МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

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

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

**Лабораторная работа № 2**

**По дисциплине «Компьютерная графика и геометрия»**

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

**Выполнил студент группы M3101  
*Дудко Матвей Владимирович***

**Проверил:  
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

**Частичное решение лабораторной работы.**

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

Аргументы передаются через командную строку:

**lab2.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).

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

## Рисование линий

Уравнение прямой линии:

Существуют несколько видов алгоритмов:

1. Со сглаживанием
2. Без сглаживания

### Алгоритм Брезенхема

Это простой алгоритм целочисленный, без сглаживания.

Псевдокод:



### Алгоритм Ву

Этот алгоритм может работать с дробными координатами, со сглаживанием, но он относительно сложный по сравнению с алгоритмом Брезенхема.

Псевдокод:



# Экспериментальная часть

**Частичное решение лабораторной работы.**

Лабораторная работа выполнена на языке C++. Стандарт языка C++14.

## Хранение данных изображения и заголовка при чтении из файла

Данные заголовка файла хранятся в переменных:

char \_char\_header;  
int width, height;  
unsigned int max\_value;

Чтение производится при помощи функции fscanf:

int scanned = fscanf(file\_in, "P%c\n%i %i\n%i\n", &\_char\_header, &width, &height, &max\_value);

Где:

* char\_header – номер версии файла (возможные варианты 5 или 6)
* width – ширина изображения
* height – высота изображения
* max\_value – максимально возможное значение яркости
* scanned – переменная для обработки ошибок при чтении файла

Общее количество байт для считывания вычисляется и записывается в переменную k\_bytes:

int k\_bytes = height \* width;

Данные пикселей нового изображения хранятся в одномерном динамическом массиве, выделяющемся в куче:

auto pix\_data = (unsigned char \*) calloc(k\_bytes, 1);

## Алгоритм отрисовки линии

Был реализован алгоритм Ву со сглаживанием:

wu\_line\_along\_y(pix\_data, width, height, brightness, x0, y0, x1, y1, gamma);

## Запись данных в файл

Запись данных в файл производится при помощи двух функций:

write\_header(file\_out, char\_header, width, height, 255);  
write\_data(file\_out, k\_bytes, pix\_data);

# Вывод

Выполнение данной лабораторной работы позволило узнать об алгоритмах рисования прямых линий как без сглаживания, так и со сглаживанием. В ходе данной лабораторной работы был реализован алгоритм Ву, позволяющий рисовать линии со сглаживанием.

# Листинг кода

Исходный код можно посмотреть на ресурсе GitHub по следующей ссылке: <https://github.com/DudkoMatt/GeometryAndGraphics/blob/master/Lab_02>

Название файла: ./GeometryAndGraphics/Lab\_02/main.cpp

Исходный код:

#include <iostream>  
#include <cmath>  
  
void write\_header(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value);  
  
void write\_data(FILE \*file\_out, int k\_bytes, unsigned char \*pix\_data);  
  
void free\_data(FILE \*file\_in, FILE \*file\_out, unsigned char \*pix\_data);  
  
void draw\_pix(unsigned char \*pix\_data, int height, int width, int x, int y, double brightness, double gamma, int max\_brightness);  
double fpart(double x);  
  
// along\_y рисует вдоль обеих осей  
void wu\_line\_along\_y(unsigned char \*pix\_data, int width, int height, int brightness, double x0, double y0, double x1, double y1, double gamma);  
  
int main(int argc, char \*argv[]) {  
  
 if (argc != 9 && argc != 10) {  
 std::cerr << "Wrong number of arguments. Syntax:\n<lab>.exe file\_in file\_out brightness thickness x0 y0 x1 y1 [gamma]";  
 return 1;  
 }  
  
 const char \*file\_in\_name = argv[1];  
 const char \*file\_out\_name = argv[2];  
  
 double brightness;  
 double thickness;  
 double x0, y0, x1, y1;  
  
 double gamma = 2.0;  
  
 // Общий случай: 2.0  
 // Лучше - 2.2  
  
 try {  
 brightness = std::stoi(argv[3]);  
 thickness = std::stod(argv[4]);  
 x0 = std::stod(argv[5]);  
 y0 = std::stod(argv[6]);  
 x1 = std::stod(argv[7]);  
 y1 = std::stod(argv[8]);  
 if (argc == 10)  
 gamma = std::stod(argv[9]);  
 } catch (std::invalid\_argument &e) {  
 std::cerr << "Cannot convert one of the numbers from string to int\n";  
 return 1;  
 }  
  
  
 // Считывание размеров из файла  
  
 char \_char\_header;  
 int width, height;  
 unsigned int max\_value;  
  
 FILE\* file\_in = fopen(file\_in\_name, "rb");  
  
 if (file\_in == nullptr) {  
 std::cerr << "Cannot open file to read: " << file\_in\_name << "\n";  
 return 1;  
 }  
  
 int scanned = fscanf(file\_in, "P%c\n%i %i\n%i\n", &\_char\_header, &width, &height, &max\_value);  
  
  
 if (width <= 0 || height <= 0 || x0 < 0 || y0 < 0 || x1 < 0 || y1 < 0 || y0 >= height || y1 >= height ||  
 x0 >= width || x1 >= width || thickness <= 0 || brightness < 0 || brightness > 255) {  
 std::cerr << "Wrong arguments\n";  
 fclose(file\_in);  
 return 1;  
 }  
  
  
 // Перевод из воспринимаемой яркости в фактическую  
 brightness = std::pow(brightness, gamma);  
  
  
 FILE \*file\_out = nullptr;  
 const char char\_header = '5';  
 int k\_bytes = height \* width;  
  
 auto pix\_data = (unsigned char \*) calloc(k\_bytes, 1);  
 if (!pix\_data) {  
 std::cerr << "Cannot allocate " << k\_bytes << " bytes of memory\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 // Initialization  
 for (int i = 0; i < k\_bytes; ++i) {  
 pix\_data[i] = 0;  
 }  
  
 file\_out = fopen(file\_out\_name, "wb");  
 if (file\_out == nullptr) {  
 std::cerr << "Cannot open file to write: " << file\_out\_name << "\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 if (x0 == x1 && y0 == y1) {  
 draw\_pix(pix\_data, height, width, x0, y0, brightness, gamma, brightness);  
 } else {  
 // Алгоритм Ву со сглаживанием  
 wu\_line\_along\_y(pix\_data, width, height, brightness, x0, y0, x1, y1, gamma);  
 }  
  
 write\_header(file\_out, char\_header, width, height, 255);  
 write\_data(file\_out, k\_bytes, pix\_data);  
  
 std::cout << (int) pix\_data[0];  
  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
}  
  
void write\_header(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value) {  
 fseek(file\_out, 0, SEEK\_SET);  
 fprintf(file\_out, "P%c\n%i %i\n%i\n", char\_header, width, height, max\_value);  
}  
  
void write\_data(FILE \*file\_out, int k\_bytes, unsigned char \*pix\_data) {  
 fwrite(pix\_data, k\_bytes, 1, file\_out);  
}  
  
void free\_data(FILE \*file\_in, FILE \*file\_out, unsigned char \*pix\_data) {  
 if (file\_in) fclose(file\_in);  
 if (file\_out) fclose(file\_out);  
 if (pix\_data) free(pix\_data);  
}  
  
void draw\_pix(unsigned char \*pix\_data, int height, int width, int x, int y, double brightness, double gamma, int max\_brightness) {  
 if (x >= width || x < 0 || y < 0 || y > height) return;  
 brightness = std::min(std::max(brightness, 0.0), 1.0);  
  
 // Гамма коррекция:  
 if (gamma > 0) {  
 brightness = std::pow(max\_brightness \* brightness, 1.0 / gamma);  
 } else {  
 // sRGB -- только частичное решение  
 }  
  
 pix\_data[width \* y + x] = std::min(std::max((int) std::round(brightness), 0), 255);  
}  
  
double fpart(double x) {  
 return x - std::floor(x);  
}  
  
void wu\_line\_along\_y(unsigned char \*pix\_data, int width, int height, int brightness, double x0, double y0, double x1, double y1, double gamma) {  
 // Swap if necessary  
 if (x1 < x0) {  
 double t = x0;  
 x0 = x1;  
 x1 = t;  
  
 t = y0;  
 y0 = y1;  
 y1 = t;  
 }  
  
 double dx = x1 - x0;  
 double dy = y1 - y0;  
  
 double gradient = dy / dx; // Угловой коэффициент прямой  
  
 if (dy > dx) {  
 gradient = dx / dy;  
  
 double t = x0;  
 x0 = y0;  
 y0 = t;  
  
 t = x1;  
 x1 = y1;  
 y1 = t;  
 }  
  
  
 // Обработка начальной точки  
 double xend = round(x0);  
 double yend = y0 + gradient \* (xend - x0);  
 double xgap = 1 - fpart(x0);  
 double xpxl1 = xend;  
 double ypxl1 = std::floor(yend);  
  
 // Отрисовка вдоль оси oY  
 if (dy > dx) {  
  
 draw\_pix(pix\_data, height, width, ypxl1, xpxl1, (1 - fpart(yend)) \* xgap, gamma, brightness);  
 draw\_pix(pix\_data, height, width, ypxl1 + 1, xpxl1, (fpart(yend)) \* xgap, gamma, brightness);  
 } else {  
 draw\_pix(pix\_data, height, width, xpxl1, ypxl1, (1 - fpart(yend)) \* xgap, gamma, brightness);  
 draw\_pix(pix\_data, height, width, xpxl1, ypxl1 + 1, (fpart(yend)) \* xgap, gamma, brightness);  
 }  
  
 double intery = yend + gradient;  
  
  
 // Обработка конечной точки  
 xend = round(x1);  
 yend = y1 + gradient \* (xend - x1);  
 xgap = 1 - fpart(x1);  
 double xpxl2 = xend;  
 double ypxl2 = std::floor(yend);  
  
 if (dy > dx) {  
 draw\_pix(pix\_data, height, width, ypxl2, xpxl2, (1 - fpart(yend)) \* xgap, gamma, brightness);  
 draw\_pix(pix\_data, height, width, ypxl2 + 1, xpxl2, (fpart(yend)) \* xgap, gamma, brightness);  
 } else {  
  
 draw\_pix(pix\_data, height, width, xpxl2, ypxl2, (1 - fpart(yend)) \* xgap, gamma, brightness);  
 draw\_pix(pix\_data, height, width, xpxl2, ypxl2 + 1, (fpart(yend)) \* xgap, gamma, brightness);  
 }  
  
 for (int x = xpxl1 + 1; x < xpxl2; ++x) {  
 if (dy > dx) {  
 draw\_pix(pix\_data, height, width, std::floor(intery), x, (1 - fpart(intery)), gamma, brightness);  
 draw\_pix(pix\_data, height, width, std::floor(intery) + 1, x, fpart(intery), gamma, brightness);  
 } else {  
 draw\_pix(pix\_data, height, width, x, std::floor(intery), (1 - fpart(intery)), gamma, brightness);  
 draw\_pix(pix\_data, height, width, x, std::floor(intery) + 1, fpart(intery), gamma, brightness);  
 }  
 intery += gradient;  
 }  
  
}