МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение высшего образования

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

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

**Отчет по лабораторной работе №4**

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

**Изучение цветовых пространств**

**Выполнил студент группы №M3101:**

***Пантелеев Ярослав Кириллович***

**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.

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:**

***Цветовые 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

***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

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

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

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

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”

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

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

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

***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”

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

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

***Пространство 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

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
```

***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.

**Экспериментальная часть**

Язык выполнения работы: C++17, компилятор Microsoft Visual C++.

Этапы работы программы:

1) Чтение файла картинки (хедера и информации о цвете каждого пикселя) с обработкой ошибок чтения (из 1 файла или из 3 файлов)
2) Выполнение заданного аргументом командной строки преобразования между цветовыми пространствами (из *from* в *to*)
3) Запись полученного изображения в выходной файл (из 1 файла или из 3 файлов)

**Вывод:**

Выполнение данной лабораторной работы позволило узнать о существовании различных цветовых пространств, об их особенностях, сферах применения и об алгоритмах преобразования изображения для перехода из одного цветового пространства в другое.

**Листинг исходного кода:**

**main.cpp**

```
#include <iostream>
#include <string>
#include <vector>
#include "Image.h"

int main(int argc, char* argv[]) {
    std::string from, to, input, output;
    int input_count = 0;
    int output_count = 0;
    for(int i = 1; i < argc; i++) {
        if(std::string(argv[i]) == "-f") {
            from = std::string(argv[i + 1]);
        }
        if(std::string(argv[i]) == "-t") {
            to = std::string(argv[i + 1]);
        }
        if(std::string(argv[i]) == "-i") {
            input_count = atoi(argv[i + 1]);
            input = std::string(argv[i + 2]);
        }
        if(std::string(argv[i]) == "-o") {
            output_count = atoi(argv[i + 1]);
            output = std::string(argv[i + 2]);
        }
    }

    bool input_format_is_correct = false;
    bool output_format_is_correct = false;
    std::vector <std::string> color_spaces = { "RGB", "HSL", "HSV", "YCbCr.601", "YCbCr.709", "
YCoCg", "CMY" };
    for (const auto& color_space : color_spaces) {
        if(color_space == from)
            input_format_is_correct = true;
        if(color_space == to)
            output_format_is_correct = true;
    }
    if (!input_format_is_correct ||
        !output_format_is_correct ||
        argc != 11 ||
        from.empty() || to.empty() || input.empty() || output.empty() ||
        input_count != 1 && input_count != 3 ||
        output_count != 1 && output_count != 3)
    {
        std::cerr << "command line arguments are invalid" << "\n";
        return 1;
    }

    Image* img;
    try {
        img = new Image(input, input_count);
    }
    catch(const std::exception& error) {
        std::cerr << error.what() << '\n';
        return 1;
    }

    img->change_color_space(from, to);
    try {
        img->write(output, output_count);
    }
    catch(const std::exception& error) {
        std::cerr << error.what() << '\n';
        delete img;
```

```
63.         return 1;
64.     }
65.
66.     delete img;
67.     return 0;
68. }
```

## Image.h

```
1.  #pragma once
2.  #include <vector>
3.  #include <fstream>
4.
5.  class Pixel {
6.  public:
7.      unsigned char a, b, c;
8.      Pixel(unsigned char, unsigned char, unsigned char);
9.      Pixel() = default;
10.     void RGB_to_HSL();       // 1 -> 2
11.     void HSL_to_RGB();       // 2 -> 1
12.     void RGB_to_HSV();       // 1 -> 3
13.     void HSV_to_RGB();       // 3 -> 1
14.     void RGB_to_YCbCr_601(); // 1 -> 4
15.     void YCbCr_601_to_RGB(); // 4 -> 1
16.     void RGB_to_YCbCr_709(); // 1 -> 5
17.     void YCbCr_709_to_RGB(); // 5 -> 1
18.     void RGB_to_YCoCg();     // 1 -> 6
19.     void YCoCg_to_RGB();     // 6 -> 1
20.     void RGB_to_CMY();       // 1 -> 7
21.     void CMY_to_RGB();       // 7 -> 1
22. };
23.
24. class Image {
25. public:
26.     Image(const std::string&, int);
27.     void write(std::string, int);
28.     void change_color_space(std::string, std::string);
29. private:
30.     int width;
31.     int height;
32.     int color_depth;
33.     std::vector <std::vector <Pixel>> image;
34. };
```

## Image.cpp

```
1.  #include "Image.h"
2.  #include <algorithm>
3.  #include <stdexcept>
4.  #include <cmath>
5.  #include <string>
6.  #include <vector>
7.  #include <fstream>
8.
9.  std::vector <std::string> parse_pattern(std::string pattern) {
10.     int separator_index = -1;
11.     for (size_t i = 0; i < pattern.size(); i++)
12.         if (pattern[i] == '.')
13.             separator_index = i;
14.     if(separator_index == -1)
15.         throw std::runtime_error("pattern has no dots, so doesn't match name.ext");
16.
17.     const std::string name = pattern.substr(0, separator_index);
18.     const std::string format = pattern.substr(separator_index);
19.     return { name + "_1" + format,
20.              name + "_2" + format,
```

```
21.              name + "_3" + format };
22. }
23.
24. Pixel::Pixel(unsigned char a, unsigned char b, unsigned char c) : a(a), b(b), c(c) {}
25.
26. Image::Image(const std::string& filename, int count) {
27.     if(count == 1) {
28.
29.         std::ifstream fin(filename, std::ios::binary);
30.         if (!fin.is_open())
31.             throw std::runtime_error("failed to open file");
32.
33.         char ch[2];
34.         fin >> ch[0] >> ch[1];
35.         if (ch[0] != 'P' || ch[1] != '6')
36.             throw std::runtime_error("expected P6 format");
37.         fin >> this->width >> this->height >> this->color_depth;
38.         if(this->color_depth != 255)
39.             throw std::runtime_error("only 255 color depth is supported");
40.         this->image.assign(this->height, std::vector <Pixel>(this->width));
41.
42.         char color;
43.         fin.read(&color, 1);
44.         for (int i = 0; i < this->height; i++) {
45.             for (int j = 0; j < this->width; j++) {
46.                 this->image[i][j] = Pixel();
47.
48.                 fin.read(&color, sizeof(unsigned char));
49.                 this->image[i][j].a = color;
50.
51.                 fin.read(&color, sizeof(unsigned char));
52.                 this->image[i][j].b = color;
53.
54.                 fin.read(&color, sizeof(unsigned char));
55.                 this->image[i][j].c = color;
56.             }
57.         }
58.         fin.close();
59.     }
60.     else {
61.         std::vector <std::string> files = parse_pattern(filename);
62.         if(files.size() != 3)
63.             throw std::runtime_error("unknown error during name generation");
64.
65.         for(int k = 0; k < 3; k++) {
66.             std::ifstream fin(files[k], std::ios::binary);
67.             if(!fin.is_open())
68.                 throw std::runtime_error("failed to open file " + files[k]);
69.
70.             char ch[2];
71.             fin >> ch[0] >> ch[1];
72.             if (ch[0] != 'P' || ch[1] != '5')
73.                 throw std::runtime_error("expected P5 format");
74.             fin >> this->width >> this->height >> this->color_depth;
75.             if (this->color_depth != 255)
76.                 throw std::runtime_error("only 255 color depth is supported");
77.             if (k == 0) {
78.                 this->image.assign(this->height, std::vector <Pixel>(this->width));
79.             }
80.             char color;
81.             fin.read(&color, 1);
82.             for (int i = 0; i < this->height; i++) {
83.                 for (int j = 0; j < this->width; j++) {
84.                     fin.read(&color, sizeof(unsigned char));
85.                     if (k == 0)
86.                         this->image[i][j].a = color;
87.                     if (k == 1)
88.                         this->image[i][j].b = color;
```

```
                    if (k == 2)
                        this->image[i][j].c = color;
                    // fin.read(&color, sizeof(unsigned char));
                    // fin.read(&color, sizeof(unsigned char));
                }
            }
            fin.close();
        }
    }
}

void Image::write(std::string filename, int count) {
    if(count == 1) {
        std::ofstream fout(filename, std::ios::binary);
        if(!fout.is_open()) {
            throw std::runtime_error("cannot open output file");
        }

        fout << "P6\n" << width << ' ' << height << '\n' << color_depth << '\n';
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                fout << image[i][j].a;
                fout << image[i][j].b;
                fout << image[i][j].c;
            }
        }
        fout.flush();
        fout.close();
    } else {
        std::vector<std::string> files = parse_pattern(filename);
        if(files.size() != 3)
            throw std::runtime_error("unknown error during name generation");

        for(int k = 0; k < 3; k++) {
            std::ofstream fout(files[k], std::ios::binary);
            if(!fout.is_open()) {
                throw std::runtime_error("cannot open output file " + files[k]);
            }

            fout << "P5\n" << width << ' ' << height << '\n' << color_depth << '\n';
            if (k == 0) {
                for (int i = 0; i < height; i++) {
                    for (int j = 0; j < width; j++) {
                        fout << image[i][j].a;
                        // fout << image[i][j].a;
                        // fout << image[i][j].a;
                    }
                }
            }
            else if (k == 1) {
                for (int i = 0; i < height; i++) {
                    for (int j = 0; j < width; j++) {
                        fout << image[i][j].b;
                        // fout << image[i][j].b;
                        // fout << image[i][j].b;
                    }
                }
            }
            else if (k == 2) {
                for (int i = 0; i < height; i++) {
                    for (int j = 0; j < width; j++) {
                        fout << image[i][j].c;
                        // fout << image[i][j].c;
                        // fout << image[i][j].c;
                    }
                }
            }
            fout.flush();
```

```
157.             fout.close();
158.         }
159.     }
160.
161. }
162.
163. void Image::change_color_space(std::string from, std::string to) {
164.     for (int i = 0; i < height; i++) {
165.         for (int j = 0; j < width; j++) {
166.             if (from == "HSL")
167.                 image[i][j].HSL_to_RGB();
168.             if (from == "HSV")
169.                 image[i][j].HSV_to_RGB();
170.             if (from == "YCbCr.601")
171.                 image[i][j].YCbCr_601_to_RGB();
172.             if (from == "YCbCr.709")
173.                 image[i][j].YCbCr_709_to_RGB();
174.             if (from == "YCoCg")
175.                 image[i][j].YCoCg_to_RGB();
176.             if (from == "CMY")
177.                 image[i][j].CMY_to_RGB();
178.         }
179.     }
180.
181.     for (int i = 0; i < height; i++) {
182.         for (int j = 0; j < width; j++) {
183.             if (to == "HSL")
184.                 image[i][j].RGB_to_HSL();
185.             if (to == "HSV")
186.                 image[i][j].RGB_to_HSV();
187.             if (to == "YCbCr.601")
188.                 image[i][j].RGB_to_YCbCr_601();
189.             if (to == "YCbCr.709")
190.                 image[i][j].RGB_to_YCbCr_709();
191.             if (to == "YCoCg")
192.                 image[i][j].RGB_to_YCoCg();
193.             if (to == "CMY")
194.                 image[i][j].RGB_to_CMY();
195.         }
196.     }
197. }
198.
199. void Pixel::HSL_to_RGB() {
200.     // H S L
201.     const double h = a / 255.0;
202.     const double s = b / 255.0;
203.     const double l = c / 255.0;
204.
205.     double q;
206.     if (l < 0.5)
207.         q = l * (s + 1.0);
208.     else
209.         q = l + s - (l * s);
210.     const double p = l * 2 - q;
211.
212.     // R
213.     double r;
214.     double tr = h + 1.0 / 3.0;
215.     if (tr < 0)
216.         tr += 1.0;
217.     else if (tr > 1.0)
218.         tr -= 1.0;
219.     if (tr < 1.0 / 6.0)
220.         r = p + (q - p) * 6.0 * tr;
221.     else if (tr >= 1.0 / 6.0 && tr < 0.5)
222.         r = q;
223.     else if (tr >= 0.5 && tr < 2.0 / 3.0)
224.         r = p + (q - p) * (2.0 / 3.0 - tr) * 6.0;
```

```
225.     else
226.         r = p;
227.
228.     // G
229.     double g;
230.     double tg = h;
231.     if (tg < 0)
232.         tg += 1.0;
233.     else if (tg > 1.0)
234.         tg -= 1.0;
235.     if (tg < 1.0 / 6.0)
236.         g = p + (q - p) * 6.0 * tg;
237.     else if (tg >= 1.0 / 6.0 && tg < 0.5)
238.         g = q;
239.     else if (tg >= 0.5 && tg < 2.0 / 3.0)
240.         g = p + (q - p) * (2.0 / 3.0 - tg) * 6.0;
241.     else
242.         g = p;
243.
244.     // B
245.     double b;
246.     double tb = h - 1.0 / 3.0;
247.     if (tb < 0)
248.         tb += 1.0;
249.     else if (tb > 1.0)
250.         tb -= 1.0;
251.     if(tb < 1.0 / 6.0)
252.         b = p + (q - p) * 6.0 * tb;
253.     else if(tb >= 1.0 / 6.0 && tb < 0.5)
254.         b = q;
255.     else if(tb >= 0.5 && tb < 2.0 / 3.0)
256.         b = p + (q - p) * (2.0 / 3.0 - tb) * 6.0;
257.     else
258.         b = p;
259.
260.     // Output
261.     this->a = static_cast<unsigned char>(r * 255);
262.     this->b = static_cast<unsigned char>(g * 255);
263.     this->c = static_cast<unsigned char>(b * 255);
264. }
265.
266. void Pixel::RGB_to_HSL() {
267.     // R G B
268.     const double r = a * 1.0 / 255;
269.     const double g = b * 1.0 / 255;
270.     const double b = c * 1.0 / 255;
271.     const double max = std::max(r, std::max(g, b));
272.     const double min = std::min(r, std::min(g, b));
273.
274.     // H S L
275.     double h;
276.     if(max == min)
277.         h = 0;
278.     else if(r == max && g >= b)
279.         h = (g - b) / (max - min) * 60;
280.     else if(r == max && g < b)
281.         h = (g - b) / (max - min) * 60 + 360;
282.     else if(max == g)
283.         h = (b - r) / (max-min) * 60 + 120;
284.     else
285.         h = (r - g) / (max-min) * 60 + 240;
286.     const double l = (max + min) / 2;
287.     const double s = (max - min) / (1 - abs(1 - (max + min)));
288.
289.     // Output
290.     this->a = static_cast<unsigned char>(round(h * 255 / 360));
291.     this->b = static_cast<unsigned char>(round(s * 255));
292.     this->c = static_cast<unsigned char>(round(l * 255));
```

```
293. }
294.
295. void Pixel::HSV_to_RGB() {
296.     // H S V
297.     const double h = a / 255.0 * 360.0;
298.     const double s = b / 255.0;
299.     const double v = c / 255.0;
300.
301.     const double c = v * s;
302.     const double x = c * (1 - abs((static_cast<int>(h / 60)) % 2 + (h / 60 - static_cast<int>(h
/ 60)) - 1));
303.     const double m = v - c;
304.
305.     // R G B
306.     double r, g, b;
307.     if(h >= 0 && h <= 60)
308.         r = c, g = x, b = 0;
309.     else if(h >= 60 && h <= 120)
310.         r = x, g = c, b = 0;
311.     else if(h >= 120 && h <= 180)
312.         r = 0, g = c, b = x;
313.     else if(h >= 180 && h <= 240)
314.         r = 0, g = x, b = c;
315.     else if(h >= 240 && h <= 300)
316.         r = x, g = 0, b = c;
317.     else
318.         r = c, g = 0, b = x;
319.
320.     // Output
321.     this->a = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>((r + m) * 255))));
322.     this->b = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>((g + m) * 255))));
323.     this->c = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>((b + m) * 255))));
324. }
325.
326. void Pixel::RGB_to_HSV() {
327.     // R G B
328.     const double r = a * 1.0 / 255;
329.     const double g = b * 1.0 / 255;
330.     const double b = c * 1.0 / 255;
331.
332.     // H S V
333.     const double v = std::max(r, std::max(g, b));
334.     const double diff = v - std::min(r, std::min(g, b));
335.     double h;
336.     if(diff == 0)
337.         h = 0;
338.     else if(v == r)
339.         h = (g - b) / diff;
340.     else if(v == g)
341.         h = 2 + (b - r) / diff;
342.     else
343.         h = 4 + (r - g) / diff;
344.     h *= 60.0;
345.     if(h < 0)
346.         h += 360;
347.     const double s = (v == 0 ? 0 : diff / v);
348.
349.     // Output
350.     this->a = static_cast<unsigned char>(round(h / 360.0 * 255.0));
351.     this->b = static_cast<unsigned char>(round(s * 255.0));
352.     this->c = static_cast<unsigned char>(round(v * 255.0));
353. }
354.
355. void Pixel::CMY_to_RGB() {
356.     this->a = static_cast<unsigned char>(255 - a);
```

```
357.     this->b = static_cast<unsigned char>(255 - b);
358.     this->c = static_cast<unsigned char>(255 - c);
359. }
360.
361. void Pixel::RGB_to_CMY() {
362.     this->a = static_cast<unsigned char>(255 - a);
363.     this->b = static_cast<unsigned char>(255 - b);
364.     this->c = static_cast<unsigned char>(255 - c);
365. }
366.
367. void Pixel::RGB_to_YCoCg() {
368.     // R G B
369.     const double r = a * 1.0 / 255;
370.     const double g = b * 1.0 / 255;
371.     const double b = c * 1.0 / 255;
372.
373.     // Y Co Cg
374.     double y = std::min(1.0, std::max(0.0, static_cast<double>(r / 4 + g / 2 + b / 4)));
375.     double co = std::min(1.0, std::max(0.0, static_cast<double>(r / 2 - b / 2 + 0.5)));
376.     double cg = std::min(1.0, std::max(0.0, static_cast<double>(-
r / 4 + g / 2 - b / 4 + 0.5)));
377.
378.     // Output
379.     this->a = static_cast<unsigned char>(y * 255);
380.     this->b = static_cast<unsigned char>(co * 255);
381.     this->c = static_cast<unsigned char>(cg * 255);
382. }
383.
384. void Pixel::YCoCg_to_RGB() {
385.     // Y Co Cg
386.     const double y = a * 1.0 / 255;
387.     const double co = b * 1.0 / 255 - 0.5;
388.     const double cg = c * 1.0 / 255 - 0.5;
389.
390.     // R G B
391.     const double r = std::min(1.0, std::max(0.0, static_cast<double>(y + co - cg)));
392.     const double g = std::min(1.0, std::max(0.0, static_cast<double>(y + cg)));
393.     const double b = std::min(1.0, std::max(0.0, static_cast<double>(y - co - cg)));
394.
395.     // Output
396.     this->a = static_cast<unsigned char>(r * 255);
397.     this->b = static_cast<unsigned char>(g * 255);
398.     this->c = static_cast<unsigned char>(b * 255);
399. }
400.
401. void Pixel::RGB_to_YCbCr_601() {
402.     // R G B
403.     const double r = a / 255.0;
404.     const double g = b / 255.0;
405.     const double b = c / 255.0;
406.
407.     // Coefficients for R G B
408.     const double coef_r = 0.299;
409.     const double coef_g = 0.587;
410.     const double coef_b = 0.114;
411.
412.     // Y Cb Cr
413.     const double Y = coef_r * r + coef_g * g + coef_b * b;
414.     const double Cb = (b - Y) / (2.0 * (1.0 - coef_b));
415.     const double Cr = (r - Y) / (2.0 * (1.0 - coef_r));
416.
417.     // Output
418.     this->a = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(Y * 255)))));
419.     this->b = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round((Cb + 0.5) * 255)))));
420.     this->c = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round((Cr + 0.5) * 255)))));
```

```
421. }
422.
423. void Pixel::YCbCr_601_to_RGB() {
424.     // Y Cb Cr
425.     const double Y = a / 255.0;
426.     const double Cb = b / 255.0 - 0.5;
427.     const double Cr = c / 255.0 - 0.5;
428.
429.     // Coefficients for R G B
430.     const double coef_r = 0.299;
431.     const double coef_g = 0.587;
432.     const double coef_b = 0.114;
433.
434.     // R G B
435.     const double r = Y + (2.0 - 2.0 * coef_r) * Cr;
436.     const double g = Y - coef_b * (2.0 - 2.0 * coef_b) * Cb / coef_g - coef_r * (2 - 2.0 * coef
_r) * Cr / coef_g;
437.     const double b = Y + (2.0 - 2.0 * coef_b) * Cb;
438.
439.     // Output
440.     this->a = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(r * 255)))));
441.     this->b = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(g * 255)))));
442.     this->c = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(b * 255)))));
443. }
444.
445. void Pixel::RGB_to_YCbCr_709() {
446.     // R G B
447.     const double r = a / 255.0;
448.     const double g = b / 255.0;
449.     const double b = c / 255.0;
450.
451.     // Coefficients for R G B
452.     const double coef_r = 0.2126;
453.     const double coef_g = 0.7152;
454.     const double coef_b = 0.0722;
455.
456.     // Y Cb Cr
457.     const double Y = coef_r * r + coef_g * g + coef_b * b;
458.     const double Cb = (b - Y) / (2.0 * (1.0 - coef_b));
459.     const double Cr = (r - Y) / (2.0 * (1.0 - coef_r));
460.
461.     // Output
462.     this->a = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(Y * 255)))));
463.     this->b = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round((Cb + 0.5) * 255)))));
464.     this->c = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round((Cr + 0.5) * 255)))));
465. }
466.
467. void Pixel::YCbCr_709_to_RGB() {
468.     // Y Cb Cr
469.     const double Y = a / 255.0;
470.     const double Cb = b / 255.0 - 0.5;
471.     const double Cr = c / 255.0 - 0.5;
472.
473.     // Coefficients for R G B
474.     const double coef_r = 0.2126;
475.     const double coef_g = 0.7152;
476.     const double coef_b = 0.0722;
477.
478.     // R G B
479.     const double r = Y + (coef_r * -2.0 + 2.0) * Cr;
480.     const double g = Y - coef_b * (2.0 - 2.0 * coef_b) * Cb / coef_g - coef_r * (2.0 - 2.0 * co
ef_r) * Cr / coef_g;
```

```
481.     const double b = Y + (coef_b * -2.0 + 2.0) * Cb;
482.
483.     // Output
484.     this->a = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(r * 255)))));
485.     this->b = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(g * 255)))));
486.     this->c = static_cast<unsigned char>(std::min(255,
std::max(0, static_cast<int>(std::round(b * 255)))));
487. }
```