

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 геометрия и графика»

Изучение цветовых пространств

Выполнил студент группы №М3101
Семенов Георгий Витальевич

Преподаватель:
Скаков Павел Сергеевич

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:

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.

Теоретическая часть

Изображения могут быть представлены в различных цветовых пространствах, описываемых одной или несколькими цветовыми координатами.

Главными аддитивными цветовыми пространствами являются RGB и CMY.

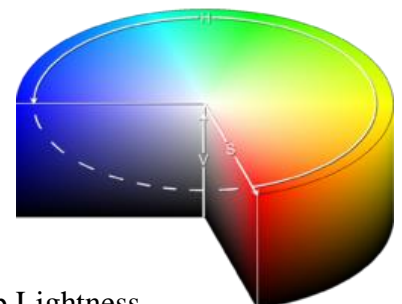
Цветовое пространство **RGB** использует каналы красного, зелёного и синего цветов; их выбор обусловлен физиологией восприятия человеческого глаза.

Цветовое пространство **CMY** (или CMYK с чёрным цветом) использует цвета голубого, жёлтого и пурпурного; используется, главным образом, в типографии.

HSV использует три канала: Hue — тон, Saturation — насыщенность, Value — яркость.

Hue определяется углом от 0 до 360 градусов на цветовом круге, Saturation определяет отдалённость от белого цвета, а Value — близость к чёрному.

HSL вместо параметра Value использует обратный параметр Lightness.



Семейство пространств **YCbCr** определяется тремя координатами: Y — яркость цвета (люма), Cb — синяя цветоразностная компонента, Cr — красная цветоразностная компонента. Несмотря на то, что оно далеко от абсолютного цветового пространства, в силу того, что человеческий глаз менее чувствителен к перепадам цвета, пространство позволяет уменьшить поток видеоданных. CbCr-плоскость при Y=1 представлена на рисунке.



Пространство **YCoCg** определяется координатами: Y — яркость цвета (люма), Co — зелёная цветоразностная компонента (хрома зелёного), Cg — оранжевая цветоразностная компонента (хрома оранжевого).

Экспериментальная часть

Язык программирования: C++.

Изображение описано в классе `baseImage`, хранящее в двумерном массиве (vector) указатели на экземпляры класса `baseColor`, в котором реализованы методы для работы с цветовыми пространствами.

При инициализации объекта `baseImage` указывается путь к файлу, количество файлов и цветовое пространство. Метод `print()` содержит аналогичную сигнатуру и производит запись изображения в файл в новом цветовом пространстве.

Перевод из RGB в новые пространства:

CMY:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$H' = \begin{cases} \text{undefined}, & \text{if } C = 0 \\ \frac{G-B}{C} \bmod 6, & \text{if } M = R \\ \frac{B-R}{C} + 2, & \text{if } M = G \\ \frac{R-G}{C} + 4, & \text{if } M = B \end{cases}$$

$$H = 60^\circ \times H'$$

$$V = \max(R, G, B) = M$$

$$Y'_{601} = 0.2989 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B$$

$$Y'_{709} = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722$$

$$C = \text{range}(R, G, B) = M - m$$

$$S_V = \begin{cases} 0, & \text{if } V = 0 \\ \frac{C}{V}, & \text{otherwise} \end{cases}$$

$$S_L = \begin{cases} 0, & \text{if } L = 1 \text{ or } L = 0 \\ \frac{C}{1-|2L-1|}, & \text{otherwise} \end{cases}$$

YCbCr:

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

YCoCg:

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

Перевод в RGB:

Для HSL

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

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

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

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

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

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

HSV

$$C = V \times S_V$$

YCbCr:

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

YCoCg:

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

CMY:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

Вывод

Изображения могут быть представлены в различных цветовых пространствах, впрочем, необязательно эквивалентно.

Листинг

<https://github.com/MrGeorgeous/ComputerGeometryAndGraphics/blob/master/README.md>

Файл: Source.cpp

```
#define _CRT_SECURE_NO_WARNINGS

#include<stdio.h>
#include <iostream>
```

```

#include <fstream>
#include <istream>
#include <iomanip>

#include<string>
#include<vector>
#include<set>
#include<queue>

#include <iostream>
#include <cmath>
#include <cfenv>

#pragma STDC FENV_ACCESS ON

using namespace std;

const int UCHAR_SIZE = sizeof(unsigned char);

typedef vector<vector<double>> matrix;

matrix matrixYCbCr601 = { {-100,0.0,0.0},{0.0,0.0,0.0},{0.0,0.0,0.0} };
matrix revMatrixYCbCr601 = { {-100,0.0,0.0},{0.0,0.0,0.0},{0.0,0.0,0.0} };
matrix matrixYCbCr709 = { {-100,0.0,0.0},{0.0,0.0,0.0},{0.0,0.0,0.0} };
matrix revMatrixYCbCr709 = { {-100,0.0,0.0},{0.0,0.0,0.0},{0.0,0.0,0.0} };

matrix matrixYCoCg = { {1.0/4,1.0/2,1.0/4},{1.0/2,0,-1.0/2},{-1.0/4,1.0/2,-1.0/4} };
matrix revMatrixYCoCg = { {1,1,-1},{1,0,1},{1,-1,-1} };

void fillYCbCr(matrix& m, double a, double b, double c) {
    // a - K_r, b - K_g, c - K_b
    m[0][0] = a;
    m[0][1] = b;
    m[0][2] = c;
    m[1][0] = -0.5 * a / (1.0 - c);
    m[1][1] = -0.5 * b / (1.0 - c);
    m[1][2] = 0.5;
    m[2][0] = 0.5;
    m[2][1] = -0.5 * b / (1.0 - a);
    m[2][2] = -0.5 * c / (1.0 - a);
}

void fillRevYCbCr(matrix& m, double a, double b, double c) {
    // a - K_r, b - K_g, c - K_b
    m[0][0] = 1.0;
    m[0][1] = 0;
    m[0][2] = 2.0 - 2.0 * a;
    m[1][0] = 1.0;
    m[1][1] = -c / b * (2.0 - 2.0 * c);
    m[1][2] = -a / b * (2.0 - 2.0 * a);
    m[2][0] = 1.0;
    m[2][1] = 2.0 - 2.0 * c;
    m[2][2] = 0;
}

```

```

class baseColor {

public:

    double red = 1; // 0 ... 1
    double green = 1; // 0 ... 1
    double blue = 1; // 0 ... 1

    baseColor() {

    }

    baseColor(baseColor * c) {
        red = c->red;
        green = c->green;
        blue = c->blue;
    }

    baseColor(double r, double g, double b) : red(r), green(g), blue(b) {

    }

    static double HueToRGB(double v1, double v2, double vH) {
        if (vH < 0)
            vH += 1;

        if (vH > 1)
            vH -= 1;

        if ((6 * vH) < 1)
            return (v1 + (v2 - v1) * 6 * vH);

        if ((2 * vH) < 1)
            return v2;

        if ((3 * vH) < 2)
            return (v1 + (v2 - v1) * ((2.0f / 3) - vH) * 6);

        return v1;
    }

    void fromHSLToRGB() {

        double H = this->red;
        double S = this->green ;
        double L = this->blue ;

        double r = 0;
        double g = 0;
        double b = 0;

        if (S == 0) {
            r = L;
            g = L;
            b = L;
        } else {
            double v1, v2;
            double hue = H;

```

```

        v2 = (L < 0.5) ? (L * (1 + S)) : ((L + S) - (L * S));
        v1 = 2 * L - v2;

        r = HueToRGB(v1, v2, hue + (1.0 / 3));
        g = HueToRGB(v1, v2, hue);
        b = HueToRGB(v1, v2, hue - (1.0 / 3));
    }

    this->red = r ;
    this->green = g ;
    this->blue = b;
}

void fromRGBToHSL() {

    double r = this->red ;
    double g = this->green;
    double b = this->blue;

    double max_c = max(r, max(g,b));
    double min_c = min(r, min(g, b));

    double h, s, l = (max_c + min_c) / 2;

    if (max_c == min_c) {
        h = 0;
        s = 0; // achromatic
    } else {
        double d = max_c - min_c;
        s = (l > 0.5) ? (d / (2 - max_c - min_c)) : (d / (max_c +
min_c));

        if (max_c == r) {
            h = (g - b) / d + ((g < b) ? 6 : 0);
        }
        if (max_c == g) {
            h = (b - r) / d + 2;
        }
        if (max_c == b) {
            h = (r - g) / d + 4;
        }

        h /= 6;
    }

    this->red = h;
    this->green = s ;
    this->blue = l ;

}

void fromHSVToRGB() {

    double fH = red * 360;

```



```

double fS = green;
double fV = blue;

double fR = 0;
double fG = 0;
double fB = 0;

double fC = fV * fS;
double fHPrime = fmod(fH / 60.0, 6);
double fX = fC * (1 - fabs(fmod(fHPrime, 2) - 1));
double fM = fV - fC;

if (0 <= fHPrime && fHPrime < 1) {
    fR = fC;
    fG = fX;
    fB = 0;
}
else if (1 <= fHPrime && fHPrime < 2) {
    fR = fX;
    fG = fC;
    fB = 0;
}
else if (2 <= fHPrime && fHPrime < 3) {
    fR = 0;
    fG = fC;
    fB = fX;
}
else if (3 <= fHPrime && fHPrime < 4) {
    fR = 0;
    fG = fX;
    fB = fC;
}
else if (4 <= fHPrime && fHPrime < 5) {
    fR = fX;
    fG = 0;
    fB = fC;
}
else if (5 <= fHPrime && fHPrime < 6) {
    fR = fC;
    fG = 0;
    fB = fX;
}
else {
    fR = 0;
    fG = 0;
    fB = 0;
}

fR += fM;
fG += fM;
fB += fM;

this->red = fR;
this->green = fG;
this->blue = fB;
}

void fromRGBtoHSV() {

```

```

double fH = 0;
double fS = 0;
double fV = 0;

double fR = red;
double fG = green;
double fB = blue;

double fCMax = max(max(fR, fG), fB);
double fCMin = min(min(fR, fG), fB);
double fDelta = fCMax - fCMin;

if (fDelta > 0) {
    if (fCMax == fR) {
        fH = 60 * (fmod(((fG - fB) / fDelta), 6));
    }
    else if (fCMax == fG) {
        fH = 60 * (((fB - fR) / fDelta) + 2);
    }
    else if (fCMax == fB) {
        fH = 60 * (((fR - fG) / fDelta) + 4);
    }

    if (fCMax > 0) {
        fS = fDelta / fCMax;
    }
    else {
        fS = 0;
    }

    fV = fCMax;
}
else {
    fH = 0;
    fS = 0;
    fV = fCMax;
}

if (fH < 0) {
    fH = 360 + fH;
}

this->red = fH / 360;
this->green = fS;
this->blue = fV;

}

void fromCMYtoRGB() {
    this->red = 1.0 - this->red;
    this->green = 1.0 - this->green;
    this->blue = 1.0 - this->blue;
}

void fromRGBtoCMY() {
    this->red = 1.0 - this->red;
    this->green = 1.0 - this->green;

```

```

        this->blue = 1.0 - this->blue;
    }

    void fromRGBtoYCbCr601() {
        matrixMultiply(matrixYCbCr601);
        this->green += 0.5;
        this->blue += 0.5;
    }

    void fromYCbCr601toRGB() {
        this->green -= 0.5;
        this->blue -= 0.5;
        matrixMultiply(revMatrixYCbCr601);
    }

    void fromRGBtoYCbCr709() {
        matrixMultiply(matrixYCbCr709);
        this->green += 0.5;
        this->blue += 0.5;
    }

    void fromYCbCr709toRGB() {
        this->green -= 0.5;
        this->blue -= 0.5;
        matrixMultiply(revMatrixYCbCr709);
    }

    void fromRGBtoYCoCg() {
        matrixMultiply(matrixYCoCg);
        this->green += 0.5;
        this->blue += 0.5;
    }

    void fromYCoCgtoRGB() {
        this->green -= 0.5;
        this->blue -= 0.5;
        matrixMultiply(revMatrixYCoCg);
    }

    void matrixMultiply(matrix & m) {
        if (m[0][0] == -100) {
            cout << "Fatality.";
        }

        double r = m[0][0] * this->red + m[0][1] * this->green + m[0][2] *
this->blue;
        double g = m[1][0] * this->red + m[1][1] * this->green + m[1][2] *
this->blue;
        double b = m[2][0] * this->red + m[2][1] * this->green + m[2][2] *
this->blue;

        /*this->red = max(0.0, min(1.0, r));
        this->green = max(0.0, min(1.0, g));
        this->blue = max(0.0, min(1.0, b));*/

        this->red = r;
        this->green = g;
        this->blue = b;
    }

```

```
};
```

```
typedef vector<vector<double>> doubleMatrix;  
typedef vector<vector<baseColor*>> pnmMatrix;  
typedef vector<char> chars;
```

```
enum palette {  
    RGB,  
    HSL,  
    HSV,  
    YCbCr601,  
    YCbCr709,  
    YCoCg,  
    CMY,  
    NO_PALETTE  
};
```

```
enum files {  
    One,  
    Three  
};
```

```
enum channel {  
    All,  
    Red,  
    Green,  
    Blue  
};
```

```
class baseImage {
```

```
public:
```

```
    int width = 0; // y  
    int height = 0; // x  
    palette colorSpace = RGB;
```

```
    pnmMatrix m;  
    chars errorEncounter;
```

```
    baseImage(size_t w, size_t h, baseColor color = baseColor()) {
```

```
        m = pnmMatrix(h, vector<baseColor*>(w, nullptr));  
        for (int i = 0; i < height; i++) {  
            for (int j = 0; j < width; j++) {  
                m[i][j] = new baseColor(color);  
            }  
        }
```

```
        width = w;  
        height = h;
```

```

    }

    ~baseImage() {

        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                delete m[i][j];
            }
        }

    }

    bool loadChannelsFromFile(string filename, channel ch = All, const double
channelDepth = 255) {

        if (!errorEncounter.empty()) {
            return false;
        }

        FILE* file = fopen(filename.c_str(), "rb");
        if ((file != NULL)) {}
        else {
            errorEncounter.push_back(1);
            return false;
        }

        char p1, p2 = ' ';
        int w = 0, h = 0, d = 0;

        fscanf(file, "%c%c\n%i %i\n%i\n", &p1, &p2, &w, &h, &d);

        if (((w <= 0) || (h <= 0)) || (p1 != 'P') || (!(p2 == '5') || (p2 ==
'6')) || !(d == 255)) {
            errorEncounter.push_back(1);
            return false;
        }

        if ((ch == All) || (ch == Red)) {
            width = w;
            height = h;
            m = pnmMatrix(h, vector<baseColor*>(w, nullptr));
            for (int i = 0; i < height; i++) {
                for (int j = 0; j < width; j++) {
                    m[i][j] = new baseColor();
                }
            }
        }

        unsigned char t;
        unsigned char r, g, b;

        for (int j = 0; j < min(h, height); j++) {
            for (int i = 0; i < min(w, width); i++) {

                if (ch == All) {

```

```

        size_t res = 0;
        res += fread(&r, UCHAR_SIZE, 1, file);
        res += fread(&g, UCHAR_SIZE, 1, file);
        res += fread(&b, UCHAR_SIZE, 1, file);

        if (res != 3 * UCHAR_SIZE) {
            errorEncounter.push_back(1);
            return false;
        }

        m[j][i]->red = double(r) / channelDepth;
        m[j][i]->green = double(g) / channelDepth;
        m[j][i]->blue = double(b) / channelDepth;

    }
    else {

        size_t r = fread(&t, UCHAR_SIZE, 1, file);
        if (r != UCHAR_SIZE) {
            errorEncounter.push_back(1);
            return false;
        }

        switch (ch) {
        case Red:
            m[j][i]->red = double(t) /
channelDepth;

            break;
        case Green:
            m[j][i]->green = double(t) /
channelDepth;

            break;
        case Blue:
            m[j][i]->blue = double(t) /
channelDepth;

            break;
        }
    }
}

}

fclose(file);
return true;
}

baseImage(string filename, palette c, files count) : colorSpace(c) {

    if (!errorEncounter.empty()) {
        return;
    }
}

```

```

    }

    if (count == Three) {

        string fn = filename.substr(0, filename.find_last_of("."));
        string ext = filename.substr(filename.find_last_of(".") + 1,
filename.size() - (filename.find_last_of(".") + 1));

        if (!loadChannelsFromFile(fn + "_1." + ext, Red)) {
            return;
        }
        if (!loadChannelsFromFile(fn + "_2." + ext, Green)) {
            return;
        }
        if (!loadChannelsFromFile(fn + "_3." + ext, Blue)) {
            return;
        }
    } else {
        if (!loadChannelsFromFile(filename, All)) {
            return;
        }
    }
}

void processConversionTo(palette c) {

    if (colorSpace == c) {
        return;
    }

    fillRevYCbCr(revMatrixYCbCr601, 0.299, 0.587, 0.114);
    fillRevYCbCr(revMatrixYCbCr709, 0.0722, 0.2126, 0.7152);

    fillYCbCr(matrixYCbCr601, 0.299, 0.587, 0.114);
    fillYCbCr(matrixYCbCr709, 0.0722, 0.2126, 0.7152);

    for (int j = 0; j < height; j++) {
        for (int i = 0; i < width; i++) {

            // Now let's convert to RGB
            switch (colorSpace) {
                case RGB:
                    break;
                case HSL:
                    m[j][i] -> fromHSLToRGB();
                    break;
                case HSV:
                    m[j][i] -> fromHSVToRGB();
                    break;
                case YCbCr601:
                    m[j][i] -> fromYCbCr601toRGB();
                    break;
                case YCbCr709:
                    m[j][i] -> fromYCbCr709toRGB();
                    break;
            }
        }
    }
}

```

```

        case YCoCg:
            m[j][i] -> fromYCoCgtoRGB();
            break;
        case CMY:
            m[j][i] -> fromCMYtoRGB();
            break;
    }

    // Now let's go back from RGB
    switch (c) {
        case RGB:
            break;
        case HSL:
            m[j][i] -> fromRGBtoHSL();
            break;
        case HSV:
            m[j][i] -> fromRGBtoHSV();
            break;
        case YCbCr601:
            m[j][i] -> fromRGBtoYCbCr601();
            break;
        case YCbCr709:
            m[j][i] -> fromRGBtoYCbCr709();
            break;
        case YCoCg:
            m[j][i] -> fromRGBtoYCoCg();
            break;
        case CMY:
            m[j][i] -> fromRGBtoCMY();
            break;
    }

    }

}

}

void print(string filename, palette c, files count) {

    if (!errorEncounter.empty()) {
        return;
    }

    processConversionTo(c);

    if (count == Three) {

        string fn = filename.substr(0, filename.find_last_of("."));
        string ext = filename.substr(filename.find_last_of(".") + 1,
filename.size() - (filename.find_last_of(".") + 1));

        if (!writeChannels(fn + "_1." + ext, Red)) {
            return;
        }
        if (!writeChannels(fn + "_2." + ext, Green)) {

```



```

        return;
    }
    if (!writeChannels(fn + "_3." + ext, Blue)) {
        return;
    }
}
else {
    if (!writeChannels(filename, All)) {
        return;
    }
}

}

bool writeChannels(string filename, channel ch = All, unsigned char
channelDepth = 255) {

    if (!errorEncounter.empty()) {
        return false;
    }

    FILE* file = fopen(filename.c_str(), "wb");
    if (!(file != NULL)) {
        errorEncounter.push_back(1);
        return false;
    }

    if (ch != All) {

        fprintf(file, "P5\n");
        fprintf(file, "%i %i\n%i\n", width, height, channelDepth);

        for (int j = 0; j < height; j++) {
            for (int i = 0; i < width; i++) {

                unsigned char t = 0;
                switch (ch) {
                    case Red:
                        t = double(max(0.0, min(1.0, m[j][i]-
>red)) * channelDepth);

                        break;
                    case Green:
                        t = double(max(0.0, min(1.0, m[j][i]-
>green)) * channelDepth);

                        break;
                    case Blue:
                        t = double(max(0.0, min(1.0, m[j][i]-
>blue)) * channelDepth);

                        break;
                }

                //switch (ch) {
                //case Red:
                //    t = double(m[j][i]->red *
channelDepth);

                //    break;
                //case Green:

```

```

channelDepth);

//      t = double(m[j][i]->green *
//      break;
//case Blue:
//      t = double(m[j][i]->blue *
channelDepth);

//      break;
//}

fwrite(&t, sizeof(unsigned char), 1, file);

    }

}

else {

    fprintf(file, "P6\n");
    fprintf(file, "%i %i\n%i\n", width, height, channelDepth);

    for (int j = 0; j < height; j++) {
        for (int i = 0; i < width; i++) {

            unsigned char r = max(0.0, min(1.0, m[j][i]-
>red)) * channelDepth;
            unsigned char g = max(0.0, min(1.0, m[j][i]-
>green)) * channelDepth;
            unsigned char b = max(0.0, min(1.0, m[j][i]-
>blue)) * channelDepth;

            //unsigned char r = m[j][i]->red *
            //unsigned char g = m[j][i]->green *
            //unsigned char b = m[j][i]->blue *

            fwrite(&r, sizeof(unsigned char), 1, file);
            fwrite(&g, sizeof(unsigned char), 1, file);
            fwrite(&b, sizeof(unsigned char), 1, file);

        }

    }

    fclose(file);
    return true;

}

};

```

```

palette stringToPalette(string arg_next) {
    palette t = RGB;
    if (arg_next == "RGB") {
        t = RGB;
    }
    if (arg_next == "HSL") {
        t = HSL;
    }
    if (arg_next == "HSV") {
        t = HSV;
    }
    if (arg_next == "YCbCr.601") {
        t = YCbCr601;
    }
    if (arg_next == "YCbCr.709") {
        t = YCbCr709;
    }
    if (arg_next == "YCoCg") {
        t = YCoCg;
    }
    if (arg_next == "CMY") {
        t = CMY;
    }
    return t;
}

```

```

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

    palette fromPalette = RGB;
    palette toPalette = RGB;
    files fromFiles = One;
    files toFiles = One;
    string in = "";
    string out = "";

    for (int i = 0; i < argc; i++) {
        if (i == 0) {
            continue;
        }

        string arg(argv[i]);
        if (arg == "-f") {
            if (i + 1 < argc) {
                fromPalette = stringToPalette(argv[i + 1]);
            }
        }
        if (arg == "-t") {
            if (i + 1 < argc) {
                toPalette = stringToPalette(argv[i + 1]);
            }
        }
        if (arg == "-i") {
            if (i + 2 < argc) {
                if (string(argv[i + 1]) == "1") {

```

```

        fromFiles = One;
    }
    if (string(argv[i + 1]) == "3") {
        fromFiles = Three;
    }
    in = string(argv[i + 2]);
}
}
if (arg == "-o") {
    if (i + 2 < argc) {
        if (string(argv[i + 1]) == "1") {
            toFiles = One;
        }
        if (string(argv[i + 1]) == "3") {
            toFiles = Three;
        }
        out = string(argv[i + 2]);
    }
}

baseImage im(in, fromPalette, fromFiles);
im.print(out, toPalette, toFiles);

if (!im.errorEncounter.empty()) {
    cerr << "Some errors encountered.";
    return 1;
}

// TEST ALL MODES
//string name = "west_1";
//string toname = "test/im";
//string revname = "output/im";

//for (int i = RGB; i != NO_PALETTE; i++) {
//    for (int j = RGB; j != NO_PALETTE; j++) {

//        string lol = toname + "_" + to_string(i) + "_" + to_string(j);
//        baseImage a(name + ".ppm", static_cast<palette>(i),
files::One);
//        a.print(lol + "_1to1.ppm", static_cast<palette>(j),
files::One);
//        baseImage b(name + ".ppm", static_cast<palette>(i),
files::One);
//        b.print(lol + "_1to3.ppm", static_cast<palette>(j),
files::Three);

//        baseImage c(name + ".ppm", static_cast<palette>(i),
files::Three);
//        c.print(lol + "_3to1.ppm", static_cast<palette>(j),
files::One);

//        baseImage d(name + ".ppm", static_cast<palette>(i),
files::Three);
//        d.print(lol + "_3to3.ppm", static_cast<palette>(j),
files::Three);

//    }

```

```

//}

//for (int i = RGB; i != NO_PALETTE; i++) {
//    for (int j = RGB; j != NO_PALETTE; j++) {

//        string lol1 = toname + "_" + to_string(i) + "_" +
to_string(j);
//        string lol2 = revname + "_" + to_string(i) + "_" +
to_string(j);
//        baseImage a(lol1 + "_1to1.ppm", static_cast<palette>(j),
files::One);
//        a.print(lol2 + "_1to1.ppm", static_cast<palette>(i),
files::One);

//    }
//}

return 0;

}

```