**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное

учреждение высшего образования

**Санкт-Петербургский исследовательский университет**

**Информационных технологий, механики и оптики**

Факультет информационных технологий и программирования

Дисциплина: компьютерная геометрия и графика

**Отчет**

по лабораторной работе №4

***Изучение цветовых пространств***

Выполнил: студент гр. M3102

Муслимов О. С.

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 соответствуют различным системам представления информации о цвете. Так как в соответствии с моделью зрения человека мы имеем 3 вида рецепторов, отвечающих за цветное зрение, то и для кодирования информации о цвете разумно использовать трёхмерное цветовое пространство. Переход от одного цветового пространства к другому можно представить себе как изменение базиса системы координат: значения меняются, но информация остаётся.

В данной лабораторной работе мы работам с 6 цветовыми пространствами.

1. 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

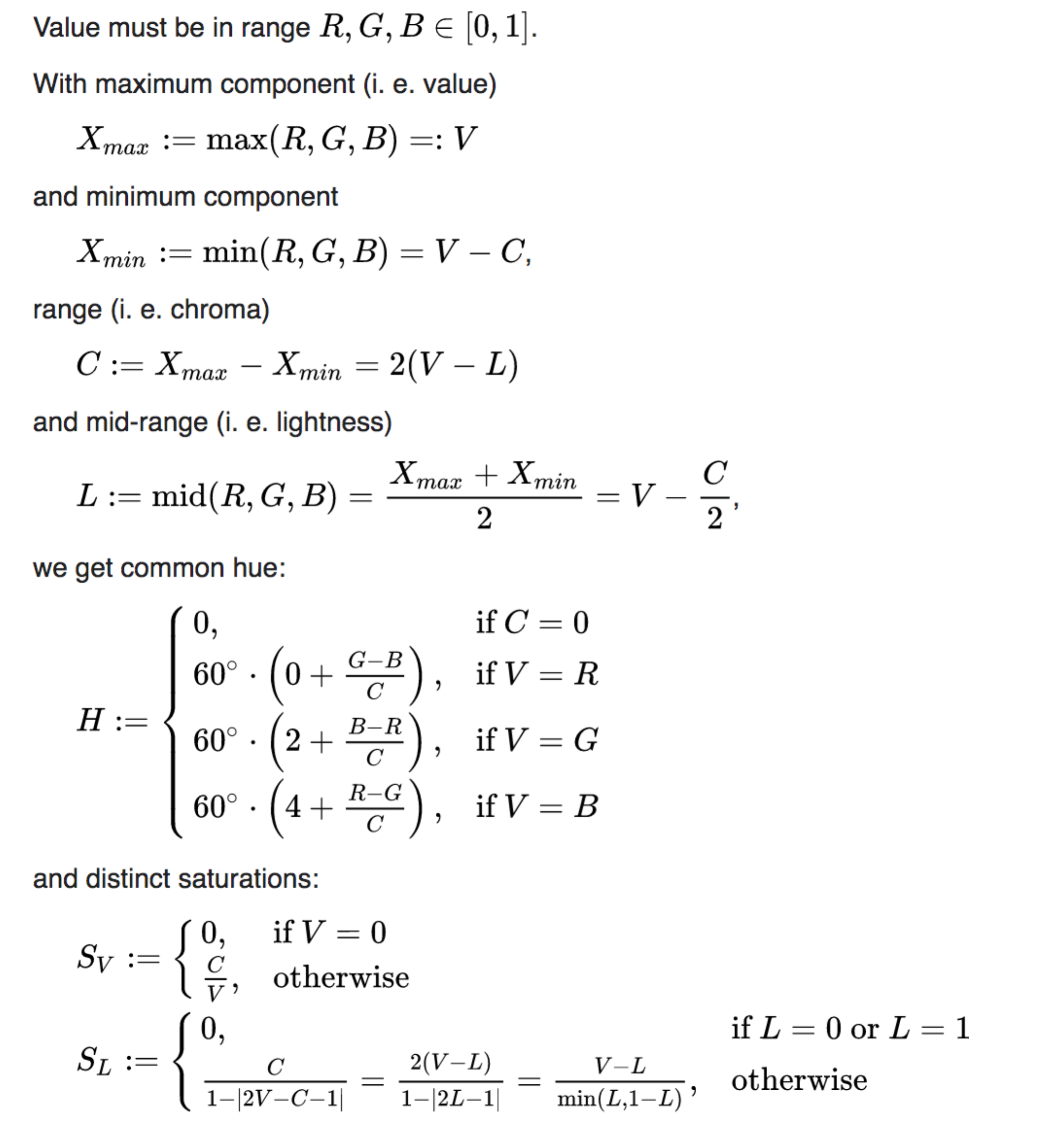
Типичный диапазон значений: 0..255 для каждой компоненты

2. HSL и HSV:

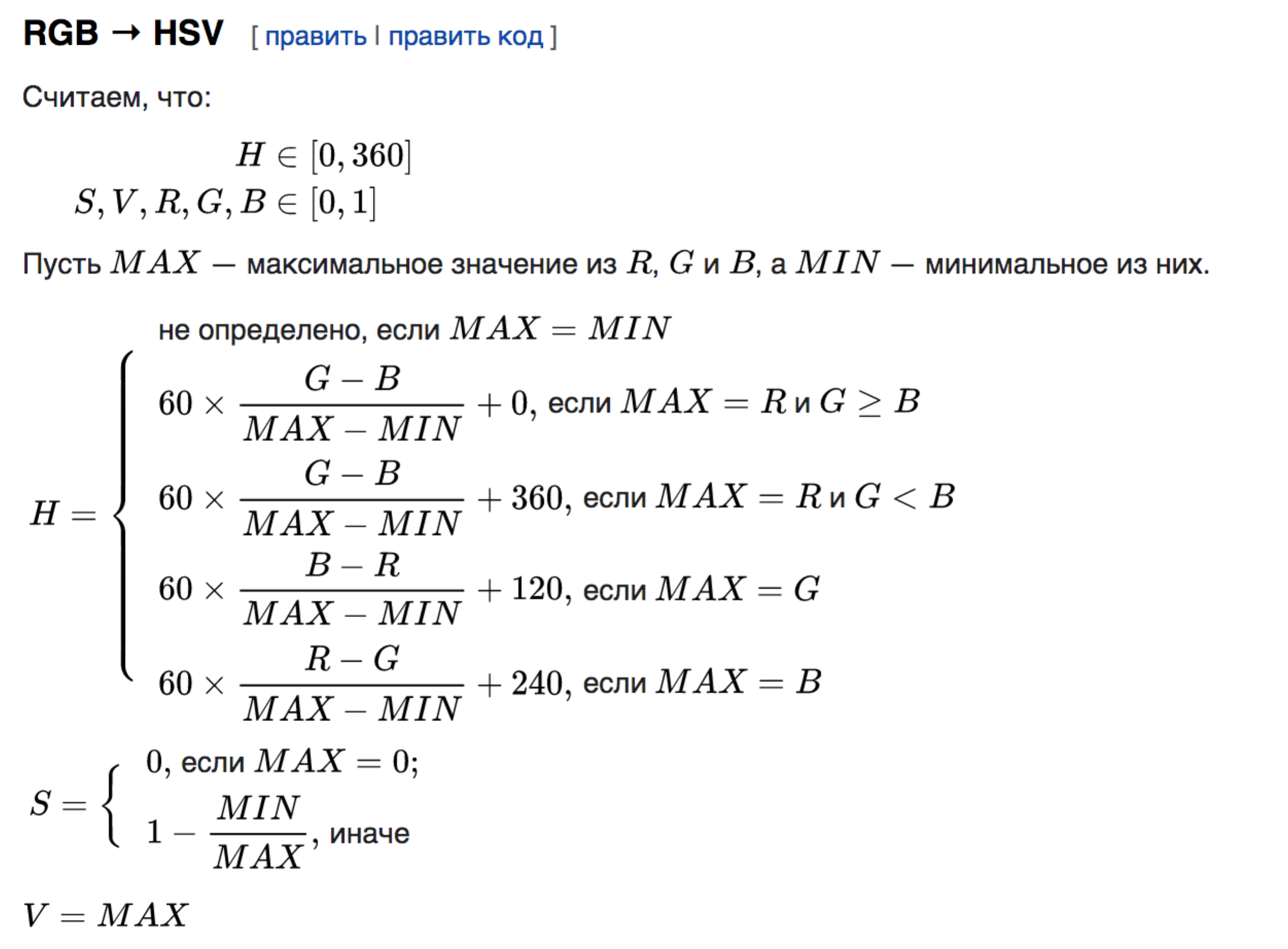
Пространства HSL (другие названия: HLS, HSI) и HSV (другое название: HSB) широко используются в интерфейсах выбора цвета. Предназначены для “интуитивно понятного” изменения таких характеристик цвета как: оттенок, насыщенность, яркость.

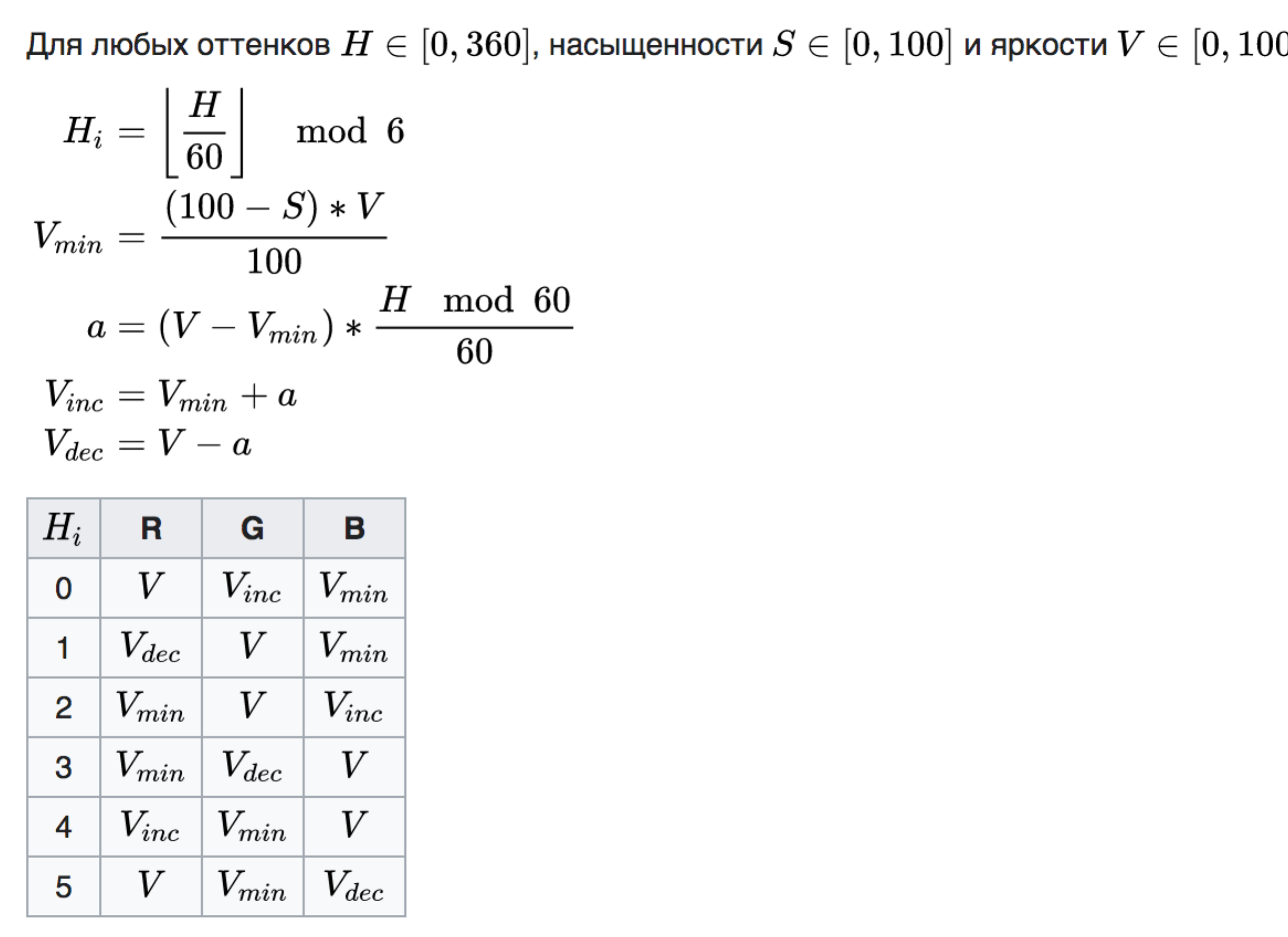
H (Hue) – оттенок: диапазон 0..360°, 0..100 или 0..1  
S (Saturation) – насыщенность: 0..100 или 0..1  
L/I (Lightness/Intensity) – “светлота”: 0..100 или 0..1  
V/B (Value/Brightness) – “яркость”: 0..100 или 0..1

RGB to HSL:

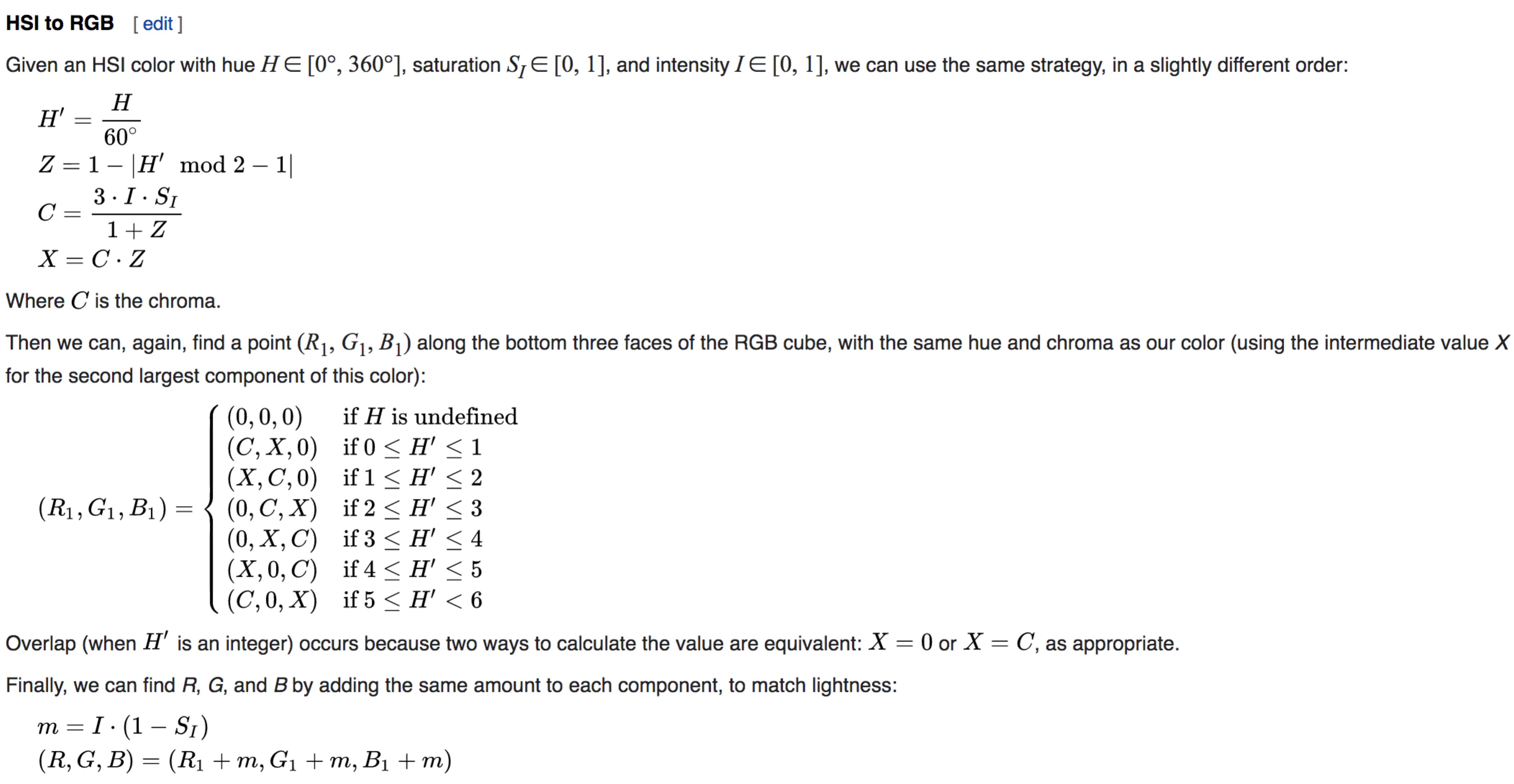


RGB to HSV:



HSV to RGB:  


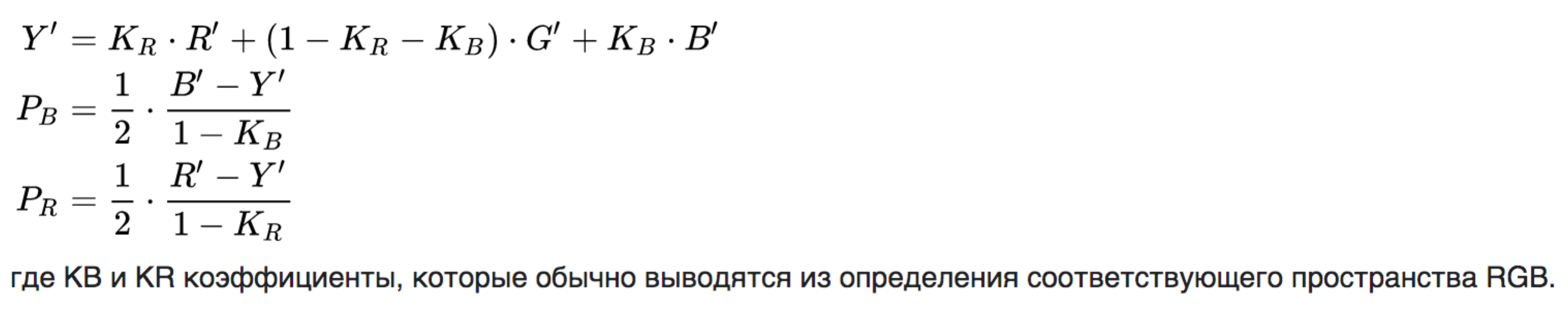
HSL to RGB:



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”

|  |  |  |
| --- | --- | --- |
|  | Kr | Kb |
| 601 | 0.299 | 0.114 |
| 709 | 0.2126 | 0.0722 |

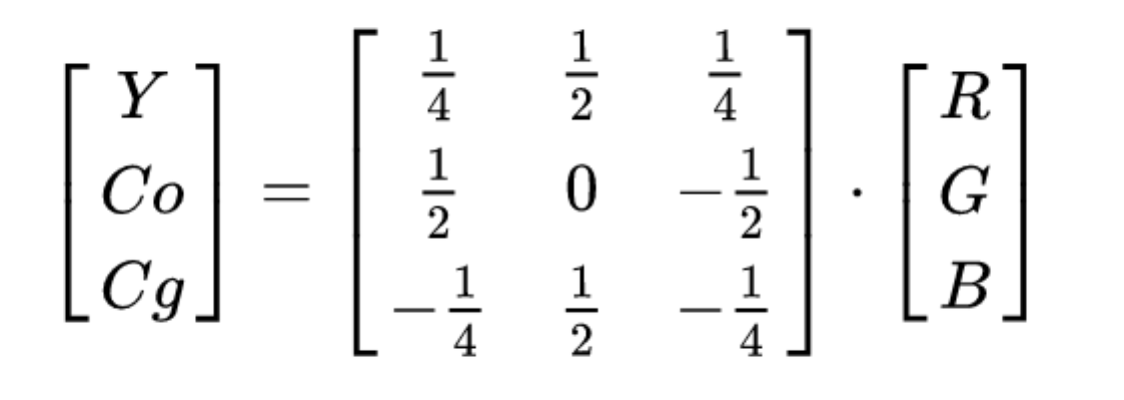


YCoCg:

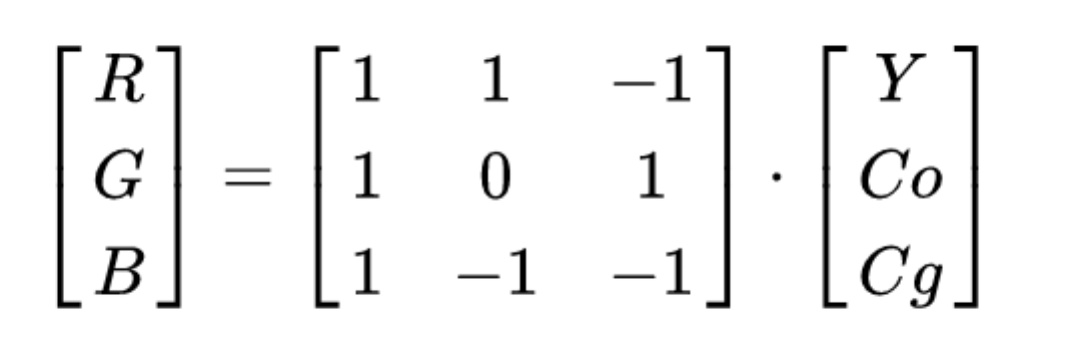
Пространство YCgCo – недавно разработанная альтернатива YCbCr. Те же принципы, но более простое преобразование в/из RGB

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

Диапазон значений:  
0 ≤ 𝑌, 𝑅, 𝐺, 𝐵 ≤ 1   
−0.5 ≤ 𝐶𝑜, 𝐶𝑔 ≤ 0.5

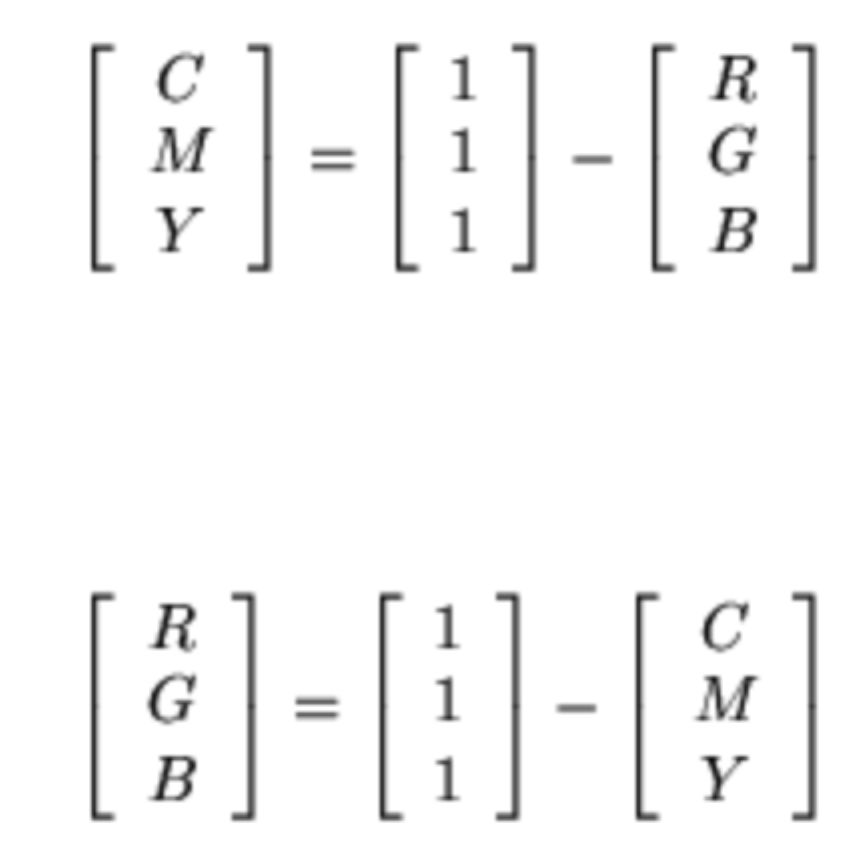
YCoCg to RGB:  


RGB to YCoCg:



CMY:

CMY (англ. Cyan, Magneta, Yellow) - схема формирования цвета, использующая голубой, пурпурный и жёлтый цвета, как основные.

RGB to CMY and CMY to RGB:  


**Экспериментальная часть**

Язык программирования: C++20.

Я сделал структуру данных RGB и создал массив этой структуры, в которой будут храниться информация о каждом канале.

Далее я из исходного цветового пространства перехожу в RGB, а потом из RGB я перехожу в нужное. Если значение может принимать диапазон вида

[**-a**;**a**] то я перед записью в массив увеличиваю значение на **a**, а когда хочу поработать с этим значением, то уменьшаю его на **a**. Еще был написан класс, который фиксирует ошибки введенных аргументов в консоль и ошибки, которые могут присутствовать в файле.

**Выводы**

В ходе проделанной работы были изучены некоторые цветовые пространства, а так же их характеристики. Еще были изучены и применены алгоритмы перехода из одного пространства в другое. Полученные результаты совпадают с теоритическими, так как после цепочки RGB->\*otherColorSpace\*->RGB полученное изображение совпадало с исходным, таким образом, написанная мной программа, сохраняет целостность данных файла.

**Листинг**

**main.cpp**

|  |
| --- |
| #include <iostream> |
|  | #include "CException.h" |
|  | #include "CImage.h" |
|  | #include "string.h" |
|  |  |
|  | using namespace std; |
|  | SInput input; |
|  |  |
|  | int main(int argc, char \*argv[]) { |
|  | try { |
|  | if (argc != 11) { |
|  | CException("Wrong amount of arguments\n"); |
|  | } |
|  | string tmp; |
|  | for (int i = 1; i < argc;) { |
|  | tmp = argv[i]; |
|  | if (tmp == "-f") { |
|  | i++; |
|  | tmp = argv[i]; |
|  | if (tmp == "RGB") { |
|  | input.from = 0; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "HSL") { |
|  | input.from = 1; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "HSV") { |
|  | input.from = 2; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "YCbCr.601") { |
|  | input.from = 3; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "YCbCr.709") { |
|  | input.from = 4; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "YCoCg") { |
|  | input.from = 5; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "CMY") { |
|  | input.from = 6; |
|  | i++; |
|  | continue; |
|  | } else { |
|  | throw CException("Other color space\n"); |
|  | } |
|  | } |
|  | tmp = argv[i]; |
|  | if (tmp == "-t") { |
|  | i++; |
|  | tmp = argv[i]; |
|  | if (tmp == "RGB") { |
|  | input.to = 0; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "HSL") { |
|  | input.to = 1; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "HSV") { |
|  | input.to = 2; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "YCbCr.601") { |
|  | input.to = 3; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "YCbCr.709") { |
|  | input.to = 4; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "YCoCg") { |
|  | input.to = 5; |
|  | i++; |
|  | continue; |
|  | } |
|  | if (tmp == "CMY") { |
|  | input.to = 6; |
|  | i++; |
|  | continue; |
|  | } else { |
|  | throw CException("Other color space\n"); |
|  | } |
|  | } |
|  | tmp = argv[i]; |
|  | if (tmp == "-i") { |
|  | i++; |
|  | input.countInput = atoi(argv[i]); |
|  | if (input.countInput != 1 && input.countInput != 3) { |
|  | throw CException("Wrong amount of input files"); |
|  | } |
|  | i++; |
|  | input.inputFile = argv[i]; |
|  | i++; |
|  | continue; |
|  | } |
|  | tmp = argv[i]; |
|  | if (tmp == "-o") { |
|  | i++; |
|  | input.countOutput = atoi(argv[i]); |
|  | if (input.countOutput != 1 && input.countOutput != 3) { |
|  | throw CException("Wrong amount of output files"); |
|  | } |
|  | i++; |
|  | input.outputFile = argv[i]; |
|  | i++; |
|  | continue; |
|  | } |
|  | } |
|  | CImage img; |
|  | if (input.countInput == 1) { |
|  | img.read1file(input); |
|  | } else { |
|  | img.read3files(input); |
|  | } |
|  | img.convert(input); |
|  | if (input.countOutput == 1) { |
|  | img.write1file(input); |
|  | } else { |
|  | img.write3files(input); |
|  | } |
|  | return 0; |
|  | } catch (CException &exception) { |
|  | cerr << exception.getError(); |
|  | if (exception.getFile()) { |
|  | fclose(exception.getFile()); |
|  | } |
|  | if (exception.getFile2()) { |
|  | fclose(exception.getFile2()); |
|  | } |
|  | if (exception.getFile3()) { |
|  | fclose(exception.getFile3()); |
|  | } |
|  | exit(1); |
|  | } |

**CException.h**

|  |
| --- |
| #ifndef COLORSPACECONVERSION\_CEXCEPTION\_H |
|  | #define COLORSPACECONVERSION\_CEXCEPTION\_H |
|  |  |
|  | #include <string> |
|  |  |
|  | using std::string; |
|  |  |
|  | class CException { |
|  | public: |
|  | CException(string error); |
|  |  |
|  | CException(string error, FILE \*f); |
|  |  |
|  | CException(string error, FILE \*f1, FILE \*f2, FILE \*f3); |
|  |  |
|  | string getError(); |
|  |  |
|  | FILE \*getFile(); |
|  |  |
|  | FILE \*getFile2(); |
|  |  |
|  | FILE \*getFile3(); |
|  |  |
|  | private: |
|  | string error; |
|  | FILE \*file = nullptr; |
|  | FILE \*file2 = nullptr; |
|  | FILE \*file3 = nullptr; |
|  | }; |
|  |  |
|  |  |
|  | #endif //COLORSPACECONVERSION\_CEXCEPTION\_H |

**CException.cpp**

|  |
| --- |
| #include "CException.h" |
|  |  |
|  | CException::CException(string error) { |
|  | this->error = error; |
|  | } |
|  |  |
|  | CException::CException(string error, FILE \*f) { |
|  | this->error = error; |
|  | this->file = f; |
|  | } |
|  |  |
|  | FILE \*CException::getFile() { |
|  | return this->file; |
|  | } |
|  |  |
|  | string CException::getError() { |
|  | return this->error; |
|  | } |
|  |  |
|  | CException::CException(string error, FILE \*f1, FILE \*f2, FILE \*f3) { |
|  | this->error = error; |
|  | this->file = f1; |
|  | this->file2 = f2; |
|  | this->file3 = f3; |
|  | } |
|  |  |
|  | FILE \*CException::getFile2() { |
|  | return this->file2; |
|  | } |
|  |  |
|  | FILE \*CException::getFile3() { |
|  | return this->file3; |
|  | } |

**CImage.h**

|  |
| --- |
| #ifndef COLORSPACECONVERSION\_CIMAGE\_H |
|  | #define COLORSPACECONVERSION\_CIMAGE\_H |
|  |  |
|  |  |
|  | #include <iostream> |
|  | #include "CException.h" |
|  | #include <cmath> |
|  |  |
|  | using namespace std; |
|  |  |
|  | struct SInput { |
|  | int from; |
|  | int to; |
|  | int countInput; |
|  | int countOutput; |
|  | char \*inputFile; |
|  | char \*outputFile; |
|  | }; |
|  |  |
|  | struct RGB { |
|  | double red; |
|  | double green; |
|  | double blue; |
|  | }; |
|  |  |
|  | struct SMetaData { |
|  | FILE \*file; |
|  | int version; |
|  | int width; |
|  | int height; |
|  | int max\_val; |
|  | int size; |
|  | }; |
|  |  |
|  | class CImage { |
|  | public: |
|  |  |
|  | void read1file(SInput console); |
|  |  |
|  | void read3files(SInput console); |
|  |  |
|  | void write1file(SInput console); |
|  |  |
|  | void write3files(SInput console); |
|  |  |
|  | void convert(SInput config); |
|  |  |
|  | ~CImage(); |
|  |  |
|  | private: |
|  | SMetaData \*data; |
|  | RGB \*pixRGB; |
|  |  |
|  |  |
|  | void RGBtoHSL(); |
|  |  |
|  | void RGBtoHSV(); |
|  |  |
|  | void RGBtoYCbCr601(); |
|  |  |
|  | void RGBtoYCbCr709(); |
|  |  |
|  | void RGBtoYCoCg(); |
|  |  |
|  | void RGBtoCMY(); |
|  |  |
|  | void HSLtoRGB(); |
|  |  |
|  | void HSVtoRGB(); |
|  |  |
|  | void YCbCr601toRGB(); |
|  |  |
|  | void YCbCr709toRGB(); |
|  |  |
|  | void YCoCgtoRGB(); |
|  |  |
|  | void CMYtoRGB(); |
|  |  |
|  | double T(double t, double q, double p); |
|  |  |
|  | }; |
|  |  |
|  |  |
|  | #endif //COLORSPACECONVERSION\_CIMAGE\_H |

**CImage.cpp**

|  |
| --- |
| #include "CImage.h" |
|  | #include <string.h> |
|  | #include <algorithm> |
|  |  |
|  | void CImage::read1file(SInput console) { |
|  | FILE \*f = fopen(console.inputFile, "rb"); |
|  | if (!f) { |
|  | throw CException("Input file didn't open\n"); |
|  | } |
|  | data = new SMetaData[1]; |
|  | data[0].file = f; |
|  | if (fscanf(f, "P%d%d%d%d\n", &this->data[0].version, &this->data[0].width, &this->data[0].height, |
|  | &data[0].max\_val) != 4) { |
|  | throw CException("Wrong amount data in file", f); |
|  | } |
|  | if (data[0].version != 6) { |
|  | throw CException("Expected version 6"); |
|  | } |
|  | data[0].size = data[0].width \* data[0].height; |
|  | unsigned char \*buffer; |
|  | buffer = new unsigned char[data[0].size \* 3]; |
|  | pixRGB = new RGB[data[0].size]; |
|  | fread(buffer, sizeof(unsigned char), data[0].size \* 3, f); |
|  | for (int i = 0, j = 0; i < data[0].size \* 3; i += 3, j++) { |
|  | pixRGB[j] = {(double) buffer[i], (double) buffer[i + 1], (double) buffer[i + 2]}; |
|  | } |
|  | delete[] buffer; |
|  | fclose(f); |
|  | } |
|  |  |
|  | void CImage::read3files(SInput console) { |
|  | FILE \*f1, \*f2, \*f3; |
|  | string filename1 = console.inputFile; |
|  | string filename2 = filename1; |
|  | string filename3 = filename1; |
|  | filename1.insert(filename1.size() - 4, "\_1"); |
|  | filename2.insert(filename2.size() - 4, "\_2"); |
|  | filename3.insert(filename3.size() - 4, "\_3"); |
|  | int n = filename1.size(); |
|  | char \*name1 = new char[filename1.size() + 1]; |
|  | char \*name2 = new char[filename2.size() + 1]; |
|  | char \*name3 = new char[filename3.size() + 1]; |
|  | strcpy(name1, filename1.c\_str()); |
|  | strcpy(name2, filename2.c\_str()); |
|  | strcpy(name3, filename3.c\_str()); |
|  | f1 = fopen(name1, "rb"); |
|  | if (!f1) { |
|  | throw CException("File \_1 didn't open"); |
|  | } |
|  | f2 = fopen(name2, "rb"); |
|  | if (!f2) { |
|  | throw CException("File \_2 didn't open"); |
|  | } |
|  | f3 = fopen(name3, "rb"); |
|  | if (!f3) { |
|  | throw CException("File \_3 didn't open"); |
|  | } |
|  | data = new SMetaData[3]; |
|  | if (fscanf(f1, "P%d%d%d%d\n", &this->data[0].version, &this->data[0].width, &this->data[0].height, |
|  | &data[0].max\_val) != 4) { |
|  | throw CException("Wrong amount of meta data in \_1"); |
|  | } |
|  | if (fscanf(f2, "P%d%d%d%d\n", &this->data[1].version, &this->data[1].width, &this->data[1].height, |
|  | &data[1].max\_val) != 4) { |
|  | throw CException("Wrong amount of meta data in \_2"); |
|  | } |
|  | if (fscanf(f3, "P%d%d%d%d\n", &this->data[2].version, &this->data[2].width, &this->data[2].height, |
|  | &data[2].max\_val) != 4) { |
|  | throw CException("Wrong amount of meta data in \_3"); |
|  | } |
|  | if (data[0].version != 5) { |
|  | throw CException("Expected version 5 in \_1", f1, f2, f3); |
|  | } |
|  | if (data[1].version != 5) { |
|  | throw CException("Expected version 5 in \_2", f1, f2, f3); |
|  | } |
|  | if (data[2].version != 5) { |
|  | throw CException("Expected version 5 in \_3", f1, f2, f3); |
|  | } |
|  | int width = data[0].width; |
|  | if (data[1].width != width || data[2].width != width) { |
|  | throw CException("Different widths", f1, f2, f3); |
|  | } |
|  | int height = data[0].height; |
|  | if (data[1].height != height || data[2].height != height) { |
|  | throw CException("Different heights", f1, f2, f3); |
|  | } |
|  | data[0].size = width \* height; |
|  | data[1].size = width \* height; |
|  | data[2].size = width \* height; |
|  | int mv = data[0].max\_val; |
|  | if (data[1].max\_val != mv || data[2].max\_val != mv) { |
|  | throw CException("Different max values", f1, f2, f3); |
|  | } |
|  | int len = data[0].size; |
|  | unsigned char \*buffer1 = new unsigned char[len]; |
|  | fread(buffer1, sizeof(unsigned char), len, f1); |
|  | unsigned char \*buffer2 = new unsigned char[len]; |
|  | fread(buffer2, sizeof(unsigned char), len, f2); |
|  | unsigned char \*buffer3 = new unsigned char[len]; |
|  | fread(buffer3, sizeof(unsigned char), len, f3); |
|  | pixRGB = new RGB[len]; |
|  | for (int i = 0; i < len; i++) { |
|  | pixRGB[i] = {(double) buffer1[i], (double) buffer2[i], (double) buffer3[i]}; |
|  | } |
|  | delete[] buffer1; |
|  | delete[] buffer2; |
|  | delete[] buffer3; |
|  | fclose(f1); |
|  | fclose(f2); |
|  | fclose(f3); |
|  | } |
|  |  |
|  | void CImage::write1file(SInput console) { |
|  | FILE \*new\_f = fopen(console.outputFile, "wb"); |
|  | if (!new\_f) { |
|  | throw CException("Output file didn't open", new\_f); |
|  | } |
|  | fprintf(new\_f, "P%d\n%d %d\n%d\n", 6, data[0].width, data[0].height, data[0].max\_val); |
|  | int len = data[0].size \* 3; |
|  | unsigned char \*buffer = new unsigned char[len]; |
|  | for (int i = 0, j = 0; i < len; i += 3, j++) { |
|  | buffer[i] = (unsigned char) pixRGB[j].red; |
|  | buffer[i + 1] = (unsigned char) pixRGB[j].green; |
|  | buffer[i + 2] = (unsigned char) pixRGB[j].blue; |
|  | } |
|  | fwrite(buffer, sizeof(unsigned char), len, new\_f); |
|  | delete[] buffer; |
|  | fclose(new\_f); |
|  | } |
|  |  |
|  | void CImage::write3files(SInput console) { |
|  | FILE \*f1, \*f2, \*f3; |
|  | string filename1 = console.outputFile; |
|  | string filename2 = filename1; |
|  | string filename3 = filename1; |
|  | filename1.insert(filename1.size() - 4, "\_1"); |
|  | filename2.insert(filename2.size() - 4, "\_2"); |
|  | filename3.insert(filename3.size() - 4, "\_3"); |
|  | int n = filename1.size(); |
|  | char \*name1 = new char[filename1.size() + 1]; |
|  | char \*name2 = new char[filename2.size() + 1]; |
|  | char \*name3 = new char[filename3.size() + 1]; |
|  | strcpy(name1, filename1.c\_str()); |
|  | strcpy(name2, filename2.c\_str()); |
|  | strcpy(name3, filename3.c\_str()); |
|  | f1 = fopen(name1, "wb"); |
|  | if (!f1) { |
|  | throw CException("Output file \_1 didn't open"); |
|  | } |
|  | f2 = fopen(name2, "wb"); |
|  | if (!f2) { |
|  | throw CException("Output file \_2 didn't open"); |
|  | } |
|  | f3 = fopen(name3, "wb"); |
|  | if (!f3) { |
|  | throw CException("Output file \_3 didn't open"); |
|  | } |
|  | fprintf(f1, "P%i\n%d %d\n%d\n", 5, data[0].width, data[0].height, data[0].max\_val); |
|  | fprintf(f2, "P%i\n%d %d\n%d\n", 5, data[0].width, data[0].height, data[0].max\_val); |
|  | fprintf(f3, "P%i\n%d %d\n%d\n", 5, data[0].width, data[0].height, data[0].max\_val); |
|  | unsigned char \*buffer1 = new unsigned char[data[0].size]; |
|  | unsigned char \*buffer2 = new unsigned char[data[0].size]; |
|  | unsigned char \*buffer3 = new unsigned char[data[0].size]; |
|  | for (int i = 0; i < data[0].size; i++) { |
|  | buffer1[i] = (unsigned char) pixRGB[i].red; |
|  | buffer2[i] = (unsigned char) pixRGB[i].green; |
|  | buffer3[i] = (unsigned char) pixRGB[i].blue; |
|  | } |
|  | fwrite(buffer1, sizeof(unsigned char), data[0].size, f1); |
|  | fwrite(buffer2, sizeof(unsigned char), data[0].size, f2); |
|  | fwrite(buffer3, sizeof(unsigned char), data[0].size, f3); |
|  | delete[] buffer1; |
|  | delete[] buffer2; |
|  | delete[] buffer3; |
|  | fclose(f1); |
|  | fclose(f2); |
|  | fclose(f3); |
|  | } |
|  |  |
|  | CImage::~CImage() { |
|  | delete[] pixRGB; |
|  | } |
|  |  |
|  | void CImage::convert(SInput config) { |
|  | switch (config.from) { |
|  | case 0: |
|  | break; |
|  | case 1: |
|  | HSLtoRGB(); |
|  | break; |
|  | case 2: |
|  | HSVtoRGB(); |
|  | break; |
|  | case 3: |
|  | YCbCr601toRGB(); |
|  | break; |
|  | case 4: |
|  | YCbCr709toRGB(); |
|  | break; |
|  | case 5: |
|  | YCoCgtoRGB(); |
|  | break; |
|  | case 6: |
|  | CMYtoRGB(); |
|  | break; |
|  | } |
|  | switch (config.to) { |
|  | case 0: |
|  | break; |
|  | case 1: |
|  | RGBtoHSL(); |
|  | break; |
|  | case 2: |
|  | RGBtoHSV(); |
|  | break; |
|  | case 3: |
|  | RGBtoYCbCr601(); |
|  | break; |
|  | case 4: |
|  | RGBtoYCbCr709(); |
|  | break; |
|  | case 5: |
|  | RGBtoYCoCg(); |
|  | break; |
|  | case 6: |
|  | RGBtoCMY(); |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | void CImage::RGBtoCMY() { |
|  | for (int i = 0; i < data->size; i++) { |
|  | pixRGB[i].red = (1 - pixRGB[i].red / 255.0) \* 255.0; |
|  | pixRGB[i].green = (1 - pixRGB[i].green / 255.0) \* 255.0; |
|  | pixRGB[i].blue = (1 - pixRGB[i].blue / 255.0) \* 255.0; |
|  | } |
|  | } |
|  |  |
|  | void CImage::CMYtoRGB() { |
|  | for (int i = 0; i < data->size; i++) { |
|  | pixRGB[i].red = (1 - pixRGB[i].red / 255.0) \* 255.0; |
|  | pixRGB[i].green = (1 - pixRGB[i].green / 255.0) \* 255.0; |
|  | pixRGB[i].blue = (1 - pixRGB[i].blue / 255.0) \* 255.0; |
|  | } |
|  | } |
|  |  |
|  | void CImage::RGBtoHSL() { |
|  | for (int i = 0; i < data->size; i++) { |
|  | double r = pixRGB[i].red / 255.0; |
|  | double g = pixRGB[i].green / 255.0; |
|  | double b = pixRGB[i].blue / 255.0; |
|  | double Max = max(r, max(g, b)); |
|  | double Min = min(r, min(g, b)); |
|  | double V = Max; |
|  | double C = Max - Min; |
|  | double L = V - C / 2.0; |
|  | double H; |
|  | if (C == 0) |
|  | H = 0; |
|  | else { |
|  | if (V == r) |
|  | H = (60.0) \* ((g - b) / C); |
|  | else if (V == g) |
|  | H = (60.0) \* (2 + (b - r) / C); |
|  | else if (V == b) |
|  | H = (60.0) \* (4 + (r - g) / C); |
|  | else |
|  | H = 0; |
|  | } |
|  | double S = ((L == 0) || (L == 1)) ? 0 : ((V - L) / min(L, 1 - L)); |
|  | pixRGB[i].blue = L \* 255.0; |
|  | pixRGB[i].green = S \* 255.0; |
|  | pixRGB[i].red = (H / 360.0) \* 255.0; |
|  | } |
|  | } |
|  |  |
|  | void CImage::HSLtoRGB() { |
|  | for (int i = 0; i < data->size; i++) { |
|  | double h = (pixRGB[i].red / 255.0) \* 360.0; |
|  | double s = pixRGB[i].green / 255.0; |
|  | double l = pixRGB[i].blue / 255.0; |
|  | double h1 = h / 60.0; |
|  | double c = (1 - abs(2 \* l - 1)) \* s; |
|  | double x = c \* (1 - abs(fmod(h1, 2) - 1)); |
|  | double m = l - c / 2.0; |
|  | m \*= 255.0; |
|  | if ((h1 >= 0) && (h1 <= 1)) { |
|  | pixRGB[i].red = c \* 255.0 + m; |
|  | pixRGB[i].green = x \* 255.0 + m; |
|  | pixRGB[i].blue = 0 + m; |
|  | } |
|  | if ((h1 > 1) && (h1 <= 2)) { |
|  | pixRGB[i].red = x \* 255.0 + m; |
|  | pixRGB[i].green = c \* 255.0 + m; |
|  | pixRGB[i].blue = 0 + m; |
|  | } |
|  | if ((h1 > 2) && (h1 <= 3)) { |
|  | pixRGB[i].red = 0 + m; |
|  | pixRGB[i].green = c \* 255.0 + m; |
|  | pixRGB[i].blue = x \* 255.0 + m; |
|  | } |
|  | if ((h1 > 3) && (h1 <= 4)) { |
|  | pixRGB[i].red = 0 + m; |
|  | pixRGB[i].green = x \* 255.0 + m; |
|  | pixRGB[i].blue = c \* 255.0 + m; |
|  | } |
|  | if ((h1 > 4) && (h1 <= 5)) { |
|  | pixRGB[i].red = x \* 255.0 + m; |
|  | pixRGB[i].green = 0 + m; |
|  | pixRGB[i].blue = c \* 255.0 + m; |
|  | } |
|  | if ((h1 > 5) && (h1 <= 6)) { |
|  | pixRGB[i].red = c \* 255.0 + m; |
|  | pixRGB[i].green = 0 + m; |
|  | pixRGB[i].blue = x \* 255.0 + m; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | double CImage::T(double t, double q, double p) { |
|  | t = (t < 0.0) ? t + 1.0 : (t > 1.0) ? t - 1.0 : t; |
|  | if (t < 1.0 / 6.0) |
|  | return p + (q - p) \* 6.0 \* t; |
|  | if (t < 1.0 / 2.0) |
|  | return q; |
|  | if (t < 2.0 / 3.0) |
|  | return p + ((q - p) \* (2.0 / 3.0 - t) \* 6.0); |
|  | return p; |
|  |  |
|  | } |
|  |  |
|  | void CImage::RGBtoHSV() { |
|  | for (int i = 0; i < data->size; i++) { |
|  | double r = pixRGB[i].red / 255.0; |
|  | double g = pixRGB[i].green / 255.0; |
|  | double b = pixRGB[i].blue / 255.0; |
|  | double MAX = max(r, max(g, b)); |
|  | double MIN = min(r, min(g, b)); |
|  | double h, s, v; |
|  | if (MAX == MIN) { |
|  | h = 0; |
|  | } else if (MAX == r && g >= b) { |
|  | h = 60.0 \* (g - b) / (MAX - MIN); |
|  | } else if (MAX == r && g < b) { |
|  | h = 60.0 \* (g - b) / (MAX - MIN) + 360.0; |
|  | } else if (MAX == g) { |
|  | h = 60.0 \* (b - r) / (MAX - MIN) + 120.0; |
|  | } else if (MAX == b) { |
|  | h = 60.0 \* (r - g) / (MAX - MIN) + 240.0; |
|  | } |
|  | if (MAX == 0) { |
|  | s = 0; |
|  | } else { |
|  | s = 1 - MIN / MAX; |
|  | } |
|  | v = MAX; |
|  | pixRGB[i].red = h / 360.0 \* 255.0; |
|  | pixRGB[i].green = s \* 255.0; |
|  | pixRGB[i].blue = v \* 255.0; |
|  | } |
|  | } |
|  |  |
|  | void CImage::HSVtoRGB() { |
|  | for (int i = 0; i < data->size; i++) { |
|  | double h = pixRGB[i].red / 255.0 \* 360.0; |
|  | double s = pixRGB[i].green / 255.0; |
|  | double V = pixRGB[i].blue / 255.0; |
|  | double r, g, b; |
|  | double Vmin = (1 - s) \* V; |
|  | double Hi = abs(fmod(h / 60.0, 6)); |
|  | pixRGB[i].red = r \* 255.0; |
|  | pixRGB[i].green = g \* 255.0; |
|  | pixRGB[i].blue = b \* 255.0; |
|  | double a = (V - Vmin) \* (fmod(h, 60)) / 60.0; |
|  | double Vinc = Vmin + a; |
|  | double Vdec = V - a; |
|  | switch ((int) Hi) { |
|  | case 0: |
|  | r = V; |
|  | g = Vinc; |
|  | b = Vmin; |
|  | break; |
|  | case 1: |
|  | r = Vdec; |
|  | g = V; |
|  | b = Vmin; |
|  | break; |
|  | case 2: |
|  | r = Vmin; |
|  | g = V; |
|  | b = Vinc; |
|  | break; |
|  | case 3: |
|  | r = Vmin; |
|  | g = Vdec; |
|  | b = V; |
|  | break; |
|  | case 4: |
|  | r = Vinc; |
|  | g = Vmin; |
|  | b = V; |
|  | break; |
|  | case 5: |
|  | r = V; |
|  | g = Vmin; |
|  | b = Vdec; |
|  | break; |
|  | } |
|  | pixRGB[i].red = r \* 255.0; |
|  | pixRGB[i].green = g \* 255.0; |
|  | pixRGB[i].blue = b \* 255.0; |
|  | } |
|  | } |
|  |  |
|  | void CImage::RGBtoYCbCr601() { |
|  | double kr = 0.299, kb = 0.114; |
|  | for (int i = 0; i < data->size; i++) { |
|  | double r = pixRGB[i].red / 255.0; |
|  | double g = pixRGB[i].green / 255.0; |
|  | double b = pixRGB[i].blue / 255.0; |
|  | double y = kr \* r + (1 - kr - kb) \* g + kb \* b; |
|  | double pb = 0.5 \* (b - y) / (1.0 - kb); |
|  | double pr = 0.5 \* (r - y) / (1.0 - kr); |
|  | pixRGB[i].red = y \* 255.0; |
|  | pixRGB[i].green = (pb + 0.5) \* 255.0; |
|  | pixRGB[i].blue = (pr + 0.5) \* 255.0; |
|  | } |
|  | } |
|  |  |
|  | void CImage::RGBtoYCbCr709() { |
|  | double kr = 0.0722, kb = 0.7152; |
|  | for (int i = 0; i < data->size; i++) { |
|  | double r = pixRGB[i].red / 255.0; |
|  | double g = pixRGB[i].green / 255.0; |
|  | double b = pixRGB[i].blue / 255.0; |
|  | double y = kr \* r + (1 - kr - kb) \* g + kb \* b; |
|  | double pb = 0.5 \* (b - y) / (1.0 - kb); |
|  | double pr = 0.5 \* (r - y) / (1.0 - kr); |
|  | pixRGB[i].red = y \* 255.0; |
|  | pixRGB[i].green = (pb + 0.5) \* 255.0; |
|  | pixRGB[i].blue = (pr + 0.5) \* 255.0; |
|  | } |
|  | } |
|  |  |
|  | void CImage::YCbCr601toRGB() { |
|  | double kr = 0.299, kb = 0.114; |
|  | for (int i = 0; i < data->size; i++) { |
|  | double y = pixRGB[i].red / 255.0; |
|  | double pb = pixRGB[i].green / 255.0 - 0.5; |
|  | double pr = pixRGB[i].blue / 255.0 - 0.5; |
|  | double r, g, b; |
|  | b = 2 \* pb \* (1 - kb) + y; |
|  | r = 2 \* pr \* (1 - kr) + y; |
|  | g = (y - kr \* r - kb \* b) / (1 - kr - kb); |
|  | r = r < 0 ? 0 : r; |
|  | r = r > 1 ? 1 : r; |
|  | b = b < 0 ? 0 : b; |
|  | b = b > 1 ? 1 : b; |
|  | g = g < 0 ? 0 : g; |
|  | g = g > 1 ? 1 : g; |
|  | pixRGB[i].red = r \* 255.0; |
|  | pixRGB[i].green = g \* 255.0; |
|  | pixRGB[i].blue = b \* 255.0; |
|  | } |
|  | } |
|  |  |
|  | void CImage::YCbCr709toRGB() { |
|  | double kr = 0.0722, kg = 0.2126, kb = 0.7152; |
|  | for (int i = 0; i < data->size; i++) { |
|  | double y = pixRGB[i].red / 255.0; |
|  | double pb = pixRGB[i].green / 255.0 - 0.5; |
|  | double pr = pixRGB[i].blue / 255.0 - 0.5; |
|  | double r, g, b; |
|  | b = 2 \* pb \* (1 - kb) + y; |
|  | r = 2 \* pr \* (1 - kr) + y; |
|  | g = (y - kr \* r - kb \* b) / (1 - kr - kb); |
|  | r = r < 0 ? 0 : r; |
|  | r = r > 1 ? 1 : r; |
|  | b = b < 0 ? 0 : b; |
|  | b = b > 1 ? 1 : b; |
|  | g = g < 0 ? 0 : g; |
|  | g = g > 1 ? 1 : g; |
|  | pixRGB[i].red = r \* 255.0; |
|  | pixRGB[i].green = g \* 255.0; |
|  | pixRGB[i].blue = b \* 255.0; |
|  | } |
|  | } |
|  |  |
|  | void CImage::RGBtoYCoCg() { |
|  | for (int i = 0; i < data->size; i++) { |
|  | double r = pixRGB[i].red / 255.0; |
|  | double g = pixRGB[i].green / 255.0; |
|  | double b = pixRGB[i].blue / 255.0; |
|  | double y = r / 4.0 + g / 2.0 + b / 4.0; |
|  | double cg = -r / 4.0 + g / 2.0 - b / 4.0; |
|  | double co = r / 2.0 - b / 2.0; |
|  | pixRGB[i].red = y \* 255.0; |
|  | pixRGB[i].green = (co + 0.5) \* 255.0; |
|  | pixRGB[i].blue = (cg + 0.5) \* 255.0; |
|  |  |
|  | } |
|  | } |
|  |  |
|  | void CImage::YCoCgtoRGB() { |
|  | for (int i = 0; i < data->size; i++) { |
|  | double y = pixRGB[i].red / 255.0; |
|  | double co = pixRGB[i].green / 255.0 - 0.5; |
|  | double cg = pixRGB[i].blue / 255.0 - 0.5; |
|  | double r = y + co - cg; |
|  | double g = y + cg; |
|  | double b = y - co - cg; |
|  | r = r < 0 ? 0 : r; |
|  | r = r > 1 ? 1 : r; |
|  | b = b < 0 ? 0 : b; |
|  | b = b > 1 ? 1 : b; |
|  | g = g < 0 ? 0 : g; |
|  | g = g > 1 ? 1 : g; |
|  | pixRGB[i].red = r \* 255.0; |
|  | pixRGB[i].green = g \* 255.0; |
|  | pixRGB[i].blue = b \* 255.0; |
|  | } |
|  | } |