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

Отчет по лабораторной работе № 2

По дисциплине «Компьютерная графика и геометрия»

Выполнил студент группы №M3102  
*Ларин Владислав Денисович*

Проверил:  
*Скаков Павел Сергеевич*

САНКТ-ПЕТЕРБУРГ  
2019

***Цель работы*:** изучить алгоритмы и реализовать программу, рисующую линию на изображении в формате PGM (P5) с учетом гамма-коррекции sRGB.

***Описание*:**

Программа должна поддерживать серые (PNM P5) изображения и рисовать прямую линию на исходном фоне с учетом входных параметров. Аргументы передаются через командную строку:

*lab#.exe <имя\_входного\_файла> <имя\_выходного\_файла> <яркость\_линии> <толщина\_линии> <x\_начальный> <y\_начальный> <x\_конечный> <y\_конечный> <гамма>*

***Теоретическая часть*:**

Описание файла pnm:

Первые два байта файла кодируют сочетание символов P5, указывающих на формат файла (оттенки серого). После них идет перевод строки. Четвертый, пятый, шестой, восьмой, девятый и десятый байты кодируют ширину и высоту соответственно (в виде трехзначного числа). Далее снова следует переход на новую строку. В третьей строке содержатся три байта, кодирующих число максимально доступного значения пикселя данного файла. Далее следует массив чисел, кодирующих саму картинку.

На фоне этой картинки рисуется прямая линия с заданными яркостью, начальными и конечными координатами, толщиной и значением гаммы. Идея гамма-коррекции заключается в том, чтобы применить инверсию гаммы монитора к окончательному цвету перед записью в файл. При наложении прямой на исходный фон используется альфа-смешение – процесс комбинирования двух объектов на экране с учетом альфа-каналов. Расчёт яркости результирующего пикселя после наложения двух пикселей друг на друга выполняется по формуле: R = B ⋅ (1 − A) + F ⋅ A, где B – яркость фонового пикселя, F – яркость накладываемого пикселя, A – непрозрачность накладываемого пикселя.

***Экспериментальная часть:***

Программа написана на языке С++ (стандарт С++17).

Данные считываются с помощью функции fread в массив pixel, память для которого выделяется с помощью функции malloc. На фоне исходной картинки мы рисуем прямую линию с заданными яркостью, начальными и конечными координатами, толщиной и значением гаммы. Расчёт яркости результирующего пикселя после наложения пикселя фона и пикселя прямой друг на друга выполняется в отдельной функции по формуле R = B ⋅ (1 − A) + F ⋅ A. Гамма-коррекция sRGB и закрашивание каждого пикселя также вынесены в отдельные функции. Для наклонных прямых алгоритм отрисовки линии проходит её вдоль основной оси, подбирая координаты по неосновной оси. В зависимости от величины ошибки, которая показывает, как далеко ушли пиксели от идеальной линии по неосновной оси, распределяется интенсивность между точками. Чем больше удалена точка от идеальной линии, тем меньше её интенсивность. Такое распределение придаёт линии одинаковую интенсивность на всём её протяжении и устраняет ступенчатость. Вычисление ошибки и проход по всем пикселям прямой реализованы в отдельных функциях. В конце программы (или в случае ошибки во входных данных) память, отведенная под массивы и выделенная с помощью функции malloc, очищается с помощью функции free, а открытые файлы закрываются.

***Вывод:***

Программа производит прорисовку прямой линии с заданными яркостью, начальными и конечными координатами, толщиной и значением гаммы. Программа использует гамма-коррекцию sRGB и альфа-смешение для наглядности выводимого изображения, а также не допускает ступенчатость или неравномерную интенсивность прямой.

В случае некорректно введенных входных данных или ошибки при выделении памяти, программа выводит соответствующий комментарий и завершает свою работу, предварительно закрыв открытые файлы и очистив выделенную память. Все полученные результаты совпадают с теоретическими выкладками.

***Листинг:***

main.cpp  
  
  
#include **<iostream>**#include **<vector>**#include **<math.h>**#include **<algorithm>  
  
typedef unsigned char** uchar;  
  
**using namespace** std;  
  
**const double** EPS = 0.00001;  
  
  
**double** dist(**double** x1, **double** y1, **double** x2, **double** y2)  
{  
 **double** dx = x1 - x2;  
 **double** dy = y1 - y2;  
 **return** sqrt(dx \* dx + dy \* dy);  
}  
  
  
**double** minDist(**double** xBegin, **double** yBegin, **double** xEnd, **double** yEnd, **double** xFind, **double** yFind, **double** thick)  
{  
 **double** toBegin = dist(xBegin, yBegin, xFind, yFind),  
 toEnd = dist(xEnd, yEnd, xFind, yFind),  
 length = dist(xBegin, yBegin, xEnd, yEnd ),  
   
 dx = (xEnd - xBegin) / length,  
 dy = (yEnd - yBegin) / length,  
   
 xNewBegin = xBegin - dx,  
 yNewBegin = yBegin - dy,  
 xNewEnd = xEnd + dx,  
 yNewEnd = yEnd + dy,  
   
 newToBegin = dist(xNewBegin, yNewBegin, xFind, yFind ),  
 newToEnd = dist(xNewEnd, yNewEnd, xFind, yFind ),  
 newLength = dist(xNewBegin, yNewBegin, xNewEnd, yNewEnd),  
   
 yDiff = yEnd - yBegin,  
 xDiff = xEnd - xBegin,  
 allDiff = yBegin \* xEnd - yEnd \* xBegin,  
 genDist = fabs(yDiff \* xFind - xDiff \* yFind + allDiff) / sqrt(yDiff \* yDiff + xDiff \* xDiff);  
   
 **if** (fabs(toBegin \* toBegin - toEnd \* toEnd) > length \* length)  
 {  
 **if** (fabs(newToBegin \* newToBegin - newToEnd \* newToEnd) > newLength \* newLength || genDist > thick + 1.0 - EPS)  
 {  
 genDist = (thick + 1) \* 2;  
 }  
 **else** {  
 **double** xProj, yProj;  
 **double** yForProj = genDist \* -dx;  
 **double** xForProj = genDist \* dy;  
 **if** (yDiff \* (xFind + xForProj) - xDiff \* (yFind + yForProj) + allDiff < EPS)  
 {  
 xProj = xFind + xForProj;  
 yProj = yFind + yForProj;  
 }  
 **else** {  
 xProj = xFind - xForProj;  
 yProj = yFind - yForProj;  
 }  
 genDist = min(dist(xProj, yProj, xBegin, yBegin), dist(xProj, yProj, xEnd, yEnd)) + thick - EPS;  
 }  
 }  
 **return** genDist;  
}  
  
  
**double** pointBright(**double** dist, **double** thick)  
{  
 **if** (dist >= thick + 1.0)  
 {  
 **return** 0.0;  
 }  
 **if** (dist <= thick)  
 {  
 **return** 1.0;  
 }  
 **return** 1.0 - dist + thick;  
}  
  
  
**double** applyReverseGamma(**double** bright, **double** gamma)  
{  
 bright /= 255;  
 **double** newBright = 0;  
  
 **if**(gamma != 2)  
 {  
 **if** (bright <= 0.04045)  
 {  
 newBright = bright / 12.92;  
 }  
 **else** {  
 newBright = pow((bright + 0.055) / 1.055, 2.4);  
 }  
 }  
 **else** {  
 newBright = pow(bright, gamma);  
 }  
 **return** newBright \* 255;  
}  
  
  
**double** applyGamma(**double** bright, **double** gamma)  
{  
 bright /= 255;  
 **double** newBright = 0;  
  
 **if** (gamma != 2)  
 {  
 **if** (bright <= 0.0031308)  
 {  
 newBright = 12.92 \* bright;  
 }  
 **else** {  
 newBright = pow(1.055 \* bright, 0.4167) - 0.055;  
 }  
 }  
 **else** {  
 newBright = pow(bright, 1 / gamma);  
 }  
 **return** newBright \* 255;  
}  
  
  
**void** drawPixel(vector<uchar> &pixel, **int** x, **int** y, **int** w, **int** h, **int** brightness, **double** bright, **double** gamma)  
{  
 **double** newBrightness = applyReverseGamma(brightness, gamma);  
 **double** background = applyReverseGamma(pixel[y \* w + x], gamma);  
  
 **double** newPixel = background \* (1 - bright) + newBrightness \* bright;  
 pixel[y \* w + x] = applyGamma(newPixel, gamma);  
}  
  
  
**void** drawLine(vector<uchar> &pixel, **double** xBegin, **double** yBegin, **double** xEnd, **double** yEnd, **int** w, **int** h, **int** bright, **double** thick, **double** gamma)  
{  
 **if** (xBegin > xEnd)  
 {  
 swap(xBegin, xEnd);  
 swap(yBegin, yEnd);  
 }  
  
 **double** thickCor = thick \* 2 + 2.0 - EPS;  
 **double** pointBrightness;  
  
 **if** (xBegin == xEnd || yBegin == yEnd)  
 {  
 **for** (**int** x = max(0, **int**(xBegin - thickCor)); x <= min(h - 1, **int**(xEnd + thickCor)); x++)  
 {  
 **for** (**int** y = max(0, **int**(yBegin - thickCor)); y <= min(w - 1, **int**(yEnd + thickCor)); y++)  
 {  
 pointBrightness = pointBright(minDist(xBegin, yBegin, xEnd, yEnd, x, y, thick), thick);  
 drawPixel(pixel, x, y, w, h, bright, pointBrightness, gamma);  
 }  
 }  
 }  
 **else** {  
 **double** grad = (yEnd - yBegin) / (xEnd - xBegin);  
 **double** yCurr = yBegin;  
 **for** (**int** x = max(0, **int**(xBegin - thickCor)); x <= min(h - 1, **int**(xEnd + thickCor)); x++)  
 {  
 **for** (**int** y = max(0, **int**(yCurr - fabs(grad) - thickCor)); y <= min(w - 1, **int**(yCurr + fabs(grad) + thickCor)); y++)  
 {  
 pointBrightness = pointBright(minDist(xBegin, yBegin, xEnd, yEnd, x, y, thick), thick);  
 drawPixel(pixel, y, x, w, h, bright, pointBrightness, gamma);  
 }  
 yCurr += (x >= xBegin && x <= xEnd ? grad : 0);  
 }  
 }  
}  
  
  
  
**int** main(**int** argc, **char** \*argv[])  
{  
 **if** (argc != 10 && argc != 9)  
 {  
 cerr << **"Invalid request"**;  
 **return** 1;  
 }  
  
 **char** \*fileName\_in = argv[1];  
 **char** \*fileName\_out = argv[2];  
 **int** bright = atoi(argv[3]);  
 **double** thick = (atof(argv[4]) - 1) / 2.0,  
 xBegin = atof(argv[5]),  
 yBegin = atof(argv[6]),  
 xEnd = atof(argv[7]),  
 yEnd = atof(argv[8]),  
 gamma;  
  
 **if** (argc == 10)  
 {  
 gamma = atof(argv[9]);  
 }  
 **else** {  
 gamma = 2;  
 }  
   
 swap(xBegin, yBegin);  
 swap(xEnd, yEnd );  
  
 FILE \*f\_in = fopen(fileName\_in, **"rb"**);  
 **if** (f\_in == NULL)  
 {  
 cerr << **"Invalid input file"**;  
 **return** 1;  
 }  
  
 **int** type, w, h, dummy;  
 **int** countFileParameters = fscanf(f\_in, **"P%i\n%i %i\n%i\n"**, &type, &w, &h, &dummy);  
 **if** (countFileParameters != 4)  
 {  
 cerr << **"Incorrect file content"**;  
 fclose(f\_in);  
 **return** 1;  
 }  
  
 **if** ((type != 5) || (w <= 0 || h <= 0) || (dummy != 255))  
 {  
 cerr << **"Incorrect parameters"**;  
 fclose(f\_in);  
 **return** 1;  
 }  
  
 **int** startPosition = ftell(f\_in);  
 fseek(f\_in, 0, SEEK\_END);  
 **int** countPixels = ftell(f\_in) - startPosition;  
 fseek(f\_in, startPosition, 0);  
 **if** (countPixels != w \* h)  
 {  
 cerr << **"Not enough data"**;  
 fclose(f\_in);  
 **return** 1;  
 }  
  
 vector<uchar> pixel(w \* h, 0);  
 fread(&pixel[0], **sizeof**(uchar), pixel.size(), f\_in);  
 fclose(f\_in);  
  
 drawLine(pixel, xBegin, yBegin, xEnd, yEnd, w, h, bright, thick, gamma);  
  
 FILE \*f\_out = fopen(fileName\_out, **"wb"**);  
 **if** (f\_out == NULL)  
 {  
 cerr << **"Invalid output file"**;  
 **return** 1;  
 }  
  
 fprintf(f\_out, **"P%i\n%i %i\n%i\n"**, type, w, h, dummy);  
 fwrite(&pixel[0], **sizeof**(uchar), pixel.size(), f\_out);  
 fclose(f\_out);  
 **return** 0;  
}