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 №M3101:

Пантелеев Ярослав Кириллович

Преподаватель:

Скаков Павел Сергеевич

САНКТ-ПЕТЕРБУРГ

2020

Цель работы: реализовать программу, которая позволяет проводить настройку автояркости изображения в различных цветовых пространствах.

Описание:

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).

Частичное решение: только преобразования 0-3 + корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок.

Полное решение: все остальное.

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:

C: fprintf(stderr, "Error\n");

C++: std::cerr

Теоретическая часть:

Коррекция яркости и контрастности изображений

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.

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:

$$y = (x - min)*255/(max - min)$$

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.

Экспериментальная часть

Язык выполнения работы: C++17, компилятор Microsoft Visual C++.

Этапы работы программы:

1) Чтение файла картинки (хедера и информации о цвете каждого пикселя) с обработкой ошибок чтения
2) Выполнение заданной аргументом командной строки коррекции яркости изображения, используя полученные из аргументов командной строки смещение и множитель (в случае 0-ого и 1-ого типа коррекции) или автоматически определяя смещение и множитель (в случае 2-5ых типов коррекции) в цветовом пространстве RGB или YCbCr.601.
3) Запись полученного изображения в выходной файл

Вывод:

Выполнение данной лабораторной работы позволило узнать о способах корректировки яркости изображения и об алгоритмах этой корректировки. Была реализована программа, демонстрирующая эти алгоритмы: «ручная» корректировка яркости заданием смещения и множителя через аргументы командной строки и автоматическая корректировка, определяющая смещение и множитель исходя из максимальной и минимальной яркостей пикселей данного изображения.

Листинг исходного кода:

main.cpp

```
#include <iostream>
#include <string>
#include "Image.h"

int main(int argc, char* argv[]) {
    double offset = -1;
    double coefficient = -1;
    if (argc < 4) {
        std::cerr << "Invalid command line arguments!" << "\n";
        return 1;
    }
    std::string input = std::string(argv[1]);
    std::string output = std::string(argv[2]);
    int type = atoi(argv[3]);
    if (type < 0 || type > 5) {
        std::cerr << "Invalid type of operation!" << "\n";
        return 1;
    }
    if (type < 2) {
        if (argc == 6) {
            offset = atof(argv[4]);
            coefficient = atof(argv[5]);
        }
        else {
            std::cerr << "Invalid command line arguments!" << "\n";
            return 1;
        }
    }

    Image* img;
    try {
        img = new Image(input);
    }
    catch(const std::exception& error) {
        std::cerr << error.what() << '\n';
        return 1;
    }

    img->correct_brightness(type, offset, coefficient);
    try {
        img->write(output);
    }
    catch(const std::exception& error) {
        std::cerr << error.what() << '\n';
        delete img;
        return 1;
    }

    delete img;
    return 0;
}
```

Image.h

```
#pragma once
#include <vector>
#include <fstream>

class Pixel {
public:
    unsigned char a, b, c;
    Pixel(unsigned char, unsigned char, unsigned char);
    Pixel() = default;
```

```
    void RGB_to_YCbCr_601();
    void YCbCr_601_to_RGB();
};

class Image {
public:
    Image(const std::string&);
    void write(std::string);
    void correct_brightness(int, int, double);
private:
    int width;
    int height;
    int color_depth;
    char format_0;
    char format_1;
    std::vector <std::vector <Pixel>> image;
};
```

Image.cpp

```
#include "Image.h"
#include <algorithm>
#include <stdexcept>
#include <cmath>
#include <string>
#include <vector>
#include <fstream>
#include <iostream>

unsigned char correct_color(unsigned char color, int offset, double coefficient) {
    int result = (static_cast<double>(color) - offset) * coefficient;
    if (result > 255)
        result = 255;
    if (result < 0)
        result = 0;
    return result;
}

Pixel::Pixel(unsigned char a, unsigned char b, unsigned char c) : a(a), b(b), c(c) {}

void Pixel::RGB_to_YCbCr_601() {
    // R G B
    const double r = a / 255.0;
    const double g = b / 255.0;
    const double b = c / 255.0;

    // Coefficients for R G B
    const double coef_r = 0.299;
    const double coef_g = 0.587;
    const double coef_b = 0.114;

    // Y Cb Cr
    const double Y = coef_r * r + coef_g * g + coef_b * b;
    const double Cb = (b - Y) / (2.0 * (1.0 - coef_b));
    const double Cr = (r - Y) / (2.0 * (1.0 - coef_r));

    // Output
    this->a = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(Y * 255)))));
    this->b = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round((Cb + 0.5) * 255)))));
    this->c = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round((Cr + 0.5) * 255)))));
}

void Pixel::YCbCr_601_to_RGB() {
```

```
44.     // Y Cb Cr
45.     const double Y = a / 255.0;
46.     const double Cb = b / 255.0 - 0.5;
47.     const double Cr = c / 255.0 - 0.5;
48.
49.     // Coefficients for R G B
50.     const double coef_r = 0.299;
51.     const double coef_g = 0.587;
52.     const double coef_b = 0.114;
53.
54.     // R G B
55.     const double r = Y + (2.0 - 2.0 * coef_r) * Cr;
56.     const double g = Y - coef_b * (2.0 - 2.0 * coef_b) * Cb / coef_g - coef_r * (2 - 2.0 * coef
_r) * Cr / coef_g;
57.     const double b = Y + (2.0 - 2.0 * coef_b) * Cb;
58.
59.     // Output
60.     this->a = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(r * 255)))));
61.     this->b = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(g * 255)))));
62.     this->c = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(b * 255)))));
63. }
64.
65. Image::Image(const std::string& filename) {
66.     std::ifstream fin(filename, std::ios::binary);
67.     if (!fin.is_open())
68.         throw std::runtime_error("failed to open file");
69.
70.     fin >> this->format_0 >> this->format_1;
71.     if (this->format_0 != 'P' || (this->format_1 != '5' && this->format_1 != '6')) {
72.         throw std::runtime_error("expected P5 or P6 format");
73.     }
74.     fin >> this->width >> this->height >> this->color_depth;
75.     if(this->color_depth != 255)
76.         throw std::runtime_error("only 255 color depth is supported");
77.     this->image.assign(this->height, std::vector <Pixel>(this->width));
78.
79.     char color;
80.     fin.read(&color, 1);
81.     for (int i = 0; i < this->height; i++) {
82.         for (int j = 0; j < this->width; j++) {
83.             this->image[i][j] = Pixel();
84.
85.             fin.read(&color, sizeof(unsigned char));
86.             this->image[i][j].a = color;
87.             this->image[i][j].b = color;
88.             this->image[i][j].c = color;
89.             if (this->format_1 == '6') {
90.                 fin.read(&color, sizeof(unsigned char));
91.                 this->image[i][j].b = color;
92.
93.                 fin.read(&color, sizeof(unsigned char));
94.                 this->image[i][j].c = color;
95.             }
96.         }
97.     }
98.     fin.close();
99. }
100.
101. void Image::write(std::string filename) {
102.     std::ofstream fout(filename, std::ios::binary);
103.     if(!fout.is_open()) {
104.         throw std::runtime_error("cannot open output file");
105.     }
106.
```

```
    fout << format_0 << format_1 << "\n" << width << '
' << height << '\n' << color_depth << '\n';
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            fout << image[i][j].a;
            if (format_1 == '6') {
                fout << image[i][j].b;
                fout << image[i][j].c;
            }
        }
    }
    fout.flush();
    fout.close();
}

void Image::correct_brightness(int type, int offset, double coefficient) {
    if (type % 2 == 1) {
        // Ïåðåâîäèì â YCbCr601
        for (int i = 0; i < this->height; i++) {
            for (int j = 0; j < this->width; j++) {
                this->image[i][j].RGB_to_YCbCr_601();
            }
        }
    }
    if (type == 0) {
        if (format_1)
        {
            for (int i = 0; i < height; i++) {
                for (int j = 0; j < width; j++) {
                    image[i][j].a = correct_color(image[i][j].a, offset, coefficient);
                    image[i][j].b = correct_color(image[i][j].b, offset, coefficient);
                    image[i][j].c = correct_color(image[i][j].c, offset, coefficient);
                }
            }
        }
        else
        {
            for (int i = 0; i < height; i++) {
                for (int j = 0; j < width; j++) {
                    int result = image[i][j].a = correct_color(image[i][j].a, offset,
coefficient);
                    image[i][j].a = result;
                    image[i][j].b = result;
                    image[i][j].c = result;
                }
            }
        }
    }
    if (type == 1) {
        if (format_1) {
            for (int i = 0; i < height; i++) {
                for (int j = 0; j < width; j++) {
                    image[i][j].a = correct_color(image[i][j].a, offset, coefficient);
                }
            }
        }
        else {
            for (int i = 0; i < height; i++) {
                for (int j = 0; j < width; j++) {
                    int result = correct_color(image[i][j].a, offset, coefficient);
                    image[i][j].a = result;
                    image[i][j].b = result;
                    image[i][j].c = result;
                }
            }
        }
    }
    if (type == 2) {
```

```
173.         unsigned char min = 255;
174.         unsigned char max = 0;
175.         if (format_1) {
176.             for (int i = 0; i < this->height; i++) {
177.                 for (int j = 0; j < this->width; j++) {
178.                     min = std::min(min, image[i][j].a);
179.                     min = std::min(min, image[i][j].b);
180.                     min = std::min(min, image[i][j].c);
181.
182.                     max = std::max(max, image[i][j].a);
183.                     max = std::max(max, image[i][j].b);
184.                     max = std::max(max, image[i][j].c);
185.                 }
186.             }
187.         }
188.         else {
189.             for (int i = 0; i < this->height; i++) {
190.                 for (int j = 0; j < this->width; j++) {
191.                     min = std::min(min, image[i][j].a);
192.                     max = std::max(max, image[i][j].a);
193.                 }
194.             }
195.         }
196.         offset = static_cast<int>(min);
197.         coefficient = 255.0 / (static_cast<int>(max) - static_cast<int>(min));
198.         std::cout << offset << " " << coefficient << "\n";
199.         for (int i = 0; i < this->height; i++) {
200.             for (int j = 0; j < this->width; j++) {
201.                 this->image[i][j].a = correct_color(this->image[i][j].a, offset, coefficient);
202.                 this->image[i][j].b = correct_color(this->image[i][j].b, offset, coefficient);
203.                 this->image[i][j].c = correct_color(this->image[i][j].c, offset, coefficient);
204.             }
205.         }
206.     }
207.     if (type == 3) {
208.         unsigned char min = 255;
209.         unsigned char max = 0;
210.         for (int i = 0; i < this->height; i++) {
211.             for (int j = 0; j < this->width; j++) {
212.                 min = std::min(min, image[i][j].a);
213.                 max = std::max(max, image[i][j].a);
214.             }
215.         }
216.         offset = static_cast<int>(min);
217.         coefficient = 255.0 / (static_cast<int>(max) - static_cast<int>(min));
218.         std::cout << offset << " " << coefficient << "\n";
219.         if (this->format_1 == '6') {
220.             for (int i = 0; i < this->height; i++) {
221.                 for (int j = 0; j < this->width; j++) {
222.                     this->image[i][j].a = correct_color(this->image[i][j].a, offset,
coefficient);
223.                 }
224.             }
225.         }
226.         else {
227.             for (int i = 0; i < this->height; i++) {
228.                 for (int j = 0; j < this->width; j++) {
229.                     this->image[i][j].a = correct_color(this->image[i][j].a, offset,
coefficient);
230.                     this->image[i][j].b = correct_color(this->image[i][j].b, offset,
coefficient);
231.                     this->image[i][j].c = correct_color(this->image[i][j].c, offset,
coefficient);;
232.                 }
233.             }
234.         }
235.     }
236.     if (type == 4) {
```

```
237.         std::vector <unsigned long long int> color_count(256, 0);
238.         auto ignoring_count = static_cast<long long int>(0.03901 * this->width * this->height);
239.         for (int i = 0; i < this->height; i++) {
240.             for (int j = 0; j < this->width; j++) {
241.                 color_count[this->image[i][j].a]++;
242.                 color_count[this->image[i][j].b]++;
243.                 color_count[this->image[i][j].c]++;
244.             }
245.         }
246.         int max_ignored_brightness = 255;
247.         int min_ignored_brightness = 0;
248.         while (ignoring_count > 0) {
249.             ignoring_count -= color_count[max_ignored_brightness];
250.             if (ignoring_count >= 0) {
251.                 max_ignored_brightness--;
252.             }
253.         }
254.         ignoring_count = static_cast<long long int>(0.03901 * this->width * this->height);
255.         while (ignoring_count > 0) {
256.             ignoring_count -= color_count[min_ignored_brightness];
257.             if (ignoring_count >= 0) {
258.                 min_ignored_brightness++;
259.             }
260.         }
261.
262.         offset = static_cast<int>(min_ignored_brightness);
263.         coefficient = 255.0 / (static_cast<int>(max_ignored_brightness) - static_cast<int>(min_
ignored_brightness));
264.         std::cout << offset << " " << coefficient << "\n";
265.         for (int i = 0; i < this->height; i++) {
266.             for (int j = 0; j < this->width; j++) {
267.                 this->image[i][j].a = correct_color(this->image[i][j].a, offset, coefficient);
268.                 this->image[i][j].b = correct_color(this->image[i][j].b, offset, coefficient);
269.                 this->image[i][j].c = correct_color(this->image[i][j].c, offset, coefficient);
270.             }
271.         }
272.     }
273.     if (type == 5) {
274.         std::vector <long long int> color_count(256, 0);
275.         auto ignoring_count = static_cast<long long int>(0.03901 * this->width * this->height);
276.         for (int i = 0; i < this->height; i++) {
277.             for (int j = 0; j < this->width; j++) {
278.                 color_count[this->image[i][j].a]++;
279.             }
280.         }
281.         int max_ignored_brightness = 255;
282.         int min_ignored_brightness = 0;
283.         while (ignoring_count > 0) {
284.             ignoring_count -= color_count[max_ignored_brightness];
285.             if (ignoring_count >= 0) {
286.                 max_ignored_brightness--;
287.             }
288.         }
289.         ignoring_count = static_cast<long long int>(0.03901 * this->width * this->height);
290.         while (ignoring_count > 0) {
291.             ignoring_count -= color_count[min_ignored_brightness];
292.             if (ignoring_count >= 0) {
293.                 min_ignored_brightness++;
294.             }
295.         }
296.
297.         offset = static_cast<int>(min_ignored_brightness);
298.         coefficient = 255.0 / (static_cast<int>(max_ignored_brightness) - static_cast<int>(min_
ignored_brightness));
299.         std::cout << offset << " " << coefficient << "\n";
300.         if (this->format_1 == '6') {
301.             for (int i = 0; i < this->height; i++) {
302.                 for (int j = 0; j < this->width; j++) {
```

```
303.                     this->image[i][j].a = correct_color(this->image[i][j].a, offset,
coefficient);
304.                 }
305.             }
306.         }
307.         else {
308.             for (int i = 0; i < this->height; i++) {
309.                 for (int j = 0; j < this->width; j++) {
310.                     this->image[i][j].a = correct_color(this->image[i][j].a, offset,
coefficient);
311.                     this->image[i][j].b = correct_color(this->image[i][j].b, offset,
coefficient);
312.                     this->image[i][j].c = correct_color(this->image[i][j].c, offset,
coefficient);
313.                 }
314.             }
315.         }
316.     }
317.     if (type % 2 == 1) {
318.         for (int i = 0; i < this->height; i++) {
319.             for (int j = 0; j < this->width; j++) {
320.                 this->image[i][j].YCbCr_601_to_RGB();
321.             }
322.         }
323.     }
324. }
```