МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

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

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа № 4

По дисциплине «Компьютерная графика и геометрия»

Изучение цветовых пространств

Выполнил студент группы М3101

Кузьмук Павел Юрьевич

Проверил:

Скаков Павел Сергеевич

САНКТ-ПЕТЕРБУРГ

2020

ЦЕЛЬ РАБОТЫ

Реализация программы, которая позволяет проводить преобразования между цветовыми пространствами.

ОПИСАНИЕ РАБОТЫ

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

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

Аргументы передаются через командную строку:

lab4.exe -f <from_color_space> -t <to_color_space> -i <count> <input_file_name> -o <count> <output_file_name>

где

- <color_space> - RGB / HSL / HSV / YCbCr.601 / YCbCr.709 / YCoCg / CMY
- <count> - 1 или 3
- <file_name>:
  - для count=1 просто имя файла; формат ppm
  - для count=3 шаблон имени вида <name.ext>, что соответствует файлам <name_1.ext>, <name_2.ext> и <name_3.ext> для каждого канала соответственно; формат pgm

Порядок аргументов (-f, -t, -i, -o) может быть произвольным.

Везде 8-битные данные и полный диапазон (0..255, PC range).

Полное решение: всё работает + корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок.

Если программе передано значение, которое не поддерживается – следует сообщить об ошибке.

Коды возврата:

0 - ошибок нет

1 - произошла ошибка

В поток вывода ничего не выводится (printf, cout).

Сообщения об ошибках выводятся в поток вывода ошибок:

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

Пространство YUV / YCbCr

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

Y – яркость

U/Cb – цветоразность “хроматический синий”

V/Cr – цветоразность “хроматический красный”

В пространстве YUV традиционно существует два диапазона значений.

Для 8-битных данных:

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

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

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

Пространство YCgCo

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

Y – яркость

Cg – цветоразность “хроматический зелёный”

Co – цветоразность “хроматический оранжевый”

Пространства CMY и CMYK

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

C (Cyan) – голубой

M (Magenta) – пурпурный

Y(Yellow) – жёлтый

K (blacK) – чёрный

ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

Все данные будем хранить в классе image, для удобства занесем чтение из 1 цветного и 3 монохромных файлов в 1 метод, таким образом после чтения мы гарантированно получаем изображение с 3 каналами.

На следующем шаге вызываем необходимое преобразование для каждого пикселя изображения, в любом случае приводя данные в диапазон 0..255

Аналогично первому шагу выводим изображение в файл.

Стоит отметить, что pnm не поддерживает цветовых пространств кроме rgb, то есть изображение, например, в hsl будет выглядеть странно в программе для просмотра(gimp в моем случае)

ВЫВОД

Выполнение данной лабораторной работы позволило узнать о существовании различных цветовых пространств, об алгоритмах преобразования данных изображения для перехода из одного цветового пространства в другое и о возможных потерях при переходе между цветовыми пространствами. Для удобства все переходы между цветовыми пространствами производятся через пространство RGB.

ЛИСТИНГ КОДА

main.cpp:

```
#include <iostream>

#include <string>
#include <vector>
#include "image.h"

using namespace std;

vector<string> color_spaces = {"RGB", "HSL", "HSV", "YCbCr.601", "YCbCr.709", "YCoCg", "CMY"};

int main(int argc, char *argv[]) {
string from, to, input, output;
int inc, outc;
bool wf = false;
for(int i = 1; i < argc; i ++)
{
if(string(argv[i]) == "-f" && i < argc - 1) {
from = string(argv[i+1]);
}
if(string(argv[i]) == "-t" && i < argc - 1) {
to = string(argv[i+1]);
}
if(string(argv[i]) == "-i" && i < argc - 2) {
inc = atoi(argv[i+1]);
input = string(argv[i+2]);
}
if(string(argv[i]) == "-o" && i < argc - 2) {
outc = atoi(argv[i+1]);
output = string(argv[i+2]);
}
}
wf |= argc != 11;
wf |= from == "" || to == "" || input == "" || output == "";
wf |= (inc != 1 && inc != 3);
wf |= (outc != 1 && outc != 3);
bool inf = false, outf = false;
for(int i = 0; i < color_spaces.size(); i ++)
{
if(color_spaces[i] == from)
inf = true;
if(color_spaces[i] == to)
outf = true;
}
```

```
wf |= (!inf) | (!outf);
if(wf)
{
cerr << "command line arguments are invalid" << endl;
return 1;
}

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

img -> change_color_space(from, to);
try
{
img -> drop(outc, output);
}
catch(const std::exception& e)
{
cerr << e.what() << '\n';
delete(img);
return 1;
}
delete(img);
return 0;
}
```

image.h:

```
#pragma once

#include <vector>
#include <algorithm>
#include <stdexcept>
#include <fstream>
#include <cmath>

struct pixel {
unsigned char f, s, t;
};
```

```
using namespace std;

class Image{
private:
int width, height, color_depth;
vector<vector<pixel> > image;
public:
Image(int, string);
void drop(int, string);
void change_color_space(string, string);
};


pixel HSL_to_RGB(pixel p);

pixel RGB_to_HSL(pixel);

pixel HSV_to_RGB(pixel);

pixel RGB_to_HSV(pixel);

pixel CMY_to_RGB(pixel);

pixel RGB_to_CMY(pixel);

pixel YCoCg_to_RGB(pixel);

pixel RGB_to_YCoCg(pixel);

pixel RGB_to_YCbCr_601(pixel);

pixel YCbCr_601_to_RGB(pixel);

pixel RGB_to_YCbCr_709(pixel);

pixel YCbCr_709_to_RGB(pixel);
```

image.cpp:

```
#include "image.h"


vector<string> get_files(string pattern) {
int lastDot = -1;
for(int i = 0; i < pattern.size(); i ++)
if(pattern[i] == '.')
lastDot = i;
```

```
        if(lastDot == -1)
        throw runtime_error("pattern has no dots, so doesn't match
name.ext");
        string fp = pattern.substr(0, lastDot), sp = pattern.substr(lastDot);
        return {fp + "_" + to_string(1) + sp, fp + "_" + to_string(2) + sp,
fp + "_" + to_string(3) + sp};
        }

        Image::Image(int count, string filename) {
        if(count == 1) {
        ifstream fin(filename, ios::binary);
        if(!fin.is_open())
        throw runtime_error("failed to open file");

        char cc[2];
        fin >> cc[0] >> cc[1];
        if(cc[0] != 'P' || cc[1] != '6')
        throw runtime_error("expected P6 format");
        fin >> width >> height >> color_depth;
        if(color_depth != 255)
        throw runtime_error("only 255 color depth is supported");
        image.assign(height, vector<pixel>(width));
        char p;
        fin.read(&p, 1);
        for(int i = 0; i < height; i ++)
        for(int j = 0; j < width; j ++)
        {
        pixel cur;
        fin.read(&p, sizeof(unsigned char));
        cur.f = p;
        fin.read(&p, sizeof(unsigned char));
        cur.s = p;
        fin.read(&p, sizeof(unsigned char));
        cur.t = p;
        image[i][j] = cur;
        }
        fin.close();
        } else {
        vector<string> files = get_files(filename);
        if(files.size() != 3)
        throw runtime_error("unknown error during name generation");
        for(int fn = 0; fn < 3; fn ++)
        {
        ifstream fin(files[fn], ios::binary);
        if(!fin.is_open())
        throw runtime_error("failed to open file " + files[fn]);
```

```
        char cc[2];
        fin >> cc[0] >> cc[1];
        if(cc[0] != 'P' || cc[1] != '5')
        throw runtime_error("expected P5 format");
        fin >> width >> height >> color_depth;
        if(color_depth != 255)
        throw runtime_error("only 255 color depth is supported");
        if(image.size() == 0)
        image.assign(height, vector<pixel>(width));
        char p;
        fin.read(&p, 1);
        for(int i = 0; i < height; i ++)
        for(int j = 0; j < width; j ++)
        {
        fin.read(&p, sizeof(unsigned char));
        if(fn == 0)
        image[i][j].f = p;
        if(fn == 1)
        image[i][j].s = p;
        if(fn == 2)
        image[i][j].t = p;
        }
        fin.close();
        }
        }
        }

        void Image::drop(int count, string filename) {
        if(count == 1) {
        ofstream fout(filename, ios::binary);
        if(!fout.is_open()) {
        throw 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].f;
        fout << image[i][j].s;
        fout << image[i][j].t;
        }
        fout.flush();
        fout.close();
        } else {
        vector<string> files = get_files(filename);
        if(files.size() != 3)
```

```
        throw runtime_error("unknown error during name generation");
        for(int fn = 0; fn < 3; fn ++)
        {
        ofstream fout(files[fn], ios::binary);
        if(!fout.is_open()) {
        throw runtime_error("cannot open output file " + files[fn]);
        }
        fout << "P5\n" << width << ' ' << height << '\n' << color_depth << '\
n';
        for(int i = 0; i < height; i ++)
        for(int j = 0; j < width; j ++)
        {
        if(fn == 0)
        fout << image[i][j].f;
        if(fn == 1)
        fout << image[i][j].s;
        if(fn == 2)
        fout << image[i][j].t;
        }
        fout.flush();
        fout.close();
        }
        }
        }

        void Image::change_color_space(string from, string to) {
        // convert to rgb
        for(int i = 0; i < height; i ++)
        for(int j = 0; j < width; j ++)
        {
        if(from == "HSL")
        image[i][j] = HSL_to_RGB(image[i][j]);
        if(from == "HSV")
        image[i][j] = HSV_to_RGB(image[i][j]);
        if(from == "CMY")
        image[i][j] = CMY_to_RGB(image[i][j]);
        if(from == "YCoCg")
        image[i][j] = YCoCg_to_RGB(image[i][j]);
        if(from == "YCbCr.601")
        image[i][j] = YCbCr_601_to_RGB(image[i][j]);
        if(from == "YCbCr.709")
        image[i][j] = YCbCr_709_to_RGB(image[i][j]);
        }

        // convert from rgb to needed
        for(int i = 0; i < height; i ++)
        for(int j = 0; j < width; j ++)
```

```
{
if(to == "HSL")
image[i][j] = RGB_to_HSL(image[i][j]);
if(to == "HSV")
image[i][j] = RGB_to_HSV(image[i][j]);
if(to == "CMY")
image[i][j] = RGB_to_CMY(image[i][j]);
if(to == "YCoCg")
image[i][j] = RGB_to_YCoCg(image[i][j]);
if(to == "YCbCr.601")
image[i][j] = RGB_to_YCbCr_601(image[i][j]);
if(to == "YCbCr.709")
image[i][j] = RGB_to_YCbCr_709(image[i][j]);
}
}

pixel HSL_to_RGB(pixel px) {
double h = px.f / 255.0;
double s = px.s / 255.0;
double l = px.t / 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;

return pixel{
(unsigned char)round(r*255),
(unsigned char)round(g*255),
(unsigned char)round(b*255)
};
}

pixel RGB_to_HSL(pixel p) {
double r = p.f * 1.0 / 255;
double g = p.s * 1.0 / 255;
double b = p.t * 1.0 / 255;
double mx = max(r, max(g, b));
double mn = min(r, min(g, b));
double h, s, l = (mx + mn) / 2;
s = (mx - mn) / (1 - abs(1 - (mx + mn)));
if(mx == mn)
h = 0;
else if(r == mx && g >= b)
h = (g - b) / (mx - mn) * 60;
else if(r == mx && g < b)
h = (g - b) / (mx - mn) * 60 + 360;
else if(mx == g)
h = (b - r) / (mx-mn) * 60 + 120;
else if(mx == b)
h = (r - g) / (mx-mn)* 60 + 240;

return pixel{
(unsigned char)round(h*255 / 360),
(unsigned char)round(s*255),
(unsigned char)round(l*255)
};
}
```

```
pixel HSV_to_RGB(pixel px) {
double h = px.f / 255.0 * 360.0;
double s = px.s / 255.0;
double v = px.t / 255.0;
double c = v * s;
double hh = h / 60;
double x = c * (1 - abs(((int) hh) % 2 + (hh - (int)hh) - 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 rr = (int)(round((r + m) * 255));
int gg = (int)(round((g + m) * 255));
int bb = (int)(round((b + m) * 255));

if(rr < 0) rr = 0;
if(rr > 255) rr = 255;
if(bb < 0) bb = 0;
if(bb > 255) bb = 255;
if(gg < 0) gg = 0;
if(gg > 255) gg = 255;
return pixel{(unsigned char)rr, (unsigned char)gg, (unsigned
char)bb};
}

pixel RGB_to_HSV(pixel px) {
double r = px.f * 1.0 / 255;
double g = px.s * 1.0 / 255;
double b = px.t * 1.0 / 255;
double mx = max(r, max(g, b));
double mn = min(r, min(g, b));
double h, s, v = mx, c = mx - mn;

if(c == 0)
h = 0;
else if(v == r)
```

```
        h = (g - b) / c;
        else if(v == g)
        h = 2 + (b - r) / c;
        else
        h = 4 + (r - g) / c;

        h *= 60.0;
        if(h < 0)
        h += 360;

        s = (v == 0 ? 0 : c / v);

        return pixel{
        (unsigned char)round(h/360.0*255.0),
        (unsigned char)round(s*255.0),
        (unsigned char)round(v*255.0)
        };
        }

        pixel CMY_to_RGB(pixel px) {
        return  pixel{(unsigned  char)(255  -  px.f),  (unsigned  char)(255  -
px.s), (unsigned char)(255 - px.t)};
        }

        pixel RGB_to_CMY(pixel px) {
        return  pixel{(unsigned  char)(255  -  px.f),  (unsigned  char)(255  -
px.s), (unsigned char)(255 - px.t)};
        }

        pixel RGB_to_YCoCg(pixel px) {
        double r = px.f * 1.0 / 255;
        double g = px.s * 1.0 / 255;
        double b = px.t * 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;
        if(y > 1.0) y = 1.0;
        if(y < 0) y = 0;
        if(co > 1.0) co = 1.0;
        if(co < 0) co = 0;
        if(cg > 1.0) cg = 1.0;
        if(cg < 0) cg = 0;
        return pixel{
        (unsigned char)(y*255),
        (unsigned char)(co*255),
        (unsigned char)(cg*255)
        };
```

```
}

pixel YCoCg_to_RGB(pixel px) {
double y = px.f * 1.0 / 255;
double co = px.s * 1.0 / 255 - 0.5;
double cg = px.t * 1.0 / 255 - 0.5;
double r = y + co - cg;
double g = y + cg;
double b = y - co - cg;

if(r > 1.0) r = 1.0;
if(r < 0) r = 0;
if(g > 1.0) g = 1.0;
if(g < 0) g = 0;
if(b > 1.0) b = 1.0;
if(b < 0) b = 0;

return pixel{
(unsigned char)(r*255),
(unsigned char)(g*255),
(unsigned char)(b*255)
};
}

pixel RGB_to_YCbCr_601(pixel px) {
double r = px.f / 255.0;
double g = px.s / 255.0;
double b = px.t / 255.0;
double K_b = 0.299;
double K_r = 0.587;
double K_g = 0.114;

double Y = K_r*r + K_g*g + K_b*b;
double C_b = (b - Y) / (2 * (1 - K_b));
double C_r = (r - Y) / (2 * (1 - K_r));

int rr = round(Y * 255), gg = round((C_b + 0.5) * 255), bb =
round((C_r + 0.5) * 255);
if(rr < 0) rr = 0;
if(rr > 255) rr = 255;
if(bb < 0) bb = 0;
if(bb > 255) bb = 255;
if(gg < 0) gg = 0;
if(gg > 255) gg = 255;
return pixel{(unsigned char)rr, (unsigned char)gg, (unsigned
char)bb};
}
```

```
	pixel YCbCr_601_to_RGB(pixel px) {
	double Y = px.f / 255.0;
	double C_b = px.s / 255.0 - 0.5;
	double C_r = px.t / 255.0 - 0.5;
	double K_b = 0.299;
	double K_r = 0.587;
	double K_g = 0.114;
	double r = Y + (2.0 - 2.0 * K_r) * C_r;
	double g = Y - K_b * (2.0 - 2.0 * K_b) * C_b / K_g - K_r * (2 - 2.0 *
K_r) * C_r / K_g;
	double b = Y + (2.0 - 2.0 * K_b) * C_b;
	int rr = round(r * 255), gg = round(g * 255), bb = round(b * 255);
	if(rr > 255) rr = 255;
	if(rr < 0) rr = 0;
	if(gg > 255) gg = 255;
	if(gg < 0) gg = 0;
	if(bb > 255) bb = 255;
	if(bb < 0) bb = 0;
	return pixel{(unsigned char)rr, (unsigned char)gg, (unsigned
char)bb};
	}

	pixel RGB_to_YCbCr_709(pixel px) {
	double r = px.f / 255.0;
	double g = px.s / 255.0;
	double b = px.t / 255.0;
	double K_b = 0.0722;
	double K_r = 0.2126;
	double K_g = 0.7152;

	double Y = K_r*r + K_g*g + K_b*b;
	double C_b = (b - Y) / (2 * (1 - K_b));
	double C_r = (r - Y) / (2 * (1 - K_r));

	int rr = round(Y * 255), gg = round((C_b + 0.5) * 255), bb =
round((C_r + 0.5) * 255);
	if(rr < 0) rr = 0;
	if(rr > 255) rr = 255;
	if(bb < 0) bb = 0;
	if(bb > 255) bb = 255;
	if(gg < 0) gg = 0;
	if(gg > 255) gg = 255;
	return pixel{(unsigned char)rr, (unsigned char)gg, (unsigned
char)bb};
	}
```

```
        pixel YCbCr_709_to_RGB(pixel px) {
        double Y = px.f / 255.0;
        double C_b = px.s / 255.0 - 0.5;
        double C_r = px.t / 255.0 - 0.5;
        double K_b = 0.0722;
        double K_r = 0.2126;
        double K_g = 0.7152;
        double r = Y + (2 - 2 * K_r) * C_r;
        double g = Y - K_b * (2 - 2 * K_b) * C_b / K_g - K_r * (2 - 2 * K_r)
* C_r / K_g;
        double b = Y + (2 - 2 * K_b) * C_b;
        int rr = round(r * 255), gg = round(g * 255), bb = round(b * 255);
        if(rr > 255) rr = 255;
        if(rr < 0) rr = 0;
        if(gg > 255) gg = 255;
        if(gg < 0) gg = 0;
        if(bb > 255) bb = 255;
        if(bb < 0) bb = 0;
        return   pixel{(unsigned   char)rr,   (unsigned   char)gg,   (unsigned
char)bb};
        }
```