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. М3102
Карпов Арсений 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:
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

Уравнение прямой из точки (x_0, y_0) в точку (x_1, y_1) имеет вид:

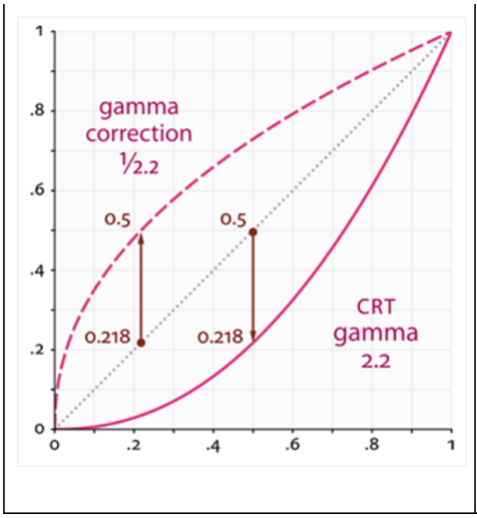
$$y - y_0 = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

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 X, то рассматриваются точки с координатами (x, y) и (x, y + 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.



- 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 (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.

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.

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.

Экспериментальная часть

Язык программирования: C++/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 x и y, 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). Начало системы координат – левый верхний угол.

В функции рисования точки поставим условия выхода, если x или y 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: $\mathbf{R = B + (F - B) \cdot A}$, где B – яркость фонового пикселя, F – яркость накладываемого пикселя, A – непрозрачность накладываемого пикселя, R – результат.

При толщине линии меньше 1 итоговая интенсивность пикселя умножается на толщину линии. Для начала разбиваем толщину линии на составляющие по x и y. После этого в цикле по икс от начальной координаты x0 – толщина линии по x пополам до конечной координаты x1 + толщина линии пополам находим границы, в которых меняется координата y линии. Далее проходим по всем возможным для данной координаты x y-кам и, если пиксель не краний, то его интенсивность – 1, иначе его яркость будет определена пропорционально его расстоянию от ближайшего пикселя по y при данном x.

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 погрешность получается при определении значения интенсивности крайних точек.

Листинг

main.cpp

```
#include <iostream>
#include <algorithm>
#include <cmath>
typedef unsigned char uchar;
double correct(double value, double gamma, int bpp)
{
    value = double(value) / bpp;
    if (value > 1)
        value = 1;
    if (gamma == 0)
```

```
    {
        if (value < 0.04045)
            return  (bpp * value) / 12.92;
        else
            return bpp * (pow((200.0 * value + 11.0) / 211.0, 2.4));
    }
    else {
        return bpp * pow(value, gamma);
    }
}
double to_real(double value, double gamma, int bpp)
{
    value = double(value)/ bpp;
    if (gamma == 0)
    {
        if (value < 0.0031308)
            return  value * 12.92 * bpp;
        else
            return bpp * ((211.0 * pow(value, 0.4166) - 11.0) / 200.0);
    }
    else {
        return bpp * pow(value, 1 / gamma);
    }
}
void print_point(bool tmp, int bpp, double bright, uchar* p2, int width, int height, double x, double y, double
&intens, double gamma, double fat1)
{
    if (!tmp)
    {
        double it = y;
        y = x;
        x = it;
    }
    if ((x >= 0) && (x < width) && (y >= 0) && (y < height)) {
        if (fat1 != 0)
            intens *= fat1;
        int index = abs(y * width + x) + 0.5;
```

```
            *(p2 + index) = correct(*(p2 + index), gamma, bpp);
            *(p2 + index) = to_real((*(p2 + index) * (1 - intens) + bright * intens), gamma, bpp);
            if (*(p2 + index) < 0)
                *(p2 + index) = 0;
            if (*(p2 + index) > bpp)
                *(p2 + index) = bpp;
        }
    }
    void line(uchar* p2, int &width, int &height, double& x0, double& x1, double & y0, double& y1, int bpp,
    double &bright, double &fatness, double gamma)
    {
        bool tmp = true;
        double dx = abs(x1-x0);
        double dy = abs(y1-y0);
        if (dx < dy) {
            std::swap(dx, dy);
            std::swap(x0, y0);
            std::swap(y1, x1);
            tmp = false;
        }
        if (x0 > x1) {
            std::swap(x0, x1);
            std::swap(y0, y1);
        }
        dy = y1 - y0;
        bright = correct(bright, gamma, bpp);
        double fat1 = 0;
        if (fatness < 1)
        {
            fat1 = fatness;
            fatness += 0.5;
        }
        double y_0, y_1, k, k_x, k_y;
        if (dy == 0) {
            for (int x = x0; x <= ceil(x1); x++)
            {
                y_0 = y1 - (fatness - 1) / 2;
```

```
        y_1 = y1 + (fatness - 1) / 2;
        for (int y = int(y_0); y <= ceil(y_1); y++)
        {
            k_y = std::min(1.0, (fatness + 1) / 2 - abs(y - y1));
            k_x = k_y;
            if (int(x1) != x1)
            {
                if (x == int(x1))
                    k_x = x + 1 - x1;
                if (x == ceil(x1))
                    k_x = x1 + 1 - x;
            }
            double intens = (k_x + k_y) / 2;
            print_point(tmp, bpp, bright, p2, width, height, x, y, intens, gamma, fat1);
        }
    }
}
else {
    double max_y = y0;
    double max_x = x0;
    double buf = y0;
    k = (x1-x0)/(y1-y0);
    k_x = abs(fatness*cos(atan(k)));
    k_y = abs(fatness*sin(atan(k)));
    for (int x = int(x0-k_x/2); x<=ceil(x1+k_x/2); x++) {
        double _y = (x-x0)/k+y0;

        y_0 = -k*(x-x0)+y0;
        y_1 = -k*(x-x1)+y1;
        if (y_0>y_1)
            std::swap(y_0, y_1);
        y_0 = std::max(_y-(k_y)*abs(1/k), y_0-1);
        y_1 = std::min(_y+(k_y)*abs(1/k), y_1)+1;
        if (max_y<y_1) {
            max_y = y_1;
            max_x = x;
        }
```

```
        for (int y = int(y_0); y<=y_1; y++) {
            double intens = 1;
            if (y==int(y_0)) {
                intens = 1-y_0+int(y_0);
                if ((intens>=0.85))
                    intens = 0.85;
                if (intens<=0.15)
                    intens = 0;
                if ((max_y==y_1) && (max_x==x))
                    buf = intens;
            }
            else if (y==int(y_1)) {
                intens = y_1-int(y_1);
                if ((intens>=0.85))
                    intens = 0.85;
                if (intens<=0.15)
                    intens = 0;
            }
            print_point(tmp, bpp, bright, p2, width, height, x, y, intens, gamma, fat1);
        }
    }
    print_point(tmp, bpp, bright, p2, width, height, max_x, max_y, buf, gamma, fat1);
  }
}
void P5_line (FILE* fout, uchar* p, int width, int height, int bpp, double x0, double x1, double y0, double
y1, double &bright, double &fatness, double &gamma)
{
  fprintf(fout, "P5\n%d %d\n%d\n", width, height, bpp);
  line(p, width, height, x0, x1, y0, y1, bpp, bright, fatness, gamma);
  fwrite(p, sizeof(uchar), width*height, fout);
}
int main(int argc, char* argv[]) {
  if ((argc != 10) && (argc != 9))
  {
    std::cerr << "Wrong arguments";
    return 1;
  }
```

```
FILE* fin = fopen(argv[1], "rb");
if (fin == NULL) {
    std::cerr << "Can't open input file";
    return 1;
}
int width, height, bpp;
double bright = atoi(argv[3]);
if ((bright < 0) || (bright > 255))
{
    std::cerr << "Incorrect brightness";
    return 1;
}
double fatness = atof(argv[4]);
if (fatness <= 0)
{
    std::cerr << "Incorrect fatness";
}
double x0 = atof(argv[5]);
double y0 = atof(argv[6]);
double x1 = atof(argv[7]);
double y1 = atof(argv[8]);
double gamma = 0;
if (argc == 10)
    gamma = atof(argv[9]);
if (gamma < 0)
{
    std::cerr << "Incorrect gamma";
    return 1;
}
int type;
int i = fscanf(fin, "P %d", &type);
if ((i < 1) || (type != 5))
{
    std::cerr << "Wrong type of the fin";
    fclose(fin);
    return 1;
}
```

```
i = fscanf(fin, "\n%d %d\n%d\n", &width, &height, &bpp);
if (i != 3)
{
    std::cerr << "Wrong type of the fin";
    fclose(fin);
    return 1;
}
if (width <= 0)
{
    std::cerr << "Wrong width";
}
if (height <= 0)
{
    std::cerr << "Wrong height";
}
if ((x0 < 0) || (x0 > width))
{
    std::cerr << "Incorrect x0 coordinate";
    return 1;
}
if ((y0 < 0) || (y0 > height))
{
    std::cerr << "Incorrect y0 coordinate";
    return 1;
}
if ((x1 < 0) || (x1 > width))
{
    std::cerr << "Incorrect x1 coordinate";
    return 1;
}
if ((y1 < 0) || (y1 > height))
{
    std::cerr << "Incorrect y1 coordinate";
    return 1;
}
uchar* p;
p = (uchar*)malloc( width * height * sizeof(uchar));
```

```
    int k = fread(p, sizeof(uchar), width * height, fin);
    if (k < 1)
    {
        std::cerr << "Wrong type of the fin";
        fclose(fin);
        return 1;
    }
    FILE* fout = fopen(argv[2], "wb");
    if (fout == NULL) {
        std::cerr << "Can't open output file";
        fclose(fin);
        return 1;
    }
    P5_line(fout, p, width, height, bpp, x0, x1, y0, y1, bright, fatness, gamma);
    free(p);
    return 0;
}
```