Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное
учреждение высшего образования
Санкт-Петербургский исследовательский университет
Информационных технологий, механики и оптики
Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Компьютерная графика и геометрия

Отчет
по лабораторной работе №2
Изучение алгоритмов отрисовки растровых линий с применением сглаживания и гамма-коррекции

Выполнил: студент гр. М3101
Ершов Михаил Юрьевич
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

Существует несколько алгоритмов отрисовки растровых линий:

1. **Алгоритм Брезенхема**

   Этот алгоритм не применяет сглаживание и может работать только с целочисленным координатами, однако он очень прост в написании и исполнении.

```
function line(int x0, int x1, int y0, int y1)
     int deltax := abs(x1 - x0)
     int deltay := abs(y1 - y0)
     real error := 0
     real deltaerr := (deltay + 1) / (deltax + 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 >= 1.0
             y := y + diry
             error := error - 1.0
```

2. **Алгоритм Ву**

   Этот алгоритм сложнее алгоритма Брезехема, однако он применяет сглаживание и способен работать с нецелыми координатами.

```
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
```

3. Тем не менее, эти алгоритмы изначально не предусматривают отрисовку линий произвольной толщины, поэтому в данной работе был использован другой алгоритм, идея которого заключается в нахождении крайних точек линии и закрашиванию пикселей, находящихся внутри области, ограниченной данными точками, то есть, по сути, закрашиванию точек внутри прямоугольника, которым является линия. Для этого найдём направляющий вектор прямой, проходящей через точки начала и конца, после чего найдём вектор, перпендикулярный вектору прямой, найдём орт полученного вектора и умножим его на половину толщину, теперь, прибавляя данный вектор к точкам начала и конца, можем получить крайние точки прямоугольника, после чего закрасим точки, координаты которых лежат внутри прямоугольника, принадлежность точки прямоугольнику определяется рассмотрением псевдоскалярных произведений векторов, составленных из данной точки и крайних точек прямоугольника. Также алгоритм предусматривает сглаживание линии, для этого точки, находящиеся около прямоугольника, частично закрашиваются, степень закрашенности определяется путем рассмотрения окрестности точки, разбиения этой окрестности на малые части и определения, принадлежит ли данная часть окрестности прямоугольнику, чем таких частей больше, тем сильнее точка закрашена. Тем не менее, для случая толщины 1, когда прямоугольник превращается в простую прямую, этот способ не подходит, поэтому для степени закрашенности рассматривается расстояние от данной точки до прямой, проходящей через начало и конец, чем расстояние меньше, тем сильнее точка закрашена.

Экспериментальная часть

Язык программирования: C++14

Полное решение.

Сначала считывается изображение из файла, после чего с помощью метода drawLine отрисовывается линия с введёнными параметрами с использованием алгоритма, приведённого выше, и учётом гамма-коррекции, затем осуществляется запись изображение в файл, указанный в входных данных.

Выводы

Выполнение данной работы позволило ознакомиться с алгоритмами отрисовки растровых линий и гамма-коррекцией, также в ходе работы был реализован собственный алгоритм отрисовки линий произвольной толщины.

Листинг

main.cpp

```
#include <iostream>
#include <string>
#include <cstdlib>
#include "pgm.h"

using namespace std;

int main(int argc, char* argv[]) {
    if (argc < 9 || argc > 10) {
        cerr << "Invalid number of arguments\n";
        return 1;
    }
    try {
        PGM image(argv[1]);
        if (argc == 9) {
            image.drawLine({ stod(argv[5]), stod(argv[6]) },
                { stod(argv[7]), stod(argv[8]) },
                atoi(argv[3]),
                stod(argv[4]));
        }
        else {
            image.drawLine({ stod(argv[5]), stod(argv[6]) },
                { stod(argv[7]), stod(argv[8]) },
                atoi(argv[3]),
                stod(argv[4]),
                stod(argv[9]));
        }
        image.print(argv[2]);
    }
    catch (const exception& e) {
        cerr << e.what();
        return 1;
    }
    return 0;
}
```

pgm.h

```
#ifndef LAB2_PGM_H
#define LAB2_PGM_H

#include <cstdio>
#include <cstring>
#include <cmath>
#include <stdexcept>

typedef unsigned char uchar;

struct Point {
    double x, y;
    Point(double x, double y) {
        this->x = x;
        this->y = y;
    }
    Point() = default;
    double length() const {
        return std::sqrt(x * x + y * y);
    }
    Point operator-(const Point& point) const {
        return { this->x - point.x, this->y - point.y };
    }

    Point operator+(const Point& point) const {
        return { this->x + point.x, this->y + point.y };
    }

    Point operator*(const double& k) const {
        return { this->x * k, this->y * k };
    }

    double operator^(const Point& point) const {
        return x * point.y - y * point.x;
    }

    double operator*(const Point& point) const {
        return x * point.x + y * point.y;
    }
};

class PGM {
private:
    char header[2];
    int width, height;
    uchar maxValue;
    uchar* data;
    void plot(Point point, double intensity, int brightness, double gamma = 0);
public:
    explicit PGM(char* fileName);
    ~PGM();
    void print(char* fileName);
    void drawLine(Point begin, Point end, int brightness, double thickness, double gamma
= 0);
};

#endif //LAB2_PGM_H
```

pgm.cpp

```
#include "pgm.h"
#include <algorithm>

PGM::PGM(char* fileName) {
    FILE* fin = fopen(fileName, "rb");
    if (fin == nullptr) {
        throw std::runtime_error("File can't be opened\n");
    }
    int tmp;
    if (fscanf(fin, "%s%d%d%d", header, &width, &height, &tmp) < 4) {
        throw std::runtime_error("Invalid header\n");
    }
    if (strcmp(header, "P5") != 0) {
        throw std::runtime_error("File is not a PGM image\n");
    }
    if (tmp > 255) {
        throw std::runtime_error("Unsupported colours\n");
    }
    maxValue = (uchar)tmp;
    if (width <= 0 || height <= 0) {
        throw std::runtime_error("Invalid header\n");
    }
    if (fgetc(fin) == EOF) {
        throw std::runtime_error("Invalid header\n");
    }
    data = new uchar[width * height];
    fread(data, 1, width * height, fin);
    if (fclose(fin) != 0) {
        throw std::runtime_error("File can't be closed\n");
    }
}

PGM::~PGM() {
    delete[] data;
}
void PGM::print(char* fileName) {
    FILE* fout = fopen(fileName, "wb");
    if (fout == nullptr) {
        throw std::runtime_error("File can't be opened or created\n");
    }
    fprintf(fout, "%s\n%i %i\n%i\n", header, width, height, maxValue);
    fwrite(data, 1, width * height, fout);
    fclose(fout);
    uchar _max = 0;
    for (int i = 0; i < width * height; i++) {
        _max = std::max(_max, data[i]);
    }
}

void PGM::plot(Point point, double intensity, int brightness, double gamma) {
    if (point.x < 0 || point.x > width || point.y < 0 || point.y > height || brightness <
0) {
        return;
    }
    int index = int(std::round(point.y)) * width + int(std::round(point.x));
    double currentBrightness = (double)data[index] / 255;
    if (gamma == 0) {
        currentBrightness = currentBrightness <= 0.04045 ?
            currentBrightness / 12.92 :
            std::pow((currentBrightness + 0.055) / 1.055, 2.4);
    }
    else {
        currentBrightness = std::pow(currentBrightness, gamma);
```

```
    }
    currentBrightness *= (1.0 - intensity);
    double relativeBrightness = (double)brightness / 255.0;
    if (gamma == 0) {
        double decodedBrightness = relativeBrightness <= 0.04045 ?
            relativeBrightness / 12.92 :
            std::pow((relativeBrightness + 0.055) / 1.055, 2.4);
        currentBrightness += intensity * decodedBrightness;
        currentBrightness = currentBrightness <= 0.0031308 ?
            currentBrightness * 12.92 :
            std::pow(currentBrightness, 1.0 / 2.4) * 1.055 - 0.055;
    }
    else {
        double decodedBrightness = pow(relativeBrightness, gamma);
        currentBrightness += intensity * decodedBrightness;
        currentBrightness = std::pow(currentBrightness, 1.0 / gamma);
    }
    if (1.0 - currentBrightness < 1e-5) {
        currentBrightness = 1.0;
    }
    data[index] = 255 * currentBrightness;
}

bool insideRectangle(const Point& x, const Point& a, const Point& b, const Point& c,
const Point& d) {
    Point ax = a - x;
    Point bx = b - x;
    Point cx = c - x;
    Point dx = d - x;
    double res[4] = { ax ^ bx, bx ^ cx, cx ^ dx, dx ^ ax };
    for (int i = 0; i < 4; i++) {
        for (int j = i + 1; j < 4; j++) {
            if (res[i] * res[j] < 0) {
                return false;
            }
        }
    }
    return true;
}

double calculateIntensity(const Point& x, const Point& a, const Point& b, const Point& c,
const Point& d, bool incline) {
    if (insideRectangle(x, a, b, c, d)) {
        return 1.0;
    }
    if (!incline) {
        return 0.0;
    }
    int insideRect = 0;
    int total = 0;
    for (double i = -0.5; i <= 0.5; i += 0.5) {
        for (double j = -0.5; j <= 0.5; j += 0.5) {
            if (insideRectangle(Point(x.x + i, x.y + j), a, b, c, d)) {
                insideRect++;
            }
            total++;
        }
    }
    return (double)insideRect / total;
}

double calculateIntensity(const Point& x, const double& A, const double& B, const double&
C) {
    double d = std::abs(A * x.x + B * x.y + C) / std::sqrt(A * A + B * B);
    if (d < 1e-2) {
```

```
        d = 0.0;
    }
    if (d > 0.9) {
        d = 1.0;
    }
    return 1.0 - d;
}

int min(double a, double b, double c, double d) {
    return round(std::min(std::min(a, b), std::min(c, d)));
}

int max(double a, double b, double c, double d) {
    return round(std::max(std::max(a, b), std::max(c, d)));
}

void PGM::drawLine(Point begin, Point end, int brightness, double thickness, double
gamma) {
    if (begin.x < 0 || begin.x >= width || begin.y < 0 || begin.y >= height) {
        throw std::runtime_error("First point is out of bounds\n");
    }
    if (end.x < 0 || end.x >= width || end.y < 0 || end.y >= height) {
        throw std::runtime_error("Second point is out of bounds\n");
    }
    if (brightness < 0 || brightness > 255) {
        throw std::runtime_error("Invalid brightness\n");
    }
    if (thickness <= 0) {
        throw std::runtime_error("Invalid thickness\n");
    }
    Point bot1, bot2, top1, top2;
    Point vector;
    vector = { begin.y - end.y, end.x - begin.x };
    bool incline = (vector.x != 0 && vector.y != 0);
    vector = { vector.x / vector.length(), vector.y / vector.length() };
    double c = thickness / 2.0;
    if (c < 1e-5) {
        c = 0.0;
    }
    Point k = vector * c;
    bot1 = begin - k;
    bot2 = end - k;
    top1 = begin + k;
    top2 = end + k;
    for (int i = std::max(0, min(bot1.x, bot2.x, top1.x, top2.x)); i <= std::min(width -
1, max(bot1.x, bot2.x, top1.x, top2.x)); i++) {
        for (int j = std::max(0, min(bot1.y, bot2.y, top1.y, top2.y)); j <=
std::min(height, max(bot1.y, bot2.y, top1.y, top2.y)); j++) {
            Point x(i, j);
            if (!incline || thickness > 1.0) {
                plot(x, calculateIntensity(x, bot1, top1, top2, bot2, incline),
brightness, gamma);
            }
            else {
                plot(x, calculateIntensity(x, begin.y - end.y, end.x - begin.x, begin ^
end), brightness, gamma);
            }
        }
    }
}
```