**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 №4

***Изучение цветовых пространств***

Выполнила: студент гр. M3101

Семина А. Д.

Преподаватель: Скаков П. С.

Санкт-Петербург

2020

**Цель работы:** реализовать программу, которая позволяет проводить преобразования между цветовыми пространствами.

Входные и выходные данные могут быть как одним файлом ppm, так и набором из 3 pgm.

**Описание работы**

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

Аргументы передаются через командную строку:

**lab4.exe -f <from\_color\_space> -t <to\_color\_space> -i <count> <input\_file\_name> -o <count> <output\_file\_name>,**

где

* <color\_space> - RGB / HSL / HSV / YCbCr.601 / YCbCr.709 / YCoCg / CMY
* <count> - 1 или 3
* <file\_name>:
  + для count=1 просто имя файла; формат ppm
  + для count=3 шаблон имени вида <name.ext>, что соответствует файлам <name\_1.ext>, <name\_2.ext> и <name\_3.ext> для каждого канала соответственно; формат pgm

Порядок аргументов (-f, -t, -i, -o) может быть произвольным.

Везде 8-битные данные и полный диапазон (**0..255, PC range**).

**Полное решение**: всё работает + корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок.

/\* да, частичного решения нет \*/

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

Коды возврата:

0 - ошибок нет

1 - произошла ошибка

В поток вывода ничего не выводится (printf, cout).

Сообщения об ошибках выводятся в поток вывода ошибок:

С: fprintf(stderr, "Error\n");

C++: std::cerr

Следующие параметры гарантировано не будут выходить за обусловленные значения:

* <count> = 1 или 3;
* width и height в файле - положительные целые значения;
* яркостных данных в файле ровно width \* height;

**Теоретическая часть**

# *Определение*

**Цветовое пространство** — модель представления цвета, основанная на использовании цветовых координат. Цветовое пространство строится таким образом, чтобы любой цвет был представлен точкой, имеющей определённые координаты. Чаще всего одному набору координат будет соответствовать один цвет, но для некоторых случаев это не так (например, для модели CMYK или, например, когда кодируется цветовой тон — данные по тону «закольцованы», и тона для максимального и минимального значений совпадают).

Цветовые пространства описываются набором цветовых координат и правилами построения цветов. К примеру, RGB является трёхмерным цветовым пространством, где каждый цвет описан набором из трёх координат — каждая из них отвечает компоненте цвета в разложении на красный, зелёный и синий цвета. Количество координат задаёт размерность пространства. Существует много цветовых пространств различной размерности — от одномерных, которые могут описать исключительно монохромное изображение, до шести- и десятимерных, таких, например, как пространство CMYKLcLm (*Cyan, Magenta, Yellow, Key color, lightCyan, lightMagenta*). Пространства высокой размерности чаще всего используются в целях печати на плоттерах или аппаратах для цветопроб.

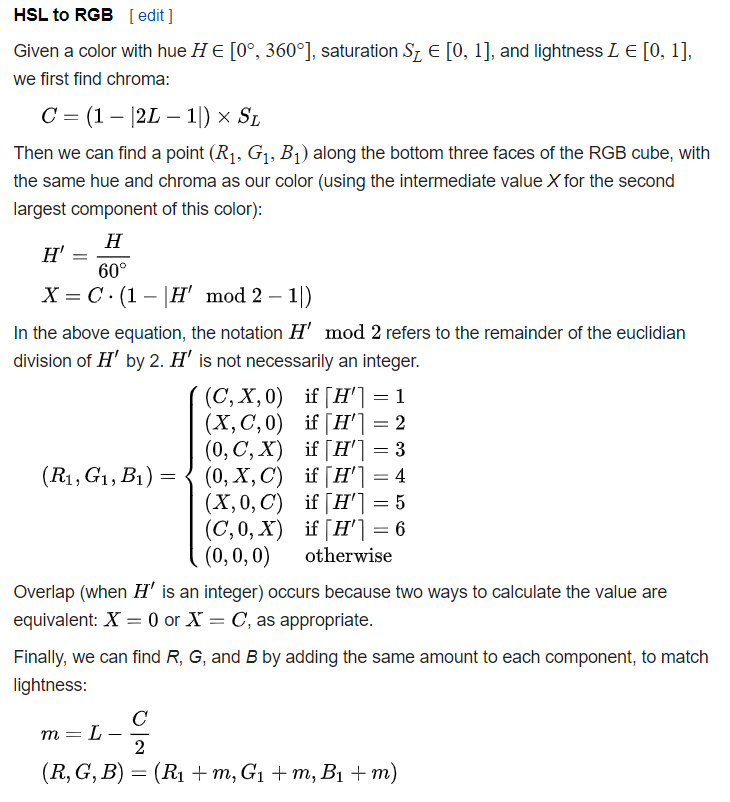
# *RGB*

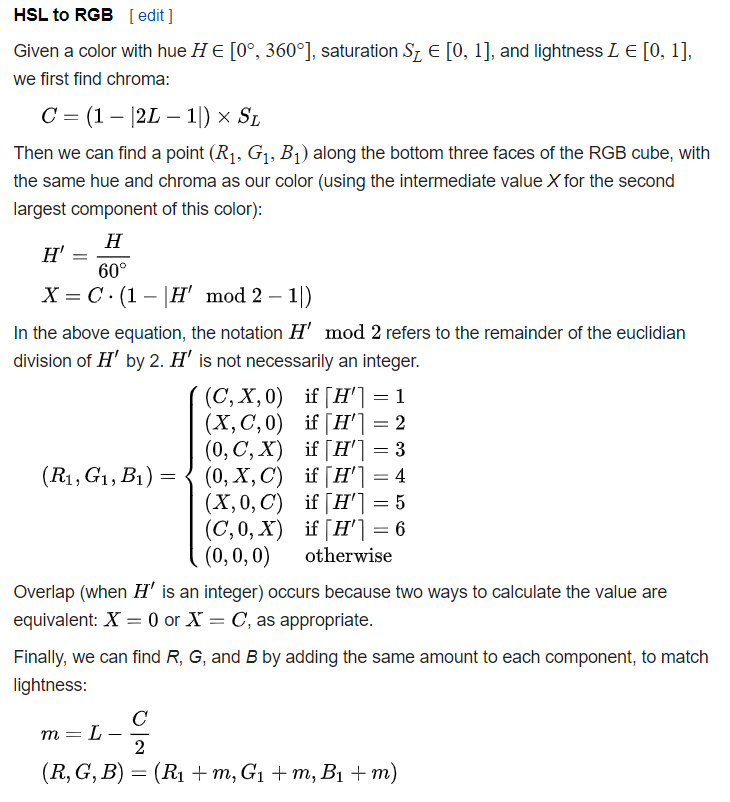
**RGB** (аббревиатура английских слов **r**ed, **g**reen, **b**lue — красный, зелёный, синий) или **КЗС** — аддитивная цветовая модель, описывающая способ кодирования цвета для цветовоспроизведения с помощью трёх цветов, которые принято называть основными. Выбор основных цветов обусловлен особенностями физиологии восприятия цвета сетчаткой человеческого глаза.

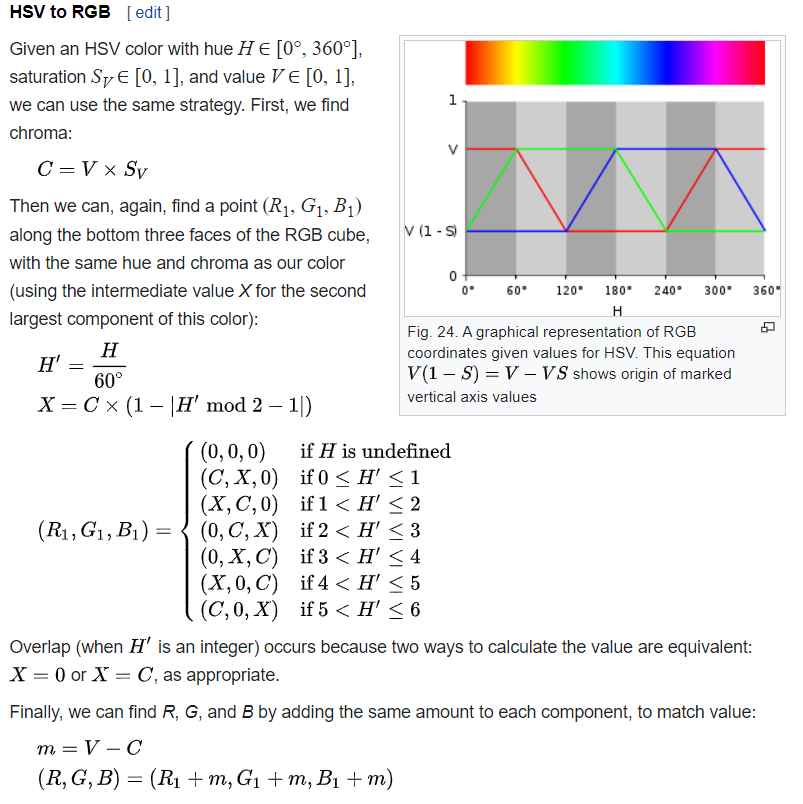
RGB-модель является аддитивной, где цвета получаются путём добавления (англ. addition) к чёрному цвету. При отсутствии краски нет никакого цвета — чёрный, максимальное смешение даёт белый. Если цвет экрана, освещённого цветным прожектором, обозначается в RGB как (r1, g1, b1), а цвет того же экрана, освещённого другим прожектором, — (r2, g2, b2), то при освещении двумя прожекторами цвет экрана будет обозначаться как (r1+r2, g1+g2, b1+b2).

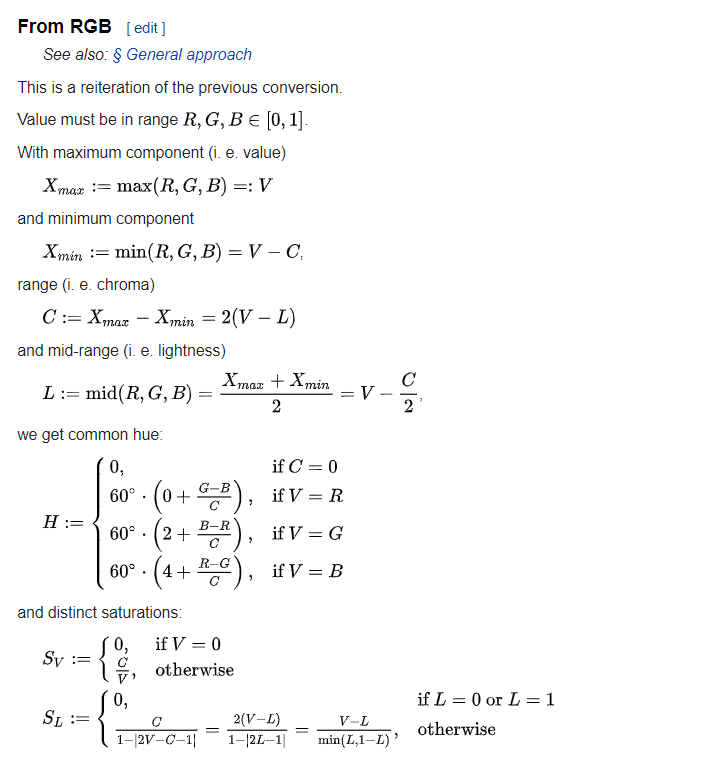
Изображение в данной цветовой модели состоит из трёх каналов. При смешении основных цветов, например, синего (B) и красного (R), получается пурпурный (M, magenta), зелёного (G) и красного (R) — жёлтый (Y, yellow), зелёного (G) и синего (B) — циановый (С, cyan). При смешении всех трёх основных цветов получается белый цвет (W, white).

# *HSL и HSV*







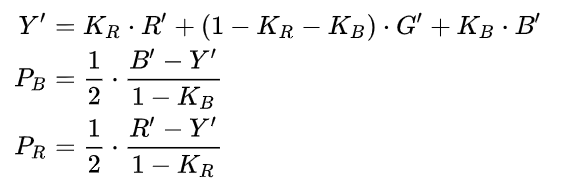


# *YCbCr.601 и YCbCr.709*

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:



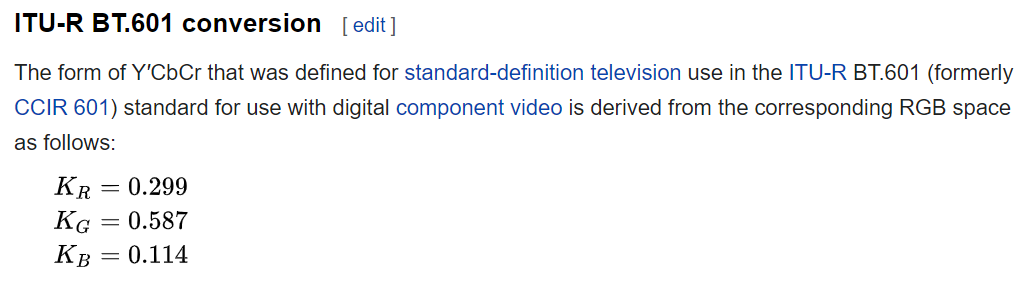
{\displaystyle {\begin{aligned}Y'&=K\_{R}\cdot R'+(1-K\_{R}-K\_{B})\cdot G'+K\_{B}\cdot B'\\P\_{B}&={\frac {1}{2}}\cdot {\frac {B'-Y'}{1-K\_{B}}}\\P\_{R}&={\frac {1}{2}}\cdot {\frac {R'-Y'}{1-K\_{R}}}\end{aligned}}}

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 предотваратить 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.

# 

# *YCoCg*

# 

# *CMY*

Так как модель CMYK применяют в основном в полиграфии при цветной печати, а бумага и прочие печатные материалы являются поверхностями, отражающими свет, удобнее считать, какое количество света отразилось от той или иной поверхности, нежели сколько поглотилось. Таким образом, если вычесть из белого три первичных цвета, RGB, мы получим тройку дополнительных цветов CMY. «Субтрактивный» означает «вычитаемый» — из белого вычитаются первичные 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. Для считывания и записи (P6) используется вспомогательный массив image.

В каждом массиве image1, image2, image3 с цветами 0..255 хранится 1 из 3 каналов. Я перевожу цветовое пространство из исходного в RGB, а потом в нужное нам по формулам, представленным в теории.

**Выводы**

В ходе проделанной работы были получены следующие результаты:

Время работы каждого из представленных алгоритмов - 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.

**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 count\_in, count\_out;

char symbol;

string color\_space\_from, color\_space\_to;

//const char\*

string input\_f, output\_f;

int number, h, w, max\_color;

vector<unsigned char> image1, image2, image3, image;

FILE \*fin, \*fout;

unsigned char find\_nearest(double color)

{

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

}

void from\_HSL\_to\_RGB()

{

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

{

double H = image1[i] \* 360. / 255.;

double S = image2[i] / 255.;

double L = image3[i] / 255.;

double C = (1 - abs(2 \* L - 1)) \* S;

double H\_ = H / 60.;

double X = C \* (1 - abs((fmod(H\_, 2.) - 1)));

double R1, G1, B1;

if (ceil(H\_) - 1. < 0.00001)

{

R1 = C;

G1 = X;

B1 = 0;

}

else if (ceil(H\_) - 2. < 0.00001)

{

R1 = X;

G1 = C;

B1 = 0;

}

else if (ceil(H\_) - 3. < 0.00001)

{

R1 = 0;

G1 = C;

B1 = X;

}

else if (ceil(H\_) - 4. < 0.00001)

{

R1 = 0;

G1 = X;

B1 = C;

}

else if (ceil(H\_) - 5. < 0.00001)

{

R1 = X;

G1 = 0;

B1 = C;

}

else if (ceil(H\_) - 6. < 0.00001)

{

R1 = C;

G1 = 0;

B1 = X;

}

else

{

R1 = 0;

G1 = 0;

B1 = 0;

}

double m = L - C / 2.;

image1[i] = find\_nearest(255 \* (R1 + m));

image2[i] = find\_nearest(255 \* (G1 + m));

image3[i] = find\_nearest(255 \* (B1 + m));

}

}

void from\_HSV\_to\_RGB()

{

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

{

double H = image1[i] \* 360. / 255.;

double S = image2[i] / 255.;

double V = image3[i] / 255.;

double C = V \* S;

double H\_ = H / 60.;

double X = C \* (1 - abs((fmod(H\_, 2.) - 1)));

double R1, G1, B1;

if (0 <= H\_ && H\_ <= 1)

{

R1 = C;

G1 = X;

B1 = 0;

}

else if (1 < H\_ && H\_ <= 2)

{

R1 = X;

G1 = C;

B1 = 0;

}

else if (2 < H\_ && H\_ <= 3)

{

R1 = 0;

G1 = C;

B1 = X;

}

else if (3 < H\_ && H\_ <= 4)

{

R1 = 0;

G1 = X;

B1 = C;

}

else if (4 < H\_ && H\_ <= 5)

{

R1 = X;

G1 = 0;

B1 = C;

}

else if (5 < H\_ && H\_ <= 6)

{

R1 = C;

G1 = 0;

B1 = X;

}

else // H undefined

{

R1 = 0;

G1 = 0;

B1 = 0;

}

double m = V - C;

image1[i] = find\_nearest(255 \* (R1 + m));

image2[i] = find\_nearest(255 \* (G1 + m));

image3[i] = find\_nearest(255 \* (B1 + m));

}

}

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 < h \* w; i++)

{

double Y = image1[i] / 255.;

double C\_b = image2[i] / 255. - 0.5;

double C\_r = image3[i] / 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;

image1[i] = find\_nearest(255 \* R);

image2[i] = find\_nearest(255 \* G);

image3[i] = find\_nearest(255 \* B);

}

}

void from\_YCbCr\_709\_to\_RGB()

{

double K\_b = 0.0722;

double K\_r = 0.2126;

double K\_g = 0.7152;//1 - K\_b - K\_r

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

{

double Y = image1[i] / 255.;

double C\_b = image2[i] / 255. - 0.5;

double C\_r = image3[i] / 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;

image1[i] = find\_nearest(255 \* R);

image2[i] = find\_nearest(255 \* G);

image3[i] = find\_nearest(255 \* B);

}

}

void from\_YCoCg\_to\_RGB()

{

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

{

double Y = image1[i] / 255.;

double Co = image2[i] / 255. - 0.5;

double Cg = image3[i] / 255. - 0.5;

image1[i] = find\_nearest(255 \* (Y + Co - Cg));

image2[i] = find\_nearest(255 \* (Y + Cg));

image3[i] = find\_nearest(255 \* (Y - Co - Cg));

}

}

void from\_CMY\_to\_RGB()

{

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

{

image1[i] = 255 - image1[i];

image2[i] = 255 - image2[i];

image3[i] = 255 - image3[i];

}

}

void from\_RGB\_to\_HSL()

{

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

{

double R = image1[i] / 255.;

double G = image2[i] / 255.;

double B = image3[i] / 255.;

double X\_max = max(R, max(G, B));

double X\_min = min(R, min(G, B));

double V = X\_max;

double L = (X\_max + X\_min) / 2.;

double C = X\_max - X\_min;

double H;

if (C == 0)

H = 0;

else if (V == R)

H = 60. \* (G - B) / C;

else if (V == G)

H = 60. \* (2 + (B - R) / C);

else //if(V == B)

H = 60. \* (4 + (R - G) / C);

if (H < 0) // V == R

H += 360;

double S;

if(L== 0 || L == 1)

S = 0;

else

{

//S = 2 \* (V - L) / (1 - abs(2 \* L- 1));

S = (V - L) / min(L, 1 - L);

}

image1[i] = find\_nearest(255 \* (H / 360.));

image2[i] = find\_nearest(255 \* S);

image3[i] = find\_nearest(255 \* L);

}

}

void from\_RGB\_to\_HSV()

{

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

{

double R = image1[i] / 255.;

double G = image2[i] / 255.;

double B = image3[i] / 255.;

double X\_max = max(R, max(G, B));

double X\_min = min(R, min(G, B));

double V = X\_max;

double C = V - X\_min;

double H;

if (C == 0)

H = 0;

else if (V == R)

H = 60. \* (G - B) / C;

else if (V == G)

H = 60. \* (2 + (B - R) / C);

else if(V == B)

H = 60. \* (4 + (R - G) / C);

if (H < 0)

H += 360;

double S;

if(V == 0)

S = 0;

else

S = C / V;

image1[i] = find\_nearest(255 \* (H / 360.));

image2[i] = find\_nearest(255 \* S);

image3[i] = find\_nearest(255 \* V);

}

}

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 < h \* w; i++)

{

double R = image1[i] / 255.;

double G = image2[i] / 255.;

double B = image3[i] / 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);

image1[i] = find\_nearest(255 \* Y);

image2[i] = find\_nearest(255 \* (C\_b + 0.5));

image3[i] = find\_nearest(255 \* (C\_r + 0.5));

}

}

void from\_RGB\_to\_YCbCr\_709()

{

double K\_b = 0.0722;

double K\_r = 0.2126;

double K\_g = 0.7152;//1 - K\_b - K\_r

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

{

double R = image1[i] / 255.;

double G = image2[i] / 255.;

double B = image3[i] / 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);

image1[i] = find\_nearest(255 \* Y);

image2[i] = find\_nearest(255 \* (C\_b + 0.5));

image3[i] = find\_nearest(255 \* (C\_r + 0.5));

}

}

void from\_RGB\_to\_YCoCg()

{

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

{

double R = image1[i] / 255.;

double G = image2[i] / 255.;

double B = image3[i] / 255.;

image1[i] = find\_nearest(255 \* (R / 4. + G / 2. + B / 4.));

image2[i] = find\_nearest(255 \* (R / 2. - B / 2. + 0.5)); // Co in[-0.5..0.5]

image3[i] = find\_nearest(255 \* (- R / 4. + G / 2. - B / 4. + 0.5));// Cg in[-0.5..0.5]

}

}

void from\_RGB\_to\_CMY()

{

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

{

image1[i] = 255 - image1[i];

image2[i] = 255 - image2[i];

image3[i] = 255 - image3[i];

}

}

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

{

//lab4.exe -f <from\_color\_space> -t <to\_color\_space> -i <count\_> <input\_file\_name> -o <count\_> <output\_file\_name>

if (argc == 11)

{

try

{

int i = 1;

while(i < 11)

{

string s = argv[i];

if(s == "-f")

{

color\_space\_from = argv[i + 1];

if(color\_space\_from != "RGB" && color\_space\_from != "HSL" && color\_space\_from != "HSV" && color\_space\_from != "YCbCr.601" && color\_space\_from != "YCbCr.709" && color\_space\_from != "YCoCg" && color\_space\_from != "CMY")

{

cerr << "Wrong color space";

return 1;

}

i += 2;

}

else if(s == "-t")

{

color\_space\_to = argv[i + 1];

if(color\_space\_to != "RGB" && color\_space\_to != "HSL" && color\_space\_to != "HSV" && color\_space\_to != "YCbCr.601" && color\_space\_to != "YCbCr.709" && color\_space\_to != "YCoCg" && color\_space\_to != "CMY")

{

cerr << "Wrong color space";

return 1;

}

i += 2;

}

else if(s == "-i")

{

count\_in = stoi(argv[i + 1]);

input\_f = argv[i + 2];

if(count\_in != 1 && count\_in != 3)

{

cerr << "count\_ != 1 && count\_ != 3";

return 1;

}

i += 3;

}

else if(s == "-o")

{

count\_out = stoi(argv[i + 1]);

output\_f = argv[i + 2];

if(count\_out != 1 && count\_out != 3)

{

cerr << "count\_ != 1 && count\_ != 3";

return 1;

}

i += 3;

}

else

{

cerr << "Wrong commands -i, -o, -f, -t "<< argv[i];

return 1;

}

}

}

catch(...)

{

cerr << "Invalid arguments";

return 1;

}

}

else

{

cerr << "Invalid number of arguments";

return 1;

}

if (count\_in == 3)

{

for(int i = 1; i < 4; i++)

{

string name\_of\_file = input\_f;

if(i == 1)

name\_of\_file.insert( name\_of\_file.find\_last\_of("."), "\_1");

else if(i == 2)

name\_of\_file.insert( name\_of\_file.find\_last\_of("."), "\_2");

else

name\_of\_file.insert( name\_of\_file.find\_last\_of("."), "\_3");

char \*name = &name\_of\_file[0];

fin = fopen(name, "rb");

if (fin == NULL)

{

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

return 1;

}

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

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

{

fclose(fin);

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

return 1;

}

if(i == 1)

{

image1.resize(h \* w, 0);

quantity = fread(&image1[0], sizeof(unsigned char), image1.size(), fin);

}

else if(i == 2)

{

image2.resize(h \* w, 0);

quantity = fread(&image2[0], sizeof(unsigned char), image2.size(), fin);

}

else

{

image3.resize(h \* w, 0);

quantity = fread(&image3[0], sizeof(unsigned char), image3.size(), fin);

}

fclose(fin);

if (quantity != h \* w)

{

cerr << "Invalid pgm file";

return 1;

}

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

}

}

else

{

char \*name = &input\_f[0];

fin = fopen(name, "rb");

if (fin == NULL)

{

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

return 1;

}

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

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

{

fclose(fin);

cerr << "Invalid ppm file";

return 1;

}

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

image1.resize(h \* w, 0);

image2.resize(h \* w, 0);

image3.resize(h \* w, 0);

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

fclose(fin);

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

{

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

return 1;

}

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

{

image1[i / 3] = image[i];

image2[i / 3] = image[i + 1];

image3[i / 3] = image[i + 2];

}

}

if(color\_space\_from == "RGB")

{

}

else if(color\_space\_from == "HSL")

from\_HSL\_to\_RGB();

else if(color\_space\_from == "HSV")

from\_HSV\_to\_RGB();

else if(color\_space\_from == "YCbCr.601")

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

else if(color\_space\_from == "YCbCr.709")

from\_YCbCr\_709\_to\_RGB();

else if(color\_space\_from == "YCoCg")

from\_YCoCg\_to\_RGB();

else if(color\_space\_from == "CMY")

from\_CMY\_to\_RGB();

else

{

cerr << "Wrong color space";

return 1;

}

if(color\_space\_to == "RGB")

{

}

else if(color\_space\_to == "HSL")

from\_RGB\_to\_HSL();

else if(color\_space\_to == "HSV")

from\_RGB\_to\_HSV();

else if(color\_space\_to == "YCbCr.601")

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

else if(color\_space\_to == "YCbCr.709")

from\_RGB\_to\_YCbCr\_709();

else if(color\_space\_to == "YCoCg")

from\_RGB\_to\_YCoCg();

else if(color\_space\_to == "CMY")

from\_RGB\_to\_CMY();

else

{

cerr << "Wrong color space";

return 1;

}

if (count\_out == 3)

{

for(int i = 1; i < 4; i++)

{

string name\_of\_file = output\_f;

if(i == 1)

name\_of\_file.insert( name\_of\_file.find\_last\_of("."), "\_1");

else if(i == 2)

name\_of\_file.insert( name\_of\_file.find\_last\_of("."), "\_2");

else

name\_of\_file.insert( name\_of\_file.find\_last\_of("."), "\_3");

char \*name = &name\_of\_file[0];

fout = fopen(name, "wb");

if (fout == NULL)

{

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

return 1;

}

fprintf(fout, "%c%d\n%d %d\n%d\n", symbol, 5, w, h, max\_color);

int quantity;

if(i == 1)

quantity = fwrite(&image1[0], sizeof(unsigned char), image1.size(), fout);

else if(i == 2)

quantity = fwrite(&image2[0], sizeof(unsigned char), image2.size(), fout);

else

quantity = fwrite(&image3[0], sizeof(unsigned char), image3.size(), fout);

fclose(fout);

if (quantity != h \* w)

{

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

return 1;

}

}

}

else

{

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

{

image[i] = image1[i / 3];

image[i + 1] = image2[i / 3];

image[i + 2] = image3[i / 3];

}

char \*name = &output\_f[0];

fout = fopen(name, "wb");

if (fout == NULL)

{

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

return 1;

}

fprintf(fout, "%c%d\n%d %d\n%d\n", symbol, 6, w, h, max\_color);

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

fclose(fout);

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

{

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

return 1;

}

}

return 0;

}