**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное

учреждение высшего образования

**Санкт-Петербургский исследовательский университет**

**Информационных технологий, механики и оптики**

Факультет: ФИТиП

Дисциплина: Компьютерная графика и геометрия

**Отчет**

по лабораторной работе № 4

***Изучение цветовых пространств.***

Выполнила: студент гр. M3101

Дымчикова А. Ч.

Преподаватель: Скаков П.С.

Санкт-Петербург

2020

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

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

**Описание работы**

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:

С: 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 представляют собой различные системы, с помощью которых можно хранить информацию о цвете. По аналогии с человеческим зрением, для которого используется три вида рецепторов, за кодирование информации отвечает трехмерное цветовое пространство. Переход от одного пространства к другому можно представить как изменение базиса системы координат: меняется только форма хранения, само значение остается неизменным.

Существуют аддитивные (RGB) и субтрактивные (CMY) цветовые пространства. В аддитивных – нули соответствуют черному цвету, а максимальные значения каждой компоненты соответствуют черному. В субтрактивных наоборот – нули отвечают за белый цвет, а максимальные значения отвечают за черный.

Рассмотрим каждое из используемых в данной лабораторной пространств подробнее:

* RGB

Данное пространство – это самое широко используемое цветовое пространство. Его компоненты примерно соответствуют трем видам наших цветовых рецепторов.

R (Red) – красный

G (Green) – зеленый

B (Blue) – синий

Обычно используется диапазон значений [0; 255].

* HSL

Это пространство используется в основном в интерфейсах выбора цвета из-за интуитивно-понятного изменения своих характеристик.

H (Hue) – оттенок: диапазон [0; 360] или [0; 1]

S (Saturation) – насыщенность: [0; 100] или [0; 1]

L (Lightness) – светлота: [0; 100] или [0; 1]

* HSV

Это пространство так же, как и HSL, используется в основном в интерфейсах выбора цвета из-за интуитивно-понятного изменения своих характеристик.

H (Hue) – оттенок: диапазон: [0; 360] или [0; 1]

S (Saturation) – насыщенность: [0; 100] или [0; 1]

V (Value) – яркость: [0; 100] или [0; 1]

* YCbCr

Данное пространство 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: [0; 1]

Cb – цветоразность «хроматический синий»: [-0,5; 0,5]

Cr – цветоразность «хроматический красный»: [-0,5; 0,5]

* YCoCg

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

Y – яркость: [0; 1]

Cg – цветоразность «хроматический зеленый»: [-0,5; 0,5]

Co – цветоразность «хроматический оранжевый»: [-0,5; 0,5]

* 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.

С (Cyan) – голубой: [0; 255]

M (Magenta) – пурпурный: [0; 255]

Y(Yellow) – жёлтый: [0; 255]

K (blacK) – чёрный

**Экспериментальная часть**

Выполнено полное решение лабораторной работы.

Язык программирования: С++17.

Необходимо выполнить переход из одного цветового пространства в другое. Для удобства исходное пространство переводится в RGB-пространство, и затем выполняется перевод из RGB-пространства в конечное пространство.

Для хранения в PNM-файле из-за ограничений формата все значения приводятся к диапазону [0; 255].

Время работы алгоритма прямопропорционально количеству пикселей в изображении и максимуму из времени работы функций перехода из исходного пространства в RGB-пространство и из него в конечное пространство.

**Выводы**

В результате выполненной работы были получены знания о представленных в лабораторной работе цветовых пространствах и переходах между ними. Также были получены знания о потерях при переходе из одного пространства в другое.

**Листинг**

**PNM\_ex.h**

#pragma once

#include <vector>

#include <string>

**class** **PNM\_Header** {

**public:**

**char** format;

**int** width;

**int** height;

**int** depth;

PNM\_Header();

PNM\_Header(**FILE** \*fin);

};

**class** **PNM\_File** {

**public:**

PNM\_Header header;

std::vector <**unsigned** **char**> data;

**int** size = **0**;

PNM\_File();

PNM\_File(**FILE** \*fin);

PNM\_File(std::vector<PNM\_File>&);

PNM\_File(PNM\_File&, **int**);

**void** **write**(**FILE** \*fout);

};

**PNM\_ex.cpp**

#include "PNM\_ex.h"

PNM\_Header::PNM\_Header() {};

PNM\_Header::PNM\_Header(**FILE** \*fin) {

**char** checker;

**int** check = fscanf(fin, "%c%c**\n**%d %d**\n**%d**\n**", &checker, &format, &width, &height, &depth);

**if** (check != **5** || checker != 'P' || format != '5' && format != '6'

|| width <= **0** || height <= **0** || depth != **255**) {

**throw** std::exception("Invalid data format!**\n**");

}

}

PNM\_File::PNM\_File() {};

PNM\_File::PNM\_File(**FILE** \*fin) {

header = PNM\_Header(fin);

**if** (header.format == '5') {

size = header.width \* header.height;

}

**else** {

size = header.width \* header.height \* **3**;

}

data.resize(size);

**int** check = fread(&data[**0**], **sizeof**(**unsigned** **char**), size, fin);

**if** (check != size) {

**throw** std::exception("Invalid data format: width or**\a**nd height are wrong!**\n**");

}

}

PNM\_File::PNM\_File(std::vector<PNM\_File> &files) {

header = files[**0**].header;

header.format = '6';

size = files[**0**].size \* **3**;

data.resize(size);

**int** w = header.width;

**int** h = header.height;

**for** (**int** i = **0**; i < h; i++) {

**for** (**int** j = **0**; j < w; j++) {

**for** (**int** k = **0**; k < **3**; k++) {

data[i \* w \* **3** + j \* **3** + k] = files[k].data[i \* w + j];

}

}

}

}

PNM\_File::PNM\_File(PNM\_File &file, **int** ch) {

header = file.header;

header.format = '5';

size = file.size / **3**;

data.resize(size);

**int** w = header.width;

**int** h = header.height;

**for** (**int** i = **0**; i < h; i++) {

**for** (**int** j = **0**; j < w; j++) {

data[i \* w + j] = file.data[i \* w \* **3** + j \* **3** + ch];

}

}

}

**void** PNM\_File::write(**FILE** \*fout) {

fprintf(fout, "P%c**\n**%d %d**\n**%d**\n**", header.format, header.width, header.height, header.depth);

fwrite(&data[**0**], **sizeof**(**unsigned** **char**), size, fout);

fclose(fout);

}

**Functs.h**

# pragma once

#include <algorithm>

#include "PNM\_ex.h"

**const** **int** INF = **1e9**;

**const** **double** EPS = **1e-9**;

**class** **component** {

**public:**

**double** a;

**double** b;

**double** c;

component();

component(**double**, **double**, **double**);

};

**void** **check**(std::vector<PNM\_File> &files);

component **get\_component**(std::vector<PNM\_File> &files, **int** i, **int** j);

**void** **write\_component**(std::vector<PNM\_File> &files, **int** i, **int** j, component &new\_color);

**void** **to\_HSV**(std::vector<PNM\_File> &files);

**void** **from\_HSV**(std::vector<PNM\_File> &files);

**void** **to\_HSL**(std::vector<PNM\_File> &files);

**void** **from\_HSL**(std::vector<PNM\_File> &files);

component **to\_YCbCr**(**double** Kb, **double** Kr, component color);

component **from\_YCbCr**(**double** Kb, **double** Kr, component color);

**void** **from\_YCbCr601**(std::vector<PNM\_File> &files);

**void** **to\_YCbCr601**(std::vector<PNM\_File> &files);

**void** **from\_YCbCr709**(std::vector<PNM\_File> &files);

**void** **to\_YCbCr709**(std::vector<PNM\_File> &files);

**void** **to\_YCoCg**(std::vector<PNM\_File> &files);

**void** **from\_YCoCg**(std::vector<PNM\_File> &files);

**void** **to\_CMY**(std::vector<PNM\_File> &files);

**void** **from\_CMY**(std::vector<PNM\_File> &files);

**void** **foo**(std::vector<PNM\_File> &files);

**void** **convert**(std::vector<PNM\_File> &files, **int** from, **int** to);

**Functs.cpp**

#include <algorithm>

#include <vector>

#include <exception>

#include "PNM\_ex.h"

#include "Functs.h"

component::component(): a(**0**), b(**0**), c(**0**) {}

component::component(**double** a, **double** b, **double** c): a(a), b(b), c(c) {}

**void** check(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

**if** (files.size() == **3** &&

(files[**1**].header.height != h || files[**2**].header.height != h ||

files[**1**].header.width != w || files[**2**].header.width != w)) {

**throw** std::exception("Wrong parametres of width or**\a**nd height!**\n**");

}

}

component get\_component(std::vector<PNM\_File> &files, **int** i, **int** j) {

**int** w = files[**0**].header.width;

component color;

**if** (files.size() == **1**) {

**if** (files[**0**].data[i \* w \* **3** + j \* **3** + **1**] == **46**) {

**int** t = **5**;

}

color.a = **double**(files[**0**].data[i \* w \* **3** + j \* **3** + **0**]) / **255**;

color.b = **double**(files[**0**].data[i \* w \* **3** + j \* **3** + **1**]) / **255**;

color.c = **double**(files[**0**].data[i \* w \* **3** + j \* **3** + **2**]) / **255**;

}

**else** {

color.a = **double**(files[**0**].data[i \* w + j]) / **255**;

color.b = **double**(files[**1**].data[i \* w + j]) / **255**;

color.c = **double**(files[**2**].data[i \* w + j]) / **255**;

}

**return** color;

}

**void** write\_component(std::vector<PNM\_File> &files, **int** i, **int** j, component &new\_color) {

**int** w = files[**0**].header.width;

component color;

**if** (files.size() == **1**) {

files[**0**].data[i \* w \* **3** + j \* **3** + **0**] = new\_color.a;

files[**0**].data[i \* w \* **3** + j \* **3** + **1**] = new\_color.b;

files[**0**].data[i \* w \* **3** + j \* **3** + **2**] = new\_color.c;

}

**else** {

files[**0**].data[i \* w + j] = new\_color.a;

files[**1**].data[i \* w + j] = new\_color.b;

files[**2**].data[i \* w + j] = new\_color.c;

}

}

**void** to\_HSV(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

check(files);

**for** (**int** j = **0**; j < h; j++) {

**for** (**int** k = **0**; k < w; k++) {

component color;

color = get\_component(files, j, k);

component new\_color;

new\_color.c = std::max(color.a, std::max(color.b, color.c));

**double** sub = new\_color.c - std::min(color.a, std::min(color.b, color.c));

**if** (sub == **0**) {

new\_color.a = **0**;

} **else** **if**(abs(new\_color.c - color.a) < EPS){

new\_color.a = (color.b - color.c) / sub;

} **else** **if** (abs(new\_color.c - color.b) < EPS) {

new\_color.a = (color.c - color.a) / sub + **2**;

} **else** {

new\_color.a = (color.a - color.b) / sub + **4**;

}

new\_color.a \*= **60.**;

**if** (new\_color.a < **0**) {

new\_color.a += **360**;

}

**if** (new\_color.c == **0**) {

new\_color.b = **0**;

} **else** {

new\_color.b = sub / new\_color.c;

}

new\_color.a = round(new\_color.a / **360** \* **255**);

new\_color.b = round(new\_color.b \* **255**);

new\_color.c = round(new\_color.c \* **255**);

write\_component(files, j, k, new\_color);

}

}

}

**void** from\_HSV(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

check(files);

**for** (**int** j = **0**; j < h; j++) {

**for** (**int** k = **0**; k < w; k++) {

component color;

color = get\_component(files, j, k);

color.a \*= **360**;

component new\_color;

**double** mul = color.b \* color.c;

**double** f = color.a / **60**;

**double** x = (**1** - abs((f - **int**(f)) + **int**(f) % **2** - **1**)) \* mul;

**if** (color.a < **60**) {

new\_color.a = mul;

new\_color.b = x;

} **else** **if**(color.a < **120**) {

new\_color.a = x;

new\_color.b = mul;

} **else** **if**(color.a < **180**) {

new\_color.b = mul;

new\_color.c = x;

} **else** **if**(color.a < **240**) {

new\_color.b = x;

new\_color.c = mul;

} **else** **if**(color.a < **300**) {

new\_color.a = x;

new\_color.c = mul;

}**else** **if**(color.a < **360**) {

new\_color.a = mul;

new\_color.c = x;

}

new\_color.a += color.c - mul;

new\_color.b += color.c - mul;

new\_color.c += color.c - mul;

new\_color.a = round(new\_color.a \* **255.**);

new\_color.b = round(new\_color.b \* **255.**);

new\_color.c = round(new\_color.c \* **255.**);

write\_component(files, j, k, new\_color);

}

}

}

**void** to\_HSL(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

check(files);

**for** (**int** j = **0**; j < h; j++) {

**for** (**int** k = **0**; k < w; k++) {

component color;

color = get\_component(files, j, k);

component new\_color;

**double** max\_c = std::max(color.a, std::max(color.b, color.c));

**double** min\_c = std::min(color.a, std::min(color.b, color.c));

new\_color.c = (max\_c + min\_c) / **2**;

**if** (new\_color.c == **0** || max\_c == min\_c) {

new\_color.b = **0**;

} **else** **if**(new\_color.c <= **0.5**) {

new\_color.b = (max\_c - min\_c) / (**2** \* new\_color.c);

} **else** {

new\_color.b = (max\_c - min\_c) / (**2** - **2** \* new\_color.c);

}

**if** (max\_c == min\_c) {

new\_color.a = **0**;

} **else** **if**(abs(max\_c - color.a) < EPS && color.b >= color.c) {

new\_color.a = (color.b - color.c) / (max\_c - min\_c) \* **60**;

} **else** **if** (abs(max\_c - color.a) < EPS && color.b < color.c) {

new\_color.a = (color.b - color.c) / (max\_c - min\_c) \* **60** + **360**;

} **else** **if** (abs(max\_c - color.b) < EPS) {

new\_color.a = (color.c - color.a) / (max\_c - min\_c) \* **60** + **120**;

} **else** **if** (abs(max\_c - color.c) < EPS) {

new\_color.a = (color.a - color.b) / (max\_c - min\_c) \* **60** + **240**;

}

new\_color.a = round(new\_color.a / **360** \* **255**);

new\_color.b = round(new\_color.b \* **255**);

new\_color.c = round(new\_color.c \* **255**);

write\_component(files, j, k, new\_color);

}

}

}

**void** from\_HSL(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

check(files);

**for** (**int** j = **0**; j < h; j++) {

**for** (**int** k = **0**; k < w; k++) {

component color;

color = get\_component(files, j, k);

component new\_color;

**double** q;

**if** (color.c < **0.5**) {

q = color.c \* (color.b + **1.**);

} **else** {

q = color.c + color.b - (color.c \* color.b);

}

**double** p = **2.** \* color.c - q;

**double** t[**3**], col[**3**];

t[**0**] = color.a + **1.** / **3.**;

t[**1**] = color.a;

t[**2**] = color.a - **1.** / **3.**;

**for** (**int** f = **0**; f < **3**; f++) {

**if** (t[f] < **0**) {

t[f] = t[f] + **1**;

} **else** **if** (t[f] > **1**) {

t[f] = t[f] - **1**;

}

**double** c;

**if** (t[f] < **1.** / **6.**) {

col[f] = p + ((q - p) \* **6** \* t[f]);

} **else** **if**(t[f] < **0.5**) {

col[f] = q;

} **else** **if** (t[f] < **2.** / **3.**) {

col[f] = p + ((q - p) \* (**2.** / **3.** - t[f]) \* **6**);

} **else** {

col[f] = p;

}

}

new\_color.a = round(col[**0**] \* **255**);

new\_color.b = round(col[**1**] \* **255**);

new\_color.c = round(col[**2**] \* **255**);

write\_component(files, j, k, new\_color);

}

}

}

component to\_YCbCr(**double** Kb, **double** Kr, component color) {

component new\_color;

new\_color.a = Kr \* color.a + (**1** - Kr - Kb) \* color.b + Kb \* color.c;

new\_color.b = **0.5** \* (color.c - new\_color.a) / (**1** - Kb);

new\_color.c = **0.5** \* (color.a - new\_color.a) / (**1** - Kr);

new\_color.a = round(new\_color.a \* **255**);

new\_color.b = round((new\_color.b + **0.5**) \* **255**);

new\_color.c = round((new\_color.c + **0.5**) \* **255**);

**return** new\_color;

}

**void** check\_borders(**double** &x) {

**if** (x > **1**) {

x = **1**;

}

**if** (x < **0**) {

x = **0**;

}

}

component from\_YCbCr(**double** Kb, **double** Kr, component color) {

color.b -= **0.5**;

color.c -= **0.5**;

component new\_color;

new\_color.a = color.a + (**1** - Kr) \* color.c / **0.5**;

new\_color.b = color.a - **2** \* Kb \* (**1** - Kb) \* color.b / (**1** - Kb - Kr) - **2** \* Kr \* (**1** - Kr) \* color.c / (**1** - Kb - Kr);

new\_color.c = color.a + (**1** - Kb) \* color.b / **0.5**;

check\_borders(new\_color.a);

check\_borders(new\_color.b);

check\_borders(new\_color.c);

new\_color.a = round(new\_color.a \* **255**);

new\_color.b = round(new\_color.b \* **255**);

new\_color.c = round(new\_color.c \* **255**);

**return** new\_color;

}

**void** from\_YCbCr601(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

check(files);

**for** (**int** j = **0**; j < h; j++) {

**for** (**int** k = **0**; k < w; k++) {

component color;

color = get\_component(files, j, k);

component new\_color = from\_YCbCr(**0.114**, **0.299**, color);

write\_component(files, j, k, new\_color);

}

}

}

**void** to\_YCbCr601(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

check(files);

**for** (**int** j = **0**; j < h; j++) {

**for** (**int** k = **0**; k < w; k++) {

component color;

color = get\_component(files, j, k);

component new\_color = to\_YCbCr(**0.114**, **0.299**, color);

write\_component(files, j, k, new\_color);

}

}

}

**void** from\_YCbCr709(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

check(files);

**for** (**int** j = **0**; j < h; j++) {

**for** (**int** k = **0**; k < w; k++) {

component color;

color = get\_component(files, j, k);

component new\_color = from\_YCbCr(**0.0722**, **0.2126**, color);

write\_component(files, j, k, new\_color);

}

}

}

**void** to\_YCbCr709(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

check(files);

**for** (**int** j = **0**; j < h; j++) {

**for** (**int** k = **0**; k < w; k++) {

component color;

color = get\_component(files, j, k);

component new\_color = to\_YCbCr(**0.0722**, **0.2126**, color);

write\_component(files, j, k, new\_color);

}

}

}

**void** to\_YCoCg(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

check(files);

**for** (**int** j = **0**; j < h; j++) {

**for** (**int** k = **0**; k < w; k++) {

component color;

color = get\_component(files, j, k);

component new\_color;

new\_color.a = **0.25** \* color.a + **0.5** \* color.b + **0.25** \* color.c;

new\_color.b = **0.5** \* color.a - **0.5** \* color.c;

new\_color.c = -**0.25** \* color.a + **0.5** \* color.b - **0.25** \* color.c;

check\_borders(new\_color.a);

new\_color.a = round(new\_color.a \* **255**);

new\_color.b = round((new\_color.b + **0.5**) \* **255**);

new\_color.c = round((new\_color.c + **0.5**) \* **255**);

write\_component(files, j, k, new\_color);

}

}

}

**void** from\_YCoCg(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

check(files);

**for** (**int** j = **0**; j < h; j++) {

**for** (**int** k = **0**; k < w; k++) {

component color;

color = get\_component(files, j, k);

color.b -= **0.5**;

color.c -= **0.5**;

component new\_color;

new\_color.a = color.a + color.b - color.c;

new\_color.b = color.a + color.c;

new\_color.c = color.a - color.b - color.c;

check\_borders(new\_color.a);

check\_borders(new\_color.b);

check\_borders(new\_color.c);

new\_color.a = round(new\_color.a \* **255**);

new\_color.b = round(new\_color.b \* **255**);

new\_color.c = round(new\_color.c \* **255**);

write\_component(files, j, k, new\_color);

}

}

}

**void** to\_CMY(std::vector<PNM\_File> &files) {

**int** h = files[**0**].header.height;

**int** w = files[**0**].header.width;

check(files);

**for** (**int** j = **0**; j < h; j++) {

**for** (**int** k = **0**; k < w; k++) {

component color;

**if** (files.size() == **1**) {

**if** (files[**0**].data[j \* w \* **3** + k \* **3** + **1**] == **133**) {

**int** t = **5**;

}

color.a = (files[**0**].data[j \* w \* **3** + k \* **3** + **0**]);

color.b = (files[**0**].data[j \* w \* **3** + k \* **3** + **1**]);

color.c = (files[**0**].data[j \* w \* **3** + k \* **3** + **2**]);

}

**else** {

color.a = (files[**0**].data[j \* w + k]);

color.b = (files[**1**].data[j \* w + k]);

color.c = (files[**2**].data[j \* w + k]);

}

component new\_color;

new\_color.a = **255** - color.a;

new\_color.b = **255** - color.b;

new\_color.c = **255** - color.c;

write\_component(files, j, k, new\_color);

}

}

}

**void** from\_CMY(std::vector<PNM\_File> &files) {

to\_CMY(files);

}

**void** foo(std::vector<PNM\_File> &files) {

//doing nothing

}

**void** convert(std::vector<PNM\_File> &files, **int** from, **int** to) {

**void**(\*convfrom[**7**])(std::vector<PNM\_File>&) = { foo, from\_HSL, from\_HSV, from\_YCbCr601, from\_YCbCr709, from\_YCoCg, from\_CMY };

**void**(\*convto[**7**])(std::vector<PNM\_File>&) = { foo, to\_HSL, to\_HSV, to\_YCbCr601, to\_YCbCr709, to\_YCoCg, to\_CMY };

convfrom[from](files);

convto[to](files);

}

**main.cpp**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <map>

#include "PNM\_ex.h"

#include "Functs.h"

#include <stdlib.h>

**using** **namespace** std;

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

**if** (argc != **11**) {

cerr << "Wrong number of arguments!";

**return** **1**;

}

vector<**FILE** \*>fin;

**int** from = -**1**;

**int** to = -**1**;

**int** count = **0**;

**int** num = **0**;

map <string, **int**> space\_numbers = { {"RGB", **0**}, {"HSL", **1**}, {"HSV", **2**}, {"YCbCr.601", **3**},

{"YCbCr.709", **4**}, {"YCoCg", **5**}, {"CMY", **6**} };

//RGB - 0, HSL - 1, HSV - 2, YCbCr.601 - 3, YCbCr.709 - 4, YCoCg - 5, CMY - 6

string name;

**for** (**int** i = **1**; i < argc;) {

**switch** (argv[i][**1**]) {

**case** 'f':

i++;

from = space\_numbers.at(argv[i]);

i++;

**break**;

**case** 't':

i++;

to = space\_numbers.at(argv[i]);

i++;

**break**;

**case** 'i':

i++;

count = atoi(argv[i]);

i++;

name = argv[i];

//char buf[1];

i++;

**for** (**int** j = **0**; j < count; j++) {

//itoa(i - st + 1, , 10);

string new\_name = name;

**if** (count == **3**) {

new\_name.insert(new\_name.find\_last\_of('.'), "\_");

new\_name.insert(new\_name.find\_last\_of('.'), to\_string(j + **1**));

}

fin.push\_back(fopen(new\_name.c\_str(), "rb"));

**if** (!fin[j]) {

**for** (**int** k = **0**; k < **3**; k++) {

**if** (fin[k]) {

fclose(fin[k]);

}

}

cerr << "Input file error!**\n**";

cerr << new\_name << '\n';

**return** **1**;

}

}

**break**;

**case** 'o':

i++;

num = atoi(argv[i]);

i++;

i++;

**break**;

**default:**

cerr << "Wrong operation!";

**return** **1**;

}

}

vector<PNM\_File> files(count);

**for** (**int** i = **0**; i < count; i++) {

try {

files[i] = PNM\_File(fin[i]);

}

**catch** (exception err) {

fclose((fin[i]));

cerr << err.what();

**return** **1**;

}

fclose(fin[i]);

}

convert(files, from, to);

vector <**FILE** \*> fout;

**for** (**int** i = **1**; i < argc;) {

**if** (argv[i][**1**] == 'o' && argv[i][**0**] == '-') {

i++;

count = atoi(argv[i]);

i++;

string name = argv[i];

i++;

**for** (**int** j = **0**; j < count; j++) {

//itoa(i - st + 1, buf, 10);

string new\_name = name;

**if** (count == **3**) {

new\_name.insert(new\_name.find\_last\_of('.'), "\_");

new\_name.insert(new\_name.find\_last\_of('.'), to\_string(j + **1**));

}

fout.push\_back(fopen(new\_name.c\_str(), "wb"));

**if** (!fout[j]) {

**for** (**int** k = **0**; k < **3**; k++) {

**if** (fout[k]) {

fclose(fout[k]);

}

}

cerr << "Output file error!**\n**";

cerr << argv[i] << '\n';

**return** **1**;

}

}

}

**else** {

i++;

}

}

vector <PNM\_File> res;

**if** (count != files.size()) {

**if** (count == **3**) {

res.push\_back(PNM\_File(files[**0**], **0**));

res.push\_back(PNM\_File(files[**0**], **1**));

res.push\_back(PNM\_File(files[**0**], **2**));

}

**else** {

res.push\_back(PNM\_File(files));

}

}

**else** {

res = files;

}

**for** (**int** i = **0**; i < count; i++) {

res[i].write(fout[i]);

}

**return** **0**;

}