**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***

Выполнила: студент гр. M3101

Семина А. Д.

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.

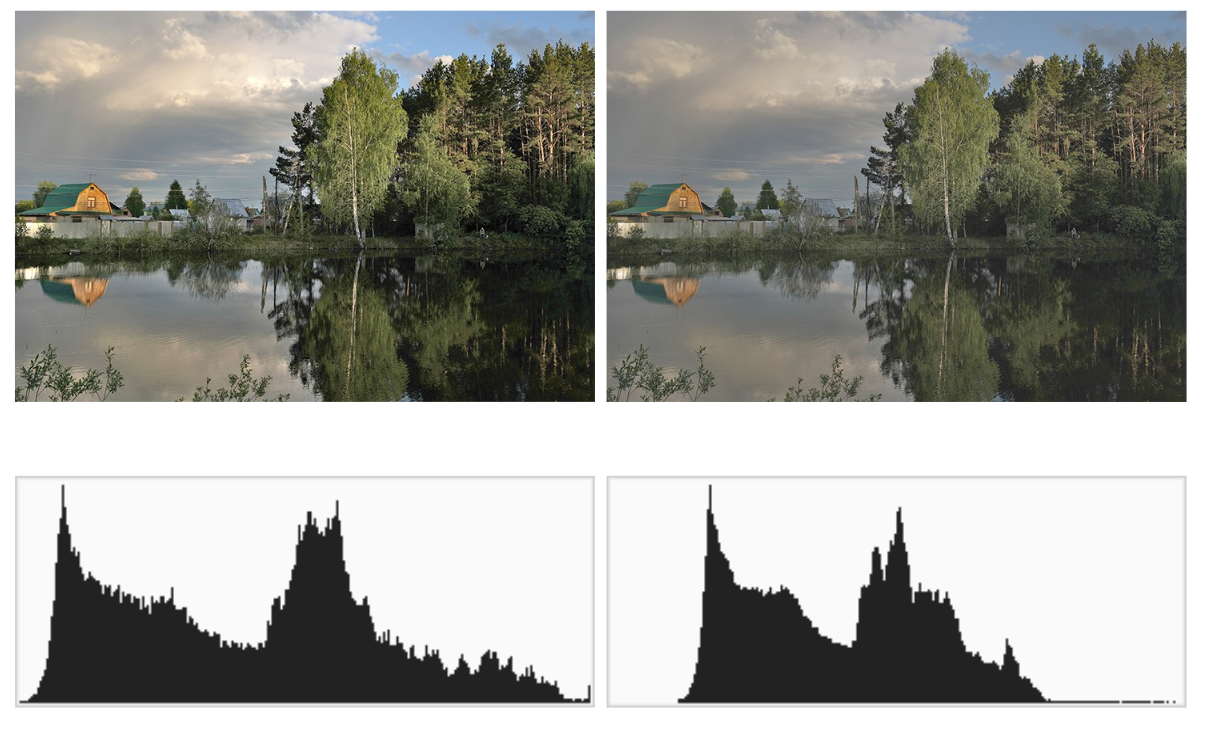
Эту проблему можно хорошо продемонстрировать, если построить распределение яркостей всех точек изображения – гистограмму.



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

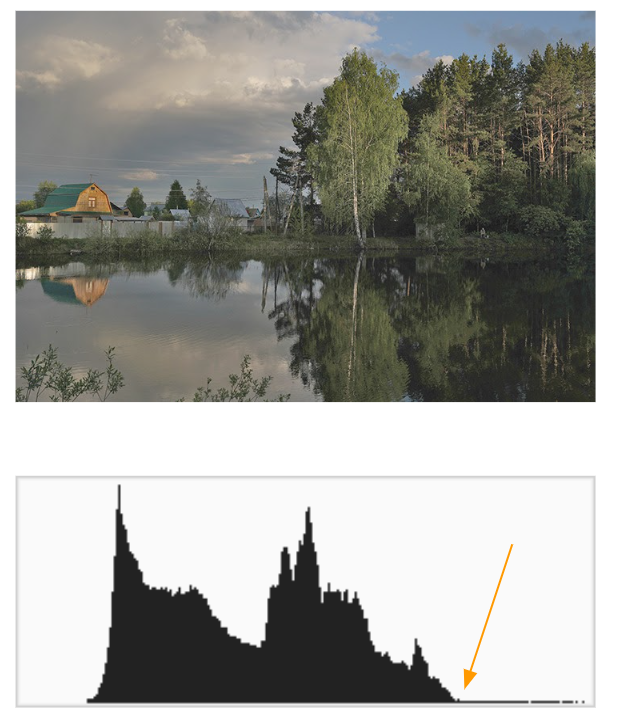
Преобразование яркости каждого пикселя можно описать простой формулой:

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.

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.

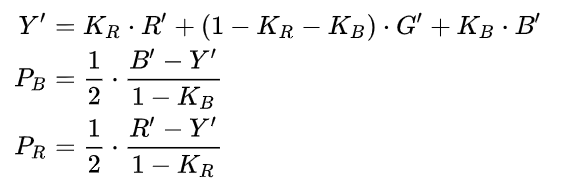


# *YCbCr.601*

YCbCr иногда сокращают до YCC. Y’CbCr часто называют YPbPr, когда речь идет о системах аналогового компонентного видео, хотя термин Y’CbCr обычно используется для обеих систем.

Y’CbCr часто путают с цветовым пространством YUV, и, как правило, термины YCbCr и YUV используются как взаимозаменяемые, что приводит к некоторой путанице. Когда речь идет о видео или сигналах в цифровой форме, термин «YUV» в основном означает «Y’CbCr».

Сигналы Y’CbCr (до нормирования и смещения для перевода сигналов в цифровую форму) называют YPbPr. Они формируются с применением гамма-коррекции из соответствующих RGB источников с помощью двух определенных констант KB и KR следующим образом:



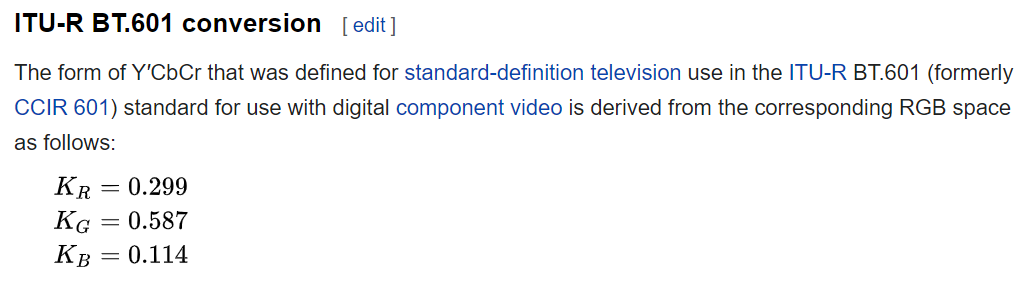
{\displaystyle {\begin{aligned}Y'&=K\_{R}\cdot R'+(1-K\_{R}-K\_{B})\cdot G'+K\_{B}\cdot B'\\P\_{B}&={\frac {1}{2}}\cdot {\frac {B'-Y'}{1-K\_{B}}}\\P\_{R}&={\frac {1}{2}}\cdot {\frac {R'-Y'}{1-K\_{R}}}\end{aligned}}}

где KB и KR коэффициенты, которые обычно выводятся из определения соответствующего пространства RGB.

Здесь апостроф ' означает компоненты с гамма-коррекцией, поэтому R', G' и B' располагаются в пределах от 0 до 1, где 0 соответствует минимальной интенсивности (например, для отображения чёрного цвета) и 1 соответствует максимуму (например, для отображения белого цвета). Результирующее значение яркости (Y) будет иметь пределы от 0 до 1, а значения цветности (PB и PR) будут расположены в пределах от −0.5 до +0.5. Обратный процесс преобразования может быть легко получен путём обращения представленных выше уравнений.

При представлении сигналов в цифровой форме, результат нормируется и округляется, и, как правило, добавляется смещение. Так, например, нормирование и смещение, применяемое к компоненте Y' согласно спецификации (например, MPEG-2[[1]](https://ru.wikipedia.org/wiki/YCbCr#cite_note-1)), приводит к значению 16 для чёрного и значению 235 для белого при использовании 8-битного представления. Стандарт имеет 8-битные цифровые версии Cb и Cr, нормированные в другом диапазоне: от 16 до 240.

Нормирование приводит к использованию меньшего диапазона цифровых значений. В этом случае имеется некоторый запас, который может быть использован в случае превышения порога входными данными, таким образом устраняя необходимость их отсечения. Дополнительные диапазоны могут быть использованы для расширения цветовой палитры, как например в пространстве xvYCC .

Так как в пространстве YCRCB можно представить существенно более широкую гамму значений сигнала, чем поддерживаемая в соответствующих диапазонах сигналов R, G и B, то существует вероятность получения таких сигналов Y, CR и CB, которые, несмотря на пригодность каждого из них по отдельности, могут, при преобразовании к RGB, привести к получению значений, лежащих вне допустимых пределов. Это можно предотвратить наложив ограничения на сигналы Y, CR и CB, также такие ограничения применяются для поддержания значений яркости и цветовых оттенков, при этом субъективные искажения минимизируются посредством потери только насыщенности цвета.

**Экспериментальная часть**

Язык программирования: C++ 14

Данные считываются в формате, указанном в описании работы. Если данные не соответствуют формату, то выводится ошибка и программа завершает свою работу.

Если введено нечетное число (P6), то сначала переводим из RGB в YCrCb (а в конце перед выводом обратно). Если преобразование 2..5, то в массив colors считаю либо все пиксели из файла (RGB), либо только из канала Y (YCrCb). Потом по формулам из теории высчитываю, сколько символов нужно проигнорировать, нахожу цвет макс и цвет мин, а уже по ним высчитываю смещение и множитель, которые потом применяю ко всем пикселям или к каналу Y.

**Выводы**

В ходе проделанной работы были получены следующие результаты:

Время работы алгоритма - O(w \* h), где w и h – высота и ширина изображения. Сделать быстрее не получится, так как хотя бы раз мы должны пройти по всем пикселям изображения. Перевод из одного цветового пространства в другое – время тоже линейное, в сумме O(n \* const) = O(n).

**Листинг**

**Название исходного файла : main.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <algorithm>

#include <vector>

#include <string>

#include <cmath>

using namespace std;

int transformation, offset, quantity;

double multiplier;

char symbol;

int number, h, w, max\_color;

vector<unsigned char> image;

FILE \*fin, \*fout;

unsigned char find\_nearest(double color)

{

return (unsigned char)min(max(color, 0.), 255.);

}

void from\_RGB\_to\_YCbCr\_601()

{

double K\_r = 0.299;

double K\_g = 0.587;

double K\_b = 0.114;

for(int i = 0; i < 3 \* h \* w; i += 3)

{

double R = image[i] / 255.;

double G = image[i + 1] / 255.;

double B = image[i + 2] / 255.;

double Y = K\_r \* R + K\_g \* G + K\_b \* B;

double C\_b = 0.5 \* (B - Y) / (1 - K\_b);

double C\_r = 0.5 \* (R - Y) / (1 - K\_r);

image[i] = find\_nearest(255 \* Y);

image[i + 1] = find\_nearest(255 \* (C\_b + 0.5));

image[i + 2] = find\_nearest(255 \* (C\_r + 0.5));

}

}

void from\_YCbCr\_601\_to\_RGB()

{

double K\_r = 0.299;

double K\_g = 0.587;

double K\_b = 0.114;

for(int i = 0; i < 3 \* h \* w; i += 3)

{

double Y = image[i] / 255.;

double C\_b = image[i + 1] / 255. - 0.5;

double C\_r = image[i + 2] / 255. - 0.5;

double R = Y + C\_r \* 2 \* (1 - K\_r);

double B = Y + C\_b \* 2 \* (1 - K\_b);

double G = (Y - K\_r \* R - K\_b \* B) / K\_g;

image[i] = find\_nearest(255 \* R);

image[i + 1] = find\_nearest(255 \* G);

image[i + 2] = find\_nearest(255 \* B);

}

}

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

{

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

if (argc == 4 || argc == 6)

{

try

{

transformation = stoi(argv[3]);

if(transformation < 0 || transformation > 5)

{

cerr << "transformation < 0 || transformation > 5";

return 1;

}

if(transformation == 0 || transformation == 1)

{

offset = stoi(argv[4]);

if(offset < -255 || offset > 255)

{

cerr << "offset < -255 || offset > 255";

return 1;

}

multiplier = stof(argv[5]);

if(multiplier < 1./255. || multiplier > 255)

{

cerr << "multiplier < 1./255. || multiplier > 255";

return 1;

}

}

}

catch(...)

{

cerr << "Invalid arguments";

return 1;

}

}

else

{

cerr << "Invalid number of arguments";

return 1;

}

fin = fopen(argv[1], "rb");

if (fin == NULL)

{

cerr << "Input file can't be open";

return 1;

}

quantity = fscanf(fin, "%c%d\n%d%d\n%d\n", &symbol, &number, &w, &h, &max\_color);

if (quantity != 5 || symbol != 'P' || (number != 5 && number != 6) || h <= 0 || w <= 0 || max\_color <= 0 || max\_color > 255)

{

fclose(fin);

cerr << "Invalid ppm file";

return 1;

}

if (number == 5)

{

image.resize(h \* w, 0);

quantity = fread(&image[0], sizeof(unsigned char), image.size(), fin);

if (quantity != h \* w)

{

fclose(fin);

cerr << "Invalid pgm file: quantity != h \* w";

return 1;

}

}

else

{

image.resize(3 \* h \* w, 0);

quantity = fread(&image[0], sizeof(unsigned char), image.size(), fin);

if (quantity != 3 \* h \* w)

{

fclose(fin);

cerr << "Invalid pnm file: quantity != 3 \* h \* w";

return 1;

}

}

int plus = 1;

if(transformation % 2 == 1 && number == 6)//YCbCr P6

{

plus = 3;

from\_RGB\_to\_YCbCr\_601();

}

if(transformation != 0 && transformation != 1)//нахожу offset, multiplier

{

//можно отсортировать, но это дольше

int colors[256] = {0};

for(int i = 0; i < quantity; i += plus)

colors[image[i]]++;

int ignor = 0;

if(transformation == 4 || transformation == 5)

ignor = ceil(0.39 \* quantity / plus / 100.);

int sum = 0, i = 0;

while(sum <= ignor)

{

sum += colors[i];

i++;

}

int min\_ = i - 1;

sum = 0;

i = 255;

while(sum <= ignor)

{

sum += colors[i];

i--;

}

int max\_ = i + 1;

offset = min\_;

multiplier = 255. / (max\_ - min\_);

cout << offset << " " << multiplier;

}

for(int i = 0; i < quantity; i += plus)

image[i] = find\_nearest((image[i] - offset) \* multiplier);

if(transformation % 2 == 1 && number == 6)//YCbCr P6

from\_YCbCr\_601\_to\_RGB();

fout = fopen(argv[2], "wb");

if (fout == NULL)

{

cerr << "Output file can't be open";

return 1;

}

fprintf(fout, "%c%d\n%d %d\n%d\n", symbol, number, w, h, max\_color);

int quantity\_out = fwrite(&image[0], sizeof(unsigned char), image.size(), fout);

fclose(fout);

if (quantity != quantity\_out)

{

cerr << "Quantity of output file != width \* height";

return 1;

}

return 0;

}