Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное
учреждение высшего образования
Санкт-Петербургский исследовательский университет
Информационных технологий, механики и оптики
Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Компьютерная графика и геометрия

Отчет

по лабораторной работе №4

Изучение цветовых пространств

Выполнил: студент гр. М3101
Ершов Михаил Юрьевич
Преподаватель: Скаков П.С.

Санкт-Петербург
2020

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

Описание работы

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

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

1. 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.
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**).

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

2. Цветовые пространства

Цветовые пространства соответствуют различным системам представления информации о цвете.

Так как в соответствии с моделью зрения человека мы имеем 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.

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.

3. 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.

4. HSL/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

5. 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	Y: 16..235
U: 0..255	U: 16..240
V: 0..255	V: 16..240

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

6. YCoCg

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

Y – яркость
Cg – цветоразность “хроматический зелёный”
Co – цветоразность “хроматический оранжевый”

7. CMY/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.

C (Cyan) – голубой
M (Magenta) – пурпурный
Y(Yellow) – жёлтый
K (blacK) – чёрный

Экспериментальная часть

Язык программирования: C++14

Считываем изображение из PPM файла, либо же из 3 PGM файлов, для каждого из 3 каналов соответственно, затем методом convert применяем конвертацию из исходного цветового пространства в требуемое и, сохранив все величины в диапазоне 0..255, выводим изображение либо в указанный PPM файл, либо по каналам выводим изображения в 3 PGM файла. Конвертация осуществляется сначала из исходного пространства в RGB, затем из RGB в требуемое, за исключением пространства HSV, которое конвертируется в HSL, и только затем, если требуется, осуществляется конвертация в RGB и перевод в требуемое. Аналогично HSL конвертируется в HSV напрямую.

Выводы

Выполнение данной работы позволило изучить различные цветовые пространства и способы перехода из одного пространства в другое.

Листинг

main.cpp

```
#include <iostream>
#include <string>
#include <cstdlib>
#include "ppm.h"

using namespace std;

int main(int argc, char* argv[]) {
    char* from, * to;
    int count_in, count_out;
    char* input;
    char* output;
    try {
        for (int i = 0; i < argc; i++) {
            if (string(argv[i]) == "-f") {
                if (i != argc - 1) {
                    from = argv[i + 1];
                }
                else {
                    cerr << "Invalid input\n";
                    return 1;
                }
            }
            if (string(argv[i]) == "-t") {
                if (i != argc - 1) {
                    to = argv[i + 1];
                }
                else {
                    cerr << "Invalid input\n";
                    return 1;
                }
            }
            if (string(argv[i]) == "-i" && i != argc - 1) {
                count_in = atoi(argv[i + 1]);
                if (i < argc - 2) {
                    input = argv[i + 2];
                }
                else {
                    cerr << "Invalid input\n";
                    return 1;
                }
            }
            if (string(argv[i]) == "-o" && i != argc - 1) {
                count_out = atoi(argv[i + 1]);
                if (i < argc - 2) {
                    output = argv[i + 2];
                }
                else {
                    cerr << "Invalid input\n";
                    return 1;
                }
            }
        }
        if (count_in != 1 && count_in != 3 || count_out != 1 && count_out != 3) {
            cerr << "Invalid input\n";
            return 1;
        }
    }
    catch (const exception& e) {
        cerr << "Invalid input\n" << e.what() << "\n";
```

```
        return 1;
    }
    try {
        if (count_in == 1) {
            PPM image(input);
            image.convert(from, to);
            if (count_out == 1) {
                image.print(output);
            }
            else {
                image.print(output, 3);
            }
        }
        else {
            PPM image(input, 3);
            image.convert(from, to);
            if (count_out == 1) {
                image.print(output);
            }
            else {
                image.print(output, 3);
            }
        }
    }
    catch (const exception& e) {
        cerr << e.what() << "\n";
        return 1;
    }
    return 0;
}
```

pgm.h

```
#ifndef LAB4_PGM_H
#define LAB4_PGM_H

#include <vector>
#include <cstdio>
#include <cstring>

typedef unsigned char uchar;

struct Pixel {
    uchar first, second, third;
    Pixel(uchar first, uchar second, uchar third) : first(first), second(second), third(third) {}
    Pixel() = default;
};

class PPM {
private:
    char header[2];
    int width, height;
    uchar maxValue;
    Pixel* data;

    static Pixel convertRGBtoHSL(const Pixel& pixel);
    static Pixel convertHSLtoRGB(const Pixel& pixel);
    static Pixel convertHSLtoHSV(const Pixel& pixel);
    static Pixel convertHSVtoHSL(const Pixel& pixel);
    static Pixel convertRGBtoYCbCr_601(const Pixel& pixel);
    static Pixel convertRGBtoYCbCr_709(const Pixel& pixel);
    static Pixel convertYCbCr_601toRGB(const Pixel& pixel);
    static Pixel convertYCbCr_709toRGB(const Pixel& pixel);
    static Pixel convertRGBtoYCoCg(const Pixel& pixel);
    static Pixel convertYCoCgtoRGB(const Pixel& pixel);
    static Pixel convertRGBtoCMY(const Pixel& pixel);
    static Pixel convertCMYtoRGB(const Pixel& pixel);
    void readP5(char* fileName, int pos);
    void printP5(char* fileName, int pos);
public:
    explicit PPM(char* fileName);
    explicit PPM(char* fileName, int num);
    ~PPM();
    void print(char* fileName);
    void print(char* fileName, int num);
    void convert(char* from, char* to);
};



#endif //LAB4_PGM_H
```

pgm.cpp

```
#include "ppm.h"
#include <stdexcept>
#include <string>
#include <algorithm>

PPM::PPM(char* fileName) {
    FILE* fin = fopen(fileName, "rb");
    if (fin == nullptr) {
        throw std::runtime_error("File can't be opened\n");
    }
    int tmp;
    if (fscanf(fin, "%s%d%d%d", header, &width, &height, &tmp) < 4) {
        throw std::runtime_error("Invalid header\n");
    }
    if (strcmp(header, "P6") != 0) {
        throw std::runtime_error("File is not a PGM image\n");
    }
    if (tmp > 255) {
        throw std::runtime_error("Unsupported colours\n");
    }
    maxValue = (uchar)tmp;
    if (width < 0 || height < 0) {
        throw std::runtime_error("Invalid header\n");
    }
    if (fgetc(fin) == EOF) {
        throw std::runtime_error("Invalid header\n");
    }
    data = new Pixel[width * height];
    fread(data, sizeof(Pixel), width * height, fin);
    if (fclose(fin) != 0) {
        throw std::runtime_error("File can't be closed\n");
    }
}

void PPM::readP5(char* fileName, int pos) {
    FILE* fin = fopen(fileName, "rb");
    if (fin == nullptr) {
        throw std::runtime_error("File can't be opened\n");
    }
    int tmp;
    if (fscanf(fin, "%s%d%d%d", header, &width, &height, &tmp) < 4) {
        throw std::runtime_error("Invalid header\n");
    }
    if (strcmp(header, "P5") != 0) {
        throw std::runtime_error("File is not a PGM image\n");
    }
    if (tmp > 255) {
        throw std::runtime_error("Unsupported colours\n");
    }
    maxValue = (uchar)tmp;
    if (width < 0 || height < 0) {
        throw std::runtime_error("Invalid header\n");
    }
    if (fgetc(fin) == EOF) {
        throw std::runtime_error("Invalid header\n");
    }
    if (data == NULL) {
        data = new Pixel[width * height];
    }
    uchar* dataP5 = new uchar[width * height];
    fread(dataP5, sizeof(uchar), width * height, fin);
    for (int i = 0; i < width * height; i++) {
        switch (pos) {
```

```
        case 0: data[i].first = dataP5[i]; break;
        case 1: data[i].second = dataP5[i]; break;
        case 2: data[i].third = dataP5[i]; break;
        }
    }
    delete[] dataP5;
    if (fclose(fin) != 0) {
        throw std::runtime_error("File can't be closed\n");
    }
}

PPM::PPM(char* fileName, int num) {
    data = NULL;
    std::string in(fileName);
    int pos = -1;
    for (int i = in.size() - 1; i >= 0; i--) {
        if (in[i] == '.') {
            pos = i;
            break;
        }
    }
    readP5((char*)std::string(in.substr(0, pos) + "_1" + in.substr(pos, in.size())).c_str(), 0);
    readP5((char*)std::string(in.substr(0, pos) + "_2" + in.substr(pos, in.size())).c_str(), 1);
    readP5((char*)std::string(in.substr(0, pos) + "_3" + in.substr(pos, in.size())).c_str(), 2);
    strcpy(header, "P6");
}

PPM::~PPM() {
    delete[] data;
}

void PPM::print(char* fileName) {
    FILE* fout = fopen(fileName, "wb");
    if (fout == nullptr) {
        throw std::runtime_error("File can't be opened or created\n");
    }
    fprintf(fout, "%s\n%i %i\n%i\n", header, width, height, maxValue);
    fwrite(data, sizeof(Pixel), width * height, fout);
    fclose(fout);
}
void PPM::printP5(char* fileName, int pos) {
    FILE* fout = fopen(fileName, "wb");
    if (fout == nullptr) {
        throw std::runtime_error("File can't be opened or created\n");
    }
    uchar* dataP5 = new uchar[width * height];
    for (int i = 0; i < width * height; i++) {
        switch (pos) {
        case 0: dataP5[i] = data[i].first; break;
        case 1: dataP5[i] = data[i].second; break;
        case 2: dataP5[i] = data[i].third; break;
        }
    }
    fprintf(fout, "%s\n%i %i\n%i\n", "P5", width, height, maxValue);
    fwrite(dataP5, 1, width * height, fout);
    fclose(fout);
}

void PPM::print(char* fileName, int num) {
    std::string in(fileName);
    int pos = -1;
    for (int i = in.size() - 1; i >= 0; i--) {
        if (in[i] == '.') {
            pos = i;
            break;
```

```
        }
    }
    printP5((char*)std::string(in.substr(0, pos) + "_1" + in.substr(pos, in.size())).c_str(), 0);
    printP5((char*)std::string(in.substr(0, pos) + "_2" + in.substr(pos, in.size())).c_str(), 1);
    printP5((char*)std::string(in.substr(0, pos) + "_3" + in.substr(pos, in.size())).c_str(), 2);
}

void PPM::convert(char* from, char* to) {
    std::string f = std::string(from);
    std::string t = std::string(to);
    if (f == "HSL" && t == "HSV") {
        for (int i = 0; i < height * width; i++) {
            data[i] = convertHSLtoHSV(data[i]);
        }
        return;
    }
    if (f == "HSV" && t == "HSL") {
        for (int i = 0; i < height * width; i++) {
            data[i] = convertHSVtoHSL(data[i]);
        }
        return;
    }

    for (int i = 0; i < height * width; i++) {
        if (f == "HSV") {
            data[i] = convertHSLtoRGB(convertHSVtoHSL(data[i]));
        }
        if (f == "HSL") {
            data[i] = convertHSLtoRGB(data[i]);
        }
        if (f == "CMY") {
            data[i] = convertCMYtoRGB(data[i]);
        }
        if (f == "YCbCr.601") {
            data[i] = convertYCbCr_601toRGB(data[i]);
        }
        if (f == "YCbCr.709") {
            data[i] = convertYCbCr_709toRGB(data[i]);
        }
        if (f == "YCoCg") {
            data[i] = convertYCoCgtoRGB(data[i]);
        }
    }

    for (int i = 0; i < height * width; i++) {
        if (t == "HSV") {
            data[i] = convertHSLtoHSV(convertRGBtoHSL(data[i]));
        }
        if (t == "HSL") {
            data[i] = convertRGBtoHSL(data[i]);
        }
        if (t == "CMY") {
            data[i] = convertRGBtoCMY(data[i]);
        }
        if (t == "YCbCr.601") {
            data[i] = convertRGBtoYCbCr_601(data[i]);
        }
        if (t == "YCbCr.709") {
            data[i] = convertRGBtoYCbCr_709(data[i]);
        }
        if (t == "YCoCg") {
            data[i] = convertRGBtoYCoCg(data[i]);
        }
    }
}
```

```
Pixel PPM::convertRGBtoHSL(const Pixel& pixel) {
    double R = (double)pixel.first / 255.0;
    double G = (double)pixel.second / 255.0;
    double B = (double)pixel.third / 255.0;
    double Cmax = std::max(std::max(R, G), B);
    double Cmin = std::min(std::min(R, G), B);
    double delta = Cmax - Cmin;
    double L = (Cmax + Cmin) / 2.0;
    double H, S;
    if (delta == 0) {
        H = 0.0;
        S = 0.0;
    }
    else {
        S = delta / (1 - std::abs(2.0 * L - 1.0));
        if (Cmax == R) {
            H = (G - B) / delta;
            int d = (int)H / 6;
            H -= d * 6.0;
        }
        if (Cmax == G) {
            H = (B - R) / delta + 2.0;
        }
        if (Cmax == B) {
            H = (R - G) / delta + 4;
        }
    }
    H = std::max(std::min(H, 6.0), 0.0);
    S = std::max(std::min(S, 1.0), 0.0);
    L = std::max(std::min(L, 1.0), 0.0);
    return Pixel((H / 6.0) * 255, S * 255, L * 255);
}

Pixel PPM::convertHSLtoHSV(const Pixel& pixel) {
    double S = (double)pixel.second / 255.0;
    double L = (double)pixel.third / 255.0;
    S *= L < 0.5 ? L : 1 - L;
    double newS = 2 * S / (L + S);
    double newV = L + S;
    newS = std::max(std::min(newS, 1.0), 0.0);
    newV = std::max(std::min(newV, 1.0), 0.0);
    return Pixel(pixel.first, 255 * newS, 255 * newV);
}

Pixel PPM::convertHSVtoHSL(const Pixel& pixel) {
    double S = (double)pixel.second / 255.0;
    double V = (double)pixel.third / 255.0;
    double L = (2.0 - S) * V / 2.0;
    if (L < 1e-5) {
        L = 0.0;
    }
    if (1 - L < 1e-5) {
        L = 1.0;
    }
    if (L != 0.0) {
        if (L == 1.0) {
            S = 0.0;
        }
        else if (L < 0.5) {
            S = S * V / (L * 2.0);
        }
        else {
            S = S * V / (2.0 - L * 2.0);
        }
    }
```

```
    S = std::max(std::min(S, 1.0), 0.0);
    L = std::max(std::min(L, 1.0), 0.0);
    return Pixel(pixel.first, 255 * S, 255 * L);
};

double HtoRGB(double p, double q, double t) {
    if (t < 0.0) {
        t += 1.0;
    }
    if (t > 1.0) {
        t -= 1.0;
    }
    if (t < 1.0 / 6.0) {
        return p + (q - p) * 6.0 * t;
    }
    if (t < 0.5) {
        return q;
    }
    if (t < 2.0 / 3.0) {
        return p + (q - p) * (2.0 / 3.0 - t) * 6.0;
    }
    return p;
}

Pixel PPM::convertHSLtoRGB(const Pixel& pixel) {
    double H = (double)pixel.first / 255.0;
    double S = (double)pixel.second / 255.0;
    double L = (double)pixel.third / 255.0;
    double R, G, B;
    if (S < 1e-5) {
        R = G = B = L;
    }
    else {
        double q = L < 0.5 ? L * (1.0 + S) : L + S - L * S;
        double p = 2.0 * L - q;
        R = HtoRGB(p, q, H + 1.0 / 3.0);
        G = HtoRGB(p, q, H);
        B = HtoRGB(p, q, H - 1.0 / 3.0);
    }
    R = std::max(std::min(R, 1.0), 0.0);
    G = std::max(std::min(G, 1.0), 0.0);
    B = std::max(std::min(B, 1.0), 0.0);
    return Pixel(R * 255, G * 255, B * 255);
}
Pixel PPM::convertCMYtoRGB(const Pixel& pixel) {
    return Pixel(255 - pixel.first, 255 - pixel.second, 255 - pixel.third);
}

Pixel PPM::convertRGBtoCMY(const Pixel& pixel) {
    return Pixel(255 - pixel.first, 255 - pixel.second, 255 - pixel.third);
}

Pixel PPM::convertRGBtoYCbCr_601(const Pixel& pixel) {
    double R = (double)pixel.first / 255.0;
    double G = (double)pixel.second / 255.0;
    double B = (double)pixel.third / 255.0;
    double Y = 0.299 * R + 0.587 * G + 0.114 * B;
    double Cb = (B - Y) / 1.772 + 0.5;
    double Cr = (R - Y) / 1.402 + 0.5;
    Y = std::max(std::min(Y, 1.0), 0.0);
    Cb = std::max(std::min(Cb, 1.0), 0.0);
    Cr = std::max(std::min(Cr, 1.0), 0.0);
    return Pixel(255 * Y, 255 * Cb, 255 * Cr);
}
```

```
Pixel PPM::convertYCbCr_601toRGB(const Pixel& pixel) {
    double Y = (double)pixel.first / 255.0;
    double Cb = (double)pixel.second / 255.0 - 0.5;
    double Cr = (double)pixel.third / 255.0 - 0.5;
    double R = Y + 1.402 * Cr;
    double G = Y - (0.299 * 1.402 / 0.587) * Cr - (0.114 * 1.772 / 0.587) * Cb;
    double B = Y + 1.772 * Cb;
    R = std::max(std::min(R, 1.0), 0.0);
    G = std::max(std::min(G, 1.0), 0.0);
    B = std::max(std::min(B, 1.0), 0.0);
    return Pixel(R * 255, G * 255, B * 255);
}

Pixel PPM::convertRGBtoYCbCr_709(const Pixel& pixel) {
    double R = (double)pixel.first / 255.0;
    double G = (double)pixel.second / 255.0;
    double B = (double)pixel.third / 255.0;
    double Y = 0.2126 * R + 0.7152 * G + 0.0722 * B;
    double Cb = (B - Y) / 1.8556 + 0.5;
    double Cr = (R - Y) / 1.5748 + 0.5;
    Y = std::max(std::min(Y, 1.0), 0.0);
    Cb = std::max(std::min(Cb, 1.0), 0.0);
    Cr = std::max(std::min(Cr, 1.0), 0.0);
    return Pixel(Y * 255, Cb * 255, Cr * 255);
}

Pixel PPM::convertYCbCr_709toRGB(const Pixel& pixel) {
    double Y = (double)pixel.first / 255.0;
    double Cb = (double)pixel.second / 255.0 - 0.5;
    double Cr = (double)pixel.third / 255.0 - 0.5;
    double R = Y + 1.5748 * Cr;
    double G = Y - (0.2126 * 1.5748 / 0.7152) * Cr - (0.0722 * 1.8556 / 0.7152) * Cb;
    double B = Y + 1.8556 * Cb;
    R = std::max(std::min(R, 1.0), 0.0);
    G = std::max(std::min(G, 1.0), 0.0);
    B = std::max(std::min(B, 1.0), 0.0);
    return Pixel(R * 255, G * 255, B * 255);
}

Pixel PPM::convertRGBtoYCoCg(const Pixel& pixel) {
    double R = (double)pixel.first / 255.0;
    double G = (double)pixel.second / 255.0;
    double B = (double)pixel.third / 255.0;
    double Y = 0.25 * R + 0.5 * G + 0.25 * B;
    double Co = 0.5 * R - 0.5 * B + 0.5;
    double Cg = -0.25 * R + 0.5 * G - 0.25 * B + 0.5;
    Y = std::max(std::min(Y, 1.0), 0.0);
    Co = std::max(std::min(Co, 1.0), 0.0);
    Cg = std::max(std::min(Cg, 1.0), 0.0);
    return Pixel(Y * 255, Co * 255, Cg * 255);
}

Pixel PPM::convertYCoCgtoRGB(const Pixel& pixel) {
    double Y = (double)pixel.first / 255.0;
    double Co = (double)pixel.second / 255.0 - 0.5;
    double Cg = (double)pixel.third / 255.0 - 0.5;
    double R = Y + Co - Cg;
    double G = Y + Cg;
    double B = Y - Co - Cg;
    R = std::max(std::min(R, 1.0), 0.0);
    G = std::max(std::min(G, 1.0), 0.0);
    B = std::max(std::min(B, 1.0), 0.0);
    return Pixel(255 * R, 255 * G, 255 * B);
}
```