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. М3101
Шмарина Л.С.
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

Описание:
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:
C: 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

Перевод из RGB в HSL:

$$H = \begin{cases} 0 & \text{if } MAX = MIN \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 0^\circ, & \text{if } MAX = R \text{ and } G \geq B \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 360^\circ, & \text{if } MAX = R \text{ and } G < B \\ 60^\circ \times \frac{B-R}{MAX-MIN} + 120^\circ, & \text{if } MAX = G \\ 60^\circ \times \frac{R-G}{MAX-MIN} + 240^\circ, & \text{if } MAX = B \end{cases}$$

$$S = \frac{MAX - MIN}{1 - |1 - (MAX + MIN)|}$$

$$L = \frac{1}{2}(MAX + MIN)$$

Перевод из HSL в RGB:

$$C = (1 - |2L - 1|) \times S_L$$

$$H' = \frac{H}{60^\circ}$$

$$X = C \cdot (1 - |H' \mod 2 - 1|)$$

$$(R_1, G_1, B_1) = \begin{cases} (C, X, 0) & \text{if } \lceil H' \rceil = 1 \\ (X, C, 0) & \text{if } \lceil H' \rceil = 2 \\ (0, C, X) & \text{if } \lceil H' \rceil = 3 \\ (0, X, C) & \text{if } \lceil H' \rceil = 4 \\ (X, 0, C) & \text{if } \lceil H' \rceil = 5 \\ (C, 0, X) & \text{if } \lceil H' \rceil = 6 \\ (0, 0, 0) & \text{otherwise} \end{cases}$$

$$m = L - \frac{C}{2}$$

$$(R, G, B) = (R_1 + m, G_1 + m, B_1 + m)$$

Перевод из RGB в HSV

$$H = \begin{cases} 60 \times \dfrac{G-B}{MAX-MIN} + 0, \text{ if } MAX = R \text{ and } G \geq B \\ 60 \times \dfrac{G-B}{MAX-MIN} + 360, \text{ if } MAX = R \text{ and } G < B \\ 60 \times \dfrac{B-R}{MAX-MIN} + 120, \text{ if } MAX = G \\ 60 \times \dfrac{R-G}{MAX-MIN} + 240, \text{ if } MAX = B \end{cases}$$

$$S = \begin{cases} 0, \text{ if } MAX = 0; \\ 1 - \dfrac{MIN}{MAX}, \text{else} \end{cases}$$

$$V = MAX$$

Перевод из HSV в RGB

$$C = V \times S_V$$

$$H' = \frac{H}{60^\circ}$$

$$X = C \times (1 - |H' \bmod 2 - 1|)$$

$$(R_1, G_1, B_1) = \begin{cases} (0,0,0) & \text{if } H \text{ equals } 0 \\ (C,X,0) & \text{if } 0 < H' \leq 1 \\ (X,C,0) & \text{if } 1 < H' \leq 2 \\ (0,C,X) & \text{if } 2 < H' \leq 3 \\ (0,X,C) & \text{if } 3 < H' \leq 4 \\ (X,0,C) & \text{if } 4 < H' \leq 5 \\ (C,0,X) & \text{if } 5 < H' \leq 6 \end{cases}$$

$$m = V - C$$

$$(R, G, B) = (R_1 + m, G_1 + m, B_1 + m)$$

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.

Перевод из RGB в YCbCr.601

$$K_R = 0.299$$
$$K_G = 0.587$$
$$K_B = 0.114$$

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2-2\cdot K_R \\ 1 & -\frac{K_B}{K_G}\cdot(2-2\cdot K_B) & -\frac{K_R}{K_G}\cdot(2-2\cdot K_R) \\ 1 & 2-2\cdot K_B & 0 \end{bmatrix} \begin{bmatrix} Y' \\ P_B \\ P_R \end{bmatrix}$$

Перевод из YCbCr.601 в RGB

$$K_R = 0.299$$
$$K_G = 0.587$$
$$K_B = 0.114$$

$$\begin{bmatrix} Y' \\ P_B \\ P_R \end{bmatrix} = \begin{bmatrix} K_R & K_G & K_B \\ -\frac{1}{2}\cdot\frac{K_R}{1-K_B} & -\frac{1}{2}\cdot\frac{K_G}{1-K_B} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2}\cdot\frac{K_G}{1-K_R} & -\frac{1}{2}\cdot\frac{K_B}{1-K_R} \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

Перевод из RGB в YCbCr.709

$$K_B = 0.0722$$
$$K_R = 0.2126$$
$$K_G = 0.7152$$

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2-2\cdot K_R \\ 1 & -\frac{K_B}{K_G}\cdot(2-2\cdot K_B) & -\frac{K_R}{K_G}\cdot(2-2\cdot K_R) \\ 1 & 2-2\cdot K_B & 0 \end{bmatrix} \begin{bmatrix} Y' \\ P_B \\ P_R \end{bmatrix}$$

Перевод из YCbCr.709 в RGB

$$K_B = 0.0722$$
$$K_R = 0.2126$$
$$K_G = 0.7152$$

$$\begin{bmatrix} Y' \\ P_B \\ P_R \end{bmatrix} = \begin{bmatrix} K_R & K_G & K_B \\ -\frac{1}{2} \cdot \frac{K_R}{1-K_B} & -\frac{1}{2} \cdot \frac{K_G}{1-K_B} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \cdot \frac{K_G}{1-K_R} & -\frac{1}{2} \cdot \frac{K_B}{1-K_R} \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

Перевод из RGB в YCgCo

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 0 & 1 \\ 1 & -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} Y \\ Co \\ Cg \end{bmatrix}$$

Перевод из YCgCo в RGB

$$\begin{bmatrix} Y \\ Co \\ Cg \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 0 & -\frac{1}{2} \\ -\frac{1}{4} & \frac{1}{2} & -\frac{1}{4} \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Перевод из RGB в CMY

```
C = 255 - R
M = 255 - G
Y = 255 - B
```

Перевод из CMY в RGB

```
R = 255 - C
G = 255 - M
B = 255 - Y
```

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

Экспериментальная часть

Для работы с изображением использовались класс для хранения изображения PPM, для хранения одного пикселя -- класс pixel. Алгоритмы реализованы через класс Converter, являющийся интерфейсом, через который реализована инверсия зависимостей, вследствие чего код является расширяемым. Каждое преобразование является отдельным классом, наследующим интерфейс. Для введения дополнительных преобразований достаточно дописать новый класс и переопределить в нем метод convert, который принимает в качестве аргумента пиксель.

Выводы

В ходе работы была разработана программа, которая переводит изображение между цветовыми пространствами. Помимо изучения алгоритмов перевода между цветовыми пространствами был изучен принцип инверсии зависимости, что позволило сделать код более читаемым и более расширяемым. При неправильной реализации преобразований полученные изображения при интерпретации их в цветовом пространстве RGB наблюдались сильные искажения изображений, что позволяло определять корректность работы алгоритмов.

Листинг

main.cpp

```
#include <string>
#include <algorithm>
#include <cstring>
#include <iostream>
#include <set>
#include "PPM.h"
#include "converters/BaseConverter.h"
```

```
#include "converters/fromRGBtoHSL.h"
#include "converters/fromRGBtoHSV.h"
#include "converters/fromRGBtoYCbCr601.h"
#include "converters/fromRGBtoYCbCr709.h"
#include "converters/fromRGBtoCMY.h"
#include "converters/fromRGBtoYCoCg.h"

#include "converters/fromCMYtoRGB.h"
#include "converters/fromHSLtoRGB.h"
#include "converters/fromHSVtoRGB.h"
#include "converters/fromYCbCr601toRGB.h"
#include "converters/fromYCbCr709toRGB.h"
#include "converters/fromYCoCgtoRGB.h"



char **getCmdOption(char **begin, char **end, const
std::string &option, int count_options) {
   char **itr = std::find(begin, end, option);
   char **answer = new char *[count_options];
   bool flag = false;
   for (int i = 0; i < count_options; i++) {
       if (itr != end) {
           itr++;
           if (strcmp(*itr, "-f") == 0 || strcmp(*itr,
"-t") == 0 || strcmp(*itr, "-i") == 0 ||
               strcmp(*itr, "-o") == 0) {
               flag = true;
               break;
```

```
            }
            answer[i] = *itr;
        } else {
            flag = true;
            break;
        }
    }
    if (flag) {
        delete[]answer;
        return nullptr;
    }

    return answer;
}

bool check_if_color_space(char *color_space) {
    std::set<char *> st;
    st.insert("RGB");
    st.insert("HSL");
    st.insert("HSV");
    st.insert("YCbCr.601");
    st.insert("YCbCr.709");
    st.insert("YCoCg");
    st.insert("CMY");
    for (const char *i : st) {
        if ((int) strcmp(color_space, i) == 0)
            return true;
```

```
    }
    return false;
}

char **parse_format(char *inp) {
    int lastdot = -1;
    int count_symbols = 0;
    for (int i = 0; inp[i] != 0; i++) {
        if (inp[i] == '.') {
            lastdot = i;
        }
        count_symbols++;
    }
    if (lastdot == -1)
        return nullptr;
    char **answer = new char *[3];
    for (int l = 0; l < 3; l++) {
        answer[l] = new char[count_symbols + 3];
        for (int i = 0; i < lastdot; i++) {
            answer[l][i] = inp[i];
        }
        answer[l][lastdot] = '_';
        answer[l][lastdot + 1] = '1' + l;
        for (int i = lastdot; i < count_symbols; i++) {
            answer[l][i + 2] = inp[i];
        }
        answer[l][count_symbols + 2] = 0;
```

```
    }
    return answer;
}

Converter *getConverterByName(bool isfrom, char *name)
{

    if (strcmp("RGB", name) == 0)
        return new BaseConverter();
    if (isfrom) { //from...toRGB
        if (strcmp("HSL", name) == 0)
            return new fromHSLtoRGB();
        if (strcmp("HSV", name) == 0)
            return new fromHSVtoRGB();
        if (strcmp("YCbCr.601", name) == 0)
            return new fromYCbCr601toRGB();
        if (strcmp("YCbCr.709", name) == 0)
            return new fromYCbCr709toRGB();
        if (strcmp("YCoCg", name) == 0)
            return new fromYCoCgtoRGB();
        if (strcmp("CMY", name) == 0)
            return new fromCMYtoRGB();
    } else { //fromRGBto...
        if (strcmp("HSL", name) == 0)
            return new fromRGBtoHSL();
        if (strcmp("HSV", name) == 0)
            return new fromRGBtoHSV();
        if (strcmp("YCbCr.601", name) == 0)
```

```
            return new fromRGBtoYCbCr601();
        if (strcmp("YCbCr.709", name) == 0)
            return new fromRGBtoYCbCr709();
        if (strcmp("YCoCg", name) == 0)
            return new fromRGBtoYCoCg();
        if (strcmp("CMY", name) == 0)
            return new fromRGBtoCMY();
    }
    return nullptr;
}

int main(int argc, char **argv) {
    try {
        char **to_space_name = getCmdOption(argv, argv +
argc, "-t", 1);
        if (to_space_name == nullptr) {
            throw std::invalid_argument("Кривая опция
-t");
        }
        if (!check_if_color_space(to_space_name[0])) {
            delete[]to_space_name;
            throw std::invalid_argument("Ты написал
невалидный colorspace");
        }
        char **from_color_space = getCmdOption(argv,
argv + argc, "-f", 1);
        if (from_color_space == nullptr) {
            delete[]to_space_name;
```

```
            throw std::invalid_argument("Кривая опция
-f");
        }
        if (!check_if_color_space(from_color_space[0]))
{
            delete[]to_space_name;
            delete[]from_color_space;
            throw std::invalid_argument("Ты написал
невалидный colorspace");
        }
        char **input_file_name = getCmdOption(argv, argv
+ argc, "-i", 2);
        if (input_file_name == nullptr) {
            delete[]to_space_name;
            delete[]from_color_space;
            throw std::invalid_argument("Кривая опция
-i");
        }
        char in_count = input_file_name[0][0];
        char **inputfilenames;
        if (in_count == '1') {
            inputfilenames = new char *[1];
            inputfilenames[0] = input_file_name[1];
        } else if (in_count == '3') {
            inputfilenames =
parse_format(input_file_name[1]);
            if (inputfilenames == nullptr) {
                delete[]to_space_name;
                delete[]from_color_space;
```

```
                delete[]input_file_name;
                throw std::invalid_argument("В пути к
файлу нет расширения");
            }
        } else {
            delete[]to_space_name;
            delete[]from_color_space;
            delete[]input_file_name;
            throw std::invalid_argument("Кривой count");
        }
        delete[] input_file_name;
        input_file_name = inputfilenames;



        char **output_file_name = getCmdOption(argv,
argv + argc, "-o", 2);
        if (output_file_name == nullptr) {
            delete[]to_space_name;
            delete[]from_color_space;
            delete[]input_file_name;
            throw std::invalid_argument("Кривая опция
-o");
        }
        char out_count = output_file_name[0][0];
        char **outputfilenames;
        if (out_count == '1') {
            outputfilenames = new char *[1];
            outputfilenames[0] = output_file_name[1];
```

```
        } else if (out_count == '3') {
            outputfilenames =
parse_format(output_file_name[1]);
            if (outputfilenames == nullptr) {
                delete[]to_space_name;
                delete[]from_color_space;
                delete[]output_file_name;
                throw std::invalid_argument("В пути к
файлу нет расширения");
            }
        } else {
            delete[]to_space_name;
            delete[]from_color_space;
            delete[]output_file_name;
            throw std::invalid_argument("Кривой count");
        }
        delete[] output_file_name;
        output_file_name = outputfilenames;

        PPM *picture = nullptr;
        ifstream file1;
        file1.unsetf(ios_base::skipws);
        ifstream file2;
        file2.unsetf(ios_base::skipws);
        ifstream file3;
        file3.unsetf(ios_base::skipws);
        if (in_count == '1') {
```

```
            file1.open(input_file_name[0],
std::ios::binary);
            if (!file1.is_open()) {
                delete[]to_space_name;
                delete[]from_color_space;
                delete[]input_file_name;
                delete[]output_file_name;
                delete picture;
                if (file1.is_open())
                    file1.close();
                throw std::invalid_argument("Cannot open
input file");
            }
            picture = new PPM(&file1);
        } else {
            file1.open(input_file_name[0],
std::ios::binary);
            file2.open(input_file_name[1],
std::ios::binary);
            file3.open(input_file_name[2],
std::ios::binary);
            if (!file1.is_open() || !file2.is_open() ||
!file3.is_open()) {
                delete[]to_space_name;
                delete[]from_color_space;
                delete[]input_file_name;
                delete[]output_file_name;
                delete picture;
                if (file1.is_open())
```

```
                file1.close();
            if (file2.is_open())
                file2.close();
            if (file3.is_open())
                file3.close();
            throw std::invalid_argument("Cannot open
input file");
        }
        picture = new PPM(&file1, &file2, &file3);
    }


    Converter *to = getConverterByName(false,
to_space_name[0]);
    Converter *from = getConverterByName(true,
from_color_space[0]);
    picture->makeConvertion(from);
    picture->makeConvertion(to);



    ofstream outfile1;
    ofstream outfile2;
    ofstream outfile3;

    if (out_count == '1') {
        outfile1.open(output_file_name[0],
std::ios::binary);
        if (!outfile1.is_open()) {
            delete[]to_space_name;
```

```
                delete[]from_color_space;
                delete[]input_file_name;
                delete[]output_file_name;
                delete picture;
                if (file1.is_open())
                    file1.close();
                if (file2.is_open())
                    file2.close();
                if (file3.is_open())
                    file3.close();
                if (outfile1.is_open())
                    outfile1.close();
                throw std::invalid_argument("Cannot open
output file");
            }
            picture->write_to_file(&outfile1);
        } else {
            outfile1.open(output_file_name[0],
std::ios::binary);
            outfile2.open(output_file_name[1],
std::ios::binary);
            outfile3.open(output_file_name[2],
std::ios::binary);
            if (!outfile1.is_open() ||
!outfile2.is_open() || !outfile3.is_open()) {
                delete[]to_space_name;
                delete[]from_color_space;
                delete[]input_file_name;
                delete[]output_file_name;
```

```
                delete picture;
                if (file1.is_open())
                    file1.close();
                if (file2.is_open())
                    file2.close();
                if (file3.is_open())
                    file3.close();
                if (outfile1.is_open())
                    outfile1.close();
                if (outfile2.is_open())
                    outfile2.close();
                if (outfile3.is_open())
                    outfile3.close();
                throw std::invalid_argument("Cannot open
output file");
            }
            picture->write_to_file(&outfile1, &outfile2,
&outfile3);
        }

        delete[]to_space_name;
        delete[]from_color_space;
        delete[]input_file_name;
        delete[]output_file_name;
        delete picture;
        if (file1.is_open())
            file1.close();
        if (file2.is_open())
```

```
            file2.close();
        if (file3.is_open())
            file3.close();
        if (outfile1.is_open())
            outfile1.close();
        if (outfile2.is_open())
            outfile2.close();
        if (outfile3.is_open())
            outfile3.close();
        return 0;

    } catch (const std::invalid_argument &error) {
        std::cerr << error.what() << std::endl;
        return 1;
    }
}
```

Converter.h

```
#ifndef LAB4_CONVERTER_H
#define LAB4_CONVERTER_H

#include "pixel_t.h"

class Converter {
public:
    virtual pixel_t convert(pixel_t pixel)=0;
};
```

```
#endif //LAB4_CONVERTER_H
```

pixel_t.h

```
#ifndef LAB4_PIXEL_T_H
#define LAB4_PIXEL_T_H
struct pixel_t {
    unsigned char R;
    unsigned char G;
    unsigned char B;

};
```

PPM.h

```
#ifndef LAB4_PPM_H
#define LAB4_PPM_H

#include <iostream>
#include <fstream>
#include <cstdint>
#include <cstdlib>
#include <cmath>
#include "pixel_t.h"
#include "Converter.h"

using namespace std;
```

```
class PPM{
private:
    ifstream *file;
    ifstream *file1;
    ifstream *file2;
    ifstream *file3;
    int count;
    uint16_t width = -1;
    uint16_t height = -1;
    uint16_t depth = -1;
    pixel_t **array;

    void read_header() {
        if(count == 1) {
            unsigned char buf;
            *this->file >> buf;
            if (buf != 'P') {
                throw logic_error("Bad file");
            }
            *this->file >> buf;
            if (buf == '1' || buf == '2' || buf == '3'
|| buf == '4' || buf == '5' || buf == '7') {
                throw logic_error("Not supported type of
NetPCM");
            } else if (buf == '6');
            else {
                throw logic_error("Bad file");
            }
```

```
            *this->file >> buf;
            *this->file >> this->width;
            *this->file >> buf;
            *this->file >> this->height;
            *this->file >> buf;
            *this->file >> this->depth;
            if (this->depth != 255) {
                throw logic_error("Not supported non 255
colors count");
            }
        }else if (count==3){
            unsigned char buf;
            *this->file1 >> buf;
            if (buf != 'P') {
                throw logic_error("Bad file");
            }
            *this->file1 >> buf;
            if (buf == '1' || buf == '2' || buf == '3'
|| buf == '4' || buf == '6' || buf == '7') {
                throw logic_error("Not supported type of
NetPCM");
            } else if (buf == '5');
            else {
                throw logic_error("Bad file");
            }
            *this->file1 >> buf;
            *this->file1 >> this->width;
            *this->file1 >> buf;
```

```
            *this->file1 >> this->height;
            *this->file1 >> buf;
            *this->file1 >> this->depth;
            if (this->depth != 255) {
                throw logic_error("Not supported non 255
colors count");
            }



            *this->file2 >> buf;
            if (buf != 'P') {
                throw logic_error("Bad file");
            }
            *this->file2 >> buf;
            if (buf == '1' || buf == '2' || buf == '3'
|| buf == '4' || buf == '6' || buf == '7') {
                throw logic_error("Not supported type of
NetPCM");
            } else if (buf == '5');
            else {
                throw logic_error("Bad file");
            }
            *this->file2 >> buf;
            *this->file2 >> this->width;
            *this->file2 >> buf;
            *this->file2 >> this->height;
            *this->file2 >> buf;
            *this->file2 >> this->depth;
```

```
            if (this->depth != 255) {
                throw logic_error("Not supported non 255
colors count");
            }

            *this->file3 >> buf;
            if (buf != 'P') {
                throw logic_error("Bad file");
            }
            *this->file3 >> buf;
            if (buf == '1' || buf == '2' || buf == '3'
|| buf == '4' || buf == '6' || buf == '7') {
                throw logic_error("Not supported type of
NetPCM");
            } else if (buf == '5');
            else {
                throw logic_error("Bad file");
            }
            *this->file3 >> buf;
            *this->file3 >> this->width;
            *this->file3 >> buf;
            *this->file3 >> this->height;
            *this->file3 >> buf;
            *this->file3 >> this->depth;
            if (this->depth != 255) {
                throw logic_error("Not supported non 255
colors count");
            }
        }
```

```
}

void read_data() {
    if(count == 1) {
        for (int i = 0; i < this->height; i++) {
            for (int j = 0; j < this->width; j++) {
                unsigned char tmp;
                *this->file >> tmp;
                this->array[i][j].R = tmp;
                *this->file >> tmp;
                this->array[i][j].G = tmp;
                *this->file >> tmp;
                this->array[i][j].B = tmp;
            }
        }
    } else if (count==3){
        for (int i = 0; i < this->height; i++) {
            for (int j = 0; j < this->width; j++) {
                unsigned char tmp;
                *this->file1 >> tmp;
                this->array[i][j].R = tmp;
                *this->file2 >> tmp;
                this->array[i][j].G = tmp;
                *this->file3 >> tmp;
                this->array[i][j].B = tmp;
            }
        }
```

```
        }
    }


public:
    void write_to_file(ofstream *outfile) {
        *outfile << "P6" << (unsigned char) (10) <<
width << " " << height << (unsigned char) (10) << depth
                 << (unsigned char) (10);
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                *outfile << array[i][j].R;
                *outfile << array[i][j].G;
                *outfile << array[i][j].B;
            }
        }
    }
    void write_to_file(ofstream *outfile1,ofstream
*outfile2,ofstream *outfile3) {
        *outfile1 << "P5" << (unsigned char) (10) <<
width << " " << height << (unsigned char) (10) << depth
                 << (unsigned char) (10);
        *outfile2 << "P5" << (unsigned char) (10) <<
width << " " << height << (unsigned char) (10) << depth
                 << (unsigned char) (10);
        *outfile3 << "P5" << (unsigned char) (10) <<
width << " " << height << (unsigned char) (10) << depth
                 << (unsigned char) (10);
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
```

```
                *outfile1 << array[i][j].R;
                *outfile2 << array[i][j].G;
                *outfile3 << array[i][j].B;
            }
        }
    }
    void makeConvertion(Converter* converter){
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                array[i][j] =
converter->convert(array[i][j]);
            }
        }
    }
    explicit
    PPM(ifstream *file) {
        this->count = 1;
        this->file = file;
        this->read_header();
        file->ignore(1);
        this->array = new pixel_t *[height];
        for (int i = 0; i < this->height; i++) {
            this->array[i] = new pixel_t[width];
        }
        read_data();
    }
    PPM(ifstream *file1,ifstream *file2,ifstream *file3)
{
```

```
        this->count=3;
        this->file1 = file1;
        this->file2 = file2;
        this->file3 = file3;
        this->read_header();
        file1->ignore(1);
        file2->ignore(1);
        file3->ignore(1);
        this->array = new pixel_t *[height];
        for (int i = 0; i < this->height; i++) {
            this->array[i] = new pixel_t[width];
        }
        read_data();
    }

    ~PPM() {
        for (int i = 0; i < height; i++) {
            delete[] array[i];
        }
        delete[] array;
    }
};

#endif //LAB4_PPM_H
```

BaseConverter.h

```
#ifndef LAB4_BASECONVERTER_H
```

```
#define LAB4_BASECONVERTER_H


#include "../Converter.h"

class BaseConverter : public Converter{
   pixel_t convert(pixel_t pixel)override {
        return pixel;
   }
};


#endif //LAB4_BASECONVERTER_H
```

fromCMYtoRGB.h

```
#ifndef LAB4_FROMCMYTORGB_H
#define LAB4_FROMCMYTORGB_H

#include "../Converter.h"

class fromCMYtoRGB : public Converter {
   pixel_t convert(pixel_t pixel) override {
        int R = pixel.R;
        int G = pixel.G;
        int B = pixel.B;
        pixel.R = (unsigned char) (round(((1 - (double)
R / 255.) * 255)));
```

```
        pixel.G = (unsigned char) (round(((1 - (double)
G / 255.) * 255)));
        pixel.B = (unsigned char) (round(((1 - (double)
B / 255.) * 255)));
        return pixel;
    }
};



#endif //LAB4_FROMCMYTORGB_H
```

fromHSLtoRGB.h

```
#ifndef LAB4_FROMHSLTORGB_H
#define LAB4_FROMHSLTORGB_H

#include "../Converter.h"

class fromHSLtoRGB : public Converter {
    pixel_t convert(pixel_t pixel) override {

        double h = pixel.R / 255.0;
        double s = pixel.G / 255.0;
        double l = pixel.B / 255.0;
        double q, p;
        if (l < 0.5)
            q = l * (s + 1.0);
        else
            q = l + s - (l * s);
```

```
        p = l * 2 - q;
        double tr = h + 1.0 / 3.0, tg = h, tb = h - 1.0
/ 3.0;
        if (tr < 0) tr += 1.0;
        if (tr > 1.0) tr -= 1.0;
        if (tg < 0) tg += 1.0;
        if (tg > 1.0) tg -= 1.0;
        if (tb < 0) tb += 1.0;
        if (tb > 1.0) tb -= 1.0;
        double r, g, b;

        if (tr < 1.0 / 6.0)
            r = p + ((q - p) * 6.0 * tr);
        else if (tr >= 1.0 / 6.0 && tr < 0.5)
            r = q;
        else if (tr >= 0.5 && tr < 2.0 / 3.0)
            r = p + ((q - p) * (2.0 / 3.0 - tr) * 6.0);
        else
            r = p;

        if (tg < 1.0 / 6.0)
            g = p + ((q - p) * 6.0 * tg);
        else if (tg >= 1.0 / 6.0 && tg < 0.5)
            g = q;
        else if (tg >= 0.5 && tg < 2.0 / 3.0)
            g = p + ((q - p) * (2.0 / 3.0 - tg) * 6.0);
        else
            g = p;
```

```
        if (tb < 1.0 / 6.0)
            b = p + ((q - p) * 6.0 * tb);
        else if (tb >= 1.0 / 6.0 && tb < 0.5)
            b = q;
        else if (tb >= 0.5 && tb < 2.0 / 3.0)
            b = p + ((q - p) * (2.0 / 3.0 - tb) * 6.0);
        else
            b = p;
        pixel.R = (unsigned char)round(r*255);
        pixel.G = (unsigned char)round(g*255);
        pixel.B = (unsigned char)round(b*255);


        return pixel;
    }
};


#endif //LAB4_FROMHSLTORGB_H
```

fromHSVtoRGB.h

```
#ifndef LAB4_FROMHSVTORGB_H
#define LAB4_FROMHSVTORGB_H
#include "../Converter.h"

class fromHSVtoRGB : public Converter{
```

```
    pixel_t convert(pixel_t pixel) override{

        double H = pixel.R / 255.0 * 360.0;
        double S = pixel.G / 255.0;
        double V = pixel.B / 255.0;
        double c = V * S;
        double x = c * (1 - abs(((int) (H/60.)) % 2 +
((H/60.) - (int)(H/60.)) - 1));
        double m = V - c;
        double r, g, b;
        if(H >= 0 && H <= 60)
            r = c, g = x, b = 0;
        else if(H >= 60 && H <= 120)
            r = x, g = c, b = 0;
        else if(H >= 120 && H <= 180)
            r = 0, g = c, b = x;
        else if(H >= 180 && H <= 240)
            r = 0, g = x, b = c;
        else if(H >= 240 && H <= 300)
            r = x, g = 0, b = c;
        else
            r = c, g = 0, b = x;

        int r_int = (int)(round((r + m) * 255));
        int g_int = (int)(round((g + m) * 255));
        int b_int = (int)(round((b + m) * 255));

        if(r_int < 0) r_int = 0;
```

```
        if(r_int > 255) r_int = 255;
        if(b_int < 0) b_int = 0;
        if(b_int > 255) b_int = 255;
        if(g_int < 0) g_int = 0;
        if(g_int > 255) g_int = 255;
        pixel.R = (unsigned char)r_int;
        pixel.G = (unsigned char)g_int;
        pixel.B = (unsigned char)b_int;
        return pixel;
    }
};


#endif //LAB4_FROMHSVTORGB_H
```

fromRGBtoHSL.h

```
#ifndef LAB4_FROMRGBTOHSL_H
#define LAB4_FROMRGBTOHSL_H

#include <algorithm>
#include <cmath>
#include "../Converter.h"

class fromRGBtoHSL : public Converter {
    pixel_t convert(pixel_t pixel) override {
        double r = pixel.R / 255.;
        double g = pixel.G / 255.;
```

```
        double b = pixel.B / 255.;
        double maximum = std::max(std::max(b, g), r);
        double minimum = std::min(std::min(r, g), b);
        double H, S, L;
        if (maximum == r && g >= b)
            H = 60 * (g - b) / (maximum - minimum);
        if (maximum == r && b < g)
            H = 60 * (g - b) / (maximum - minimum) +
360;
        if (maximum == g)
            H = 60 * (b - r) / (maximum - minimum) +
120;
        if (maximum == b)
            H = 60 * (r - g) / (maximum - minimum) +
240;
        L = 1. / 2. * (maximum + minimum);
        S = (maximum - minimum) / (1 - std::abs(1 -
maximum - minimum));
        pixel.R = (unsigned char) round(H / 360 * 255);
        pixel.G = (unsigned char) round(S * 255);
        pixel.B = (unsigned char) round(L * 255);
        return pixel;
    }
};


#endif //LAB4_FROMRGBTOHSL_H
```

fromRGBtoHSV.h

```
#ifndef LAB4_FROMRGBTOHSV_H
#define LAB4_FROMRGBTOHSV_H


#include "../Converter.h"

class fromRGBtoHSV : public Converter {
   pixel_t convert(pixel_t pixel) override {
        double r = pixel.R / 255.;
        double g = pixel.G / 255.;
        double b = pixel.B / 255.;
        double maximum = std::max(std::max(b, g), r);
        double minimum = std::min(std::min(r, g), b);
        double H, S;
        double V = maximum;
        double c = maximum - minimum;

        if (c == 0)
            H = 0;
        else if (V == g)
            H = 2 + (b - r) / c;
        else if (V == r)
            H = (g - b) / c;
        else
            H = 4 + (r - g) / c;
        H *= 60.;
        if (H < 0)
```

```
            H += 360;

        if (V == 0) {
            S = 0;
        } else {
            S = c / V;
        }
        pixel.R = (unsigned char) round(H / 360. *
255.);
        pixel.G = (unsigned char) round(S * 255.);
        pixel.B = (unsigned char) round(V * 255.);

        return pixel;
    }
};


#endif //LAB4_FROMRGBTOHSV_H
```

fromRGBtoCMY.h

```
#ifndef LAB4_FROMRGBTOCMY_H
#define LAB4_FROMRGBTOCMY_H

#include "../Converter.h"

class fromRGBtoCMY : public Converter {
    pixel_t convert(pixel_t pixel) override {
        int R = pixel.R;
```

```
        int G = pixel.G;
        int B = pixel.B;
        pixel.R = (unsigned char) (round(((1 - (double)
R / 255.) * 255)));
        pixel.G = (unsigned char) (round(((1 - (double)
G / 255.) * 255)));
        pixel.B = (unsigned char) (round(((1 - (double)
B / 255.) * 255)));

        return pixel;
    }
};


#endif //LAB4_FROMRGBTOCMY_H
```

fromRGBtoYCbCr601.h

```
#ifndef LAB4_FROMRGBTOYCBCR601_H
#define LAB4_FROMRGBTOYCBCR601_H

#include "../Converter.h"

class fromRGBtoYCbCr601 : public Converter {
    pixel_t convert(pixel_t pixel) override {
        double R = pixel.R / 255.;
        double G = pixel.G / 255.;
        double B = pixel.B / 255.;
```

```
        double KR = 0.299;
        double KG = 0.587;
        double KB = 0.114;

        double Y = KR * R + KG * G + KB * B;
        double PB = 1 / 2. * (B - Y) / (1.0 - KB);
        double PR = 1 / 2. * (R - Y) / (1.0 - KR);
        pixel.R = (unsigned char)
(std::max(std::min(255., round(Y * 255)), 0.));
        pixel.G = (unsigned char)
(std::max(std::min(255., round((PB + 0.5) * 255)),
0.));
        pixel.B = (unsigned char)
(std::max(std::min(255., round((PR + 0.5) * 255)),
0.));
        return pixel;
    }
};


#endif //LAB4_FROMRGBTOYCBCR601_H
```

fromRGBtoYCbCr709.h

```
#ifndef LAB4_FROMRGBTOYCBCR709_H
#define LAB4_FROMRGBTOYCBCR709_H

#include "../Converter.h"

class fromRGBtoYCbCr709 : public Converter{
```

```
    pixel_t convert(pixel_t pixel) override {
        double R = pixel.R / 255.;
        double G = pixel.G / 255.;
        double B = pixel.B / 255.;

        double KR = 0.2126;
        double KG = 0.7152;
        double KB = 0.0722;

        double Y = KR * R + KG * G + KB * B;
        double PB = 1 / 2. * (B - Y) / (1.0 - KB);
        double PR = 1 / 2. * (R - Y) / (1.0 - KR);
        pixel.R = (unsigned char)
(std::max(std::min(255., round(Y * 255)), 0.));
        pixel.G = (unsigned char)
(std::max(std::min(255., round((PB + 0.5) * 255)),
0.));
        pixel.B = (unsigned char)
(std::max(std::min(255., round((PR + 0.5) * 255)),
0.));
        return pixel;
    }
};


#endif //LAB4_FROMRGBTOYCBCR709_H
```

fromRGBtoYCoCg.h

```
#ifndef LAB4_FROMRGBTOYCOCG_H
```

```
#define LAB4_FROMRGBTOYCOCG_H

#include "../Converter.h"

class fromRGBtoYCoCg : public Converter {
   pixel_t convert(pixel_t pixel) override {
        double r = pixel.R * 1.0 / 255;
        double g = pixel.G * 1.0 / 255;
        double b = pixel.B * 1.0 / 255;
        double y = r / 4 + g / 2 + b / 4;
        double co = r / 2 - b / 2 + 0.5;
        double cg = -r / 4 + g / 2 - b / 4 + 0.5;
        pixel.R = (unsigned char)
(std::max(std::min(255., round(y * 255)), 0.));
        pixel.G = (unsigned char)
(std::max(std::min(255., round(co * 255)), 0.));
        pixel.B = (unsigned char)
(std::max(std::min(255., round(cg * 255)), 0.));

        return pixel;
   }

};

#endif //LAB4_FROMRGBTOYCOCG_H
```

fromYCbCr601toRGB.h

```
#ifndef LAB4_FROMYCBCR601TORGB_H
#define LAB4_FROMYCBCR601TORGB_H

#include "../Converter.h"

class fromYCbCr601toRGB : public Converter {
   pixel_t convert(pixel_t pixel) override {
        double Y = pixel.R / 255.;
        double CB = pixel.G / 255. - 0.5;
        double CR = pixel.B / 255. - 0.5;
        double KR = 0.299;
        double KG = 0.587;
        double KB = 0.114;
        double r = Y + (2.0 - 2.0 * KR) * CR;
        double g = Y - KB * (2.0 - 2.0 * KB) * CB / KG -
KR * (2 - 2.0 * KR) * CR / KG;
        double b = Y + (2.0 - 2.0 * KB) * CB;
        pixel.R = (unsigned char)
(std::max(std::min(255., round(r * 255)), 0.));
        pixel.G = (unsigned char)
(std::max(std::min(255., round(g * 255)), 0.));
        pixel.B = (unsigned char)
(std::max(std::min(255., round(b * 255)), 0.));
        return pixel;
   }
};



#endif //LAB4_FROMYCBCR601TORGB_H
```

fromYCbCr709toRGB.h

```
#ifndef LAB4_FROMYCBCR709TORGB_H
#define LAB4_FROMYCBCR709TORGB_H

#include "../Converter.h"

class fromYCbCr709toRGB : public Converter{
   pixel_t convert(pixel_t pixel) override {
        double Y = pixel.R / 255.;
        double CB = pixel.G / 255. - 0.5;
        double CR = pixel.B / 255. - 0.5;
        double KR = 0.2126;
        double KG = 0.7152;
        double KB = 0.0722;
        double r = Y + (2.0 - 2.0 * KR) * CR;
        double g = Y - KB * (2.0 - 2.0 * KB) * CB / KG -
KR * (2 - 2.0 * KR) * CR / KG;
        double b = Y + (2.0 - 2.0 * KB) * CB;
        pixel.R = (unsigned char)
(std::max(std::min(255., round(r * 255)), 0.));
        pixel.G = (unsigned char)
(std::max(std::min(255., round(g * 255)), 0.));
        pixel.B = (unsigned char)
(std::max(std::min(255., round(b * 255)), 0.));
        return pixel;
   }
};
```

```
#endif //LAB4_FROMYCBCR709TORGB_H
```

fromYCoCgtoRGB.h

```
#ifndef LAB4_FROMYCOCGTORGB_H
#define LAB4_FROMYCOCGTORGB_H

#include "../Converter.h"

class fromYCoCgtoRGB : public Converter {
 pixel_t convert(pixel_t pixel) override {
        double y = pixel.R * 1.0 / 255;
    double co = pixel.G * 1.0 / 255 - 0.5;
    double cg = pixel.B * 1.0 / 255 - 0.5;
        double r = y + co - cg;
          double g = y + cg;
        double b = y - co - cg;
      pixel.R = (unsigned char)
(std::max(std::min(255., round(r * 255)), 0.));
      pixel.G = (unsigned char)
(std::max(std::min(255., round(g * 255)), 0.));
      pixel.B = (unsigned char)
(std::max(std::min(255., round(b * 255)), 0.));
            return pixel;
            }
          };
```