**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 гамма-коррекции***

Выполнила: студентка гр. M3102

Абрамова У.В.

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**

Уравнение прямой из точки в точку имеет вид:

Для реализации рисования такой линии используется алгоритм Брезенхема. ***Алгоритм Брезенхема*** — это алгоритм, определяющий, какие точки двумерного растра нужно закрасить, чтобы получить близкое приближение прямой линии между двумя заданными точками.

Но при использовании данного алгоритма линия получается «ступенчатая». Поэтому здесь логичнее использовать его модернизацию – алгоритм Ву. ***Алгоритм Ву*** — это алгоритм разложения отрезка в растр со сглаживанием.

Отличие от алгоритма Брезенхема состоит в том, что в алгоритме Ву на каждом шаге устанавливается не одна, а две точки. Например, если основной осью является Х, то рассматриваются точки с координатами (х, у) и (х, у + 1). В зависимости от величины ошибки, которая показывает, как далеко ушли пиксели от идеальной линии по неосновной оси, распределяется интенсивность между этими двумя точками. Чем больше удалена точка от идеальной линии, тем меньше её интенсивность.

Также, в данной лабораторной работе к изображению необходимо применить гамма-коррекцию. ***Гамма*** определяет отношение между численным значением пикселя и его действительной светимостью. Идея гамма-коррекции заключается в том, чтобы применить инверсию гаммы монитора к окончательному цвету перед выводом на монитор. Для гамма-коррекции используются две формулы:

*– прямая формула гамма-коррекции*

*– обратная формула гамма-коррекции*

И также, необходимо ввести понятие 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 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.

*– прямая формула sRgb*

*– обратная формула sRGB*

**Экспериментальная часть**

Язык программирования: С++/17.

Для рисования линий толщины 1, используется обычный алгоритм Ву. Алгоритм работает только для линий, у которых приращение больше по иксу, чем по игреку, поэтому реализуем функцию, которая меняет местами х и у, и в дальнейшем, при рисовании точки, координаты меняются обратно. Также, необходимо поменять начало и конец линии, если линию задали справа налево (алгоритм работает для линий слева направо). Еще стоит обозначить, что для удобства адресации в массиве, ось у направим вниз, ось х – вправо.

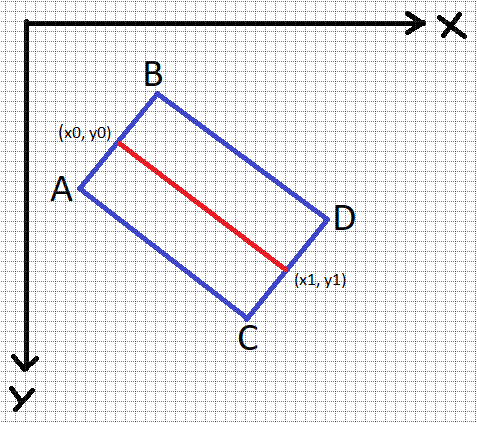
В функции рисования точки поставим условия выхода, если х или у вышли за границы изображения, или интенсивность равна нулю.

При работе с фоном, применяем прямую гамму или обратное sRGB. Также изначально применяем эти же формулы на введенную яркость. При записи в файл применяем обратную гамму или прямое sRGB.

Также используется формула альфа-смешивания: **R = B + (F − B) ⋅ A**, где В – яркость фонового пикселя, F – яркость накладываемого пикселя, А – непрозрачность накладываемого пикселя, R – результат.

Теперь про реализацию различной толщины. Здесь ситуация делится на два случая: 1) Толщина < 1 – тогда линия рисуется также как в случае толщины = 1, только с пропорционально меньшей яркостью. 2) Толщина > 1. В этом случае воспользуемся геометрией.

Нужная нам линия будет иметь вид:

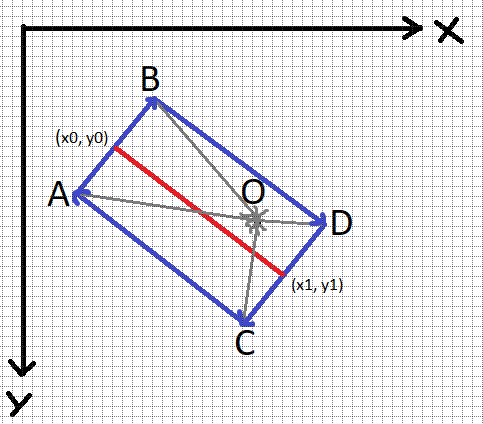


Далее, зная координаты х0, у0, х1 и у1, рассчитаем координаты точек A, B, C, D по формуле перемещения точки на заданный вектор:

где – толщина пополам, = . Найдя данные точки, с помощью уже реализованного алгоритма Ву, построим линии из А в В, из В в D, из D в С и из С в А.

Для горизонтальных и вертикальных линий немного по-другому: для горизонтальных, к координатам х точек А и С прибавляем 1, а для вертикальных к координатам у точек А и В прибавляем 1 – это чтобы линии краев (если они разных яркостей) не заслоняли друг друга.

Теперь нужно заполнить образовавшийся прямоугольник цветом. Реализуем функцию, проверяющую, принадлежит ли точка прямоугольнику. Для этого нужно ввести новые векторы:



Найдем углы BOA, BOD, DOC и COA. Если их сумма будет равна 360 – точка находится внутри прямоугольника. Таким образом закрашиваем линию.

Для угла в 45 градусов немного иначе: мы сначала закрашиваем теоретический прямоугольник, а потом рисуем контуры. И причем контуры рисуем, слегка измененным алгоритмом Ву: мы рисуем не два пикселя за раз, а один – иначе, из-за небольшой погрешности вычислений, контур скрывается за заливкой.

Также, реализована дополнительная функция проверки корректности, т.к. из-за работы с вещественными числами точки, за редким исключением, могут ошибочно не быть признаны находящимися внутри прямоугольника. Данная функция проверяет – если все соседи по клеткам точки закрашены, а сама точка нет, значит соответственно это точка не закрашена ошибочно и ее следует закрасить. Таким образом нейтрализуется погрешность проверки.

**Выводы**

В ходе проделанной работы была получена программа, реализующая рисование сглаженных линий по заданным координатам заданной толщины с применением гамма-коррекции или sRGB.

Полученные результаты не совсем совпадают с теоретическими, из-за того, что вычисляемые значения точек – вещественные числа, а координаты пикселей – целые. Также проявляется множество погрешностей при вычислении координат необходимых точек и значений углов, из-за невозможности хранения значений с бесконечной дробной частью в памяти компьютера. Поэтому получаемая линия не всегда полностью совпадает с желаемой.

**Листинг**

**main.cpp**

#include <iostream>  
#include <algorithm>  
#define \_USE\_MATH\_DEFINES  
#include <cmath>  
#include <string.h>  
using namespace std;  
typedef unsigned char uchar;  
  
uchar\* arr;  
int\* buffer;  
int format, width, height, a;  
int bright;  
bool dir;  
double gamma\_ = -1;  
double fatness;  
double xa, ya, xb, yb, xc, yc, xd, yd;  
double alpha;  
char filename\_in[100], filename\_out[100];  
  
double GammaCorrection(double Ipix, int flag) { // 1 - прямая гамма, -1 - обратная  
 double I = pow(Ipix / 255, pow(gamma\_, flag));  
 return I \* 255;  
}  
  
double sRGB(double pix) {  
 pix /= 255;  
 if(pix <= 0.0031308)  
 return 12.92 \* pix \* 255;  
 else return (1.055 \* pow(pix, 0.4167) - 0.055) \* 255;  
}  
  
double sRGB\_reverse(double pix) {  
 pix /= 255;  
 if(pix <= 0.04045)  
 return pix / 12.92 \* 255;  
 else return pow((pix + 0.055) / 1.055, 2.4) \* 255;  
}  
  
void Plot(int x, int y, double intensity) {  
 if((x >= width) || (y >= height) || (intensity == 0) || (x < 0) || (y < 0))  
 return;  
 if(dir)  
 swap(x, y);  
 if(fatness < 1)  
 intensity \*= fatness;  
 double back = 0;  
 if(intensity != 1) {  
 back = (int) arr[y \* width + x];  
 if(gamma\_ != -1)  
 back = GammaCorrection(back, 1);  
 else back = sRGB\_reverse(back);  
 }  
 int z = (back + ((double)(bright - back) \* intensity));  
 if(gamma\_ != -1)  
 buffer[y \* width + x] = GammaCorrection(z, -1);  
 else buffer[y \* width + x] = sRGB(z);  
}  
  
void OutputImage() {  
 FILE\* fout = fopen(filename\_out, "wb");  
 if(fout == NULL) {  
 std::cerr << "Cannot create output file!\n";  
 return;  
 }  
 fprintf\_s(fout, "P%d\n%d %d\n%d\n", format, width, height, a);  
 fwrite(arr, 1, width \* height, fout);  
 fclose(fout);  
}  
  
void ChangeDirection(double &x0, double &y00, double &x, double &y) {  
 dir = abs(y - y00) > abs(x - x0);  
 if(dir) {  
 swap(x, y);  
 swap(y00, x0);  
 }  
 if(x0 > x) {  
 swap(x, x0);  
 swap(y, y00);  
 }  
}  
  
void Bresenhem(double x0, double y00, double x, double y) {  
 ChangeDirection(x0, y00, x, y);  
 double dx = x - x0;  
 double dy = y - y00;  
 double delta = dx == 0 ? 0 : dy / dx;  
 double y\_curr = y00;  
 for (int x\_curr = x0; x\_curr <= x; x\_curr++) {  
 Plot(x\_curr, y\_curr, 1 - (y\_curr - (int) y\_curr));  
 if ((alpha \* 180 / M\_PI != 45) || (fatness <= 1))  
 Plot(x\_curr, y\_curr + 1, y\_curr - (int) y\_curr);  
 y\_curr += delta;  
 }  
}  
  
class Vector {  
public:  
 Vector(long double x\_, long double y\_) {  
 x = x\_;  
 y = y\_;  
 FindL();  
 }  
 void FindL() {  
 l = sqrt(x \* x + y \* y);  
 }  
 double operator\*(const Vector &other) {  
 return this->x \* other.x + this->y \* other.y;  
 }  
 Vector operator-() {  
 Vector c = {-(this->x), -(this->y)};  
 return c;  
 }  
 long double x;  
 long double y;  
 long double l;  
};  
  
bool Inside(double x, double y) {  
 Vector ab = {xb - xa, yb - ya};  
 Vector bd = {xd - xb, yd - yb};  
 Vector dc = {xc - xd, yc - yd};  
 Vector ca = {xa - xc, ya - yc};  
 Vector ao = {x - xa, y - ya}; if(ao.l == 0) return true;  
 Vector bo = {x - xb, y - yb}; if(bo.l == 0) return true;  
 Vector Do = {x - xd, y - yd}; if(Do.l == 0) return true;  
 Vector co = {x - xc, y - yc}; if(co.l == 0) return true;  
 long double oab = acos(ao \* ab / (ao.l \* ab.l)) \* 180 / M\_PI;  
 long double oac = acos(ao \* (-ca) / (ao.l \* ca.l)) \* 180 / M\_PI;  
 long double oba = acos(bo \* (-ab) / (bo.l \* ab.l)) \* 180 / M\_PI;  
 long double obd = acos(bo \* bd / (bo.l \* bd.l)) \* 180 / M\_PI;  
 long double odc = acos(Do \* dc / (Do.l \* dc.l)) \* 180 / M\_PI;  
 long double odb = acos(Do \* (-bd) / (Do.l \* bd.l)) \* 180 / M\_PI;  
 long double ocd = acos(co \* (-dc) / (co.l \* dc.l)) \* 180 / M\_PI;  
 long double oca = acos(co \* ca / (co.l \* ca.l)) \* 180 / M\_PI;  
 long double sum = (180 - oab - oba) + (180 - obd - odb) + (180 - odc - ocd) + (180 - oca - oac);  
 if((y >= (yd - yc) / (xd - xc) \* (x - xc) + yc - 1) && !((xa == xc) || (ya == yc)) && (alpha \* 180 / M\_PI == 45))  
 return false;  
 return (sum < 360.0000001) && (sum > 359.9999999);  
}  
  
void CheckIsRight() {  
 for(int i = 1; i < height - 1; i++)  
 for(int j = 1; j < width - 1; j++)  
 if(buffer[i \* width + j] == -1) {  
 int count = 0;  
 if(buffer[i \* width + j + 1] != -1) count++;  
 if(buffer[i \* width + j - 1] != -1) count++;  
 if(buffer[(i + 1) \* width + j] != -1) count++;  
 if(buffer[(i - 1) \* width + j] != -1) count++;  
 if(count == 4)  
 Plot(j, i, 1);  
 }  
}  
  
void FillRectangle(double x, double y) {  
 if(Inside(x, y))  
 Plot(x, y, 1);  
}  
  
int main(int argc, char\* argv[]) {  
 if((argc < 9) || (argc > 10)) {  
 std::cerr << "Wrong arguments!\n";  
 return 1;  
 }  
 double x0, y00, x, y;  
 strcpy(filename\_in, argv[1]);  
 strcpy(filename\_out, argv[2]);  
 bright = atoi(argv[3]);  
 fatness = atof(argv[4]);  
 x0 = atof(argv[5]);  
 y00 = atof(argv[6]);  
 x = atof(argv[7]);  
 y = atof(argv[8]);  
 if(argc == 10)  
 gamma\_ = atof(argv[9]);  
 FILE\* fin = fopen(filename\_in, "rb");  
 if(fin == NULL) {  
 std::cerr << "Cannot open file!\n";  
 return 1;  
 }  
 if(fscanf(fin, "P%d\n%d %d\n%d\n", &format, &width, &height, &a) != 4) {  
 std::cerr << "Something wrong with input file.\n";  
 return 1;  
 }  
 if(format != 5) {  
 std::cerr << "File has wrong format!\n";  
 return 1;  
 }  
 arr = new uchar[height \* width];  
 buffer = new int[height \* width];  
 for(int i = 0; i < height \* width; i++)  
 buffer[i] = -1;  
 fread(arr, 1, width \* height, fin);  
 if(gamma\_ == -1)  
 bright = sRGB\_reverse(bright);  
 else bright = GammaCorrection(bright, 1);  
 if(fatness > 1) {  
 alpha = x - x0 == 0 ? 0 : (90 \* M\_PI / 180) - abs(atan((y - y00) / (x - x0)));  
 if((y - y00) / (x - x0) > 0) {  
 xa = x0 - (fatness / 2) \* cos(alpha);  
 ya = y00 + (fatness / 2) \* sin(alpha);  
 xb = x0 + (fatness / 2) \* cos(alpha);  
 yb = y00 - (fatness / 2) \* sin(alpha);  
 xc = x - (fatness / 2) \* cos(alpha);  
 yc = y + (fatness / 2) \* sin(alpha);  
 xd = x + (fatness / 2) \* cos(alpha);  
 yd = y - (fatness / 2) \* sin(alpha);  
 if (xa > xc) {  
 swap(xa, xc);  
 swap(ya, yc);  
 swap(xb, xd);  
 swap(yb, yd);  
 }  
 }  
 else {  
 xa = x0 - (fatness / 2) \* cos(alpha);  
 ya = y00 - (fatness / 2) \* sin(alpha);  
 xb = x - (fatness / 2) \* cos(alpha);  
 yb = y - (fatness / 2) \* sin(alpha);  
 xc = x0 + (fatness / 2) \* cos(alpha);  
 yc = y00 + (fatness / 2) \* sin(alpha);  
 xd = x + (fatness / 2) \* cos(alpha);  
 yd = y + (fatness / 2) \* sin(alpha);  
 if(xb < xa) {  
 swap(xa, xb);  
 swap(ya, yb);  
 swap(xc, xd);  
 swap(yc, yd);  
 }  
 }  
 if(alpha \* 180 / M\_PI == 45) {  
 dir = false;  
 for (int i = 0; i < height; i++)  
 for (int j = 0; j < width; j++)  
 FillRectangle(j, i);  
 }  
 if((xa == xc) && (xb == xd) && (y != y00)) {  
 Bresenhem(xa + 1, ya, xb, yb);  
 Bresenhem(xc + 1, yc, xd, yd);  
 }  
 else {  
 Bresenhem(xa, ya, xb, yb);  
 Bresenhem(xc, yc, xd, yd);  
 }  
 if((ya == yb) && (yc == yd) && (x0 != x)) {  
 Bresenhem(xa, ya + 1, xc, yc);  
 Bresenhem(xb, yb + 1, xd, yd);  
 }  
 else {  
 Bresenhem(xa, ya, xc, yc);  
 Bresenhem(xb, yb, xd, yd);  
 }  
 if(alpha \* 180 / M\_PI != 45) {  
 dir = false;  
 for (int i = 0; i < height; i++)  
 for (int j = 0; j < width; j++)  
 FillRectangle(j, i);  
 }  
 }  
 else Bresenhem(x0, y00, x, y);  
 dir = false;  
 CheckIsRight();  
 for(int i = 0; i < height \* width; i++)  
 if(buffer[i] != -1)  
 arr[i] = buffer[i];  
 OutputImage();  
 fclose(fin);  
 delete[] arr;  
 return 0;  
}