**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное

учреждение высшего образования

**Санкт-Петербургский исследовательский университет**

**Информационных технологий, механики и оптики**

Факультет информационных технологий и программирования

Дисциплина: компьютерная геометрия и графика

**Отчет**

по лабораторной работе №5

***Изучение алгоритма настройки автояркости изображения***

Выполнил: студент гр. M3102

Муслимов О.С.

Преподаватель: Скаков П.С.

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).

**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:

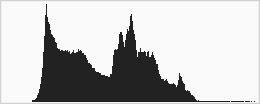
С: fprintf(stderr, "Error\n");

C++: std::cerr

**Теоретическая часть**

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

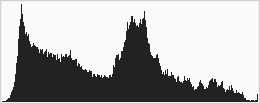
Эту проблему можно хорошо продемонстрировать, если построить распределение яркостей всех точек изображения – гистограмму.



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. 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 программирования: C++20.

Я сделал структуру данных RGB и создал массив этой структуры, в которой будут храниться информация о каждом канале. При нечётных преобразованиях мы переходим в другое цветовое пространство - YCbCr.601.

Для преобразований 0 и 1 для каждого пикселя применяется формула: **(X-<смещение>)\*<множитель>** (Для YCbCr только для канала Y). При преобразованиях 2 и 3 находится минимальное и максимальное значения и к каждому пикселю применяется формула: y = (x - min)\*255/(max - min) (Для YCbCr только для канала Y). Для преобразований 4 и 5 мы находим максимальный, но не учитывается heidght\*width\*0.0039 штук, и так же с минимальным. Дальше применяется алгоритм для 4 и 5 преобразований. Еще был написан класс, который фиксирует ошибки введенных аргументов в консоль и ошибки, которые могут присутствовать в файле.

**Выводы**

В ходе проделанной работы были получены следующие результаты:

Получена программа, которая, применяя вышеперечисленные алгоритмы, настраивает автояркость изображения в разных цветовых пространствах и тем самым на выход идет более приятное для восприятия глаз изображение (в случае автоматического режима) или изображение, настройка яркости которой удовлетворяет пользователя (в случае введенных аргументов таких, как <смещения> и <множителя>).

**Листинг**

**main.cpp**

|  |
| --- |
| #include <iostream> |
|  | #include "CException.h" |
|  | #include "CImage.h" |
|  |  |
|  | using namespace std; |
|  | SInput input; |
|  |  |
|  | int main(int argc, char \*argv[]) { |
|  | try { |
|  | if (argc < 4 || argc > 6) { |
|  | throw CExpension("Wrong amount of arguments"); |
|  | } |
|  | for (int i = 1; i < argc; i++) { |
|  | switch (i) { |
|  | case 1: |
|  | input.inputName = argv[i]; |
|  | break; |
|  | case 2: |
|  | input.outputName = argv[i]; |
|  | break; |
|  | case 3: |
|  | input.mode = atoi(argv[i]); |
|  | if (input.mode < 0 || input.mode > 5) { |
|  | throw CExpension("Wrong mode"); |
|  | } |
|  | break; |
|  | case 4: |
|  | input.offset = atoi(argv[i]); |
|  | if (input.offset < -255 || input.offset > 255) { |
|  | throw CExpension("Wrong offset"); |
|  | } |
|  | break; |
|  | case 5: |
|  | input.multiplier = atof(argv[i]); |
|  | if (input.multiplier < 1.0 / 255.0 || input.multiplier > 255.0) { |
|  | throw CExpension("Wrong multiplier"); |
|  | } |
|  | } |
|  | } |
|  | if ((input.mode == 0 || input.mode == 1) && argc != 6) { |
|  | throw CExpension("Wrong amount of arguments"); |
|  | } |
|  | CImage image(input); |
|  | image.settingUpOfBrightntess(input); |
|  | image.writeToFile(input.outputName); |
|  | } catch (CExpension &expension) { |
|  | cerr << expension.getError(); |
|  | if (expension.getFile()) { |
|  | fclose(expension.getFile()); |
|  | } |
|  | exit(1); |
|  | } |
|  | return 0; |
|  | } |

**CException.h**

|  |
| --- |
| #ifndef UNTITLED5\_CEXCEPTION\_H |
|  | #define UNTITLED5\_CEXCEPTION\_H |
|  |  |
|  |  |
|  | #include <string> |
|  |  |
|  | using namespace std; |
|  |  |
|  | class CExpension { |
|  | public: |
|  | CExpension(string error); |
|  |  |
|  | CExpension(string error, FILE \*file); |
|  |  |
|  | string getError(); |
|  |  |
|  | FILE \*getFile(); |
|  |  |
|  | private: |
|  | string error\_; |
|  | FILE \*file\_ = nullptr; |
|  | }; |
|  |  |
|  |  |
|  | #endif //UNTITLED5\_CEXCEPTION\_H |

**CException.cpp**

|  |
| --- |
| #include "CException.h" |
|  | CExpension::CExpension(string error) { |
|  | this->error\_ = error; |
|  | } |
|  |  |
|  | CExpension::CExpension(string error, FILE \*file) { |
|  | this->error\_ = error; |
|  | this->file\_ = file; |
|  | } |
|  |  |
|  | string CExpension::getError() { |
|  | return this->error\_; |
|  | } |
|  |  |
|  | FILE \*CExpension::getFile() { |
|  | return this->file\_; |
|  | } |

**CImage.h**

|  |
| --- |
| #ifndef UNTITLED5\_CIMAGE\_H |
|  | #define UNTITLED5\_CIMAGE\_H |
|  |  |
|  | #include <iostream> |
|  | #include "CException.h" |
|  | #include <climits> |
|  | #include <algorithm> |
|  | #include <vector> |
|  |  |
|  | struct SInput { |
|  | char \*inputName; |
|  | char \*outputName; |
|  | int mode; |
|  | int offset; |
|  | double multiplier; |
|  | }; |
|  |  |
|  | struct SMinMax { |
|  | int min; |
|  | int max; |
|  | }; |
|  |  |
|  | struct RGB { |
|  | double red; |
|  | double green; |
|  | double blue; |
|  | }; |
|  |  |
|  | class CImage { |
|  | public: |
|  | CImage(SInput config); |
|  |  |
|  | void settingUpOfBrightntess(SInput config); |
|  |  |
|  | void writeToFile(char \*fileName); |
|  |  |
|  | private: |
|  | FILE \*file; |
|  | int version; |
|  | int width; |
|  | int height; |
|  | int max\_val; |
|  | int size; |
|  | RGB \*pix; |
|  | bool colorSpace; |
|  |  |
|  | int getFirstD(int i, vector<int> &p); |
|  |  |
|  | int getFirstB(int i, vector<int> &p); |
|  |  |
|  | void userMode(double offset, double multilier); |
|  |  |
|  | void autoMode(SInput config); |
|  |  |
|  | double cut(double x); |
|  |  |
|  | SMinMax getDarkestAndBrightest(bool is0039); |
|  |  |
|  | void RGBtoYCbCr601(); |
|  |  |
|  | void YCbCr601toRGB(); |
|  | }; |
|  |  |
|  |  |
|  | #endif //UNTITLED5\_CIMAGE\_H |

**CImage.cpp**

|  |
| --- |
| #include "CImage.h" |
|  |  |
|  | CImage::CImage(SInput config) { |
|  | FILE \*f = fopen(config.inputName, "rb"); |
|  | file = f; |
|  | if (fscanf(f, "P%i%i%i%i\n", &this->version, &this->width, &this->height, &max\_val) != 4) { |
|  | throw CExpension("Wrong amount data in file", f); |
|  | } |
|  | size = width \* height; |
|  | unsigned char \*buffer; |
|  | if (version == 6) { |
|  | buffer = new unsigned char[size \* 3]; |
|  | pix = new RGB[size]; |
|  | fread(buffer, sizeof(unsigned char), size \* 3, f); |
|  | for (int i = 0, j = 0; i < size \* 3; i += 3, j++) { |
|  | pix[j].red = (double) buffer[i]; |
|  | pix[j].green = (double) buffer[i + 1]; |
|  | pix[j].blue = (double) buffer[i + 2]; |
|  | } |
|  | } else if (version == 5) { |
|  | buffer = new unsigned char[size]; |
|  | pix = new RGB[size]; |
|  | fread(buffer, sizeof(unsigned char), size, f); |
|  | for (int i = 0; i < size; i++) { |
|  | pix[i].red = (double) buffer[i]; |
|  | pix[i].green = (double) buffer[i]; |
|  | pix[i].blue = (double) buffer[i]; |
|  | } |
|  | } else { |
|  | throw CExpension("Wrong version of file", f); |
|  | } |
|  | colorSpace = (config.mode == 0 || config.mode == 2 || config.mode == 4);//False is YCbCr601 |
|  | delete[] buffer; |
|  | fclose(f); |
|  | } |
|  |  |
|  | void CImage::writeToFile(char \*fileName) { |
|  | FILE \*f = fopen(fileName, "wb"); |
|  | if (!f) { |
|  | throw CExpension("Input file didn't open"); |
|  | } |
|  | fprintf(f, "P%i\n%i %i\n%i\n", version, width, height, max\_val); |
|  | unsigned char \*buffer; |
|  | if (version == 5) { |
|  | buffer = new unsigned char[size]; |
|  | for (int i = 0; i < size; i++) { |
|  | buffer[i] = (unsigned char) pix[i].red; |
|  | } |
|  | fwrite(buffer, sizeof(unsigned char), size, f); |
|  | } else { |
|  | buffer = new unsigned char[size \* 3]; |
|  | for (int i = 0, j = 0; i < size \* 3; i += 3, j++) { |
|  | buffer[i] = (unsigned char) pix[j].red; |
|  | buffer[i + 1] = (unsigned char) pix[j].green; |
|  | buffer[i + 2] = (unsigned char) pix[j].blue; |
|  | } |
|  | fwrite(buffer, sizeof(unsigned char), size \* 3, f); |
|  | } |
|  | delete[] buffer; |
|  | fclose(f); |
|  | } |
|  |  |
|  | void CImage::settingUpOfBrightntess(SInput config) { |
|  | if (config.mode % 2 != 0) { |
|  | RGBtoYCbCr601(); |
|  | } |
|  | if (config.mode == 0 || config.mode == 1) { |
|  | userMode(config.offset, config.multiplier); |
|  | } else { |
|  | autoMode(config); |
|  | } |
|  | if (config.mode % 2 != 0) { |
|  | YCbCr601toRGB(); |
|  | } |
|  | } |
|  |  |
|  | void CImage::userMode(double offset, double multilier) { |
|  | if (version == 5 && colorSpace) { |
|  | for (int i = 0; i < size; i++) { |
|  | int newValue = cut((pix[i].red - offset) \* multilier); |
|  | pix[i].red = newValue; |
|  | pix[i].green = newValue; |
|  | pix[i].blue = newValue; |
|  | } |
|  | } else { |
|  | if (colorSpace) { |
|  | for (int i = 0; i < size; i++) { |
|  | int newRed = cut((pix[i].red - offset) \* multilier); |
|  | pix[i].red = newRed; |
|  | int newGreen = cut((pix[i].green - offset) \* multilier); |
|  | pix[i].green = newGreen; |
|  | int newBlue = cut((pix[i].blue - offset) \* multilier); |
|  | pix[i].blue = newBlue; |
|  | } |
|  | } else { |
|  | for (int i = 0; i < size; i++) { |
|  | int newValue = cut((pix[i].red - offset) \* multilier); |
|  | pix[i].red = newValue; |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void CImage::autoMode(SInput config) { |
|  | SMinMax tmp; |
|  | tmp = getDarkestAndBrightest(config.mode == 4 || config.mode == 5); |
|  | std::cout << tmp.min << " " << 255.0 / (tmp.max - tmp.min) << "\n"; |
|  | if (version == 5 && colorSpace) { |
|  | for (int i = 0; i < size; i++) { |
|  | int newValue = cut((pix[i].red - tmp.min) \* 255.0 / (tmp.max - tmp.min)); |
|  | pix[i].red = newValue; |
|  | pix[i].green = newValue; |
|  | pix[i].blue = newValue; |
|  | } |
|  | } else { |
|  | if (colorSpace) { |
|  | for (int i = 0; i < size; i++) { |
|  | int newRed = cut((pix[i].red - tmp.min) \* 255.0 / (tmp.max - tmp.min)); |
|  | pix[i].red = newRed; |
|  | int newGreen = cut((pix[i].green - tmp.min) \* 255.0 / (tmp.max - tmp.min)); |
|  | pix[i].green = newGreen; |
|  | int newBlue = cut((pix[i].blue - tmp.min) \* 255.0 / (tmp.max - tmp.min)); |
|  | pix[i].blue = newBlue; |
|  | } |
|  | } else { |
|  | for (int i = 0; i < size; i++) { |
|  | int newValue = cut((pix[i].red - tmp.min) \* 255.0 / (tmp.max - tmp.min)); |
|  | pix[i].red = newValue; |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | SMinMax CImage::getDarkestAndBrightest(bool is0039) { |
|  | vector<int> p(256, 0); |
|  | SMinMax tmp; |
|  | tmp.min = 0; |
|  | tmp.max = 255; |
|  | for (int i = 0; i < size; i++) { |
|  | p[pix[i].red]++; |
|  | if (version == 6 && colorSpace) { |
|  | p[pix[i].green]++; |
|  | p[pix[i].blue]++; |
|  | } |
|  | } |
|  | if (is0039) { |
|  | int j = size \* 0.0039; |
|  | j = version == 6 && colorSpace ? j \* 3 : j; |
|  | int i = 0; |
|  | int bright = 255, dark = 0; |
|  | while (i < j) { |
|  | bright = getFirstB(bright, p); |
|  | p[bright]--; |
|  | dark = getFirstD(dark, p); |
|  | p[dark]--; |
|  | i++; |
|  | } |
|  | } |
|  | bool flagB = false, flagD = false; |
|  | for (int i = 0; i < p.size(); i++) { |
|  | if (p[i] > 0 && !flagD) { |
|  | tmp.min = i; |
|  | flagD = true; |
|  | } |
|  | if (p[255 - i] > 0 && !flagB) { |
|  | tmp.max = 255 - i; |
|  | flagB = true; |
|  | } |
|  | } |
|  | return tmp; |
|  | } |
|  |  |
|  | int CImage::getFirstD(int i, vector<int> &p) { |
|  | while (p[i] == 0) { |
|  | i++; |
|  | } |
|  | return i; |
|  | } |
|  |  |
|  | int CImage::getFirstB(int i, vector<int> &p) { |
|  | while (p[i] == 0) { |
|  | i--; |
|  | } |
|  | return i; |
|  | } |
|  |  |
|  | double CImage::cut(double x) { |
|  | x = x > 255.0 ? 255.0 : x; |
|  | x = x < 0.0 ? 0.0 : x; |
|  | return x; |
|  | } |
|  |  |
|  | void CImage::YCbCr601toRGB() { |
|  | double kr = 0.299, kb = 0.114; |
|  | for (int i = 0; i < size; i++) { |
|  | double y = pix[i].red / 255.0; |
|  | double pb = pix[i].green / 255.0 - 0.5; |
|  | double pr = pix[i].blue / 255.0 - 0.5; |
|  | double r, g, b; |
|  | b = 2 \* pb \* (1 - kb) + y; |
|  | r = 2 \* pr \* (1 - kr) + y; |
|  | g = (y - kr \* r - kb \* b) / (1 - kr - kb); |
|  | r = r < 0 ? 0 : r; |
|  | r = r > 1 ? 1 : r; |
|  | b = b < 0 ? 0 : b; |
|  | b = b > 1 ? 1 : b; |
|  | g = g < 0 ? 0 : g; |
|  | g = g > 1 ? 1 : g; |
|  | pix[i].red = r \* 255.0; |
|  | pix[i].green = g \* 255.0; |
|  | pix[i].blue = b \* 255.0; |
|  | } |
|  | } |
|  |  |
|  | void CImage::RGBtoYCbCr601() { |
|  | double kr = 0.299, kb = 0.114; |
|  | for (int i = 0; i < size; i++) { |
|  | double r = pix[i].red / 255.0; |
|  | double g = pix[i].green / 255.0; |
|  | double b = pix[i].blue / 255.0; |
|  | double y = kr \* r + (1 - kr - kb) \* g + kb \* b; |
|  | double pb = 0.5 \* (b - y) / (1.0 - kb); |
|  | double pr = 0.5 \* (r - y) / (1.0 - kr); |
|  | pix[i].red = y \* 255.0; |
|  | pix[i].green = (pb + 0.5) \* 255.0; |
|  | pix[i].blue = (pr + 0.5) \* 255.0; |
|  | } |
|  | } |