МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное 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 №5**

**По дисциплине «Компьютерная геометрия и графика»**

**Изучение алгоритма настройки автояркости изображения**

**Выполнил студент группы №M3234:**

***Ота Никита Терентьевич***

**Преподаватель:**

***Скаков Павел Сергеевич***

**САНКТ-ПЕТЕРБУРГ**

**2020**

**Цель работы:**

Реализовать программу, которая позволяет проводить настройку автояркости изображения в различных цветовых пространствах.

**Описание работы:**

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

Аргументы передаются через командную строку:

lab5.exe <имя\_входного\_файла> <имя\_выходного\_файла> <преобразование> [<смещение> <множитель>],

где

* <преобразование>:

0 - применить указанные значения <смещение> и <множитель> в пространстве RGB к каждому каналу;

1 - применить указанные значения <смещение> и <множитель> в пространстве YCbCr.601 к каналу Y;

2 - автояркость в пространстве RGB: <смещение> и <множитель> вычисляются на основе минимального и максимального значений пикселей;

3 - аналогично 2 в пространстве YcbCr.601;

4 - автояркость в пространстве RGB: <смещение> и <множитель> вычисляются на основе минимального и максимального значений пикселей, после игнорирования 0.39% самых светлых и тёмных пикселей;

5 - аналогично 4 в пространстве YCbCr.601.

* <смещение> - целое число, только для преобразований 0 и 1 в диапазоне [-255..255];
* <множитель> - дробное положительное число, только для преобразований 0 и 1 в диапазоне [1/255..255].

Значение пикселя X изменяется по формуле: (X-<смещение>)\*<множитель>.

YCbCr.601 в PC диапазоне: [0, 255].

Входные/выходные данные: PNM P5 или P6 (RGB).

Если программе передано значение, которое не поддерживается – следует сообщить об ошибке.

Коды возврата:

0 - ошибок нет

1 - произошла ошибка

В поток вывода (printf, cout) выводится только следующая информация: для преобразований 2-5 найденные значения <смещение> и <множитель> в формате: "<смещение> <множитель>".

Сообщения об ошибках выводятся в поток вывода ошибок:

* С: fprintf(stderr, "Error\n");
* C++: std::cerr

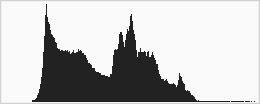
**Теоретическая часть:**

Нередко можно наблюдать изображения с плохой контрастностью: тёмные участки изображения недостаточно тёмные и/или светлые недостаточно светлые.

Эту проблему можно хорошо продемонстрировать, если построить распределение яркостей всех точек изображения – гистограмму.



Картика



Её гистограмма

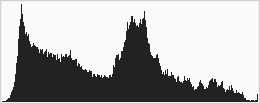
Для улучшения контрастности гистограмму нужно растянуть на весь диапазон значений: минимальное значение пикселя должно стать 0 в новом изображении, а максимальное – 255.

Преобразование яркости каждого пикселя можно описать простой формулой:

y = (x - min)\*255/(max — min)



Картинка после коррекции яркости



Её гистограмма

При нахождении минимального и максимального значений пикселей имеет смысл игнорировать небольшой процент самых тёмных и светлых пикселей, что обычно соответствует шуму.

На примере гистограммы видно, что в качестве минимума здесь можно взять абсолютный минимум, а для максимума имеет смысл взять указанное стрелкой значение, игнорируя существующие, но малочисленные более светлые пиксели.

Корректировать контрастность можно как работая в пространстве RGB, одинаково изменяя все каналы (а не отдельно каждый), так и в других цветовых пространствах.

Например, широко используется корректировка контрастности в пространстве YCbCr. Здесь изменяются значения только канала Y, соответствующего яркости изображения, а каналы Cb и Cr остаются неизменными. Как правило, это даёт более контрастные, но менее насыщенные изображения, чем автоконтрастность в пространстве RGB.

**Практическая часть:**

Язык, на котором написана программа, решающая поставленную задачу: С++.

Ключи компиляции: -std=c++11 -fsanitize=address -fno-stack-limit.

Компилятор: g++.

**План работы программы:**

1. Чтение данных из командной строки, обработка неправильных введённых данных.
2. Чтение данных с введённого пользователем файла входа с обработкой ошибок.
3. Применение корректирования яркости изображения в RGB или в YСbCr.601, в зависимости от введённых значений
4. Вывод результата в файл с обработкой ошибок.

**Код программ:**

**main.cpp**

#include <iostream>

#include <string>

#include <fstream>

#include "brightness\_changer.h"

int main(int argc, char\* argv[]) {

if (argc != 4 && argc != 6) {

std::cerr << "Got " << argc << " arguments\n";

std::cerr << "Usage: <input file> <output file> <change>\n if change is 0 or 1 then you should put also <shift> <factor>\n";

return EXIT\_FAILURE;

}

int change;

try {

change = std::stoi(argv[3]);

} catch (std::invalid\_argument const& e) {

std::cerr << "<change> must be an int value\n";

std::cerr << e.what() << '\n';

return EXIT\_FAILURE;

} catch (std::out\_of\_range const& e) {

std::cerr << "<change> must be in [0..5]\n";

std::cerr << e.what() << '\n';

return EXIT\_FAILURE;

}

int shift;

double factor;

switch (change) {

case 0:

case 1:

if (argc == 4) {

std::cerr << "Not ample arguments: sought <shift> <factor>, but couldn't find them\n";

return EXIT\_FAILURE;

}

try {

shift = std::stoi(argv[4]);

} catch (std::invalid\_argument const& e) {

std::cerr << "<shift> must be an int value\n";

std::cerr << e.what() << '\n';

return EXIT\_FAILURE;

} catch (std::out\_of\_range const& e) {

std::cerr << "<shift> must be in [-255..255]\n";

std::cerr << e.what() << '\n';

return EXIT\_FAILURE;

}

if (shift < -255 || shift > 255) {

std::cerr << "<shift> must be in [-255..255]\n";

return EXIT\_FAILURE;

}

try {

factor = std::stod(argv[5]);

} catch (std::invalid\_argument const& e) {

std::cerr << "<factor> must be a positive number\n";

std::cerr << e.what() << '\n';

}

if (factor < 1.0 / 255 || factor > 255) {

std::cerr << "<factor> must be in [1/255..255]\n";

return EXIT\_FAILURE;

}

break;

case 2:

case 3:

case 4:

case 5:

if (argc == 6) {

std::cerr << "Redundant arguments: " << argv[4] << ' ' << argv[5] << " are not needed\n";

return EXIT\_FAILURE;

}

break;

default:

std::cerr << "<change> must be in [0..5]\n";

return EXIT\_FAILURE;

}

std::ifstream in(argv[1], std::ios::binary);

if (!in.is\_open()) {

std::cerr << "Couldn't open " << argv[1] << '\n';

return EXIT\_FAILURE;

}

char p;

int type;

in >> p >> type;

if (p != 'P' || type < 5 || type > 6) {

std::cerr << "Couldn't read first line in the file or got unextected encoding, different from P5 or P6 in " << argv[1] << '\n';

in.close();

return EXIT\_FAILURE;

}

int width, height;

in >> width >> height;

if (width <= 0 || height <= 0) {

std::cerr << "Couldn't get width an height info or they are not positive in " << argv[1] << '\n';

in.close();

return EXIT\_FAILURE;

}

int brightness;

in >> brightness;

char line\_separator = in.get();

if (brightness != MAX\_BRIGHTNESS || line\_separator != '\n') {

std::cerr << "Couldn't get the highest pixel brightness or it is not equals " << MAX\_BRIGHTNESS << " in " << argv[1] << '\n';

in.close();

return EXIT\_FAILURE;

}

int colors = (type == 5 ? MONOCHROME\_COLORS : RGB\_COLORS);

int \*\*\*data = new int\*\*[height];

for (int h = 0; h < height; ++h) {

data[h] = new int\*[width];

for (int w = 0; w < width; ++w) {

data[h][w] = new int[colors];

}

}

for (int h = 0; h < height; ++h) {

for (int w = 0; w < width; ++w) {

for (int color = 0; color < colors; ++color) {

data[h][w][color] = in.get();

if (data[h][w][color] == -1 || in.gcount() == 0 || in.eof()) {

std::cerr << "Unexpected data size: not ample bytes in " << argv[1] << '\n';

in.close();

clear\_array(data, height, width);

return EXIT\_FAILURE;

}

}

}

}

if (!(in.get() == -1) || !in.eof()) {

std::cerr << "Unexpected data size: redundant bytes in " << argv[1] << '\n';

in.close();

clear\_array(data, height, width);

return EXIT\_FAILURE;

}

picture pic(height, width, brightness, colors, data);

try {

pic.switch\_brightness(change, shift, factor);

} catch (std::runtime\_error const& e) {

std::cerr << e.what() << '\n';

return EXIT\_FAILURE;

}

std::ofstream out(argv[2], std::ios::binary);

if (!out.is\_open()) {

std::cerr << "Couldn't open " << argv[2] << '\n';

return EXIT\_FAILURE;

}

out << p << type << line\_separator;

out << width << ' ' << height << line\_separator;

out << brightness << line\_separator;

for (int h = 0; h < height; ++h) {

for (int w = 0; w < width; ++w) {

for (int color = 0; color < colors; ++color) {

if (!out || out.bad()) {

std::cerr << "An error occurred while writing data\n";

out.close();

// array data will be cleared in picture destructor

return EXIT\_FAILURE;

}

out << static\_cast<unsigned char>(data[h][w][color]);

}

}

}

return EXIT\_SUCCESS;

}

**brightness\_changer.h**

#include <functional>

#include <utility>

#include <cstdlib>

int const MAX\_BRIGHTNESS = 255;

int const MONOCHROME\_COLORS = 1;

int const RGB\_COLORS = 3;

double const ONE\_SECOND = 1.0 / 2;

double const K\_R\_601 = 0.299;

double const K\_G\_601 = 0.587;

double const K\_B\_601 = 0.114;

double const PERCENTAGE = 0.39 / 100;

#define manual\_args int const shift, double const factor

typedef std::function<void(int const, double const)> brightness\_changer\_function\_t;

void clear\_array(int \*\*\*arr, int n, int m);

struct picture {

picture() = default;

picture(int const h, int const w, int const m, int const c, int\*\*\* d);

~picture();

int get\_width() const;

int get\_height() const;

int get\_max\_brightness() const;

int get\_data();

void switch\_brightness(int const change, int const shift, double const factor);

private:

int width = 0;

int height = 0;

int max\_brightness = MAX\_BRIGHTNESS;

int colors = 0;

int \*\*\*data = nullptr;

void auto\_changing(brightness\_changer\_function\_t const& manual, double const min\_value, double const max\_value);

void auto\_changing\_with\_skip(brightness\_changer\_function\_t const& manual);

void manual\_RGB(int shift, double factor);

void manual\_YCbCr(int const shift, double const factor);

void auto\_RGB();

void auto\_YCbCr();

void auto\_with\_skip\_RGB();

void auto\_with\_skip\_YCbCr();

};

**brightness\_changer.cpp**

#include <math.h>

#include <stdexcept>

#include <algorithm>

#include <vector>

#include <iostream>

#include "brightness\_changer.h"

void clear\_array(int \*\*\*arr, int n, int m) {

if (arr == nullptr) {

return;

}

for (int i = 0; i < n; ++i) {

if (arr[i] != nullptr) {

for (int j = 0; j < m; ++j) {

if (arr[i][j] != nullptr) {

delete[] arr[i][j];

arr[i][j] = nullptr;

}

}

delete[] arr[i];

arr[i] = nullptr;

}

}

delete[] arr;

arr = nullptr;

}

picture::picture(int const h, int const w, int const m, int const c, int \*\*\*data)

: height(h), width(w), max\_brightness(m), colors(c), data(data) {}

picture::~picture() {

clear\_array(data, height, width);

}

inline int correct\_bounds(double const val) {

return static\_cast<int>(fmin(MAX\_BRIGHTNESS, fmax(floor(val), 0)));

}

int change\_brightness(double const val, double const shift, double const factor) {

return correct\_bounds((val - shift) \* factor);

}

void picture::manual\_RGB(int shift, double factor) {

for (int h = 0; h < height; ++h) {

for (int w = 0; w < width; ++w) {

for (int color = 0; color < colors; ++color) {

data[h][w][color] = change\_brightness(data[h][w][color], shift, factor);

}

}

}

}

void RGB\_to\_YCbCr(double \*pixel, double const K\_R, double const K\_G, double const K\_B) {

double R = pixel[0] / MAX\_BRIGHTNESS;

double G = pixel[1] / MAX\_BRIGHTNESS;

double B = pixel[2] / MAX\_BRIGHTNESS;

double Y = K\_R \* R + K\_G \* G + K\_B \* B;

double Cb = ONE\_SECOND \* (B - Y) / (1 - K\_B);

double Cr = ONE\_SECOND \* (R - Y) / (1 - K\_R);

pixel[0] = MAX\_BRIGHTNESS \* (Y);

pixel[1] = MAX\_BRIGHTNESS \* (Cb + ONE\_SECOND);

pixel[2] = MAX\_BRIGHTNESS \* (Cr + ONE\_SECOND);

}

void YCbCr\_to\_RGB(double \*pixel, double const K\_R, double const K\_G, double const K\_B) {

double Y = pixel[0] / MAX\_BRIGHTNESS;

double Cb = pixel[1] / MAX\_BRIGHTNESS - ONE\_SECOND;

double Cr = pixel[2] / MAX\_BRIGHTNESS - ONE\_SECOND;

pixel[0] = fmin(MAX\_BRIGHTNESS, fmax(MAX\_BRIGHTNESS \* (Y + Cr \* (2 - 2 \* K\_R)), 0));

pixel[1] = fmin(MAX\_BRIGHTNESS, fmax(MAX\_BRIGHTNESS \* (Y - (K\_B / K\_G) \* (2 - 2 \* K\_B) \* Cb - (K\_R / K\_G) \* (2 - 2 \* K\_R) \* Cr), 0));

pixel[2] = fmin(MAX\_BRIGHTNESS, fmax(MAX\_BRIGHTNESS \* (Y + (2 - 2 \* K\_B) \* Cb), 0));

}

// delete[] is necassary

double\* obtain\_pixel(int \*pixel, int const colors) {

double \*res = new double[colors];

for (int color = 0; color < colors; ++color) {

res[color] = static\_cast<double>(pixel[color]);

}

return res;

}

void picture::manual\_YCbCr(int const shift, double const factor) {

// idk what to do if P5

for (int h = 0; h < height; ++h) {

for (int w = 0; w < width; ++w) {

double \*pixel = obtain\_pixel(data[h][w], colors);

RGB\_to\_YCbCr(pixel, K\_R\_601, K\_G\_601, K\_B\_601);

pixel[0] = change\_brightness(pixel[0], shift, factor);

YCbCr\_to\_RGB(pixel, K\_R\_601, K\_G\_601, K\_B\_601);

for (int color = 0; color < colors; ++color) {

data[h][w][color] = correct\_bounds(pixel[color]);

}

delete[] pixel;

}

}

}

void picture::auto\_RGB() {

int max\_value = 0, min\_value = MAX\_BRIGHTNESS;

for (int h = 0; h < height; ++h) {

for (int w = 0; w < width; ++w) {

for (int color = 0; color < colors; ++color) {

max\_value = std::max(max\_value, data[h][w][color]);

min\_value = std::min(min\_value, data[h][w][color]);

}

}

}

// auto\_changing([this] (manual\_args) { manual\_RGB(shift, factor); }, min\_value, max\_value);

int const shift = min\_value;

double const factor = MAX\_BRIGHTNESS / static\_cast<double>(max\_value - min\_value);

std::cout << shift << ' ' << factor << '\n';

manual\_RGB(shift, factor);

}

void picture::auto\_YCbCr() {

double min\_value = MAX\_BRIGHTNESS, max\_value = 0;

for (int h = 0; h < height; ++h) {

for (int w = 0; w < width; ++w) {

double \*pixel = obtain\_pixel(data[h][w], colors);

RGB\_to\_YCbCr(pixel, K\_R\_601, K\_G\_601, K\_B\_601);

min\_value = fmin(min\_value, pixel[0]);

max\_value = fmax(max\_value, pixel[0]);

delete[] pixel;

}

}

// auto\_changing([this] (manual\_args) { manual\_YCbCr(shift, factor); }, min\_value, max\_value);

double const shift = min\_value;

double const factor = MAX\_BRIGHTNESS / (max\_value - min\_value);

std::cout << shift << ' ' << factor << '\n';

manual\_YCbCr(shift, factor);

}

void picture::auto\_with\_skip\_RGB() {

std::vector<int> brightnesses;

for (int h = 0; h < height; ++h) {

for (int w = 0; w < width; ++w) {

for (int color = 0; color < colors; ++color) {

brightnesses.push\_back(data[h][w][color]);

}

}

}

int pos = static\_cast<int>(PERCENTAGE \* height \* width \* colors);

std::nth\_element(brightnesses.begin(), brightnesses.begin() + pos, brightnesses.end());

int min\_value = brightnesses[pos];

std::nth\_element(brightnesses.begin(), brightnesses.begin() + pos, brightnesses.end(), std::greater<int>());

int max\_value = brightnesses[pos];

// auto\_changing\_with\_skip([this] (manual\_args) { manual\_RGB(shift, factor); });

int const shift = min\_value;

double const factor = MAX\_BRIGHTNESS / static\_cast<double>(max\_value - min\_value);

std::cout << shift << ' ' << factor << '\n';

manual\_RGB(shift, factor);

}

void picture::auto\_with\_skip\_YCbCr() {

std::vector<double> brightnesses;

for (int h = 0; h < height; ++h) {

for (int w = 0; w < width; ++w) {

double \*pixel = obtain\_pixel(data[h][w], colors);

RGB\_to\_YCbCr(pixel, K\_R\_601, K\_G\_601, K\_B\_601);

brightnesses.push\_back(pixel[0]);

delete[] pixel;

}

}

double pos = static\_cast<int>(PERCENTAGE \* height \* width \* colors);

std::nth\_element(brightnesses.begin(), brightnesses.begin() + pos, brightnesses.end());

double min\_value = brightnesses[pos];

std::nth\_element(brightnesses.begin(), brightnesses.begin() + pos, brightnesses.end(), std::greater<double>());

int max\_value = brightnesses[pos];

// auto\_changing\_with\_skip([this] (manual\_args) { manual\_YCbCr(shift, factor); });

double const shift = min\_value;

double const factor = MAX\_BRIGHTNESS / (max\_value - min\_value);

std::cout << shift << ' ' << factor << '\n';

manual\_YCbCr(shift, factor);

}

void picture::switch\_brightness(int const change, int const shift, double const factor) {

if (change % 2 == 1 && colors == MONOCHROME\_COLORS) {

throw std::runtime\_error("P5 RGB to YCbCr conversion is not supported yet");

}

std::cout.precision(5) << '\n';

std::cout << std::fixed;

switch (change) {

case 0:

manual\_RGB(shift, factor);

break;

case 1:

manual\_YCbCr(shift, factor);

break;

case 2:

auto\_RGB();

break;

case 3:

auto\_YCbCr();

break;

case 4:

auto\_with\_skip\_RGB();

break;

case 5:

auto\_with\_skip\_YCbCr();

break;

}

}