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 гаммакоррекции

Выполнила: студент гр. М3101
Шмарина Л.С.
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:
C: 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

Существуют несколько видов алгоритмов:

1. **Со сглаживанием**
2. **Без сглаживания**

Алгоритм Брезенхема

Это простой алгоритм целочисленный, без сглаживания.

Псевдокод:

```
function line(int x0, int x1, int y0, int y1)
     int deltax := abs(x1 - x0)
     int deltay := abs(y1 - y0)
     int error := 0
     int deltaerr := (deltay + 1)
     int y := y0
     int diry := y1 - y0
     if diry > 0
         diry = 1
     if diry < 0
         diry = -1
     for x from x0 to x1
         plot(x,y)
         error := error + deltaerr
         if error >= (deltax + 1)
             y := y + diry
             error := error - (deltax + 1)
```

Алгоритм Ву

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

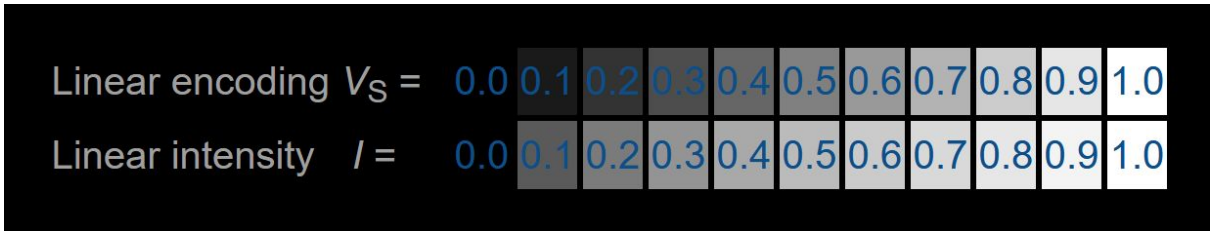
Псевдокод:

```
function plot(x, y, c) is
    // рисует точку с координатами (x, y)
    // и яркостью c (где 0 ≤ c ≤ 1)

function ipart(x) is
    return целая часть от x

function round(x) is
    return ipart(x + 0.5) // округление до ближайшего целого

function fpart(x) is
    return дробная часть x

function draw_line(x1,y1,x2,y2) is
   if x2 < x1 then
       swap(x1, x2)
       swap(y1, y2)
   end if

   dx := x2 - x1
   dy := y2 - y1
   gradient := dy ÷ dx

   // обработать начальную точку
   xend := round(x1)
   yend := y1 + gradient × (xend - x1)
   xgap := 1 - fpart(x1 + 0.5)
   xpxl1 := xend  // будет использоваться в основном цикле
   ypxl1 := ipart(yend)
   plot(xpxl1, ypxl1, (1 - fpart(yend)) × xgap)
   plot(xpxl1, ypxl1 + 1, fpart(yend) × xgap)
   intery := yend + gradient // первое y-пересечение для цикла

   // обработать конечную точку
   xend := round(x2)
   yend := y2 + gradient × (xend - x2)
   xgap := fpart(x2 + 0.5)
   xpxl2 := xend  // будет использоваться в основном цикле
   ypxl2 := ipart(yend)
   plot(xpxl2, ypxl2, (1 - fpart(yend)) × xgap)
   plot(xpxl2, ypxl2 + 1, fpart(yend) × xgap)

   // основной цикл
   for x from xpxl1 + 1 to xpxl2 - 1 do
           plot(x, ipart(intery), 1 - fpart(intery))
           plot(x, ipart(intery) + 1, fpart(intery))
           intery := intery + gradient
   repeat
end function
```

Гамма-коррекция

Рассмотрим сначала частичное решение (работаем на черном фоне), когда вы уже вычислили интенсивность пикселя, который нужно нарисовать. Самое время отобразить их на мониторе. На заре цифровой обработки изображений большинство мониторов имели электронно-лучевые трубки (ЭЛТ). Этот тип мониторов имел физическую особенность: повышение входного напряжение в два раза не означало двукратного увеличения яркости. Зависимость между входным напряжением и яркостью выражалась степенной функцией, с показателем примерно 2.2, также известным как гамма монитора.

Эта особенность мониторов (по случайному совпадению) очень напоминает то, как люди воспринимают яркость: с подобной же (но обратной) степенной зависимостью. Чтобы лучше это понять, взгляните на следующее изображение:

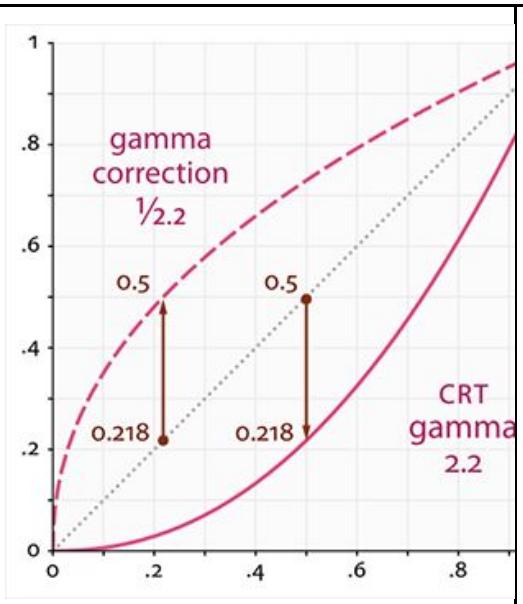


Vs показывает как воспринимается яркость человеческим глазом: при увеличении яркости в 2 раза (например, от 0.1 до 0.2) картинка действительно выглядит так, будто она в два раза ярче: изменения видны довольно отчетливо. Однако, когда мы говорим о физической яркости света, как, например, о количестве фотонов, выходящих из источника света, верную картину дает нижняя шкала (I). На ней удвоение значения дает правильную с физической точки зрения яркость, но поскольку наши глаза более восприимчивы к изменениям темных цветов, это кажется несколько странным.

Поскольку для человеческого глаза более привычен верхний вариант, мониторы и по сей день используют степенную зависимость при выводе цветов, так что исходные, в физическом смысле, значения яркости преобразуются в нелинейные значения яркости, изображенные на верхней шкале. В основном это сделано потому, что так выглядит лучше.

Эта особенность мониторов действительно делает картинку лучше для наших глаз, но когда дело доходит до рендеринга графики появляется одна проблема: все параметры цвета и яркости, которые мы устанавливаем в

наших приложениях, основаны на том, что мы видим на мониторе. А это означает что все эти параметры на самом деле являются нелинейными.



- 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

Когда мы хотим получить в 2 раза более яркий цвет в линейном пространстве, мы просто берем и удваиваем его значение. Например, возьмем яркость 0.5. Если бы мы удвоили его значение в линейном пространстве, он стал бы равным 1. С другой стороны, при выводе на дисплей, он будет преобразован в цветовое пространство монитора как 0.218, как видно из графика. Вот здесь и возникает проблема: удваивая цвет в линейном пространстве, мы фактически делаем его более чем в 4.5 раза ярче на мониторе.

До этого момента мы предполагали, что работали в линейном пространстве, но на самом деле мы работали в цветовом пространстве, определяемом монитором, поэтому все установленные нами цвета были физически не корректны, а всего лишь выглядели правильными конкретно на нашем мониторе. Руководствуясь данным предположением принято устанавливать значения освещения ярче, чем они должны быть (т.к. монитор затемняет их), что в результате делает большинство последующих вычислений в линейном пространстве неверными. Также обратите внимание, что оба графика начинаются и заканчиваются в одних и тех же точках, затемнению на дисплее подвержены только промежуточные цвета.

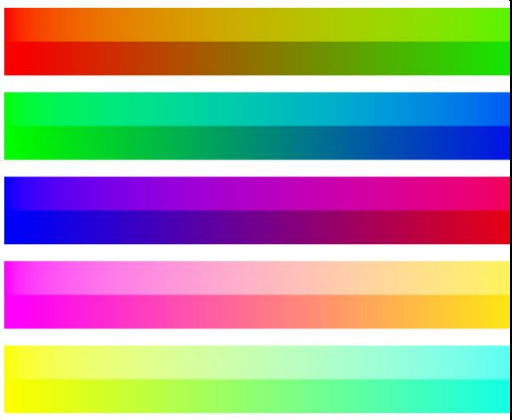
Идея гамма-коррекции заключается в том, чтобы применить инверсию гаммы монитора к окончательному цвету перед выводом на монитор

(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.

Приведем еще один пример. Допустим, у нас опять есть наш серый цвет 0.5. Перед отображением этого цвета на монитор мы сперва применяем кривую гамма-коррекции к его компонентам. Значения цвета в линейном пространстве при отображении на мониторе возводятся в степень, приблизительно равную 2.2, поэтому инверсия требует от нас возведения значений в степень 1/2.2. Таким образом, наш цвет с гамма-коррекцией становится 0.5 ^ 1/2.2 = 0.73. Затем этот скорректированные цвет выводится на монитор, и в результате он отображается как 0.73 ^ 2.2 = 0.5.

Как видите, когда мы используем гамма-коррекцию монитор отображает цвета, точно такими, какими мы задаем их в линейном пространстве в нашем приложении.

Гамма равная 2.2 это дефолтное значение, которое приблизительно выражает среднюю гамму большинства дисплеев. Цветовое пространство в результате применения этой гаммы называется цветовым пространством sRGB. Каждый монитор имеет свои собственные гамма-кривые, но значение 2.2 дает хорошие результаты на большинстве мониторов. Из-за этих небольших отличий многие игры позволяют игрокам изменять настройку гаммы.



- Пример на цветных градиентах:
- сверху - с применением гамма коррекции
- снизу - без применения гамма коррекции

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.

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. Приблизительно можно считать, что гамма равна 2.2. Гамма может изменяться от 1.0 до 2.3.

Альфа смешивание

Альфа-значение определяет степень прозрачности конкретного пикселя. Объекты могут иметь различную прозрачность, например 8-битный альфа-канал может представлять 256 уровней прозрачности: от 0 (всё растровое изображение прозрачное) до 255 (всё растровое изображение непрозрачное).

Альфа-смешение есть процесс комбинирования двух объектов на экране с учётом их альфа-каналов. Альфа-смешение используется для: антиалиасинга; создания прозрачности, теней, зеркал, тумана.

Расчёт яркости результирующего пикселя после наложения двух пикселей друг на друга выполняется по формуле:

$R = B \cdot (1 - A) + F \cdot A$ или в иной записи: $R = B + (F - B) \cdot A$

Обозначения:

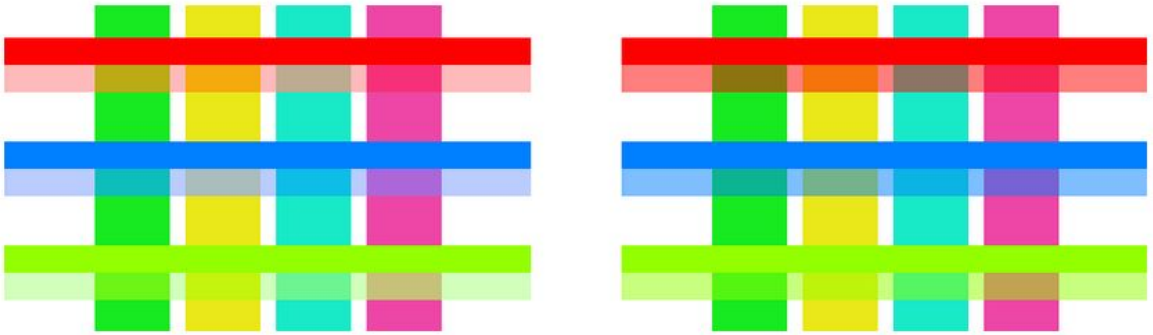
B — яркость фонового пикселя;

F — яркость накладываемого пикселя;

$A \in [0..1]$ — непрозрачность накладываемого пикселя;

R — результат.

Вторая запись отображает следующий смысл: значение A указывает относительное положение на отрезке [B..F].



Эффекты гамма-некорректного альфа-смешения. Для каждой пары вертикальных столбцов верхняя рисуется с непрозрачностью 100%, а нижняя с 50%.
Левое изображение - гамма-правильное.

Экспериментальная часть

Язык программирования: C++ 14

Выполнено полное решение

Для работы с отрисовкой линии использовался следующий алгоритм: прямая задается двумя точками. Проведем к прямой серединный перпендикуляр. Для этого найдем две точки, лежащие на нем. Зададим y или x(если x2==x1) точки серединного перпендикуляра по формуле в точках 0 и 1.

$$x = \frac{y_2-y_1}{x_2-x_1}y + \frac{x_1+x_2}{2} + \frac{y_2-y_1}{x_2-x_1}\frac{y_1+y_2}{2}$$

$$y = \frac{x_2-x_1}{y_2-y_1}x + \frac{y_1+y_2}{2} + \frac{x_2-x_1}{y_2-y_1}\frac{x_1+x_2}{2}$$

Получим две прямые, которые пересекаются в центре линии. Рассчитаем половину длины линии.

$$l = \sqrt{(y_2-y_1)^2 + (x_2-x_1)^2}$$

Представим линию как прямоугольник. Тогда точка лежит в прямоугольнике, если расстояние от этой точки до исходной линии меньше, чем половина толщины, а до серединного перпендикуляра меньше длины. Воспользуемся формулой расстояния от точки до прямой, заданной двумя точками:

$$dist = \left| \frac{(y_2-y_1)x-(x_2-x_1)y+x_2y_1-y_2x_1}{\sqrt{(y_2-y_1)^2+(x_2-x_1)^2}} \right|$$

Так как нам нужна линия со сглаживанием, воспользуемся следующим способом для определения яркости пикселя в точке: разделим каждый пиксель на 256 маленьких квадратиков. Будем искать расстояние от каждого маленького квадратика до двух прямых и считать, сколько квадратиков каждого пикселя попадают на прямую и закрашивать этот пиксель в соответствии с этим. Немного оптимизируем алгоритм и чтобы не производить вычисления для всех пикселей, сначала выделим пиксели, которые вообще нужно обрабатывать, а потом будем делить на части только их. В решении используется дополнительная память на хранение пикселей, которые нужно будет закрашивать, что ухудшает эффективность по памяти, но улучшает по времени.

Выводы

В ходе работы был разработан алгоритм, позволяющий рисовать красивые сглаженные линии, а также изучены алгоритмы гамма-коррекции. Алгоритм рисования линий можно было бы улучшить с точки зрения эффективности по памяти, если бы я не складывала в set пиксели, которые нужно обрабатывать, а сразу находила из цвет.

Листинг

main.cpp

```
#include <iostream>
#include <cstdlib>
#include <set>
#include "NetPBM.h"

using namespace std;

int main(int argc, char **argv) {
   try {
       if (argc == 666) {
```

```
            throw logic_error("Invalid user");
        }
        if (argc != 9 && argc != 10) {
            throw logic_error("Invalid argument count");
        }
        char *infilename = argv[1];
        char *outfilename = argv[2];
        int brightness = atoi(argv[3]);
        double line_width = atof(argv[4]);
        auto *point_1 = new point_t();
        (*point_1).x = atof(argv[5]);
        (*point_1).y = atof(argv[6]);
        auto *point_2 = new point_t();
        (*point_2).x = atof(argv[7]);
        (*point_2).y = atof(argv[8]);
        double gamma = -1;
        if (argc == 10) {
            gamma = atof(argv[9]);
            if (gamma <= 0) {
                throw logic_error("Invalid gamma");
            }
        }
        auto * file = new ifstream (infilename,
std::ios::binary);
        (*file).unsetf(ios_base::skipws);
        if (!(*file).is_open()) {
            delete file;
            delete point_1;
            delete point_2;
            throw logic_error("Can't open file to read");
        }
```

```
        auto *netPcm = new NetPBM(file);
        netPcm->draw_thick_line(point_1, point_2, line_width,
brightness, gamma);
        auto* outfile = new ofstream (outfilename,
std::ios::binary);
        if (!outfile->is_open()) {
            file->close();
            delete file;
            delete outfile;
            delete netPcm;
            delete point_1;
            delete point_2;
            throw logic_error("Can't open file to write");
        }
        netPcm->write_to_file(outfile);
        outfile->close();
        file->close();
        delete file;
        delete outfile;
        delete netPcm;
        delete point_1;
        delete point_2;
        return 0;
    }
    catch (const logic_error &error) {
        cerr << error.what() << endl;
        return 1;
    }
}
```

NetPBM.h

```cpp
#include <iostream>
#include <fstream>
#include <cstdint>
#include <cstdlib>
#include <cmath>
#include <set>

#ifndef LAB2KGIG_NETPBM_H
#define LAB2KGIG_NETPBM_H

using namespace std;
struct point_t {
   double x;
   double y;
};

class NetPBM {
private:
   ifstream *file;
   uint16_t type = -1;
   uint16_t width = -1;
   uint16_t height = -1;
   uint16_t depth = -1;
   unsigned char **array;

   void read_header() {
       unsigned char buf;
       *this->file >> buf;
       if (buf != 'P') {
           throw logic_error("Bad file");
       }
```

```
        *this->file >> buf;
        if (buf == '1' || buf == '2' || buf == '3' || buf == '4'
|| buf == '6' || buf == '7') {
            throw logic_error("Not supported type of NetPCM");
        } else if (buf == '5')
            this->type = 5;
        else {
            throw logic_error("Bad file");
        }
        *this->file >> buf;
        *this->file >> this->width;
        *this->file >> buf;
        *this->file >> this->height;
        *this->file >> buf;
        *this->file >> this->depth;
        if (this->depth != 255) {
            throw logic_error("Not supported non 255 colors
count");
        }
    }

    void read_data() {
        for (int i = 0; i < this->height; i++) {
            for (int j = 0; j < this->width; j++) {
//                array[i][j] = 0;
                *this->file >> this->array[i][j];
            }
        }
    }
```

```
//
https://en.wikipedia.org/wiki/SRGB#The_sRGB_transfer_function_(%
22gamma%22)
   static double gammasRGB(double u) {
       if (u <= 0.0031308) {
           return 323 * u / 25.;
       } else {
           return (211. * pow(u, 5. / 12.) - 11.) / 200.;
       }
   }

   static double ungammasRGB(double u) {
       if (u <= 0.04045) {
           return 25 * u / 323;
       } else {
           return pow((200. * u + 11.) / 211., 12. / 5.);
       }
   }

   static double gamma(double u, double gamma) {
       return pow(u, 1.0 / gamma);
   }

   static double ungamma(double u, double gamma) {
       return pow(u, gamma);
   }

   void draw_point(int x, int y, double pixel_color, int
brightness, double gamma_value) {
       double brightness_value = ((double) brightness) /
(double) depth;
```

```
        if (!(x < width && y < height && x >= 0 && y >= 0 &&
brightness >= 0 && brightness <= depth))
            throw invalid_argument("Invalid argument");


        double value = ((double) this->array[y][x]) /
this->depth;
        if (gamma_value == -1) {
            value = ungammasRGB(value);
            brightness_value = ungammasRGB(brightness_value);
        } else {
            value = ungamma(value, gamma_value);
            brightness_value = ungamma(brightness_value,
gamma_value);
        }
        value = value + pixel_color * (brightness_value - value);
        if (gamma_value == -1) {
            value = gammasRGB(value);
        } else {
            value = gamma(value, gamma_value);
        }
        if (value >= 1 - 1 / 1e9)
            value = 1;
        int temp = round(this->depth * value);
        if (temp > 255)
            temp = 255;
        this->array[y][x] = (unsigned char) temp;
    }


    static void swap(point_t *a, point_t *b) {
        auto *buf = new point_t;
        *buf = *a;
        *a = *b;
```

```
        *b = *buf;
        delete buf;
    }

public:
    int16_t getType() const {
        return type;
    }

    double perp_point_y(double x1, double y1, double x2, double
y2, double x) {
        return (-(x2 - x1) / (y2 - y1) * x + (y1 + y2) / 2 + (x2
- x1) / (y2 - y1) * (x1 + x2) / 2);
    }

    double perp_point_x(double x1, double y1, double x2, double
y2, double y) {
        return (-(y2 - y1) / (x2 - x1) * y + (x1 + x2) / 2 + (y2
- y1) / (x2 - x1) * (y1 + y2) / 2);
    }

    double dist(double x1, double y1, double x2, double y2,
double x, double y) {
        return abs((y2 - y1) * x - (x2 - x1) * y + x2 * y1 - y2 *
x1) /
               (sqrt(pow((y2 - y1), 2) + pow((x2 - x1), 2)));
    }

    void draw_thick_line(point_t *point_1, point_t *point_2,
double thickness, int brightness, double gamma_value) {
        double dy = abs(point_2->y - point_1->y);
        auto *rline_points = new set<pair<int, int>>;
        double dx = abs(point_2->x - point_1->x);
```

```
        if (dx != 0) {
            if (point_2->x < point_1->x) {
                swap(point_1, point_2);
            }
            double grad = 3 * thickness / dx;
            for (double i = point_1->x; i <= point_2->x;) {
                double j = (i - point_1->x) * (point_2->y -
point_1->y) / (point_2->x - point_1->x) + point_1->y;
                for (int x = (int) (i - thickness / 2.) - 3;
                     x <= (int) (i + thickness / 2. + 1) + 3;
x++) { // Find pixels in circle by searching in square
                    for (int y = (int) (j - thickness / 2.) - 3;
y <= (int) (1 + j + thickness / 2.) + 3; y++) {
                        if (x >= 0 && y >= 0 && x < this->width
&& y < this->height)
                            rline_points->insert(make_pair(x,
y));
                    }
                }
                i += grad;
            }
        } else {

            if (point_2->y < point_1->y) {
                swap(point_1, point_2);
            }
            double grad = 3 * thickness / dy;
            for (double i = point_1->y; i <= point_2->y;) {
                double j = (i - point_1->y) * (point_2->x -
point_1->x) / (point_2->y - point_1->y) + point_1->x;
                for (int y = (int) (i - thickness / 2.) - 3;
                     y <= (int) (i + thickness / 2. + 1) + 3;
y++) { // Find pixels in circle by searching in square
```

```
                    for (int x = (int) (j - thickness / 2.) - 3;
x <= (int) (1 + j + thickness / 2.) + 5; x++) {
                        if (x >= 0 && y >= 0 && x < this->width
&& y < this->height)
                            rline_points->insert(make_pair(x,
y));
                    }
                }
                i += grad;
            }
        }
        double koef;
        double y_perp_0, y_perp_1, x_perp_0, x_perp_1;
        if (dy != 0) {
            y_perp_0 = perp_point_y(point_1->x, point_1->y,
point_2->x, point_2->y, 0);
            y_perp_1 = perp_point_y(point_1->x, point_1->y,
point_2->x, point_2->y, 1);
        } else {
            x_perp_0 = perp_point_x(point_1->x, point_1->y,
point_2->x, point_2->y, 0);
            x_perp_1 = perp_point_x(point_1->x, point_1->y,
point_2->x, point_2->y, 1);


        }
        double line_length = (sqrt(pow((point_2->y - point_1->y),
2) + pow((point_2->x - point_1->x), 2)));


        for (pair<int, int> p : *rline_points) {
            int color = 0;
            int x = p.first;
            int y = p.second;
            for (int i = 0; i < 16; i++) {
```

```
                for (int j = 0; j < 16; j++) {
                    double x_choord = (double) x + ((double) i) /
16. + 1 / 32.;
                    double y_choord = (double) y + ((double) j) /
16. + 1 / 32.;
                    double s = dist(point_1->x, point_1->y,
point_2->x, point_2->y, x_choord, y_choord);
                    double r;
                    if (dy != 0)
                        r = dist(0, y_perp_0, 1, y_perp_1,
x_choord, y_choord);
                    else
                        r = dist(x_perp_1, 1, x_perp_0, 0,
x_choord, y_choord);
                    if (s < thickness / 2. && r < line_length /
2.)
                        color++;

                }
            }
            if (color > 255)
                color = 255;
            draw_point(x, y, color / 255., brightness,
gamma_value);
        }

//        }
        delete rline_points;
    }

    void write_to_file(ofstream *outfile) {
        *outfile << "P5" << (unsigned char) (10) << width << " "
<< height << (unsigned char) (10) << depth
```

```
                << (unsigned char) (10);
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                *outfile << array[i][j];
            }
        }
    }

    explicit
    NetPBM(ifstream
           *file) {
        this->file = file;
        this->read_header();
        file->ignore(1);
        this->array = new unsigned char *[height];
        for (int i = 0; i < this->height; i++) {
            this->array[i] = new unsigned char[width];
        }
        read_data();
    }

    ~NetPBM() {
        for (int i = 0; i < height; i++) {
            delete[] array[i];
        }
        delete[] array;
    }
};


#endif //LAB2KGIG_NETPBM_H
```