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 № 5**

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

**Изучение алгоритма настройки автояркости изображения**

**Выполнил студент группы M3101  
*Дудко Матвей Владимирович***

**Проверил:  
Скаков Павел Сергеевич**

***САНКТ-ПЕТЕРБУРГ***

***2020***

# Цель работы

Реализация программы, которая позволяет проводить настройку автояркости изображения в различных цветовых пространствах.

# Описание работы

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

Аргументы передаются через командную строку:

**lab5.exe <имя\_входного\_файла> <имя\_выходного\_файла> <преобразование> [<смещение> <множитель>]**

где

* <преобразование>:
  + 0 - применить указанные значения <смещение> и <множитель> в пространстве RGB к каждому каналу;
  + 1 - применить указанные значения <смещение> и <множитель> в пространстве YCbCr.601 к каналу Y;
  + 2 - автояркость в пространстве RGB: <смещение> и <множитель> вычисляются на основе минимального и максимального значений пикселей;
  + 3 - аналогично 2 в пространстве YCbCr.601;
  + 4 - автояркость в пространстве RGB: <смещение> и <множитель> вычисляются на основе минимального и максимального значений пикселей, после игнорирования 0.39% самых светлых и тёмных пикселей;
  + 5 - аналогично 4 в пространстве YCbCr.601.
* <смещение> - целое число, только для преобразований 0 и 1 в диапазоне [-255..255]
* <множитель> - дробное положительное число, только для преобразований 0 и 1 в диапазоне [1/255..255]

Значение пикселя X изменяется по формуле: **(X-<смещение>)\*<множитель>**.

YCbCr.601 в PC диапазоне: [0, 255].

Входные/выходные данные: PNM P5 или P6 (RGB).

**Частичное решение:** только преобразования 0-3 + корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок.

**Полное решение:** все остальное.

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

Коды возврата:

0 - ошибок нет

1 - произошла ошибка

В поток вывода (printf, cout) выводится только следующая информация: для преобразований 2-5 найденные значения <смещение> и <множитель> в формате: "<смещение> <множитель>".

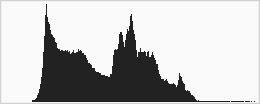
Сообщения об ошибках выводятся в поток вывода ошибок:

С: fprintf(stderr, "Error\n");

C++: std::cerr

# Теоретическая часть

Нередко можно наблюдать изображения с плохой контрастностью: тёмные участки изображения недостаточно тёмные и/или светлые недостаточно светлые.

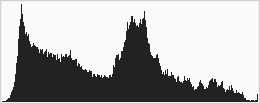
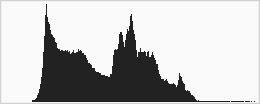
Эту проблему можно хорошо продемонстрировать, если построить распределение яркостей всех точек изображения – гистограмму.

Для улучшения контрастности гистограмму нужно растянуть на весь диапазон значений: минимальное значение пикселя должно стать 0 в новом изображении, а максимальное – 255.

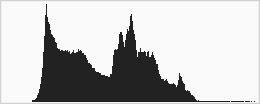
Преобразование яркости каждого пикселя можно описать простой формулой:

y = (x - min)\*255/(max - min)

При нахождении минимального и максимального значений пикселей имеет смысл игнорировать небольшой процент самых тёмных и светлых пикселей, что обычно соответствует шуму.

На примере гистограммы видно, что в качестве минимума здесь можно взять абсолютный минимум, а для максимума имеет смысл взять указанное стрелкой значение, игнорируя существующие, но малочисленные более светлые пиксели.

Корректировать контрастность можно как работая в пространстве RGB, **одинаково** изменяя все каналы (а не отдельно каждый), так и в других цветовых пространствах.  
Например, широко используется корректировка контрастности в пространстве YCbCr. Здесь изменяются значения только канала Y, соответствующего яркости изображения, а каналы Cb и Cr остаются неизменными. Как правило, это даёт более контрастные, но менее насыщенные изображения, чем автоконтрастность в пространстве RGB.



YCbCR

RGB

Исходное изображение

# Экспериментальная часть

Лабораторная работа выполнена на языке C++. Стандарт языка C++14.

Для удобства все основные функции для работы с изображением были вынесены в отдельную библиотеку.

## Хранение данных изображения и заголовка при чтении из файла

Данные заголовка файла хранятся в переменных:

char char\_header;  
int width, height;  
unsigned int max\_value;

Чтение производится при помощи функции fscanf:

fscanf(file\_in, "P%c\n%i %i\n%i\n", &char\_header, &width, &height, &max\_value);

Где:

* char\_header – номер версии файла (возможные варианты 5 или 6)
* width – ширина изображения
* height – высота изображения
* max\_value – максимально возможное значение яркости

Общее количество байт для считывания вычисляется и записывается в переменную k\_bytes:

int k\_bytes = height \* width;

if (char\_header == '6')  
 k\_bytes \*= 3;

Данные пикселей хранятся в одномерном динамическом массиве, выделяющемся в куче:

auto pix\_data = (unsigned char \*) calloc(k\_bytes, 1);

И считываются при помощи функции read\_data:

read\_data(file\_in, k\_bytes, pix\_data);

## Хранение данных нового изображения

Данные пикселей исходного изображения преобразуются “на месте” без выделения дополнительной памяти и хранятся в pix\_data.

auto pix\_data = (unsigned char \*) calloc(k\_bytes, 1);

## Алгоритм автояркости и преобразования

Суть алгоритма автояркости заключается в том, чтобы растянуть гистограмму яркостей на весь диапазон значений.

### 0. Применить указанные значения <смещение> и <множитель> в пространстве RGB к каждому каналу

Преобразование выполняется при помощи функции без выделения дополнительной памяти:

change\_offset\_multiply\_all(pix\_data, k\_bytes, offset, multiplier);

При помощи формулы:

\*(pix\_data + i) = limit\_brightness(((int) \*(pix\_data + i) - offset) \* multiplier);

### 1. Применить указанные значения <смещение> и <множитель> в пространстве YCbCr.601 к каналу Y

Преобразование выполняется аналогично преобразованию 0, но дополнительно перед преобразованием цветного изображения производится его преобразование в YCrCb и аналогично в RGB после:

if (conversion % 2 == 1 && char\_header == '6') {  
 conversions::rgb\_2\_YCbCr\_601(pix\_data, k\_bytes);  
}

if (char\_header == '5')  
 change\_offset\_multiply\_Y\_gray(pix\_data, k\_bytes, offset, multiplier);  
else  
 change\_offset\_multiply\_Y(pix\_data, k\_bytes, offset, multiplier);

if (conversion % 2 == 1 && char\_header == '6') {  
 conversions::YCbCr\_601\_2\_rgb(pix\_data, k\_bytes);  
}

### 2. Автояркость в пространстве RGB: <смещение> и <множитель> вычисляются на основе минимального и максимального значений пикселей

Для данного и следующих преобразований подсчитываются количество яркостей каждого канала (или одного для черно-белого изображения). Дополнительная память O(1).

std::vector<long long> nums\_brightnesses = std::vector<long long>(256, 0);  
if (conversion % 2 == 0)  
 count\_all\_brightnesses\_RGB(pix\_data, k\_bytes, nums\_brightnesses);

Далее вычисляются коэффициенты:

int min\_idx = 0;  
int max\_idx = 255;  
while (nums\_brightnesses[min\_idx] == 0) ++min\_idx;  
while (nums\_brightnesses[max\_idx] == 0) --max\_idx;  
offset = min\_idx;  
multiplier = 255.0l / (max\_idx - min\_idx);  
std::cout << "Calculated offset: " << offset << "\n";  
std::cout << "Multiplier: " << std::setprecision(5) << multiplier << "\n";

И производится преобразование:

if (conversion == 2) {  
 change\_offset\_multiply\_all(pix\_data, k\_bytes, offset, multiplier);  
}

### 3. Аналогично 2 в пространстве YCbCr.601

Аналогично преобразованию 2:

Подсчет значений:

std::vector<long long> nums\_brightnesses = std::vector<long long>(256, 0);  
if (conversion % 2 == 0)  
 count\_all\_brightnesses\_RGB(pix\_data, k\_bytes, nums\_brightnesses);  
else  
 count\_all\_brightnesses\_YCbCr(pix\_data, k\_bytes, char\_header, nums\_brightnesses);

Далее вычисляются коэффициенты:

int min\_idx = 0;  
int max\_idx = 255;  
while (nums\_brightnesses[min\_idx] == 0) ++min\_idx;  
while (nums\_brightnesses[max\_idx] == 0) --max\_idx;  
offset = min\_idx;  
multiplier = 255.0l / (max\_idx - min\_idx);  
std::cout << "Calculated offset: " << offset << "\n";  
std::cout << "Multiplier: " << std::setprecision(5) << multiplier << "\n";

И производится преобразование:

if (char\_header == '5')  
 change\_offset\_multiply\_Y\_gray(pix\_data, k\_bytes, offset, multiplier);  
else  
 change\_offset\_multiply\_Y(pix\_data, k\_bytes, offset, multiplier);

### 4. Автояркость в пространстве RGB: <смещение> и <множитель> вычисляются на основе минимального и максимального значений пикселей, после игнорирования 0.39% самых светлых и тёмных пикселей

Подсчет значений аналогичен пункту 2:

std::vector<long long> nums\_brightnesses = std::vector<long long>(256, 0);  
if (conversion % 2 == 0)  
 count\_all\_brightnesses\_RGB(pix\_data, k\_bytes, nums\_brightnesses);

Фильтрация 0.39 % значений и расчет коэффициентов

// Игнорируем 0.39% пикселей с двух сторон  
  
int pixels\_count = width \* height;  
auto ignoring\_count = (long long) std::round(0.0039l \* pixels\_count);  
  
int min\_idx = 0;  
int max\_idx = 255;  
// min\_idx <= 255 - излишне, но оставим на всякий случай  
while (min\_idx <= 255 && (nums\_brightnesses[min\_idx] == 0 || ignoring\_count > 0)) {  
 if (nums\_brightnesses[min\_idx] == 0) {  
 ++min\_idx;  
 continue;  
 }  
  
 if (ignoring\_count >= nums\_brightnesses[min\_idx]) {  
 ignoring\_count -= nums\_brightnesses[min\_idx];  
 ++min\_idx;  
 } else {  
 break;  
 }  
}  
  
ignoring\_count = (long long) std::round(0.0039l \* pixels\_count);  
// max\_idx >= 0 - излишне, но оставим на всякий случай  
while (max\_idx >= 0 && (nums\_brightnesses[max\_idx] == 0|| ignoring\_count > 0)) {  
 if (nums\_brightnesses[max\_idx] == 0) {  
 --max\_idx;  
 continue;  
 }  
 if (ignoring\_count >= nums\_brightnesses[min\_idx]) {  
 ignoring\_count -= nums\_brightnesses[min\_idx];  
 --max\_idx;  
 } else {  
 break;  
 }  
}  
  
offset = min\_idx;  
multiplier = 255.0l / (max\_idx - min\_idx);  
std::cout << "Calculated offset: " << offset << "\n";  
std::cout << "Multiplier: " << std::setprecision(5) << multiplier << "\n";

Преобразование:

if (conversion == 4) {  
 change\_offset\_multiply\_all(pix\_data, k\_bytes, offset, multiplier);  
}

### 5. Аналогично 4 в пространстве YCbCr.601

Подсчет значений аналогичен пункту 2:

std::vector<long long> nums\_brightnesses = std::vector<long long>(256, 0);  
if (conversion % 2 == 0)  
 count\_all\_brightnesses\_RGB(pix\_data, k\_bytes, nums\_brightnesses);

Фильтрация 0.39 % значений и расчет коэффициентов

// Игнорируем 0.39% пикселей с двух сторон  
  
int pixels\_count = width \* height;  
auto ignoring\_count = (long long) std::round(0.0039l \* pixels\_count);  
  
int min\_idx = 0;  
int max\_idx = 255;  
// min\_idx <= 255 - излишне, но оставим на всякий случай  
while (min\_idx <= 255 && (nums\_brightnesses[min\_idx] == 0 || ignoring\_count > 0)) {  
 if (nums\_brightnesses[min\_idx] == 0) {  
 ++min\_idx;  
 continue;  
 }  
  
 if (ignoring\_count >= nums\_brightnesses[min\_idx]) {  
 ignoring\_count -= nums\_brightnesses[min\_idx];  
 ++min\_idx;  
 } else {  
 break;  
 }  
}  
  
ignoring\_count = (long long) std::round(0.0039l \* pixels\_count);  
// max\_idx >= 0 - излишне, но оставим на всякий случай  
while (max\_idx >= 0 && (nums\_brightnesses[max\_idx] == 0|| ignoring\_count > 0)) {  
 if (nums\_brightnesses[max\_idx] == 0) {  
 --max\_idx;  
 continue;  
 }  
 if (ignoring\_count >= nums\_brightnesses[min\_idx]) {  
 ignoring\_count -= nums\_brightnesses[min\_idx];  
 --max\_idx;  
 } else {  
 break;  
 }  
}  
  
offset = min\_idx;  
multiplier = 255.0l / (max\_idx - min\_idx);  
std::cout << "Calculated offset: " << offset << "\n";  
std::cout << "Multiplier: " << std::setprecision(5) << multiplier << "\n";

Преобразование:

// conversion == 5  
if (char\_header == '5')  
 change\_offset\_multiply\_Y\_gray(pix\_data, k\_bytes, offset, multiplier);  
else  
 change\_offset\_multiply\_Y(pix\_data, k\_bytes, offset, multiplier);

## Запись полученных данных в файлы

Используется функции записи color::write\_to\_file и gray::write\_to\_file. Имя файла берется из аргументов командной строки

if (char\_header == '5')  
 gray::write\_to\_file(file\_out, width, height, max\_value, pix\_data);  
else  
 color::write\_to\_file(file\_out, width, height, max\_value, pix\_data);

# Вывод

Выполнение данной лабораторной работы позволило узнать о формате представления распределения яркостей всех точек изображения в виде гистограммы для анализа изображения, а также об алгоритме автоматической коррекции яркости и контрастности изображений. Была реализована программа, позволяющая изменять яркость и контрастность изображения при помощи вводимых данных из командной строки, а также в автоматическом режиме, позволяющем как игнорировать 0.39% яркостей изображения для лучшего преобразования, так и не игнорировать эти данные. Преобразования яркостей пикселей реализованы как для цветового пространства RGB, так и YCrCb.601.

# Листинг кода

Исходный код можно посмотреть на ресурсе GitHub по следующей ссылке: <https://github.com/DudkoMatt/GeometryAndGraphics/blob/master/Lab_05>

Содержание проекта:

./GeometryAndGraphics/Lab\_05/

main.cpp

conversions\_color\_spaces.cpp

conversions\_color\_spaces.h

Дополнительные зависимости:

./GeometryAndGraphics/Lab\_04/

write\_data\_pnm\_v2.h

write\_data\_pnm\_v2.cpp

Название файла: ./GeometryAndGraphics/Lab\_05/main.cpp

Исходный код:

#include <iostream>  
#include <vector>  
#include <cmath>  
#include <iomanip>  
#include <algorithm>  
#include <string>  
#include "../Lab\_04/write\_data\_pnm\_v2.h"  
#include "conversions\_color\_spaces.h"  
  
unsigned char limit\_brightness(long double brightness) {  
 return (unsigned char) std::min(255.0l, std::max(0.0l, brightness));  
}  
  
void change\_offset\_multiply\_all(unsigned char \*pix\_data, int all\_bytes, int offset, long double multiplier) {  
 for (int i = 0; i < all\_bytes; ++i) {  
 \*(pix\_data + i) = limit\_brightness(((int) \*(pix\_data + i) - offset) \* multiplier);  
 }  
}  
  
// RGB  
void change\_offset\_multiply\_Y(unsigned char \*pix\_data, int all\_bytes, int offset, long double multiplier) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 \*(pix\_data + i) = limit\_brightness(((int) \*(pix\_data + i) - offset) \* multiplier);  
 }  
}  
  
// Gray  
void change\_offset\_multiply\_Y\_gray(unsigned char \*pix\_data, int all\_bytes, int offset, long double multiplier) {  
 for (int i = 0; i < all\_bytes; ++i) {  
 \*(pix\_data + i) = limit\_brightness(((int) \*(pix\_data + i) - offset) \* multiplier);  
 }  
}  
  
  
void  
count\_all\_brightnesses\_RGB(const unsigned char \*pix\_data, int all\_bytes, std::vector<long long> &nums) {  
 for (int i = 0; i < all\_bytes; ++i) {  
 nums[\*(pix\_data + i)] += 1;  
 }  
}  
  
void  
count\_all\_brightnesses\_YCbCr(const unsigned char \*pix\_data, int all\_bytes, char char\_header, std::vector<long long> &nums) {  
 if (char\_header == '5') {  
 // Gray  
 for (int i = 0; i < all\_bytes; ++i) {  
 nums[\*(pix\_data + i)] += 1;  
 }  
 } else {  
 // RGB -> YCbCr  
 for (int i = 0; i < all\_bytes; i += 3) {  
 nums[\*(pix\_data + i)] += 1;  
 }  
 }  
}  
  
int main(int argc, char \*argv[]) {  
  
 if (argc != 4 && argc != 6) {  
 std::cerr  
 << "Wrong number of arguments. Syntax:\n<lab5>.exe <file\_in> <file\_out> <conversion> [<offset> <multiplier>]\n";  
 return 1;  
 }  
  
 const char \*file\_in\_name = argv[1];  
 const char \*file\_out\_name = argv[2];  
  
 unsigned conversion = argv[3][0] - '0';  
  
 int offset = 0;  
 long double multiplier = 0;  
 if (argc == 6) {  
 offset = std::stoi(argv[4]);  
 multiplier = std::stold(argv[5]);  
 }  
  
 if (conversion > 5 || ((conversion == 0 || conversion == 1) && argc != 6) || (conversion > 1 && argc == 6)  
 || (argc == 6 && (std::abs(offset) > 255 || multiplier < 0 || multiplier - 1.0l / 255 < 0.001l  
 || multiplier - 255.0l > 0.001l))) {  
 std::cerr << "Wrong arguments\n";  
 return 1;  
 }  
  
 char char\_header;  
 int width, height;  
 unsigned int max\_value;  
  
 FILE \*file\_in = fopen(file\_in\_name, "rb");  
  
 if (file\_in == nullptr) {  
 std::cerr << "Cannot open file to read: " << file\_in\_name << "\n";  
 return 1;  
 }  
  
 fscanf(file\_in, "P%c\n%i %i\n%i\n", &char\_header, &width, &height, &max\_value);  
  
 if (width <= 0 || height <= 0 || (char\_header != '5' && char\_header != '6')) {  
 std::cerr << "Wrong file attributes\n";  
 fclose(file\_in);  
 return 1;  
 }  
  
 FILE \*file\_out = nullptr;  
 int k\_bytes = height \* width;  
  
 if (char\_header == '6')  
 k\_bytes \*= 3;  
  
 auto pix\_data = (unsigned char \*) calloc(k\_bytes, 1);  
 if (!pix\_data) {  
 std::cerr << "Cannot allocate " << k\_bytes << " bytes of memory\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 read\_data(file\_in, k\_bytes, pix\_data);  
  
 if (conversion % 2 == 1 && char\_header == '6') {  
 conversions::rgb\_2\_YCbCr\_601(pix\_data, k\_bytes);  
 }  
  
 // Преобразования  
 if (conversion == 0) {  
 change\_offset\_multiply\_all(pix\_data, k\_bytes, offset, multiplier);  
 } else if (conversion == 1) {  
 if (char\_header == '5')  
 change\_offset\_multiply\_Y\_gray(pix\_data, k\_bytes, offset, multiplier);  
 else  
 change\_offset\_multiply\_Y(pix\_data, k\_bytes, offset, multiplier);  
 } else {  
 std::vector<long long> nums\_brightnesses = std::vector<long long>(256, 0);  
 if (conversion % 2 == 0)  
 count\_all\_brightnesses\_RGB(pix\_data, k\_bytes, nums\_brightnesses);  
 else  
 count\_all\_brightnesses\_YCbCr(pix\_data, k\_bytes, char\_header, nums\_brightnesses);  
  
 if (conversion == 2 || conversion == 3) {  
 int min\_idx = 0;  
 int max\_idx = 255;  
 while (nums\_brightnesses[min\_idx] == 0) ++min\_idx;  
 while (nums\_brightnesses[max\_idx] == 0) --max\_idx;  
 offset = min\_idx;  
 multiplier = 255.0l / (max\_idx - min\_idx);  
 std::cout << "Calculated offset: " << offset << "\n";  
 std::cout << "Multiplier: " << std::setprecision(5) << multiplier << "\n";  
  
 if (conversion == 2) {  
 change\_offset\_multiply\_all(pix\_data, k\_bytes, offset, multiplier);  
 } else {  
 // conversion == 3  
 if (char\_header == '5')  
 change\_offset\_multiply\_Y\_gray(pix\_data, k\_bytes, offset, multiplier);  
 else  
 change\_offset\_multiply\_Y(pix\_data, k\_bytes, offset, multiplier);  
 }  
  
 } else {  
  
 // Игнорируем 0.39% пикселей с двух сторон  
  
 int pixels\_count = width \* height;  
 auto ignoring\_count = (long long) std::round(0.0039l \* pixels\_count);  
  
 int min\_idx = 0;  
 int max\_idx = 255;  
 // min\_idx <= 255 - излишне, но оставим на всякий случай  
 while (min\_idx <= 255 && (nums\_brightnesses[min\_idx] == 0 || ignoring\_count > 0)) {  
 if (nums\_brightnesses[min\_idx] == 0) {  
 ++min\_idx;  
 continue;  
 }  
  
 if (ignoring\_count >= nums\_brightnesses[min\_idx]) {  
 ignoring\_count -= nums\_brightnesses[min\_idx];  
 ++min\_idx;  
 } else {  
 break;  
 }  
 }  
  
 ignoring\_count = (long long) std::round(0.0039l \* pixels\_count);  
 // max\_idx >= 0 - излишне, но оставим на всякий случай  
 while (max\_idx >= 0 && (nums\_brightnesses[max\_idx] == 0|| ignoring\_count > 0)) {  
 if (nums\_brightnesses[max\_idx] == 0) {  
 --max\_idx;  
 continue;  
 }  
 if (ignoring\_count >= nums\_brightnesses[min\_idx]) {  
 ignoring\_count -= nums\_brightnesses[min\_idx];  
 --max\_idx;  
 } else {  
 break;  
 }  
 }  
  
 offset = min\_idx;  
 multiplier = 255.0l / (max\_idx - min\_idx);  
 std::cout << "Calculated offset: " << offset << "\n";  
 std::cout << "Multiplier: " << std::setprecision(5) << multiplier << "\n";  
  
 if (conversion == 4) {  
 change\_offset\_multiply\_all(pix\_data, k\_bytes, offset, multiplier);  
 } else {  
 // conversion == 5  
 if (char\_header == '5')  
 change\_offset\_multiply\_Y\_gray(pix\_data, k\_bytes, offset, multiplier);  
 else  
 change\_offset\_multiply\_Y(pix\_data, k\_bytes, offset, multiplier);  
 }  
 }  
  
 }  
  
 if (conversion % 2 == 1 && char\_header == '6') {  
 conversions::YCbCr\_601\_2\_rgb(pix\_data, k\_bytes);  
 }  
  
  
 file\_out = fopen(file\_out\_name, "wb");  
 if (file\_out == nullptr) {  
 std::cerr << "Cannot open file to write: " << file\_out\_name << "\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 if (char\_header == '5')  
 gray::write\_to\_file(file\_out, width, height, max\_value, pix\_data);  
 else  
 color::write\_to\_file(file\_out, width, height, max\_value, pix\_data);  
  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
}

Название файла: ./GeometryAndGraphics/Lab\_05/conversions\_color\_spaces.h

Исходный код:

//  
// Created by dudko on 02.05.2020.  
//  
  
#ifndef LAB\_05\_CONVERSIONS\_COLOR\_SPACES\_H  
#define LAB\_05\_CONVERSIONS\_COLOR\_SPACES\_H  
  
namespace conversions {  
 bool check\_color\_space(char \*color\_space);  
  
 void add\_channel(unsigned char \*pix\_data, int k\_channel, int k\_bytes\_channel, const unsigned char \*channel);  
  
 void  
 get\_separate\_channel(const unsigned char \*pix\_data, int all\_bytes, int k\_channel, unsigned char \*channel\_data);  
  
 void rgb\_2\_hsv(unsigned char \*pix\_data, int all\_bytes);  
  
 void rgb\_2\_hsl(unsigned char \*pix\_data, int all\_bytes);  
  
 template<class T>  
 bool in\_range(T left, T x, T right);  
  
 void hsv\_2\_rgb(unsigned char \*pix\_data, int all\_bytes);  
  
 void hsl\_2\_rgb(unsigned char \*pix\_data, int all\_bytes);  
  
 void rgb\_2\_YCbCr\_601(unsigned char \*pix\_data, int all\_bytes);  
  
 void YCbCr\_601\_2\_rgb(unsigned char \*pix\_data, int all\_bytes);  
  
 void rgb\_2\_YCbCr\_709(unsigned char \*pix\_data, int all\_bytes);  
  
 void YCbCr\_709\_2\_rgb(unsigned char \*pix\_data, int all\_bytes);  
  
 void rgb\_2\_YCoCg(unsigned char \*pix\_data, int all\_bytes);  
  
 void YCoCg\_2\_rgb(unsigned char \*pix\_data, int all\_bytes);  
  
 void rgb\_2\_cmy(unsigned char \*pix\_data, int all\_bytes);  
  
 void cmy\_2\_rgb(unsigned char \*pix\_data, int all\_bytes);  
}  
  
#endif //LAB\_05\_CONVERSIONS\_COLOR\_SPACES\_H

Название файла: ./GeometryAndGraphics/Lab\_05/conversions\_color\_spaces.cpp

Исходный код:

//  
// Created by dudko on 02.05.2020.  
//  
  
#include "conversions\_color\_spaces.h"  
#include "../Lab\_04/write\_data\_pnm\_v2.h"  
#include <cstring>  
#include <cmath>  
#include <algorithm>  
  
namespace conversions {  
 bool check\_color\_space(char \*color\_space) {  
 if (color\_space == nullptr)  
 return false;  
 return strcmp(color\_space, "RGB") == 0 ||  
 strcmp(color\_space, "HSL") == 0 ||  
 strcmp(color\_space, "HSV") == 0 ||  
 strcmp(color\_space, "YCbCr.601") == 0 ||  
 strcmp(color\_space, "YCbCr.709") == 0 ||  
 strcmp(color\_space, "YCoCg") == 0 ||  
 strcmp(color\_space, "CMY") == 0;  
 }  
  
 void add\_channel(unsigned char \*pix\_data, int k\_channel, int k\_bytes\_channel, const unsigned char \*channel) {  
 for (int i = 0; i < k\_bytes\_channel; ++i) {  
 \*(pix\_data + i \* 3 + k\_channel) = \*(channel + i);  
 }  
 }  
  
 void  
 get\_separate\_channel(const unsigned char \*pix\_data, int all\_bytes, int k\_channel, unsigned char \*channel\_data) {  
 for (int i = 0; i < all\_bytes / 3; ++i) {  
 \*(channel\_data + i) = \*(pix\_data + 3 \* i + k\_channel);  
 }  
 }  
  
 void rgb\_2\_hsv(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double R = pix\_data[i] / 255.0;  
 double G = pix\_data[i + 1] / 255.0;  
 double B = pix\_data[i + 2] / 255.0;  
  
 double MAX = std::max(R, std::max(G, B));  
 double MIN = std::min(R, std::min(G, B));  
  
 double V = MAX;  
 double C = MAX - MIN;  
  
 // All in [0.0 .. 1.0]  
 // H in [0 .. 360]  
 double H;  
  
 // Calculating Hue  
 if (C == 0) {  
 H = 0;  
 } else if (V == R) {  
 H = (G - B) / C;  
 } else if (V == G) {  
 H = 2 + (B - R) / C;  
 } else {  
 // V == B  
 H = 4 + (R - G) / C;  
 }  
 H \*= 60;  
  
 if (H < 0)  
 H += 360;  
  
 double S\_v;  
 S\_v = V == 0 ? 0 : C / V;  
  
 // Transform to PC range: [0 .. 255]  
 // H:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(H / 360.0 \* 255));  
 // S:  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round(S\_v \* 255));  
 // V:  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round(V \* 255));  
 }  
 }  
  
 void rgb\_2\_hsl(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double R = pix\_data[i] / 255.0;  
 double G = pix\_data[i + 1] / 255.0;  
 double B = pix\_data[i + 2] / 255.0;  
  
 double MAX = std::max(R, std::max(G, B));  
 double MIN = std::min(R, std::min(G, B));  
  
 double V = MAX;  
 double C = MAX - MIN;  
 double L = (MAX + MIN) / 2.0;  
  
 // All in [0.0 .. 1.0]  
 // H in [0 .. 360]  
 double H;  
  
 // Calculating Hue  
 if (C == 0) {  
 H = 0;  
 } else if (V == R) {  
 H = (G - B) / C;  
 } else if (V == G) {  
 H = 2 + (B - R) / C;  
 } else {  
 // V == B  
 H = 4 + (R - G) / C;  
 }  
 H \*= 60;  
  
 if (H < 0)  
 H += 360;  
  
 double S\_l;  
 S\_l = L == 0 || L == 1 ? 0 : (V - L) / std::min(L, 1 - L);  
  
 // Transform to PC range: [0 .. 255]  
 // H:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round((H / 360.0 \* 255)));  
 // S:  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((S\_l \* 255)));  
 // V:  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((L \* 255)));  
 }  
 }  
  
 template<class T>  
 bool in\_range(T left, T x, T right) {  
 return (left <= x) && (x <= right);  
 }  
  
 void hsv\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double H = pix\_data[i] / 255.0 \* 360;  
 double S\_v = pix\_data[i + 1] / 255.0;  
 double V = pix\_data[i + 2] / 255.0;  
  
 double C = V \* S\_v;  
 double \_H = H / 60;  
 double X = C \* (1 - std::abs(((int) \_H) % 2 + (\_H - (int) \_H) - 1));  
  
 double \_R, \_G, \_B;  
  
 if (in\_range(0.0, \_H, 1.0)) {  
 \_R = C;  
 \_G = X;  
 \_B = 0;  
 } else if (in\_range(1.0, \_H, 2.0)) {  
 \_R = X;  
 \_G = C;  
 \_B = 0;  
 } else if (in\_range(2.0, \_H, 3.0)) {  
 \_R = 0;  
 \_G = C;  
 \_B = X;  
 } else if (in\_range(3.0, \_H, 4.0)) {  
 \_R = 0;  
 \_G = X;  
 \_B = C;  
 } else if (in\_range(4.0, \_H, 5.0)) {  
 \_R = X;  
 \_G = 0;  
 \_B = C;  
 } else {  
 \_R = C;  
 \_G = 0;  
 \_B = X;  
 }  
  
 double m = V - C;  
  
 // To RGB:  
 // R:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round((\_R + m) \* 255));  
 // G  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((\_G + m) \* 255));  
 // B  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((\_B + m) \* 255));  
 }  
 }  
  
 void hsl\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double H = pix\_data[i] / 255.0 \* 360;  
 double S\_l = pix\_data[i + 1] / 255.0;  
 double L = pix\_data[i + 2] / 255.0;  
  
 double C = (1 - std::abs(2 \* L - 1)) \* S\_l;  
 double \_H = H / 60;  
 double X = C \* (1 - std::abs(((int) \_H) % 2 + (\_H - (int) \_H) - 1));  
  
 double \_R, \_G, \_B;  
  
 if (std::ceil(\_H) - 1.0 < 0.001) {  
 \_R = C;  
 \_G = X;  
 \_B = 0;  
 } else if (std::ceil(\_H) - 2.0 < 0.001) {  
 \_R = X;  
 \_G = C;  
 \_B = 0;  
 } else if (std::ceil(\_H) - 3.0 < 0.001) {  
 \_R = 0;  
 \_G = C;  
 \_B = X;  
 } else if (std::ceil(\_H) - 4.0 < 0.001) {  
 \_R = 0;  
 \_G = X;  
 \_B = C;  
 } else if (std::ceil(\_H) - 5.0 < 0.001) {  
 \_R = X;  
 \_G = 0;  
 \_B = C;  
 } else if (std::ceil(\_H) - 6.0 < 0.001) {  
 \_R = C;  
 \_G = 0;  
 \_B = X;  
 } else {  
 \_R = 0;  
 \_G = 0;  
 \_B = 0;  
 }  
  
 double m = L - C / 2.0;  
  
 // To RGB:  
 // R:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round((\_R + m) \* 255));  
 // G  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((\_G + m) \* 255));  
 // B  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((\_B + m) \* 255));  
 }  
 }  
  
 void rgb\_2\_YCbCr\_601(unsigned char \*pix\_data, int all\_bytes) {  
 double K\_b = 0.299;  
 double K\_r = 0.587;  
 double K\_g = 0.114;  
  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double R = pix\_data[i] / 255.0;  
 double G = pix\_data[i + 1] / 255.0;  
 double B = pix\_data[i + 2] / 255.0;  
  
 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));  
  
 // To YCbCr\_601:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(Y \* 255));  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((C\_b + 0.5) \* 255));  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((C\_r + 0.5) \* 255));  
 }  
 }  
  
 void YCbCr\_601\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 double K\_b = 0.299;  
 double K\_r = 0.587;  
 double K\_g = 0.114;  
  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double Y = pix\_data[i] / 255.0;  
 double C\_b = pix\_data[i + 1] / 255.0 - 0.5;  
 double C\_r = pix\_data[i + 2] / 255.0 - 0.5;  
  
 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;  
  
 // To RGB:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(R \* 255));  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round(G \* 255));  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round(B \* 255));  
 }  
 }  
  
 void rgb\_2\_YCbCr\_709(unsigned char \*pix\_data, int all\_bytes) {  
 double K\_b = 0.0722;  
 double K\_r = 0.2126;  
 double K\_g = 0.7152;  
  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double R = pix\_data[i] / 255.0;  
 double G = pix\_data[i + 1] / 255.0;  
 double B = pix\_data[i + 2] / 255.0;  
  
 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));  
  
 // To YCbCr\_709:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(Y \* 255));  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((C\_b + 0.5) \* 255));  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((C\_r + 0.5) \* 255));  
 }  
 }  
  
 void YCbCr\_709\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 double K\_b = 0.0722;  
 double K\_r = 0.2126;  
 double K\_g = 0.7152;  
  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double Y = pix\_data[i] / 255.0;  
 double C\_b = pix\_data[i + 1] / 255.0 - 0.5;  
 double C\_r = pix\_data[i + 2] / 255.0 - 0.5;  
  
 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;  
  
 // To RGB:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(R \* 255));  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round(G \* 255));  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round(B \* 255));  
 }  
 }  
  
 void rgb\_2\_YCoCg(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double R = pix\_data[i] / 255.0;  
 double G = pix\_data[i + 1] / 255.0;  
 double B = pix\_data[i + 2] / 255.0;  
  
 // Y  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round(((R + B) / 4.0 + G / 2.0) \* 255));  
 // Co  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round(((R - B) / 2.0 + 0.5) \* 255));  
 // Cg  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((G / 2.0 - (B + R) / 4.0 + 0.5) \* 255));  
 }  
 }  
  
 void YCoCg\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; i += 3) {  
 double Y = pix\_data[i] / 255.0;  
 double Co = pix\_data[i + 1] / 255.0 - 0.5;  
 double Cg = pix\_data[i + 2] / 255.0 - 0.5;  
  
 // To RGB:  
 \*(pix\_data + i) = (unsigned char) limit\_brightness(std::round((Y - Cg + Co) \* 255));  
 \*(pix\_data + i + 1) = (unsigned char) limit\_brightness(std::round((Y + Cg) \* 255));  
 \*(pix\_data + i + 2) = (unsigned char) limit\_brightness(std::round((Y - Cg - Co) \* 255));  
 }  
 }  
  
 void rgb\_2\_cmy(unsigned char \*pix\_data, int all\_bytes) {  
 for (int i = 0; i < all\_bytes; ++i) {  
 \*(pix\_data + i) = 255 - \*(pix\_data + i);  
 }  
 }  
  
 void cmy\_2\_rgb(unsigned char \*pix\_data, int all\_bytes) {  
 rgb\_2\_cmy(pix\_data, all\_bytes);  
 }  
}

Название файла: ./GeometryAndGraphics/Lab\_04/ write\_data\_pnm\_v2.h

Исходный код:

//  
// Created by dudko on 11.03.2020.  
//  
  
#ifndef LAB\_03\_WRITE\_DATA\_PNM\_H  
#define LAB\_03\_WRITE\_DATA\_PNM\_H  
  
#include <iostream>  
  
void write\_header(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value);  
void write\_data(FILE \*file\_out, int k\_bytes, unsigned char \*pix\_data);  
int read\_header(char &char\_header, int &width, int &height, unsigned int &max\_value, FILE \*file\_in);  
void read\_data(FILE \*file\_in, int k\_bytes, unsigned char \*pix\_data);  
void free\_data(FILE \*file\_in, FILE \*file\_out, unsigned char \*pix\_data);  
void free\_data(FILE \*file, unsigned char \*pix\_data);  
void free\_data(FILE \*file);  
void free\_data(unsigned char \*pix\_data);  
namespace color {  
 void write\_header(FILE \*file\_out, int width, int height, unsigned int max\_value);  
 void write\_to\_file(FILE \*file\_out, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data);  
}  
  
namespace gray {  
 void write\_header(FILE \*file\_out, int width, int height, unsigned int max\_value);  
 void write\_to\_file(FILE \*file\_out, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data);  
}  
  
void write\_to\_file(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data);  
  
unsigned char limit\_brightness(double brightness);  
  
double to\_sRGB(double \_brightness);  
double to\_sRGB(int brightness);  
double from\_sRGB(double \_brightness);  
double from\_sRGB(int brightness);  
  
double change\_pix\_gamma\_to\_print(double \_brightness, double gamma);  
double change\_pix\_gamma\_to\_print(unsigned char pix\_data, double gamma);  
  
double change\_pix\_gamma\_from\_file(double \_brightness, double gamma);  
double change\_pix\_gamma\_from\_file(unsigned char pix\_data, double gamma);  
  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, int brightness, double gamma);  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, double \_brightness, double gamma);  
  
void decode\_gamma\_from\_file(unsigned char \*pix\_data, int k, double gamma);  
  
#endif //LAB\_03\_WRITE\_DATA\_PNM\_H

Название файла: ./GeometryAndGraphics/Lab\_04/ write\_data\_pnm\_v2.cpp

Исходный код:

//  
// Created by dudko on 11.03.2020.  
//  
  
#include "write\_data\_pnm\_v2.h"  
#include <cmath>  
#include <algorithm>  
#include <string>  
  
void write\_header(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value) {  
 fseek(file\_out, 0, SEEK\_SET);  
 fprintf(file\_out, "P%c\n%i %i\n%i\n", char\_header, width, height, max\_value);  
}  
  
void write\_data(FILE \*file\_out, int k\_bytes, unsigned char \*pix\_data) {  
 fwrite(pix\_data, k\_bytes, 1, file\_out);  
}  
  
int read\_header(char &char\_header, int &width, int &height, unsigned int &max\_value, FILE \*file\_in) {  
 int scanned = fscanf(file\_in, "P%c\n%i %i\n%i\n", &char\_header, &width, &height, &max\_value);  
 if (scanned != 4)  
 return 1;  
 if (width <= 0 || height <= 0 || max\_value <= 0 || max\_value > 255 || (char\_header != '5' && char\_header != '6'))  
 return 2;  
 return 0;  
}  
  
void read\_data(FILE \*file\_in, int k\_bytes, unsigned char \*pix\_data) {  
 if (!pix\_data)  
 return;  
 fread(pix\_data, 1, k\_bytes, file\_in);  
}  
  
void free\_data(FILE \*file\_in, FILE \*file\_out, unsigned char \*pix\_data) {  
 if (file\_in) fclose(file\_in);  
 if (file\_out) fclose(file\_out);  
 if (pix\_data) free(pix\_data);  
}  
  
void free\_data(FILE \*file, unsigned char \*pix\_data) {  
 free\_data(nullptr, file, pix\_data);  
}  
  
void free\_data(FILE \*file) {  
 free\_data(file, nullptr);  
}  
  
void free\_data(unsigned char \*pix\_data) {  
 free\_data(nullptr, pix\_data);  
}  
  
namespace color {  
 void write\_header(FILE \*file\_out, int width, int height, unsigned int max\_value) {  
 fseek(file\_out, 0, SEEK\_SET);  
 fprintf(file\_out, "P%c\n%i %i\n%i\n", '6', width, height, max\_value);  
 }  
  
 void write\_to\_file(FILE \*file\_out, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data) {  
 color::write\_header(file\_out, width, height, max\_value);  
 write\_data(file\_out, 3 \* width \* height, pix\_data);  
 }  
}  
  
namespace gray {  
 void write\_header(FILE \*file\_out, int width, int height, unsigned int max\_value) {  
 fseek(file\_out, 0, SEEK\_SET);  
 fprintf(file\_out, "P%c\n%i %i\n%i\n", '5', width, height, max\_value);  
 }  
  
 void write\_to\_file(FILE \*file\_out, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data) {  
 gray::write\_header(file\_out, width, height, max\_value);  
 write\_data(file\_out, width \* height, pix\_data);  
 }  
}  
  
void write\_to\_file(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data) {  
 write\_header(file\_out, char\_header, width, height, max\_value);  
 write\_data(file\_out, width \* height, pix\_data);  
}  
  
// brightness in [0..255] scale  
unsigned char limit\_brightness(double brightness) {  
 return (unsigned char) std::min(255.0, std::max(0.0, brightness));  
}  
  
double from\_sRGB(double \_brightness) {  
 if (\_brightness <= 0.0031308) {  
 return 323.0 \* \_brightness / 25.0;  
 } else {  
 return (211 \* pow(\_brightness, 5.0 / 12.0) - 11) / 200.0;  
 }  
}  
  
double from\_sRGB(int brightness) {  
 double \_brightness = brightness / 255.0;  
 return from\_sRGB(\_brightness);  
}  
  
double to\_sRGB(double \_brightness) {  
 if (\_brightness <= 0.04045) {  
 return 25.0 \* \_brightness / 323;  
 } else {  
 return pow((200 \* \_brightness + 11) / 211.0, 12.0 / 5.0);  
 }  
}  
  
double to\_sRGB(int brightness) {  
 double \_brightness = brightness / 255.0;  
 return to\_sRGB(\_brightness);  
}  
  
double change\_pix\_gamma\_to\_print(double \_brightness, double gamma) {  
 // Гамма коррекция:  
 if (gamma > 0) {  
 \_brightness = std::pow(\_brightness, 1.0 / gamma);  
 } else {  
 \_brightness = from\_sRGB(\_brightness);  
 }  
  
 return \_brightness;  
}  
  
double change\_pix\_gamma\_to\_print(unsigned char pix\_data, double gamma) {  
 double \_brightness = pix\_data / 255.0;  
 return change\_pix\_gamma\_to\_print(\_brightness, gamma);  
}  
  
double change\_pix\_gamma\_from\_file(double \_brightness, double gamma) {  
 if (gamma > 0) {  
 \_brightness = std::pow(\_brightness, gamma);  
 } else {  
 \_brightness = to\_sRGB(\_brightness);  
 }  
  
 return \_brightness;  
}  
  
double change\_pix\_gamma\_from\_file(unsigned char pix\_data, double gamma) {  
 double \_brightness = pix\_data / 255.0;  
 return change\_pix\_gamma\_from\_file(\_brightness, gamma);  
}  
  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, int brightness, double gamma) {  
 double \_brightness = brightness / 255.0;  
  
 // Гамма коррекция:  
 change\_pix\_gamma\_to\_print(\_brightness, gamma);  
  
 pix\_data[width \* y + x] = limit\_brightness(255 \* \_brightness);  
}  
  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, double \_brightness, double gamma) {  
 change\_pix\_gamma\_to\_print(\_brightness, gamma);  
 pix\_data[width \* y + x] = limit\_brightness(255 \* \_brightness);  
}  
  
void decode\_gamma\_from\_file(unsigned char \*pix\_data, int k, double gamma) {  
 for (int i = 0; i < k; ++i) {  
 \*(pix\_data + i) = limit\_brightness(255 \* change\_pix\_gamma\_from\_file(\*(pix\_data + i), gamma));  
 }  
}