**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное

учреждение высшего образования

**Санкт-Петербургский исследовательский университет**

**Информационных технологий, механики и оптики**

Факультет: ФИТиП

Дисциплина: Компьютерная графика и геометрия

**Отчет**

по лабораторной работе № 2

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

Выполнила: студент гр. M3101

Дымчикова А. Ч.

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**

Так как человеческий глаз воспринимает изменения темных цветов лучше, чем светлых, то линейное увеличение физической яркости света для человека выглядит совсем иначе. По этой причине и используется гамма-коррекция: мониторы по сей день используют степенную зависимость при выводе цветов, так что исходные, в физическом смысле, значения яркости преобразуются в нелинейные значения яркости, более приятные для человека.

|  |  |
| --- | --- |
| D:\Downloads\screenshot-docs.google.com-2020.07.02-17_21_28.png | * 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) |

Идея гамма-коррекции заключается в том, чтобы применить инверсию гаммы монитора к окончательному цвету перед выводом на монитор. Если посмотреть на график гамма-кривой, то линию, обозначенная штрихами, будет являться обратной для гамма-кривой монитора. При умножении выводимых значений цветов в линейном пространстве на эту обратную гамма-кривую цвета становятся ярче, и, как только они будут выведены на монитор, к ним применится гамма-кривая монитора, - результирующие цвета снова станут линейными. По сути, яркость промежуточных цветов увеличивается, чтобы сбалансировать затенение цветов монитором.

Так, к примеру, если взять серый цвет 0.5, то перед отображением этого цвета на монитор нужно сначала применить 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, данный цвет с гамма-коррекцией становится равным 0.5 ^ 1/2.2 = 0.73. Затем этот скорректированный цвет выводится на монитор, и в результате он отображается как 0.73 ^ 2.2 = 0.5.

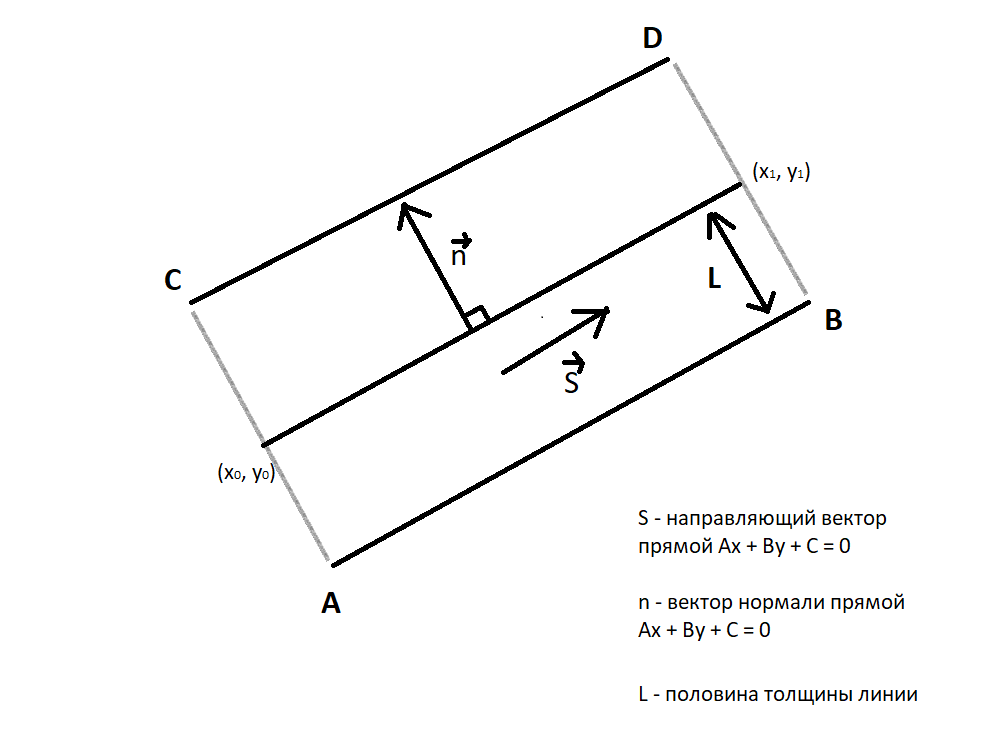
Гамма, равная 2.2, - это дефолтное значение, которое приблизительно выражает среднюю гамму большинства дисплеев. Цветовое пространство в результате применения этой гаммы называется цветовым пространством sRGB. Каждый монитор имеет свои собственные гамма-кривые, но значение 2.2 дает хорошие результаты на большинстве мониторов.

**Экспериментальная часть**

Выполнено полное решение лабораторной работы.

Язык программирования: С++17.

Необходимо нарисовать растровую линию между двумя заданными точками. Можно представить данную линию, как прямоугольник данной толщины, заданный с помощью координат своих вершин.

Найдем уравнение данной прямой.

Представление линии как прямоугольника:

– вектор нормали к данной прямой. Если сдвинуть точку начала или конца прямой по вектору нормали на половину заданной толщины вверх или вниз, то получится вершина прямоугольника. Найдем вершины прямоугольника таким образом.

– уравнение прямой, перпендикулярной данной, и проходящей через точку в параметрическом виде.

Тогда .

С помощью представленных выше формул находятся вершины прямоугольника, а для каждого пикселя высчитывается его пересечение с найденной фигурой. В зависимости от отношения площади пересечения к площади пикселя считается итоговая яркость данного пикселя.

Затем находится яркость этого пикселя с учетом гамма-коррекции и записывается в файл.

Время работы алгоритма прямопропорционально количеству пикселей в изображении и времени работы функции для нахождения пересечения пикселя и прямоугольника.

**Выводы**

В результате выполненной работы был реализован алгоритм отрисовки растровых линий с применением сглаживания и гамма-коррекции. Возможно улучшение времени работы программы при выборе более быстрой функции для нахождения пересечения пикселя и прямоугольника.

**Листинг**

**PNM\_ex.h**

#pragma once

#include <vector>

#include <string>

**class** **PNM\_Header** {

**public:**

**char** format;

**int** width;

**int** height;

**int** depth;

PNM\_Header();

PNM\_Header(**FILE** \*fin);

};

**class** **PNM\_File** {

**public:**

PNM\_Header header;

std::vector <**unsigned** **char**> data;

**int** size = **0**;

PNM\_File();

PNM\_File(**FILE** \*fin);

};

**PNM\_ex.cpp**

#include "PNM\_ex.h"

PNM\_Header::PNM\_Header() {};

PNM\_Header::PNM\_Header(**FILE** \*fin) {

**char** checker;

**int** check = fscanf(fin, "%c%c**\n**%d %d**\n**%d**\n**", &checker, &format, &width, &height, &depth);

**if** (check != **5** || checker != 'P' || format != '5' && format != '6'

|| width <= **0** || height <= **0** || depth != **255**) {

**throw** std::exception("Invalid data format!**\n**");

}

}

PNM\_File::PNM\_File() {};

PNM\_File::PNM\_File(**FILE** \*fin) {

header = PNM\_Header(fin);

**if** (header.format == '5') {

size = header.width \* header.height;

}

**else** {

size = header.width \* header.height \* **3**;

}

data.resize(size);

**int** check = fread(&data[**0**], **sizeof**(**unsigned** **char**), size, fin);

**if** (check != size) {

**throw** std::exception("Invalid data format: width or**\a**nd height are wrong!**\n**");

}

}

**geom.h**

#pragma once

**const** **double** EPS = **0.1**;

**class** **Point** {

**public:**

**double** x, y;

Point();

Point(**double**, **double**);

};

**class** **Vect** {

**public:**

**double** x, y;

Vect(**double**, **double**);

Vect(Point, Point);

**double** **len**();

};

**class** **Rect** {

**public:**

Point A, B, C, D;

Rect(Point, Point, Point, Point);

};

**geom.cpp**

#include "geom.h"

#include <cmath>

Point::Point() : x(**0**), y(**0**) {}

Point::Point(**double** x, **double** y) : x(x), y(y) {}

**double** Vect::len() {

**return** std::sqrt(x \* x + y \* y);

}

Vect::Vect(**double** x, **double** y) {

x = x;

y = y;

}

Vect::Vect(Point a, Point b) {

x = b.x - a.x;

y = b.y - a.y;

}

Rect::Rect(Point a, Point b, Point c, Point d) : A(a), B(b), C(c), D(d) {};

**Functs.h**

#pragma once

#include <algorithm>

#include "PNM\_ex.h"

#include "geom.h"

**void** **inverse**(PNM\_File &file);

**void** **vertical\_reflection**(PNM\_File &file);

**void** **horisontal\_reflection**(PNM\_File &file);

**void** **rotate\_counter\_clockwise**(PNM\_File &file);

**void** **rotate\_clockwise**(PNM\_File &file);

**double** **get\_vect\_mult**(Point a, Point b);

Vect **get\_norm**(Point a, Point b);

Rect **get\_line**(Point a, Point b, **double** thick);

**bool** **get\_pos**(Point a, Rect rec);

**double** **get\_conj**(**int** x, **int** y, Rect rec);

**double** **do\_gamma**(**double** color, **double** gamma);

**void** **draw\_line**(PNM\_File &file, **int** bright, **double** thick, Point s, Point f, **double** gamma);

**Functs.cpp**

#include <algorithm>

#include <vector>

#include "PNM\_ex.h"

#include "geom.h"

**double** **get\_vect\_mult**(Vect a, Vect b) {

**return** a.x \* b.y - a.y \* b.x;

}

Vect **get\_norm**(Point a, Point b) {

**return** Vect(a, b);

}

Rect **get\_line**(Point a, Point b, **double** thick) {

Vect s = get\_norm(a, b);

**double** l = s.len();

s.x = s.x \* thick / (**2.** \* l);

s.y = s.y \* thick / (**2.** \* l);

**return** Rect(Point(-s.y + a.x, s.x + a.y), Point(s.y + a.x, -s.x + a.y),

Point(s.y + b.x, -s.x + b.y), Point(-s.y + b.x, s.x + b.y));

}

**bool** **get\_pos**(Point a, Rect rec) {

**double** mult = get\_vect\_mult(Vect(rec.A, rec.B), Vect(a, rec.A));

**double** sign = mult / (mult == **0** ? **1** : abs(mult));

mult = get\_vect\_mult(Vect(rec.B, rec.C), Vect(a, rec.B));

**if** (sign == **0** && mult != **0**) {

sign = mult / abs(mult);

}

**if** (sign != mult / (mult == **0** ? **1** : abs(mult))) {

**return** false;

}

mult = get\_vect\_mult(Vect(rec.C, rec.D), Vect(a, rec.C));

**if** (sign == **0** && mult != **0**) {

sign = mult / abs(mult);

}

**if** (sign != mult / (mult == **0** ? **1** : abs(mult))) {

**return** false;

}

mult = get\_vect\_mult(Vect(rec.D, rec.A), Vect(a, rec.D));

**if** (sign != mult / (mult == **0** ? **1** : abs(mult))) {

**return** false;

}

**return** true;

}

**double** **get\_conj**(**int** x, **int** y, Rect rec) {

**int** cnt = **0**;

**int** size = **0**;

**for** (**double** i = x; i < x + **1**; i += EPS) {

**for** (**double** j = y; j < y + **1**; j += EPS, size++) {

**if** (get\_pos(Point(i, j), rec)) {

cnt++;

}

}

}

**return** **double**(cnt) / size;

}

**double** **do\_gamma**(**double** color, **double** gamma) {

**return** pow(color, gamma);

}

**void** **centring**(Point &a) {

**if** (a.x - **int**(a.x) <= EPS) {

a.x += **0.5**;

}

**if** (a.y - **int**(a.y) < EPS) {

a.y += **0.5**;

}

}

**void** **draw\_line**(PNM\_File &file, **int** bright, **double** thick, Point s, Point f, **double** gamma) {

**int** w = file.header.width;

**int** h = file.header.height;

centring(s);

centring(f);

Rect rec = get\_line(s, f, thick);

**for** (**int** x = **0**; x < h; x++) {

**for** (**int** y = **0**; y < w; y++) {

**double** conj = get\_conj(x, y, rec);

**double** tmp = do\_gamma(file.data[x \* w + y] / **255.**, gamma) \* (**1.0** - conj) + conj \* do\_gamma(**double**(bright) / **255.**, gamma);

tmp = do\_gamma(tmp, **1.** / gamma);

**if** (tmp > **0.9999**) {

tmp = **1.0**;

}

file.data[x \* w + y] = tmp \* **255**;

}

}

}

**main.cpp**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include "PNM\_ex.h"

#include "Functs.h"

#include "geom.h"

#include <stdlib.h>

**using** **namespace** std;

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

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

cerr << "Wrong number of arguments!";

**return** **1**;

}

**FILE** \*fin = fopen(argv[**1**], "rb");

**if** (!fin) {

cerr << "Input file error!**\n**";

cerr << argv[**1**];

**return** **1**;

}

PNM\_File file;

try {

file = PNM\_File(fin);

}

**catch** (exception err) {

fclose(fin);

cerr << err.what();

**return** **1**;

}

fclose(fin);

draw\_line(file, atoi(argv[**3**]), atof(argv[**4**]), Point(atof(argv[**6**]), atof(argv[**5**])), Point(atof(argv[**8**]), atof(argv[**7**])), (argc == **10** ? atof(argv[**9**]) : **2.2**));

**FILE** \*fout = fopen(argv[**2**], "wb");

**if** (!fout) {

cerr << "Output file error!";

**return** **1**;

}

fprintf(fout, "P%c**\n**%d %d**\n**%d**\n**", file.header.format, file.header.width, file.header.height, file.header.depth);

fwrite(&file.data[**0**], **sizeof**(**unsigned** **char**), file.size, fout);

fclose(fout);

**return** **0**;

}