**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное

учреждение высшего образования

**Санкт-Петербургский 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

Выполнила: студент гр. M3101

Тарасов Денис Евгеньевич

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**

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>: =1;
* <x,y>: координаты внутри изображения, (0;0) соответствует левому верхнему углу, дробные числа (целые значения соответствуют центру пикселей). X и Y – целые числа
* <гамма>: =2

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");

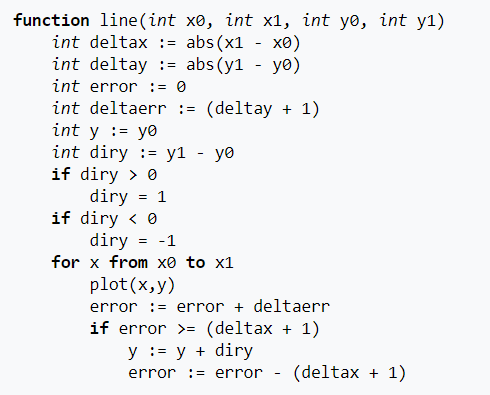
C++: std::cerr

Следующие параметры гарантировано не будут выходить за обусловленные значения:

* <яркость\_линии> = целое число 0..255;
* 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];

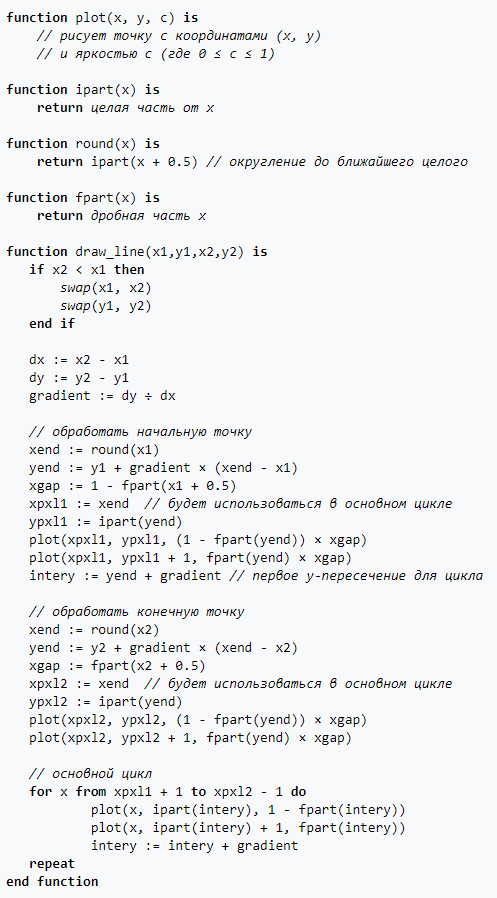
**Теоретическая часть**

Рассмотрим различные алгоритмы рисования линий:

Алгоритм Брензенхэма (Псевдокод):  


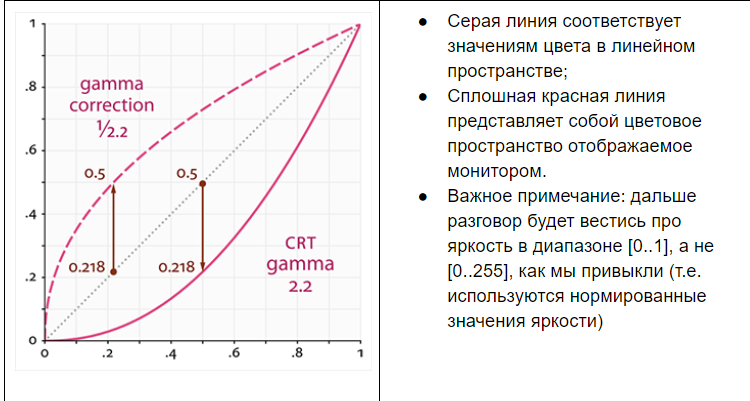
Этот алгоритм рисует линию без сглаживания

Алгоритм Ву

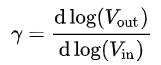


Данный алгоритм рисует линию со сглаживанием. Именно этот алгоритм мы и используем.

Гамма коррекция



**Идея гамма-коррекции** заключается в том, чтобы применить инверсию гаммы монитора к окончательному цвету перед выводом на монитор (записью в файл). Снова посмотрим на график гамма-кривой, обратив внимание на еще одну линию, обозначенную штрихами, которая является обратной для гамма-кривой монитора. Мы умножаем выводимые значения цветов в линейном пространстве на эту обратную гамма-кривую ( делаем их ярче), и как только они будут выведены на монитор, к ним применится гамма-кривая монитора, и результирующие цвета снова станут линейными. По сути мы делаем промежуточные цвета ярче, чтобы сбалансировать их затенение монитором.

Все преобразования, связанные с гамма-коррекцией, получаются из формулы: 

Тогда формула для входа:

Формула для выхода:

**Экспериментальная часть**

Язык программирования: С++ 17, частичное решение

Этапы работы программы:

1. Читаем картинку, применяя гамма-коррекцию для входа.
2. Выясняем: линия горизонтальная, вертикальная или нет?
3. Если нет, то используем алгоритм ВУ, если да, то просто чертим линию.
4. Выводим картинку, применяя гамма-коррекцию для выхода.

**Выводы**

Выполнение работы позволило узнать и опробовать алгоритм рисования линий со сглаживанием. Так же это позволило узнать о гамма-коррекции.

**Листинг**

Source.cpp

#include "PGM.h"

#include <iostream>

using namespace std;

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

if (argc != 10) {

cerr << "Wrong command";

return 1;

}

string inpFileName = string(argv[1]);

string outFileName = string(argv[2]);

double x1=atoi(argv[5]);

double y1 = atoi(argv[6]);

double x2=atoi(argv[7]);

double y2=atoi(argv[8]);

int bright = atoi(argv[3]);

PGM\* pict;

try {

pict = new PGM(inpFileName);

pict->wu\_algo(x1, y1, x2, y2, bright);

pict->write(outFileName);

}

catch (exception& ex) {

cerr << ex.what();

return 1;

}

delete pict;

return 0;

}

PGM.h

#pragma once

#include <vector>

#include <iostream>

#include <fstream>

#include <cmath>

using namespace std;

class PGM {

private:

char vers[2];

int width;

int height;

int colorDepth;

vector<vector<unsigned char>> pict;

public:

PGM(string inpFileName) {

ifstream inpFile(inpFileName, ios::binary);

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

throw exception("Cant open input File");

}

inpFile >> vers[0] >> vers[1] >> width >> height >> colorDepth;

char read\_char[1];

inpFile.read(read\_char, 1);

pict.assign(height, vector<unsigned char>(width, unsigned char(0)));

inpFile.close();

}

void write(string outFileName) {

ofstream outFile(outFileName, ios::binary);

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

throw exception("Cant open output file");

}

outFile << vers[0] << vers[1] << '**\n**' << width << ' ' << height << '**\n**' << colorDepth << '**\n**';

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

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

outFile << pict[i][j];

}

}

outFile.close();

}

void plot(int x, int y, double bright,int color) {

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

return;

}

double gamma = 2.0;

double ColorCorrect = double(color) / colorDepth;

ColorCorrect = pow(ColorCorrect, gamma);

ColorCorrect \*= bright;

ColorCorrect = pow(ColorCorrect, 1.0 / gamma);

pict[x][y] = unsigned char(colorDepth \* ColorCorrect);

}

double fpart(double x) {

return x - floor(x);

}

double rfpart(double x) {

return 1 - fpart(x);

}

void draw\_vertical(double x0, double y0, double x1, double y1, int bright) {

if (y0 > y1) {

swap(y0, y1);

}

for (int i = y0; i <= y1; i++) {

plot(x0, i, 1,bright);

}

}

void draw\_horizontal(double x0, double y0, double x1, double y1,int bright) {

if (x0 > x1) {

swap(x0, x1);

}

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

plot(i, y0, 1,bright);

}

}

void wu\_algo(double x0, double y0, double x1, double y1,int color) {

swap(x0, y0);

swap(x1, y1);

if (y0 == y1) {

draw\_horizontal(x0, y0, x1, y1,color);

return;

}

if (x0 == x1) {

draw\_vertical(x0, y0, x1, y1, color);

return;

}

bool steep = abs(y1 - y0) > abs(x1 - x0);

if (steep) {

swap(x0, y0);

swap(x1, y1);

}

if (x0 > x1) {

swap(x0, x1);

swap(y0, y1);

}

double dx = x1 - x0;

double dy = y1 - y0;

double grad;

if (dx == 0) {

grad = 1;

}

else {

grad = dy / dx;

}

int xend = round(x0);

int yend = y0 + grad \* (xend - x0);

double xgap = rfpart(x0 + 0.5);

int xpxl1 = xend;

int ypxl1 = floor(yend);

if (steep) {

plot(ypxl1, xpxl1, rfpart(yend) \* xgap,color);

plot(ypxl1 + 1, xpxl1, fpart(yend) \* xgap,color);

}

else {

plot(xpxl1, ypxl1, rfpart(yend) \* xgap,color);

plot(xpxl1, ypxl1 + 1, fpart(yend) \* xgap,color);

}

double intery = yend + grad;

xend = round(x1);

yend = y1 + grad \* (xend - x1);

xgap = fpart(x1 + 0.5);

int xpxl2 = xend;

int ypxl2 = floor(yend);

if (steep) {

plot(ypxl2, xpxl2, rfpart(yend) \* xgap,color);

plot(ypxl2 + 1, xpxl2, fpart(yend) \* xgap,color);

}

else {

plot(xpxl2, ypxl2, rfpart(yend) \* xgap,color);

plot(xpxl2, ypxl2 + 1, fpart(yend) \* xgap,color);

}

if (steep) {

for (int x = xpxl1 + 1; x <= xpxl2 - 1; x++) {

plot(floor(intery), x, rfpart(intery),color);

plot(floor(intery) + 1, x, fpart(intery),color);

intery += grad;

}

}

else {

for (int x = xpxl1 + 1; x <= xpxl2 - 1; x++) {

plot(x, floor(intery), rfpart(intery),color);

plot(x, floor(intery) + 1, fpart(intery),color);

intery += grad;

}

}

}

};