МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

**Лабораторная работа № 4**

**По дисциплине «Компьютерная графика и геометрия»**

**Изучение цветовых пространств**

**Выполнил студент группы M3101  
*Дудко Матвей Владимирович***

**Проверил:  
Скаков Павел Сергеевич**

***САНКТ-ПЕТЕРБУРГ***

***2020***

# Цель работы

Реализация программы, которая позволяет проводить преобразования между цветовыми пространствами.

# Описание работы

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

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

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.

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

## Модель зрения человека

Представление и обработка графической информации в вычислительных системах основаны на наших знаниях о модели зрения человека.

Не только регистрация и отображение изображений стараются соответствовать системе зрения человека, но и алгоритмы кодирования и сжатия данных становятся намного эффективнее при учёте того, что видит и не видит человек.

Согласно современным представлениям, система зрения человека имеет 4 вида рецепторов:

* 3 вида “колбочек”: S (short), M (medium), L (long), отвечающих за цветное зрение. Работают только при высокой освещённости.
* 1 вид “палочек”: R (rods), позволяющих регистрировать яркость. Работают только при низкой освещённости.

Система цветного зрения человека трёхкомпонентная: воспринимаемый цвет описывается тремя значениями. Любые спектры излучения, приводящие к одинаковым этим трём значениям, неразличимы для человека.

Регистрация спектра L, M и S рецепторами лежит в основе цветовой модели RGB, описывающей цвет как комбинацию красного, синего и зелёного.

Однако, M и L рецепторы чувствительны далеко не только к чистым “зелёному” и “красному” цветам, а воспринимают довольно широкие спектры, которые ещё и значительно перекрываются. При непосредственном восприятии S, M, L значений было бы очень трудно различать красно-зелёные оттенки.

Но система зрения человека решила эту проблему тем, что производится “предварительная обработка”.

SML сигнал (что условно соответствует RGB) преобразуется следующим образом:

Y = S + M + L

A = L - M

B = (L + M) – S

То есть, представление красный-зелёный-синий превращается в яркость (Y) и две цветоразницы: красно-зелёную (A) и жёлто-синюю (B).В мозг передаётся обработанный сигнал: YAB. Кроме того, количество нейронов для компонент Y, A, B различна: о яркости передаётся гораздо больше информации, чем о цветоразностях.

Всё это послужило основой для различных цветоразностных систем представления цвета, например, YUV (альтернативное название: YCbCr), широко используемых при эффективном кодировании и сжатии графической информации.

## Общие сведения о цветовых пространствах

Цветовые пространства соответствуют различным системам представления информации о цвете.

Так как в соответствии с моделью зрения человека существует 3 вида рецепторов, отвечающих за цветное зрение, то и для кодирования информации о цвете разумно использовать трёхмерное цветовое пространство.

Переход от одного цветового пространства к другому можно представить себе как изменение базиса системы координат: значения меняются, но информация остаётся.

## Аддитивные и субтрактивные пространства

Цветовые пространства бывают аддитивные (например, RGB) и субтрактивные (например, CMY).

В аддитивных пространствах 0 соответствует чёрному цвету, а 100% всех компонент – белому. Это отражает работу источников света, например, отображение информации на мониторе.

В субтрактивных наоборот: отсутствие компонент – это белый, а полное присутствие – чёрный. Это соответствует смешению красок на бумаге.

## Цветовые пространства

### Пространство RGB

Пространство RGB – это самое широко используемое цветовое пространство. Его компоненты примерно соответствуют трём видам наших цветовых рецепторов: L, M, S.

R (Red) – красный

G (Green) – зелёный

B (Blue) – синий

Типичный диапазон значений: 0..255 для каждой компоненты, но возможны и другие значения, например, 0..1023 для 10-битных данных.

### Пространства HSL и HSV

Пространства HSL (другие названия: HLS, HSI) и HSV (другое название: HSB) широко используются в интерфейсах выбора цвета. Предназначены для “интуитивно понятного” изменения таких характеристик цвета как: оттенок, насыщенность, яркость.

H (Hue) – оттенок: диапазон 0..360°, 0..100 или 0..1

S (Saturation) – насыщенность: 0..100 или 0..1

L/I (Lightness/Intensity) – “светлота”: 0..100 или 0..1

V/B (Value/Brightness) – “яркость”: 0..100 или 0..1

### Пространство YUV / YCbCr

Пространство YUV (другое название: YCbCr) крайне широко используется для обработки и хранения графической и видео информации. Отдельные компоненты примерно соответствуют разложению нашей зрительной системой информации о цвете на яркость и две цветоразницы.

Y – яркость

U/Cb – цветоразность “хроматический синий”

V/Cr – цветоразность “хроматический красный”

В пространстве YUV традиционно существует два диапазона значений.

Для 8-битных данных:

|  |  |
| --- | --- |
| PC уровни | TV уровни |
| Y: 0..255  U: 0..255  V: 0..255 | Y: 16..235  U: 16..240  V: 16..240 |

При этом значения U и V – числа со знаком, закодированные в форме со смещением +128.

В данной лабораторной работе используется диапазон 0..255 (PC range)

### Пространство YCgCo

Пространство YCgCo – недавно разработанная альтернатива YCbCr. Те же принципы, но более простое преобразование в/из RGB.

Y – яркость

Cg – цветоразность “хроматический зелёный”

Co – цветоразность “хроматический оранжевый”

### Пространства CMY и CMYK

Пространства CMY и CMYK соответствуют устройству цветных принтеров. CMYK для улучшения эффективности использования красок добавляет компонент, соответствующий чёрной краске: без него получение широко востребованного чёрного требует смешивания всех трёх красок.

С (Cyan) – голубой

M (Magenta) – пурпурный

Y(Yellow) – жёлтый

K (blacK) – чёрный

# Экспериментальная часть

Лабораторная работа выполнена на языке C++. Стандарт языка C++14.

Для удобства все основные функции для работы с изображением были вынесены в отдельную библиотеку.

## Хранение данных изображения и заголовка при чтении из файла

### В случае чтения из одного файла

Данные заголовка файла хранятся в переменных:

char char\_header = 0;  
int width = 0, height = 0;  
unsigned int max\_value = 0;

Имя входного файла берется из аргументов командной строки. Чтение производится при помощи функции read\_header:

read\_header(char\_header, width, height, max\_value, file\_in);

Где:

* char\_header – номер версии файла (возможные варианты 5 или 6)
* width – ширина изображения
* height – высота изображения
* max\_value – максимально возможное значение яркости
* file\_in – входной файл

Общее количество байт для считывания вычисляется и записывается в переменную k\_bytes:

int k\_bytes = 3 \* height \* width;

Данные пикселей хранятся в одномерном динамическом массиве, выделяющемся в куче:

auto pix\_data = (unsigned char \*) calloc(k\_bytes, 1);

И считываются при помощи функции read\_data:

read\_data(file\_in, k\_bytes, pix\_data);

### В случае чтения из трех файлов

Данные заголовка хранятся и читаются аналогичным способом. Данные изображения хранятся также в переменной pix\_data.

Имя очередного файла формируется через переменную цикла:

for (int i = 1; i <= 3; ++i) {  
 std::string tmp\_file\_name(file\_in\_name);  
 std::string idx("\_");  
 idx.append(1, '0' + i);  
 tmp\_file\_name.insert(tmp\_file\_name.begin() + tmp\_file\_name.find\_last\_of('.'), idx.begin(), idx.end());

//...

}

Где: tmp\_file\_name – имя очередного файла.

После чтения заголовка производится чтение данных изображения:

При чтении первого файла производится выделение дополнительной памяти в куче:

auto tmp\_pix\_data = (unsigned char \*) calloc(k\_bytes, 1);

Затем само чтение:

read\_data(file\_in, k\_bytes, tmp\_pix\_data);

А затем эти данные добавляются в соответствующий канал изображения в переменную pix\_data при помощи функции add\_channel:

add\_channel(pix\_data, i - 1, k\_bytes, tmp\_pix\_data);

## Хранение данных нового изображения

Данные пикселей исходного изображения преобразуются “на месте” без выделения дополнительной памяти и хранятся в pix\_data.

auto pix\_data = (unsigned char \*) calloc(k\_bytes, 1);

## Алгоритмы преобразования между цветовыми пространствами

Для упрощения задачи были реализованы лишь переходы между цветовым пространством RGB и всеми остальными в обе стороны.

Переход между двумя различными пространствами реализован с помощью перехода изначально в RGB, а затем в нужное пространство.

Все переходы из одного цветового пространства в RGB (и наоборот) были названы соответственно. Расходы дополнительной памяти при переходах: O(1). Каждый переход проходит лишь 1 раз по всем данным. Сложность: O(n)

Преобразования:

// Conversions:  
// RGB <=> any  
  
if (strcmp(from\_color\_space, to\_color\_space) != 0) {  
  
 // Изначально преобразуем в RGB  
   
 if (strcmp(from\_color\_space, "HSL") == 0) {  
 hsl\_2\_rgb(pix\_data, k\_bytes);  
 } else if (strcmp(from\_color\_space, "HSV") == 0) {  
 hsv\_2\_rgb(pix\_data, k\_bytes);  
 } else if (strcmp(from\_color\_space, "YCbCr.601") == 0) {  
 YCbCr\_601\_2\_rgb(pix\_data, k\_bytes);  
 } else if (strcmp(from\_color\_space, "YCbCr.709") == 0) {  
 YCbCr\_709\_2\_rgb(pix\_data, k\_bytes);  
 } else if (strcmp(from\_color\_space, "YCoCg") == 0) {  
 YCoCg\_2\_rgb(pix\_data, k\_bytes);  
 } else if (strcmp(from\_color\_space, "CMY") == 0) {  
 cmy\_2\_rgb(pix\_data, k\_bytes);  
 }  
  
 // Затем в нужное пространство  
  
 if (strcmp(to\_color\_space, "HSL") == 0) {  
 rgb\_2\_hsl(pix\_data, k\_bytes);  
 } else if (strcmp(to\_color\_space, "HSV") == 0) {  
 rgb\_2\_hsv(pix\_data, k\_bytes);  
 } else if (strcmp(to\_color\_space, "YCbCr.601") == 0) {  
 rgb\_2\_YCbCr\_601(pix\_data, k\_bytes);  
 } else if (strcmp(to\_color\_space, "YCbCr.709") == 0) {  
 rgb\_2\_YCbCr\_709(pix\_data, k\_bytes);  
 } else if (strcmp(to\_color\_space, "YCoCg") == 0) {  
 rgb\_2\_YCoCg(pix\_data, k\_bytes);  
 } else if (strcmp(to\_color\_space, "CMY") == 0) {  
 rgb\_2\_cmy(pix\_data, k\_bytes);  
 }  
}

## Запись полученных данных в файлы

### При выводе в 1 файл

Используется функция записи color::write\_to\_file. Имя файла берется из аргументов командной строки

color::write\_to\_file(file\_out, width, height, max\_value, pix\_data);

### При выводе в 3 файла

Имя очередного файла формируется через переменную цикла:

for (int i = 1; i <= 3; ++i) {  
 std::string tmp\_file\_name(file\_out\_name);  
 std::string idx("\_");  
 idx.append(1, '0' + i);  
 tmp\_file\_name.insert(tmp\_file\_name.begin() + tmp\_file\_name.find\_last\_of('.'), idx.begin(), idx.end());

//...

}

tmp\_file\_name – имя очередного файла для вывода

Данные для записи файла записываются в дополнительно выделяемую память при помощи функции get\_separate\_channel:

auto tmp\_channel\_data = (unsigned char \*) calloc(k\_bytes / 3, 1);  
// ...

get\_separate\_channel(pix\_data, k\_bytes, i - 1, tmp\_channel\_data);

И записываются при помощи функции gray::write\_to\_file:

gray::write\_to\_file(file\_out, width, height, max\_value, tmp\_channel\_data);

# Вывод

Выполнение данной лабораторной работы позволило узнать о модели зрения человека, существовании различных цветовых пространств, а также об алгоритмах преобразования данных изображения для перехода из одного цветового пространства в другое. Для удобства все переходы между цветовыми пространствами производятся через пространство RGB. В ходе выполнения лабораторной работы были реализованы следующие алгоритмы:

|  |  |
| --- | --- |
| Преобразования из пространства RGB:   1. rgb\_2\_hsv 2. rgb\_2\_hsl 3. rgb\_2\_YCbCr\_601 4. rgb\_2\_YCbCr\_709 5. rgb\_2\_YCoCg 6. rgb\_2\_cmy | Преобразования в пространство RGB:   1. hsv\_2\_rgb 2. hsl\_2\_rgb 3. YCbCr\_601\_2\_rgb 4. YCbCr\_709\_2\_rgb 5. YCoCg\_2\_rgb 6. cmy\_2\_rgb |

# Листинг кода

Исходный код можно посмотреть на ресурсе GitHub по следующей ссылке: <https://github.com/DudkoMatt/GeometryAndGraphics/blob/master/Lab_04>

Содержание проекта:

./GeometryAndGraphics/Lab\_04/

main.cpp

write\_data\_pnm\_v2.cpp

write\_data\_pnm\_v2.h

Название файла: ./GeometryAndGraphics/Lab\_04/main.cpp

Исходный код:

#include <iostream>  
#include <cstring>  
#include <string>  
#include <algorithm>  
#include <cmath>  
#include "write\_data\_pnm\_v2.h"  
  
bool check\_color\_space(char \*color\_space) {  
 if (color\_space == nullptr)  
 return false;  
 return strcmp(color\_space, "RGB") == 0 ||  
 strcmp(color\_space, "HSL") == 0 ||  
 strcmp(color\_space, "HSV") == 0 ||  
 strcmp(color\_space, "YCbCr.601") == 0 ||  
 strcmp(color\_space, "YCbCr.709") == 0 ||  
 strcmp(color\_space, "YCoCg") == 0 ||  
 strcmp(color\_space, "CMY") == 0;  
}  
  
void add\_channel(unsigned char \*pix\_data, int k\_channel, int k\_bytes\_channel, const unsigned char \*channel) {  
 for (int i = 0; i < k\_bytes\_channel; ++i) {  
 \*(pix\_data + i \* 3 + k\_channel) = \*(channel + i);  
 }  
}  
  
void get\_separate\_channel(const unsigned char \*pix\_data, int all\_bytes, int k\_channel, unsigned char \*channel\_data) {  
 for (int i = 0; i < all\_bytes / 3; ++i) {  
 \*(channel\_data + i) = \*(pix\_data + 3 \* i + k\_channel);  
 }  
}  
  
void rgb\_2\_hsv(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double R = pix\_data[i] / 255.0;  
 double G = pix\_data[i + 1] / 255.0;  
 double B = pix\_data[i + 2] / 255.0;  
  
 double MAX = std::max(R, std::max(G, B));  
 double MIN = std::min(R, std::min(G, B));  
  
 double V = MAX;  
 double C = MAX - MIN;  
  
 // All in [0.0 .. 1.0]  
 // H in [0 .. 360]  
 double H;  
  
 // Calculating Hue  
 if (C == 0) {  
 H = 0;  
 } else if (V == R) {  
 H = (G - B) / C;  
 } else if (V == G) {  
 H = 2 + (B - R) / C;  
 } else {  
 // V == B  
 H = 4 + (R - G) / C;  
 }  
 H \*= 60;  
  
 if (H < 0)  
 H += 360;  
  
 double S\_v;  
 S\_v = V == 0 ? 0 : C / V;  
  
 // Transform to PC range: [0 .. 255]  
 // H:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(H / 360.0 \* 255));  
 // S:  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round(S\_v \* 255));  
 // V:  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round(V \* 255));  
 }  
}  
  
void rgb\_2\_hsl(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double R = pix\_data[i] / 255.0;  
 double G = pix\_data[i + 1] / 255.0;  
 double B = pix\_data[i + 2] / 255.0;  
  
 double MAX = std::max(R, std::max(G, B));  
 double MIN = std::min(R, std::min(G, B));  
  
 double V = MAX;  
 double C = MAX - MIN;  
 double L = (MAX + MIN) / 2.0;  
  
 // All in [0.0 .. 1.0]  
 // H in [0 .. 360]  
 double H;  
  
 // Calculating Hue  
 if (C == 0) {  
 H = 0;  
 } else if (V == R) {  
 H = (G - B) / C;  
 } else if (V == G) {  
 H = 2 + (B - R) / C;  
 } else {  
 // V == B  
 H = 4 + (R - G) / C;  
 }  
 H \*= 60;  
  
 if (H < 0)  
 H += 360;  
  
 double S\_l;  
 S\_l = L == 0 || L == 1 ? 0 : (V - L) / std::min(L, 1 - L);  
  
 // Transform to PC range: [0 .. 255]  
 // H:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round((H / 360.0 \* 255)));  
 // S:  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((S\_l \* 255)));  
 // V:  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((L \* 255)));  
 }  
}  
  
template <class T>  
bool in\_range(T left, T x, T right) {  
 return (left <= x) && (x <= right);  
}  
  
void hsv\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double H = pix\_data[i] / 255.0 \* 360;  
 double S\_v = pix\_data[i + 1] / 255.0;  
 double V = pix\_data[i + 2] / 255.0;  
  
 double C = V \* S\_v;  
 double \_H = H / 60;  
 double X = C \* (1 - std::abs(((int) \_H) % 2 + (\_H - (int) \_H) - 1));  
  
 double \_R, \_G, \_B;  
  
 if (in\_range(0.0, \_H, 1.0)) {  
 \_R = C;  
 \_G = X;  
 \_B = 0;  
 } else if (in\_range(1.0, \_H, 2.0)) {  
 \_R = X;  
 \_G = C;  
 \_B = 0;  
 } else if (in\_range(2.0, \_H, 3.0)) {  
 \_R = 0;  
 \_G = C;  
 \_B = X;  
 } else if (in\_range(3.0, \_H, 4.0)) {  
 \_R = 0;  
 \_G = X;  
 \_B = C;  
 } else if (in\_range(4.0, \_H, 5.0)) {  
 \_R = X;  
 \_G = 0;  
 \_B = C;  
 } else {  
 \_R = C;  
 \_G = 0;  
 \_B = X;  
 }  
  
 double m = V - C;  
  
 // To RGB:  
 // R:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round((\_R + m) \* 255));  
 // G  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((\_G + m) \* 255));  
 // B  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((\_B + m) \* 255));  
 }  
}  
  
void hsl\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double H = pix\_data[i] / 255.0 \* 360;  
 double S\_l = pix\_data[i + 1] / 255.0;  
 double L = pix\_data[i + 2] / 255.0;  
  
 double C = (1 - std::abs(2 \* L - 1)) \* S\_l;  
 double \_H = H / 60;  
 double X = C \* (1 - std::abs(((int) \_H) % 2 + (\_H - (int) \_H) - 1));  
  
 double \_R, \_G, \_B;  
  
 if (std::ceil(\_H) - 1.0 < 0.001) {  
 \_R = C;  
 \_G = X;  
 \_B = 0;  
 } else if (std::ceil(\_H) - 2.0 < 0.001) {  
 \_R = X;  
 \_G = C;  
 \_B = 0;  
 } else if (std::ceil(\_H) - 3.0 < 0.001) {  
 \_R = 0;  
 \_G = C;  
 \_B = X;  
 } else if (std::ceil(\_H) - 4.0 < 0.001) {  
 \_R = 0;  
 \_G = X;  
 \_B = C;  
 } else if (std::ceil(\_H) - 5.0 < 0.001) {  
 \_R = X;  
 \_G = 0;  
 \_B = C;  
 } else if (std::ceil(\_H) - 6.0 < 0.001) {  
 \_R = C;  
 \_G = 0;  
 \_B = X;  
 } else {  
 \_R = 0;  
 \_G = 0;  
 \_B = 0;  
 }  
  
 double m = L - C / 2.0;  
  
 // To RGB:  
 // R:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round((\_R + m) \* 255));  
 // G  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((\_G + m) \* 255));  
 // B  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((\_B + m) \* 255));  
 }  
}  
  
void rgb\_2\_YCbCr\_601(unsigned char \*pix\_data, int all\_bytes) {  
 double K\_b = 0.299;  
 double K\_r = 0.587;  
 double K\_g = 0.114;  
  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double R = pix\_data[i] / 255.0;  
 double G = pix\_data[i + 1] / 255.0;  
 double B = pix\_data[i + 2] / 255.0;  
  
 double Y = K\_r \* R + K\_g \* G + K\_b \* B;  
 double C\_b = (B - Y) / (2 \* (1 - K\_b));  
 double C\_r = (R - Y) / (2 \* (1 - K\_r));  
  
 // To YCbCr\_601:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(Y \* 255));  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((C\_b + 0.5) \* 255));  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((C\_r + 0.5) \* 255));  
 }  
}  
  
void YCbCr\_601\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 double K\_b = 0.299;  
 double K\_r = 0.587;  
 double K\_g = 0.114;  
  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double Y = pix\_data[i] / 255.0;  
 double C\_b = pix\_data[i + 1] / 255.0 - 0.5;  
 double C\_r = pix\_data[i + 2] / 255.0 - 0.5;  
  
 double R = Y + (2 - 2 \* K\_r) \* C\_r;  
 double G = Y - K\_b \* (2 - 2 \* K\_b) \* C\_b / K\_g - K\_r \* (2 - 2 \* K\_r) \* C\_r / K\_g;  
 double B = Y + (2 - 2 \* K\_b) \* C\_b;  
  
 // To RGB:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(R \* 255));  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round(G \* 255));  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round(B \* 255));  
 }  
}  
  
void rgb\_2\_YCbCr\_709(unsigned char \*pix\_data, int all\_bytes) {  
 double K\_b = 0.0722;  
 double K\_r = 0.2126;  
 double K\_g = 0.7152;  
  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double R = pix\_data[i] / 255.0;  
 double G = pix\_data[i + 1] / 255.0;  
 double B = pix\_data[i + 2] / 255.0;  
  
 double Y = K\_r \* R + K\_g \* G + K\_b \* B;  
 double C\_b = (B - Y) / (2 \* (1 - K\_b));  
 double C\_r = (R - Y) / (2 \* (1 - K\_r));  
  
 // To YCbCr\_709:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(Y \* 255));  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((C\_b + 0.5) \* 255));  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((C\_r + 0.5) \* 255));  
 }  
}  
  
void YCbCr\_709\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 double K\_b = 0.0722;  
 double K\_r = 0.2126;  
 double K\_g = 0.7152;  
  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double Y = pix\_data[i] / 255.0;  
 double C\_b = pix\_data[i + 1] / 255.0 - 0.5;  
 double C\_r = pix\_data[i + 2] / 255.0 - 0.5;  
  
 double R = Y + (2 - 2 \* K\_r) \* C\_r;  
 double G = Y - K\_b \* (2 - 2 \* K\_b) \* C\_b / K\_g - K\_r \* (2 - 2 \* K\_r) \* C\_r / K\_g;  
 double B = Y + (2 - 2 \* K\_b) \* C\_b;  
  
 // To RGB:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(R \* 255));  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round(G \* 255));  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round(B \* 255));  
 }  
}  
  
void rgb\_2\_YCoCg(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double R = pix\_data[i] / 255.0;  
 double G = pix\_data[i + 1] / 255.0;  
 double B = pix\_data[i + 2] / 255.0;  
  
 // Y  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(((R + B) / 4.0 + G / 2.0) \* 255));  
 // Co  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round(((R - B) / 2.0 + 0.5) \* 255));  
 // Cg  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((G / 2.0 - (B + R) / 4.0 + 0.5) \* 255));  
 }  
}  
  
void YCoCg\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double Y = pix\_data[i] / 255.0;  
 double Co = pix\_data[i + 1] / 255.0 - 0.5;  
 double Cg = pix\_data[i + 2] / 255.0 - 0.5;  
  
 // To RGB:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round((Y - Cg + Co) \* 255));  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((Y + Cg) \* 255));  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((Y - Cg - Co) \* 255));  
 }  
}  
  
void rgb\_2\_cmy(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; ++i) {  
 \*(pix\_data + i) = 255 - \*(pix\_data + i);  
 }  
}  
  
void cmy\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 rgb\_2\_cmy(pix\_data, all\_bytes);  
}  
  
int main(int argc, char \*argv[]) {  
  
 // Часть 1: разбор аргументов командной строки  
  
 if (argc != 11) {  
 std::cerr << "Wrong number of arguments. Syntax:\n<lab4>.exe -f <from\_color\_space> -t <to\_color\_space> -i <count> <input\_file\_name> -o <count> <output\_file\_name>\n";  
 return 1;  
 }  
  
 char \_f = 0, \_t = 0, \_i = 0, \_o = 0;  
 char \*from\_color\_space = nullptr, \*to\_color\_space = nullptr;  
 char \*file\_in\_name = nullptr, \*file\_out\_name = nullptr;  
 int i\_count = 0, o\_count = 0;  
  
 for (int i = 1; i < argc; ++i) {  
 if (strlen(argv[i]) == 2) {  
 if (strncmp(argv[i], "-f", 2) == 0) {  
 \_f = 1;  
 from\_color\_space = argv[++i];  
 } else if (strncmp(argv[i], "-t", 2) == 0) {  
 \_t = 1;  
 to\_color\_space = argv[++i];  
 } else if (strncmp(argv[i], "-i", 2) == 0) {  
 \_i = 1;  
 i\_count = std::stoi(argv[++i]);  
 file\_in\_name = argv[++i];  
 } else if (strncmp(argv[i], "-o", 2) == 0) {  
 \_o = 1;  
 o\_count = std::stoi(argv[++i]);  
 file\_out\_name = argv[++i];  
 } else {  
 fprintf(stderr, "Invalid argument: %s\n", argv[i]);  
 return 1;  
 }  
 } else {  
 fprintf(stderr, "Invalid argument: %s\n", argv[i]);  
 return 1;  
 }  
 }  
   
 if (\_f + \_t + \_i + \_o < 4 || (i\_count != 1 && i\_count != 3) || (o\_count != 1 && o\_count != 3)  
 || !check\_color\_space(from\_color\_space) || !check\_color\_space(to\_color\_space)  
 || file\_in\_name == nullptr || file\_out\_name == nullptr) {  
 std::cerr << "Wrong arguments\n";  
 return 1;  
 }  
  
 // Часть 2: проверка на существование файла и чтение файла на ввод  
  
 unsigned char \*pix\_data = nullptr;  
 FILE \*file\_in = nullptr, \*file\_out = nullptr;  
  
 char char\_header = 0;  
 int width = 0, height = 0;  
 unsigned int max\_value = 0;  
 int k\_bytes = 0;  
  
 if (i\_count == 1) {  
  
 file\_in = fopen(file\_in\_name, "rb");  
 if (file\_in == nullptr) {  
 std::cerr << "Cannot open file to read: " << file\_in\_name << "\n";  
 return 1;  
 }  
  
 read\_header(char\_header, width, height, max\_value, file\_in);  
 if (char\_header != '6') {  
 std::cerr << "Expected PPM file format\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 k\_bytes = 3 \* width \* height;  
  
 pix\_data = (unsigned char \*) calloc(k\_bytes, 1);  
 if (!pix\_data) {  
 std::cerr << "Cannot allocate " << k\_bytes << " bytes of memory\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 read\_data(file\_in, k\_bytes, pix\_data);  
  
 fclose(file\_in);  
 file\_in = nullptr;  
  
 } else {  
 unsigned char \*tmp\_pix\_data = nullptr;  
 int prev\_k\_bytes = -1;  
  
 for (int i = 1; i <= 3; ++i) {  
 std::string tmp\_file\_name(file\_in\_name);  
 std::string idx("\_");  
 idx.append(1, '0' + i);  
 tmp\_file\_name.insert(tmp\_file\_name.begin() + tmp\_file\_name.find\_last\_of('.'), idx.begin(), idx.end());  
  
 file\_in = fopen(tmp\_file\_name.data(), "rb");  
  
 if (file\_in == nullptr) {  
 std::cerr << "Cannot open file to read: " << tmp\_file\_name << "\n";  
 return 1;  
 }  
  
 read\_header(char\_header, width, height, max\_value, file\_in);  
 if (char\_header != '5') {  
 std::cerr << "Expected PGM file format\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 k\_bytes = width \* height;  
  
 if (k\_bytes != prev\_k\_bytes && prev\_k\_bytes != -1) {  
 std::cerr << "Different amount of data in files\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 if (tmp\_pix\_data)  
 free(tmp\_pix\_data);  
 return 1;  
 } else {  
 prev\_k\_bytes = k\_bytes;  
 }  
   
 if (i == 1) {  
 pix\_data = (unsigned char \*) calloc(3 \* k\_bytes, 1);  
 if (!pix\_data) {  
 std::cerr << "Cannot allocate " << 3 \* k\_bytes << " bytes of memory\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
   
 // Initialization  
 for (int \_\_i = 0; \_\_i < k\_bytes; ++\_\_i) {  
 pix\_data[\_\_i] = 0;  
 }  
  
 tmp\_pix\_data = (unsigned char \*) calloc(k\_bytes, 1);  
 if (!tmp\_pix\_data) {  
 std::cerr << "Cannot allocate " << k\_bytes << " bytes of memory\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
 }  
  
 for (int \_\_i = 0; \_\_i < k\_bytes; ++\_\_i) {  
 \*(tmp\_pix\_data + i) = 0;  
 }  
  
 read\_data(file\_in, k\_bytes, tmp\_pix\_data);  
 add\_channel(pix\_data, i - 1, k\_bytes, tmp\_pix\_data);  
  
 fclose(file\_in);  
 file\_in = nullptr;  
 }  
  
 if (tmp\_pix\_data)  
 free(tmp\_pix\_data);  
 k\_bytes \*= 3;  
 }  
  
  
 // Часть 3: преобразования  
  
 // Conversions:  
 // RGB <=> any  
  
 if (strcmp(from\_color\_space, to\_color\_space) != 0) {  
  
 // Изначально преобразуем в RGB  
   
 if (strcmp(from\_color\_space, "HSL") == 0) {  
 hsl\_2\_rgb(pix\_data, k\_bytes);  
 } else if (strcmp(from\_color\_space, "HSV") == 0) {  
 hsv\_2\_rgb(pix\_data, k\_bytes);  
 } else if (strcmp(from\_color\_space, "YCbCr.601") == 0) {  
 YCbCr\_601\_2\_rgb(pix\_data, k\_bytes);  
 } else if (strcmp(from\_color\_space, "YCbCr.709") == 0) {  
 YCbCr\_709\_2\_rgb(pix\_data, k\_bytes);  
 } else if (strcmp(from\_color\_space, "YCoCg") == 0) {  
 YCoCg\_2\_rgb(pix\_data, k\_bytes);  
 } else if (strcmp(from\_color\_space, "CMY") == 0) {  
 cmy\_2\_rgb(pix\_data, k\_bytes);  
 }  
  
 // Затем в нужное пространство  
  
 if (strcmp(to\_color\_space, "HSL") == 0) {  
 rgb\_2\_hsl(pix\_data, k\_bytes);  
 } else if (strcmp(to\_color\_space, "HSV") == 0) {  
 rgb\_2\_hsv(pix\_data, k\_bytes);  
 } else if (strcmp(to\_color\_space, "YCbCr.601") == 0) {  
 rgb\_2\_YCbCr\_601(pix\_data, k\_bytes);  
 } else if (strcmp(to\_color\_space, "YCbCr.709") == 0) {  
 rgb\_2\_YCbCr\_709(pix\_data, k\_bytes);  
 } else if (strcmp(to\_color\_space, "YCoCg") == 0) {  
 rgb\_2\_YCoCg(pix\_data, k\_bytes);  
 } else if (strcmp(to\_color\_space, "CMY") == 0) {  
 rgb\_2\_cmy(pix\_data, k\_bytes);  
 }  
 }  
  
 // Часть 4: вывод в файл(-ы)  
  
 if (o\_count == 1) {  
  
 file\_out = fopen(file\_out\_name, "wb");  
 if (file\_out == nullptr) {  
 std::cerr << "Cannot open file to write: " << file\_out\_name << "\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 color::write\_to\_file(file\_out, width, height, max\_value, pix\_data);  
 fclose(file\_out);  
  
 } else {  
 unsigned char \*tmp\_channel\_data = nullptr;  
  
 for (int i = 1; i <= 3; ++i) {  
 std::string tmp\_file\_name(file\_out\_name);  
 std::string idx("\_");  
 idx.append(1, '0' + i);  
 tmp\_file\_name.insert(tmp\_file\_name.begin() + tmp\_file\_name.find\_last\_of('.'), idx.begin(), idx.end());  
  
 file\_out = fopen(tmp\_file\_name.data(), "wb");  
 if (file\_out == nullptr) {  
 std::cerr << "Cannot open file to write: " << tmp\_file\_name << "\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 if (tmp\_channel\_data == nullptr) {  
 tmp\_channel\_data = (unsigned char \*) calloc(k\_bytes / 3, 1);  
 if (tmp\_channel\_data == nullptr) {  
 std::cerr << "Cannot allocate " << k\_bytes / 3 << " bytes of memory\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
 }  
  
 for (int j = 0; j < k\_bytes / 3; ++j) {  
 \*(tmp\_channel\_data + i) = 0;  
 }  
  
 get\_separate\_channel(pix\_data, k\_bytes, i - 1, tmp\_channel\_data);  
 gray::write\_to\_file(file\_out, width, height, max\_value, tmp\_channel\_data);  
 fclose(file\_out);  
 file\_out = nullptr;  
 }  
  
 if (tmp\_channel\_data)  
 free(tmp\_channel\_data);  
 }  
  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
}

Название файла: ./GeometryAndGraphics/Lab\_04/write\_data\_pnm\_v2.cpp

Исходный код:

//  
// Created by dudko on 11.03.2020.  
//  
  
#include "write\_data\_pnm\_v2.h"  
#include <cmath>  
  
void write\_header(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value) {  
 fseek(file\_out, 0, SEEK\_SET);  
 fprintf(file\_out, "P%c\n%i %i\n%i\n", char\_header, width, height, max\_value);  
}  
  
void write\_data(FILE \*file\_out, int k\_bytes, unsigned char \*pix\_data) {  
 fwrite(pix\_data, k\_bytes, 1, file\_out);  
}  
  
int read\_header(char &char\_header, int &width, int &height, unsigned int &max\_value, FILE \*file\_in) {  
 int scanned = fscanf(file\_in, "P%c\n%i %i\n%i\n", &char\_header, &width, &height, &max\_value);  
 if (scanned != 4)  
 return 1;  
 if (width <= 0 || height <= 0 || max\_value <= 0 || max\_value > 255 || (char\_header != '5' && char\_header != '6'))  
 return 2;  
 return 0;  
}  
  
void read\_data(FILE \*file\_in, int k\_bytes, unsigned char \*pix\_data) {  
 if (!pix\_data)  
 return;  
 fread(pix\_data, 1, k\_bytes, file\_in);  
}  
  
void free\_data(FILE \*file\_in, FILE \*file\_out, unsigned char \*pix\_data) {  
 if (file\_in) fclose(file\_in);  
 if (file\_out) fclose(file\_out);  
 if (pix\_data) free(pix\_data);  
}  
  
void free\_data(FILE \*file, unsigned char \*pix\_data) {  
 free\_data(nullptr, file, pix\_data);  
}  
  
void free\_data(FILE \*file) {  
 free\_data(file, nullptr);  
}  
  
void free\_data(unsigned char \*pix\_data) {  
 free\_data(nullptr, pix\_data);  
}  
  
namespace color {  
 void write\_header(FILE \*file\_out, int width, int height, unsigned int max\_value) {  
 fseek(file\_out, 0, SEEK\_SET);  
 fprintf(file\_out, "P%c\n%i %i\n%i\n", '6', width, height, max\_value);  
 }  
  
 void write\_to\_file(FILE \*file\_out, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data) {  
 color::write\_header(file\_out, width, height, max\_value);  
 write\_data(file\_out, 3 \* width \* height, pix\_data);  
 }  
}  
  
namespace gray {  
 void write\_header(FILE \*file\_out, int width, int height, unsigned int max\_value) {  
 fseek(file\_out, 0, SEEK\_SET);  
 fprintf(file\_out, "P%c\n%i %i\n%i\n", '5', width, height, max\_value);  
 }  
  
 void write\_to\_file(FILE \*file\_out, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data) {  
 gray::write\_header(file\_out, width, height, max\_value);  
 write\_data(file\_out, width \* height, pix\_data);  
 }  
}  
  
void write\_to\_file(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data) {  
 write\_header(file\_out, char\_header, width, height, max\_value);  
 write\_data(file\_out, width \* height, pix\_data);  
}  
  
// brightness in [0..255] scale  
unsigned char limit\_brightness(double brightness) {  
 return (unsigned char) std::min(255.0, std::max(0.0, brightness));  
}  
  
double from\_sRGB(double \_brightness) {  
 if (\_brightness <= 0.0031308) {  
 return 323.0 \* \_brightness / 25.0;  
 } else {  
 return (211 \* pow(\_brightness, 5.0 / 12.0) - 11) / 200.0;  
 }  
}  
  
double from\_sRGB(int brightness) {  
 double \_brightness = brightness / 255.0;  
 return from\_sRGB(\_brightness);  
}  
  
double to\_sRGB(double \_brightness) {  
 if (\_brightness <= 0.04045) {  
 return 25.0 \* \_brightness / 323;  
 } else {  
 return pow((200 \* \_brightness + 11) / 211.0, 12.0 / 5.0);  
 }  
}  
  
double to\_sRGB(int brightness) {  
 double \_brightness = brightness / 255.0;  
 return to\_sRGB(\_brightness);  
}  
  
double change\_pix\_gamma\_to\_print(double \_brightness, double gamma) {  
 // Гамма коррекция:  
 if (gamma > 0) {  
 \_brightness = std::pow(\_brightness, 1.0 / gamma);  
 } else {  
 \_brightness = from\_sRGB(\_brightness);  
 }  
  
 return \_brightness;  
}  
  
double change\_pix\_gamma\_to\_print(unsigned char pix\_data, double gamma) {  
 double \_brightness = pix\_data / 255.0;  
 return change\_pix\_gamma\_to\_print(\_brightness, gamma);  
}  
  
double change\_pix\_gamma\_from\_file(double \_brightness, double gamma) {  
 if (gamma > 0) {  
 \_brightness = std::pow(\_brightness, gamma);  
 } else {  
 \_brightness = to\_sRGB(\_brightness);  
 }  
  
 return \_brightness;  
}  
  
double change\_pix\_gamma\_from\_file(unsigned char pix\_data, double gamma) {  
 double \_brightness = pix\_data / 255.0;  
 return change\_pix\_gamma\_from\_file(\_brightness, gamma);  
}  
  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, int brightness, double gamma) {  
 double \_brightness = brightness / 255.0;  
  
 // Гамма коррекция:  
 change\_pix\_gamma\_to\_print(\_brightness, gamma);  
  
 pix\_data[width \* y + x] = limit\_brightness(255 \* \_brightness);  
}  
  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, double \_brightness, double gamma) {  
 change\_pix\_gamma\_to\_print(\_brightness, gamma);  
 pix\_data[width \* y + x] = limit\_brightness(255 \* \_brightness);  
}  
  
void decode\_gamma\_from\_file(unsigned char \*pix\_data, int k, double gamma) {  
 for (int i = 0; i < k; ++i) {  
 \*(pix\_data + i) = limit\_brightness(255 \* change\_pix\_gamma\_from\_file(\*(pix\_data + i), gamma));  
 }  
}

Название файла: ./GeometryAndGraphics/Lab\_04/write\_data\_pnm\_v2.h

Исходный код:

//  
// Created by dudko on 11.03.2020.  
//  
  
#ifndef LAB\_03\_WRITE\_DATA\_PNM\_H  
#define LAB\_03\_WRITE\_DATA\_PNM\_H  
  
#include <iostream>  
  
void write\_header(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value);  
void write\_data(FILE \*file\_out, int k\_bytes, unsigned char \*pix\_data);  
int read\_header(char &char\_header, int &width, int &height, unsigned int &max\_value, FILE \*file\_in);  
void read\_data(FILE \*file\_in, int k\_bytes, unsigned char \*pix\_data);  
void free\_data(FILE \*file\_in, FILE \*file\_out, unsigned char \*pix\_data);  
void free\_data(FILE \*file, unsigned char \*pix\_data);  
void free\_data(FILE \*file);  
void free\_data(unsigned char \*pix\_data);  
namespace color {  
 void write\_header(FILE \*file\_out, int width, int height, unsigned int max\_value);  
 void write\_to\_file(FILE \*file\_out, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data);  
}  
  
namespace gray {  
 void write\_header(FILE \*file\_out, int width, int height, unsigned int max\_value);  
 void write\_to\_file(FILE \*file\_out, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data);  
}  
  
void write\_to\_file(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data);  
  
unsigned char limit\_brightness(double brightness);  
  
double to\_sRGB(double \_brightness);  
double to\_sRGB(int brightness);  
double from\_sRGB(double \_brightness);  
double from\_sRGB(int brightness);  
  
double change\_pix\_gamma\_to\_print(double \_brightness, double gamma);  
double change\_pix\_gamma\_to\_print(unsigned char pix\_data, double gamma);  
  
double change\_pix\_gamma\_from\_file(double \_brightness, double gamma);  
double change\_pix\_gamma\_from\_file(unsigned char pix\_data, double gamma);  
  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, int brightness, double gamma);  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, double \_brightness, double gamma);  
  
void decode\_gamma\_from\_file(unsigned char \*pix\_data, int k, double gamma);  
  
#endif //LAB\_03\_WRITE\_DATA\_PNM\_H