МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа № 2

По дисциплине «Компьютерная графика и геометрия»

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

Выполнил студент группы М3101

Кузьмук Павел Юрьевич

Проверил:

Скаков Павел Сергеевич

САНКТ-ПЕТЕРБУРГ

2020

ЦЕЛЬ РАБОТЫ

Изучить алгоритмы растрирования векторов на существующем изображении и реализовать программу, рисующую линию на изображении в формате PGM (P5) с учетом гамма-коррекции.

ОПИСАНИЕ РАБОТЫ

Полное решение лабораторной работы.

Программа должна быть написана на C/C++ и не использовать внешние библиотеки.

Аргументы передаются через командную строку:

lab2 <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.

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

Рисование линий

Уравнение прямой линии:

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

Существуют несколько видов алгоритмов:

1) Со сглаживанием
2) Без сглаживания

1. 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)
```

2. 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
```

3. Алгоритм, придуманный «на коленке», который удовлетворяет требованиям ЛР и дает лучшие результаты(и будет использован):

Абстрагируемся от изображения и координат пикселей, таким образом линия — просто прямоугольник какой-то толщины на плоскости, найдем координаты 4 его вершин. Для этого сначала найдем «медиану» - линия с бесконечно малой толщиной, проходящая через середины боковых сторон прямоугольника, потом через ее концы проведем перпендикуляры длиной thickness / 2, получим очертания линии, которую собираемся рисовать.

Пройдем по всем пикселям изображения, для каждого из них узнаем, какой процент от его площади покрыт прямоугольником. С такой яркостью поверх него мы и наложим цвет линии, которую рисуем. Для высчитывания площади пересечения двух прямоугольников(пикселя и линии) можно использовать геометрию, однако несложно с точки зрения процессорного времени просто разделить пиксель на 100 частей и проверить, сколько из них лежит в прямоугольнике — это и будет искомым процентом.

ВЫВОД

Выполнение данной лабораторной работы позволило узнать об алгоритмах рисования прямых линий со сглаживанием. В ходе данной лабораторной работы был реализован собственный алгоритм растрирования линий произвольной толщины.

Листинг кода

Использован язык C++ 14

main.cpp:

```
#include <iostream>

#include <string>
#include "pgm_image.h"

using namespace std;

int main(int argc, char *argv[]) {
if(argc != 9 && argc != 10) {
cerr << "command line arguments are invalid" << endl;
return 1;
}
string fin = string(argv[1]);
string fout = string(argv[2]);
double thikness, x_s, y_s, x_f, y_f, gamma = 2.4;
bool srgb = true;
int brightness;
try {
brightness = atoi(argv[3]);
thikness = stod(argv[4]);
x_s = stod(argv[5]);
y_s = stod(argv[6]);
x_f = stod(argv[7]);
y_f = stod(argv[8]);
}
catch (const exception& e) {
cerr << e.what() << endl;
return 1;
}
if(argc == 10) {
try {
gamma = stod(argv[9]);
srgb = false;
}
catch (const exception& e) {
cerr << e.what() << endl;
return 1;
}
}

PGM_Image* image;
try {
```

```
image = new PGM_Image(fin);
}
catch (const exception& e) {
cerr << e.what() << endl;
return 1;
}

image -> draw_line(Point{x_s, y_s}, Point{x_f, y_f}, thikness,
brightness, gamma, srgb);

try {
image -> drop(fout);
delete(image);
}
catch (const exception& e) {
cerr << e.what() << endl;
return 1;
}
}
```

pgm_image.h:

```
#pragma once

#include <vector>
#include <algorithm>
#include <stdexcept>
#include <fstream>

using namespace std;

class Point{
public:
double x, y;
Point(double, double);
};

class PGM_Image{
private:
int width, height, color_depth;
vector<vector<unsigned char>> image;
void plot(int, int, double, int, double, bool);
public:
PGM_Image(string);
void drop(string);
void draw_line(Point, Point, double, int, double, bool);
};
```

pgm_image.cpp:

```
#include "pgm_image.h"

#include <cmath>

double sqr(double a) {
return a * a;
}

double dist(double x1, double y1, double x2, double y2) {
return sqrt(sqr(x1 - x2) + sqr(y1 - y2));
}

double find_y(int x, double gradient, Point begin) {
return begin.y + gradient * (x - begin.x);
}

double triangle_size(Point p1, Point p2, Point p3) {
double a = sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y
- p2.y));
double b = sqrt((p1.x - p3.x) * (p1.x - p3.x) + (p1.y - p3.y) * (p1.y
- p3.y));
double c = sqrt((p3.x - p2.x) * (p3.x - p2.x) + (p3.y - p2.y) * (p3.y
- p2.y));
double p = (a + b + c) / 2;
return sqrt(p * (p-a) * (p-b) * (p-c));
}

bool point_in_rect(Point p1, Point p2, Point p3, Point p4, Point pp)
{
double h = sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y
- p2.y));
double w = sqrt((p1.x - p3.x) * (p1.x - p3.x) + (p1.y - p3.y) * (p1.y
- p3.y));
double rect_size = h * w;
double tr_size = triangle_size(pp, p1, p2) + triangle_size(pp, p1,
p3) + triangle_size(pp, p3, p4) + triangle_size(pp, p4, p2);
return fabs(rect_size - tr_size) < 1e-5;
}

double intersection_size(Point p1, Point p2, Point p3, Point p4, int
px, int py) {
if(
point_in_rect(p1, p2, p3, p4, Point{px*1.0, py*1.0}) &&
point_in_rect(p1, p2, p3, p4, Point{px+1.0, py*1.0}) &&
point_in_rect(p1, p2, p3, p4, Point{px*1.0, py+1.0}) &&
```

```
point_in_rect(p1, p2, p3, p4, Point{px+1.0, py+1.0})
) return 1.0;
int ins = 0, totp = 0;
for(double i = px + 0.1; i + 0.1 <= px + 1; i += 0.1)
for(double j = py + 0.1; j + 0.1 <= py + 1; j += 0.1)
{
totp ++;
if(point_in_rect(p1, p2, p3, p4, Point{i, j}))
ins++;
}
return ins * 1.0 / totp;
}

Point::Point(double xx, double yy) {
this -> x = xx;
this -> y = yy;
}

PGM_Image::PGM_Image(string filename) {
ifstream fin(filename, ios::binary);
if(!fin.is_open())
throw runtime_error("failed to open file");

char cc[2];
fin >> cc[0] >> cc[1];
if(cc[0] != 'P' || cc[1] != '5')
throw runtime_error("expected P5 format");
fin >> width >> height >> color_depth;
image.assign(height, vector<unsigned char>(width));
char pixel;
fin.read(&pixel, 1);
for(int i = 0; i < height; i ++)
for(int j = 0; j < width; j ++)
{
fin.read(&pixel, sizeof(unsigned char));
image[i][j] = pixel;
}
fin.close();
}

void PGM_Image::drop(string filename) {
ofstream fout(filename, ios::binary);
if(!fout.is_open()) {
throw runtime_error("cannot open output file");
}
fout << "P5\n" << width << ' ' << height << '\n' << color_depth << '\
n';
```

```
        for(int i = 0; i < height; i ++)
        for(int j = 0; j < width; j ++)
        fout << (unsigned char)(image[i][j]);
        fout.flush();
        fout.close();
        }

        void PGM_Image::plot(int x, int y, double brightness, int color,
double gamma, bool srgb) {
        if(x < 0 || x >= width || y < 0 || y >= height || brightness < 0) {
        return;
        }
        double old = double((unsigned char)image[y][x]) / color_depth;
        if(srgb)
        old = (old < 0.04045 ? old / 12.92 : pow((old + 0.055) / 1.055,
gamma));
        else
        old = pow(old, gamma);
        old *= (1.0 - brightness);
        double corrected_color = color * 1.0 / 255;
        if(srgb)
        corrected_color = (corrected_color < 0.04045 ? corrected_color /
12.92 : pow((corrected_color + 0.055) / 1.055, gamma));
        else
        corrected_color = pow(corrected_color, gamma);
        old += brightness * corrected_color;
        if(srgb)
        old = (old <= 0.0031308 ? old * 12.92 : pow(old, 1.0/gamma)*1.055 -
0.055);
        else
        old = pow(old, 1.0 / gamma);
        if(old >= 0.9999) old = 1.0;
        image[y][x] = color_depth * old;
        }

        void PGM_Image::draw_line(Point begin, Point end, double thikness,
int color, double gamma, bool srgb) {
        if(begin.x == end.x) {
        begin.x += 0.5;
        end.x += 0.5;
        }
        if(begin.y == end.y) {
        begin.y += 0.5;
        end.y += 0.5;
        }
        Point line_vector = {end.x - begin.x, end.y - begin.y};
        Point th_vector = {1.0, 0.0};
```

```
        if(line_vector.x != 0)
        th_vector = {-line_vector.y / line_vector.x, 1.0};
        double    thikness_coef    =    sqrt((thikness*thikness    /    4)    /
(th_vector.x*th_vector.x + th_vector.y*th_vector.y));
        Point   p1   =   {begin.x   +   thikness_coef*th_vector.x,   begin.y   +
thikness_coef*th_vector.y};
        Point   p2   =   {begin.x   -   thikness_coef*th_vector.x,   begin.y   -
thikness_coef*th_vector.y};
        Point   p3   =   {end.x   +   thikness_coef*th_vector.x,   end.y   +
thikness_coef*th_vector.y};
        Point   p4   =   {end.x   -   thikness_coef*th_vector.x,   end.y   -
thikness_coef*th_vector.y};
        for(int i = max(0, int(min(min(p1.y, p2.y), min(p3.y, p4.y))) - 3); i
< min(height, int(max(max(p1.y, p2.y), max(p3.y, p4.y))) + 3); i ++)
        for(int j = max(0, int(min(min(p1.x, p2.x), min(p3.x, p4.x))) - 3); j
< min(width, int(max(max(p1.x, p2.x), max(p3.x, p4.x))) + 3); j ++) {
        plot(j, i, intersection_size(p1, p2, p3, p4, j, i), color, gamma,
srgb);
        }
        }
```