву).
3. Вывод данных в файл с обработкой ошибок.

**Код программ:**

**main.cpp**

#include <iostream>

#include <fstream>

#include <string>

#include "line\_drawer.h"

void clear\_array(int \*\*data, int height) {

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

delete[] data[h];

}

delete[] data;

}

int show\_message(std::ifstream &in, std::string const &message) {

std::cerr << message << '\n';

in.close();

return 1;

}

int show\_message(std::ifstream &in, char const \*message, char const \*file) {

std::cerr << message << file << '\n';

in.close();

return 1;

}

int show\_message(std::ifstream &in, char const \*message, char const \*file,

int \*\*data, int height) {

clear\_array(data, height);

return show\_message(in, message, file);

}

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

if (argc != 9 && argc != 10) {

std::cerr << "Got " << argc << " arguments\n";

std::cerr << "Usage: <input filename> <output filename> <line brightness> <line thickness> <x0> <y0> <x1> <y1> <gamma?>\n";

return 1;

}

std::ifstream in(argv[1]);

if (!in.is\_open()) {

std::cerr << "Couldn't open " << argv[1];

return 1;

}

int brightness;

double thickness, x0, y0, x1, y1, gamma = 2.4;

bool sRGB = true;

try {

brightness = std::stoi(argv[3]);

if (brightness < 0) {

return show\_message(in, "Brightness must be a positive integer");

}

std::string positive\_error = " must be a positive number";

thickness = std::stod(argv[4]);

if (thickness < 0) {

return show\_message(in, "Thickness" + positive\_error);

}

std::string coord = " coordinate";

x0 = std::stod(argv[5]);

if (x0 < 0) {

return show\_message(in, "x0" + coord + positive\_error);

}

y0 = std::stod(argv[6]);

if (y0 < 0) {

return show\_message(in, "y0" + coord + positive\_error);

}

x1 = std::stod(argv[7]);

if (x0 < 0) {

return show\_message(in, "x1" + coord + positive\_error);

}

y1 = std::stod(argv[8]);

if (y1 < 0) {

return show\_message(in, "y1" + coord + positive\_error);

}

if (argc == 10) {

gamma = std::stod(argv[9]);

sRGB = false;

}

} catch (std::invalid\_argument const& e) {

std::cerr << "Got not a number value\n";

std::cerr << e.what();

return 1;

} catch (std::out\_of\_range const& e) {

std::cerr << "Brightness value is too big\n";

std::cerr << e.what();

return 1;

}

std::cout << "Succsessfully read all data from console\n";

char p, line\_separator;

int type;

in >> p >> type;

line\_separator = in.get();

if (p != 'P' || (type < 5 && type > 6) || line\_separator != '\n') {

return show\_message(in, "Expected P5 or P6 at a first line of ", argv[1]);

}

int width, height;

in >> width >> height;

line\_separator = in.get();

if (width <= 0 || height <= 0 || line\_separator != '\n') {

return show\_message(in, "Expected two positive integer values at a second line of ", argv[1]);

}

if (x0 >= width || x1 >= width || y0 >= height || y1 >= height) {

return show\_message(in, "Line coordinates are out of bounds: [0.." + std::to\_string(width) + ")\*[0.." + std::to\_string(height) + ")");

}

int max\_brightness;

in >> max\_brightness;

line\_separator = in.get();

if (max\_brightness != MAX\_BRIGHTNESS || line\_separator != '\n') {

return show\_message(in, "Expected 255 at a third line of ", argv[1]);

}

if (brightness > max\_brightness) {

return show\_message(in, "Brightness given in arguments is larger than max brightness in file ", argv[1]);

}

int \*\*data;

data = new int\*[height];

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

data[h] = new int[width];

}

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

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

data[h][w] = in.get();

if (data[h][w] == -1 || in.gcount() == 0 || in.eof()) {

return show\_message(in, "Unexpected data size: not enough bytes in ", argv[1], data, height);

}

}

}

if (!(in.get() == -1) || !in.eof()) {

return show\_message(in, "Unexpected data size: redundant bytes in ", argv[1], data, height);

}

std::cout << "Succsessfully read all data from file\n";

line\_drawer drawer(brightness, thickness, x0, y0, x1, y1, gamma, sRGB, height, width, max\_brightness, data);

drawer.draw\_line();

std::cout << "Line was successfully drawn\n";

std::ofstream out(argv[2]);

if (!out.is\_open()) {

std::cerr << "Couldn't open " << argv[2];

in.close();

return 1;

}

out << "P5\n";

out << width << ' ' << height << '\n';

out << max\_brightness << '\n';

data = drawer.get\_picture\_data();

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

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

if (!out || out.bad()) {

std::cerr << "An error occurred while writing data\n";

out.close();

// array data will be cleared in pictore destructor (in line\_drawer)

return 1;

}

out << static\_cast<unsigned char>(data[h][w]);

}

}

std::cout << "Successfully written picture with line data to file\n";

out.close();

return 0;

}

**line\_drawer.h**

#ifndef LINE\_DRAWER

#define LINE\_DRAWER

double const MAX\_BRIGHTNESS = 255.0;

struct point {

public:

explicit point(double x, double y);

// point(std::initializer\_list<double> const& list);

~point() = default;

double get\_x() const;

double get\_y() const;

void swap\_axis();

private:

double x;

double y;

};

struct picture {

public:

picture(int h, int w, int m, int \*\*data);

~picture();

int get\_h() const;

int get\_w() const;

int\*\* get\_data();

void fill\_pixel(int x, int y, bool swapped, int brightness, double percentage, bool sRGB, double gamma);

private:

int height;

int width;

int max\_brightness;

int \*\*data = nullptr;

};

struct line\_drawer {

public:

explicit line\_drawer(int b, double t,

double x0, double y0, double x1, double y1,

double gamma, bool sRGB,

int h, int w, int m, int \*\*data);

~line\_drawer() = default;

void draw\_line();

int\*\* get\_picture\_data();

private:

int brightness;

double thickness;

point from = point(-1, -1);

point to = point(-2, -2);

double gamma;

bool sRGB;

picture pic = picture(0, 0, 0, nullptr);

void draw\_part\_of\_line(int x0, int x1, double y, double gradient, bool swapped);

};

#endif

**line\_drawer.cpp**

#include <algorithm>

#include <stdexcept>

#include <math.h>

#include "line\_drawer.h"

double const EPS = 1e-5;

point::point(double x, double y): x(x), y(y) {}

// point::point(std::initializer\_list<double> const& list) {

// if (list.size() != 2) {

// throw std::range\_error("initializer\_list size must be equals to 2");

// }

// x = \*list.begin();

// y = \*(list.begin() + 1);

// }

double point::get\_x() const {

return x;

}

double point::get\_y() const {

return y;

}

void point::swap\_axis() {

std::swap(x, y);

}

picture::picture(int h, int w, int m, int \*\*data)

: height(h), width(w), max\_brightness(m), data(data) {}

picture::~picture() {

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

delete[] data[h];

}

delete[] data;

}

int picture::get\_h() const {

return height;

}

int picture::get\_w() const {

return width;

}

int\*\* picture::get\_data() {

return data;

}

inline double correction(double val, double percentage, int brightness) {

val \*= 1 - percentage;

val += percentage \* brightness / MAX\_BRIGHTNESS;

return val;

}

template <typename T>

inline void do\_swap(bool swapped, T &t1, T &t2) {

if (swapped) {

std::swap(t1, t2);

}

}

void picture::fill\_pixel(int x, int y, bool const swapped, int const brightness,

double const percentage, bool const sRGB, double const gamma) {

do\_swap(swapped, x, y);

// std::ofstream out("log.txt", std::ios::app);

// out << x << ' ' << y << ": ";

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

// out << "not allowed\n";

// out.flush();

return;

}

// out << " allowed!\n";

// out.flush();

// out << "data[" << x << "][" << y << "] / 255 = ";

// out.flush();

double value = data[y][x] / MAX\_BRIGHTNESS;

// out << value << '\n';

// out.flush();

if (sRGB) {

// reverse transformation

if (value <= 0.04045) {

value /= 12.92;

} else {

// gamma is 2.4 here

value = pow((value + 0.055) / 1.055, gamma);

}

value = correction(value, percentage, brightness);

// forward transformation

if (value <= 0.0031308) {

value \*= 12.92;

} else {

value = 1.055 \* pow(value, 1 / 2.4) - 0.055;

}

} else {

value = pow(value, gamma);

value = correction(value, percentage, brightness);

value = pow(value, 1 / gamma);

}

value \*= MAX\_BRIGHTNESS;

data[y][x] = static\_cast<int>(value);

}

line\_drawer::line\_drawer(int b, double t, double x0, double y0, double x1,

double y1, double gamma, bool sRGB, int h, int w,

int m, int\*\* data)

: brightness(b),

thickness(t),

gamma(gamma),

sRGB(sRGB),

from(x0, y0),

to(x1, y1),

pic(h, w, m, data) {}

int\*\* line\_drawer::get\_picture\_data() {

return pic.get\_data();

}

void line\_drawer::draw\_part\_of\_line(int x0, int x1, double y, double gradient, bool swapped) {

int arg1, arg2;

double fy = y - floor(y);

for (int x = x0; x <= x1; ++x) {

// std::cout << x << ' ' << y << '\n';

arg1 = x;

arg2 = static\_cast<int>(floor(y));

// do\_swap(swapped, arg1, arg2);

pic.fill\_pixel(arg1, arg2, swapped, brightness, 1 - fy, sRGB, gamma);

arg1 = x;

arg2 = (int) floor(y) + 1;

// do\_swap(swapped, arg1, arg2);

pic.fill\_pixel(arg1, arg2, swapped, brightness, fy, sRGB, gamma);

y += gradient;

}

}

bool check\_equals(double a, double b) {

return abs(a - b) < EPS;

}

void line\_drawer::draw\_line() {

bool swapped = abs(from.get\_x() - to.get\_x()) < abs(from.get\_y() - to.get\_y());

if (swapped) {

from.swap\_axis();

to.swap\_axis();

// std::swap(dx, dy);

// swapped = true;

}

if (to.get\_x() < from.get\_x()) {

std::swap(from, to);

}

double dx = to.get\_x() - from.get\_x(), dy = to.get\_y() - from.get\_y();

double gradient = dy / dx;

double ht = thickness / 2;

// std::cout << from.get\_x() << ' ' << from.get\_y() << ' ' << to.get\_x() << ' ' << to.get\_y() << ' ' << ht << ' ' << dx << ' ' << dy << ' ' << gradient << '\n';

if (dy == 0) {

gradient = 1;

std::cout << "Drawing vertical or horisontal line\n";

for (int x = static\_cast<int>(round(from.get\_x())); x <= static\_cast<int>(round(to.get\_x())); ++x) {

// std::cout << "\nx = " << x << '\n';

for (int y = static\_cast<int>(round(from.get\_y() - ht)); y <= static\_cast<int>(round(from.get\_y() + ht)); ++y) {

// std::cout << y << ' ';

pic.fill\_pixel(x, y, swapped, brightness, 1, sRGB, gamma);

}

}

return;

}

double curr\_x = from.get\_x(), curr\_y = from.get\_y() + gradient;

draw\_part\_of\_line(static\_cast<int>(round(curr\_x)), static\_cast<int>(round(to.get\_x())),

curr\_y - ht, gradient, swapped);

if (check\_equals(thickness, 1)) {

return;

}

draw\_part\_of\_line(static\_cast<int>(round(curr\_x)), static\_cast<int>(round(to.get\_x())),

curr\_y + ht - 1, gradient, swapped);

if (check\_equals(thickness, 2)) {

return;

}

while (curr\_x <= to.get\_x()) {

for (int y = static\_cast<int>(round(curr\_y - ht + 1)); y <= static\_cast<int>(curr\_y + ht - 1); ++y) {

pic.fill\_pixel(static\_cast<int>(round(curr\_x)), y, swapped, brightness, 1, sRGB, gamma);

}

// pic.fill\_pixel(static\_cast<int>(round(curr\_x)), curr\_y + ht - 1, brightness, 1, sRGB, gamma);

// pic.fill\_pixel(static\_cast<int>(round(curr\_x)), curr\_y - ht + 1, brightness, 1, sRGB, gamma);

++curr\_x;

curr\_y += gradient;

}

}