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[]) {
    // Îáðàáîòêà àðãóìåíòîâ êîìàíäíîé ñòðîêè è ïðîâåðêà èõ êîððåêòíîñòè
    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;
        }
    }

    // Ðàáîòà ñ îáúåêòîì êëàññà Image
    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;
```

```
8.      Pixel(unsigned char, unsigned char, unsigned char);
9.      Pixel() = default;
10.     void RGB_to_YCbCr_601();
11.     void YCbCr_601_to_RGB();
12. };
13.
14. class Image {
15. public:
16.     Image(const std::string&);
17.     void write(std::string);
18.     void correct_brightness(int, int, double);
19. private:
20.     int width;
21.     int height;
22.     int color_depth;
23.     char format_0;
24.     char format_1;
25.     std::vector <std::vector <Pixel>> image;
26. };
```

Image.cpp

```
1.  #include "Image.h"
2.  #include <algorithm>
3.  #include <stdexcept>
4.  #include <cmath>
5.  #include <string>
6.  #include <vector>
7.  #include <fstream>
8.  #include <iostream>
9.
10. unsigned char correct_color(unsigned char color, int offset, double coefficient) {
11.     int result = (static_cast<double>(color) - offset) * coefficient;
12.     if (result > 255)
13.         result = 255;
14.     if (result < 0)
15.         result = 0;
16.     return result;
17. }
18.
19. // Êîíſòðóêòîð
20. Pixel::Pixel(unsigned char a, unsigned char b, unsigned char c) : a(a), b(b), c(c) {}
21.
22. void Pixel::RGB_to_YCbCr_601() {
23.     // R G B
24.     const double r = a / 255.0;
25.     const double g = b / 255.0;
26.     const double b = c / 255.0;
27.
28.     // Coefficients for R G B
29.     const double coef_r = 0.299;
30.     const double coef_g = 0.587;
31.     const double coef_b = 0.114;
32.
33.     // Y Cb Cr
34.     const double Y = coef_r * r + coef_g * g + coef_b * b;
35.     const double Cb = (b - Y) / (2.0 * (1.0 - coef_b));
36.     const double Cr = (r - Y) / (2.0 * (1.0 - coef_r));
37.
38.     // Output
39.     this->a = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(Y * 255)))));
40.     this->b = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round((Cb + 0.5) * 255)))));
41.     this->c = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round((Cr + 0.5) * 255)))));
```

```
42. }
43.
44. void Pixel::YCbCr_601_to_RGB() {
45.     // Y Cb Cr
46.     const double Y = a / 255.0;
47.     const double Cb = b / 255.0 - 0.5;
48.     const double Cr = c / 255.0 - 0.5;
49.
50.     // Coefficients for R G B
51.     const double coef_r = 0.299;
52.     const double coef_g = 0.587;
53.     const double coef_b = 0.114;
54.
55.     // R G B
56.     const double r = Y + (2.0 - 2.0 * coef_r) * Cr;
57.     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;
58.     const double b = Y + (2.0 - 2.0 * coef_b) * Cb;
59.
60.     // Output
61.     this->a = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(r * 255)))));
62.     this->b = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(g * 255)))));
63.     this->c = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(b * 255)))));
64. }
65.
66.
67. // Êîíſòðóêòîð
68. Image::Image(const std::string& filename) {
69.     // Îòêðûâàåì ôàéë
70.     std::ifstream fin(filename, std::ios::binary);
71.     if (!fin.is_open())
72.         throw std::runtime_error("failed to open file");
73.
74.     // ×èòàåì õåäåð
75.     fin >> this->format_0 >> this->format_1;
76.     if (this->format_0 != 'P' || (this->format_1 != '5' && this->format_1 != '6')) {
77.         throw std::runtime_error("expected P5 or P6 format");
78.     }
79.     fin >> this->width >> this->height >> this->color_depth;
80.     if(this->color_depth != 255)
81.         throw std::runtime_error("only 255 color depth is supported");
82.     this->image.assign(this->height, std::vector <Pixel>(this->width));
83.
84.     // ×èòàåì ïèêñåëè
85.     char color;
86.     fin.read(&color, 1);
87.     for (int i = 0; i < this->height; i++) {
88.         for (int j = 0; j < this->width; j++) {
89.             this->image[i][j] = Pixel();
90.
91.             fin.read(&color, sizeof(unsigned char));
92.             this->image[i][j].a = color;
93.             this->image[i][j].b = color;
94.             this->image[i][j].c = color;
95.             if (this->format_1 == '6') {
96.                 fin.read(&color, sizeof(unsigned char));
97.                 this->image[i][j].b = color;
98.
99.                 fin.read(&color, sizeof(unsigned char));
100.                this->image[i][j].c = color;
101.            }
102.        }
103.    }
104.    fin.close();
105. }
```

```
106.
107. // Çàïèñü êàðòèíêè â ôàéë(-û)
108. void Image::write(std::string filename) {
109.     std::ofstream fout(filename, std::ios::binary);
110.     if(!fout.is_open()) {
111.         throw std::runtime_error("cannot open output file");
112.     }
113.
114.     fout << format_0 << format_1 << "\n" << width << '
' << height << '\n' << color_depth << '\n';
115.     for (int i = 0; i < height; i++) {
116.         for (int j = 0; j < width; j++) {
117.             fout << image[i][j].a;
118.             if (format_1 == '6') {
119.                 fout << image[i][j].b;
120.                 fout << image[i][j].c;
121.             }
122.         }
123.     }
124.     fout.flush();
125.     fout.close();
126. }
127.
128. void Image::correct_brightness(int type, int offset, double coefficient) {
129.     if (type % 2 == 1) {
130.         // Ïåðåâîäèì â YCbCr601
131.         for (int i = 0; i < this->height; i++) {
132.             for (int j = 0; j < this->width; j++) {
133.                 this->image[i][j].RGB_to_YCbCr_601();
134.             }
135.         }
136.     }
137.     if (type == 0) {
138.         if (format_1)
139.         {
140.             for (int i = 0; i < height; i++) {
141.                 for (int j = 0; j < width; j++) {
142.                     image[i][j].a = correct_color(image[i][j].a, offset, coefficient);
143.                     image[i][j].b = correct_color(image[i][j].b, offset, coefficient);
144.                     image[i][j].c = correct_color(image[i][j].c, offset, coefficient);
145.                 }
146.             }
147.         }
148.         else
149.         {
150.             for (int i = 0; i < height; i++) {
151.                 for (int j = 0; j < width; j++) {
152.                     int result = image[i][j].a = correct_color(image[i][j].a, offset,
coefficient);
153.                     image[i][j].a = result;
154.                     image[i][j].b = result;
155.                     image[i][j].c = result;
156.                 }
157.             }
158.         }
159.     }
160.     if (type == 1) {
161.         if (format_1) {
162.             for (int i = 0; i < height; i++) {
163.                 for (int j = 0; j < width; j++) {
164.                     image[i][j].a = correct_color(image[i][j].a, offset, coefficient);
165.                 }
166.             }
167.         }
168.         else {
169.             for (int i = 0; i < height; i++) {
170.                 for (int j = 0; j < width; j++) {
171.                     int result = correct_color(image[i][j].a, offset, coefficient);
```

```
172.                     image[i][j].a = result;
173.                     image[i][j].b = result;
174.                     image[i][j].c = result;
175.                 }
176.             }
177.         }
178.     }
179.     if (type == 2) {
180.         unsigned char min = 255;
181.         unsigned char max = 0;
182.         if (format_1) {
183.             for (int i = 0; i < this->height; i++) {
184.                 for (int j = 0; j < this->width; j++) {
185.                     // Èùåì ìèíèìóì
186.                     min = std::min(min, image[i][j].a);
187.                     min = std::min(min, image[i][j].b);
188.                     min = std::min(min, image[i][j].c);
189. 
190.                     // Èùåì ìàêñèìóì
191.                     max = std::max(max, image[i][j].a);
192.                     max = std::max(max, image[i][j].b);
193.                     max = std::max(max, image[i][j].c);
194.                 }
195.             }
196.         }
197.         else {
198.             for (int i = 0; i < this->height; i++) {
199.                 for (int j = 0; j < this->width; j++) {
200.                     // Èùåì ìèíèìóì
201.                     min = std::min(min, image[i][j].a);
202.                     // Èùåì ìàêñèìóì
203.                     max = std::max(max, image[i][j].a);
204.                 }
205.             }
206.         }
207.         offset = static_cast<int>(min);
208.         coefficient = 255.0 / (static_cast<int>(max) - static_cast<int>(min));
209.         std::cout << offset << " " << coefficient << "\n";
210.         for (int i = 0; i < this->height; i++) {
211.             for (int j = 0; j < this->width; j++) {
212.                 this->image[i][j].a = correct_color(this->image[i][j].a, offset, coefficient);
213.                 this->image[i][j].b = correct_color(this->image[i][j].b, offset, coefficient);
214.                 this->image[i][j].c = correct_color(this->image[i][j].c, offset, coefficient);
215.             }
216.         }
217.     }
218.     if (type == 3) {
219.         unsigned char min = 255;
220.         unsigned char max = 0;
221.         for (int i = 0; i < this->height; i++) {
222.             for (int j = 0; j < this->width; j++) {
223.                 // Èùåì ìèíèìóì
224.                 min = std::min(min, image[i][j].a);
225.                 // Èùåì ìàêñèìóì
226.                 max = std::max(max, image[i][j].a);
227.             }
228.         }
229.         offset = static_cast<int>(min);
230.         coefficient = 255.0 / (static_cast<int>(max) - static_cast<int>(min));
231.         std::cout << offset << " " << coefficient << "\n";
232.         if (this->format_1 == '6') {
233.             for (int i = 0; i < this->height; i++) {
234.                 for (int j = 0; j < this->width; j++) {
235.                     this->image[i][j].a = correct_color(this->image[i][j].a, offset,
coefficient);
236.                 }
237.             }
238.         }
```

```
239.         else {
240.             for (int i = 0; i < this->height; i++) {
241.                 for (int j = 0; j < this->width; j++) {
242.                     this->image[i][j].a = correct_color(this->image[i][j].a, offset,
coefficient);
243.                     this->image[i][j].b = correct_color(this->image[i][j].b, offset,
coefficient);
244.                     this->image[i][j].c = correct_color(this->image[i][j].c, offset,
coefficient);;
245.                 }
246.             }
247.         }
248.     }
249.     if (type == 4) {
250.         // Ñïèñîê âñòðå÷àåìîñòè ÿðêîñòè öâåòà
251.         std::vector <unsigned long long int> color_count(256, 0);
252.         auto ignoring_count = static_cast<long long int>(0.03901 * this->width * this->height);
253.         // Ñ÷èòàåì âñå âñòðå÷àåìîñòè ÿðêîñòåé
254.         for (int i = 0; i < this->height; i++) {
255.             for (int j = 0; j < this->width; j++) {
256.                 color_count[this->image[i][j].a]++;
257.                 color_count[this->image[i][j].b]++;
258.                 color_count[this->image[i][j].c]++;
259.             }
260.         }
261.         int max_ignored_brightness = 255; // âåðõíÿÿ ãðàíèöà èãíîðèðîâàíèÿ
262.         int min_ignored_brightness = 0; // íèæíÿÿ ãðàíèöà èãíîðèðîâàíèÿ
263.         while (ignoring_count > 0) {
264.             ignoring_count -= color_count[max_ignored_brightness];
265.             if (ignoring_count >= 0) {
266.                 max_ignored_brightness--;
267.             }
268.         }
269.         ignoring_count = static_cast<long long int>(0.03901 * this->width * this->height);
270.         while (ignoring_count > 0) {
271.             ignoring_count -= color_count[min_ignored_brightness];
272.             if (ignoring_count >= 0) {
273.                 min_ignored_brightness++;
274.             }
275.         }
276.
277.         offset = static_cast<int>(min_ignored_brightness);
278.         coefficient = 255.0 / (static_cast<int>(max_ignored_brightness) - static_cast<int>(min_
ignored_brightness));
279.         std::cout << offset << " " << coefficient << "\n";
280.         for (int i = 0; i < this->height; i++) {
281.             for (int j = 0; j < this->width; j++) {
282.                 this->image[i][j].a = correct_color(this->image[i][j].a, offset, coefficient);
283.                 this->image[i][j].b = correct_color(this->image[i][j].b, offset, coefficient);
284.                 this->image[i][j].c = correct_color(this->image[i][j].c, offset, coefficient);
285.             }
286.         }
287.     }
288.     if (type == 5) {
289.         // Ñïèñîê âñòðå÷àåìîñòè ÿðêîñòè öâåòà
290.         std::vector <long long int> color_count(256, 0);
291.         auto ignoring_count = static_cast<long long int>(0.03901 * this->width * this->height);
292.         // Ñ÷èòàåì âñå âñòðå÷àåìîñòè ÿðêîñòåé
293.         for (int i = 0; i < this->height; i++) {
294.             for (int j = 0; j < this->width; j++) {
295.                 color_count[this->image[i][j].a]++;
296.             }
297.         }
298.         int max_ignored_brightness = 255; // âåðõíÿÿ ãðàíèöà èãíîðèðîâàíèÿ
299.         int min_ignored_brightness = 0; // íèæíÿÿ ãðàíèöà èãíîðèðîâàíèÿ
300.         while (ignoring_count > 0) {
301.             ignoring_count -= color_count[max_ignored_brightness];
302.             if (ignoring_count >= 0) {
```

```
303.                 max_ignored_brightness--;
304.             }
305.         }
306.         ignoring_count = static_cast<long long int>(0.03901 * this->width * this->height);
307.         while (ignoring_count > 0) {
308.             ignoring_count -= color_count[min_ignored_brightness];
309.             if (ignoring_count >= 0) {
310.                 min_ignored_brightness++;
311.             }
312.         }
313.
314.         offset = static_cast<int>(min_ignored_brightness);
315.         coefficient = 255.0 / (static_cast<int>(max_ignored_brightness) - static_cast<int>(min_
ignored_brightness));
316.         std::cout << offset << " " << coefficient << "\n";
317.         if (this->format_1 == '6') {
318.             for (int i = 0; i < this->height; i++) {
319.                 for (int j = 0; j < this->width; j++) {
320.                     this->image[i][j].a = correct_color(this->image[i][j].a, offset,
coefficient);
321.                 }
322.             }
323.         }
324.         else {
325.             for (int i = 0; i < this->height; i++) {
326.                 for (int j = 0; j < this->width; j++) {
327.                     this->image[i][j].a = correct_color(this->image[i][j].a, offset,
coefficient);
328.                     this->image[i][j].b = correct_color(this->image[i][j].b, offset,
coefficient);
329.                     this->image[i][j].c = correct_color(this->image[i][j].c, offset,
coefficient);
330.                 }
331.             }
332.         }
333.     }
334.     if (type % 2 == 1) {
335.         // Ïåðåâîäèì íáðàòíî â RGB
336.         for (int i = 0; i < this->height; i++) {
337.             for (int j = 0; j < this->width; j++) {
338.                 this->image[i][j].YCbCr_601_to_RGB();
339.             }
340.         }
341.     }
342. }
```