**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное

учреждение высшего образования

**Санкт-Петербургский исследовательский университет**

**Информационных технологий, механики и оптики**

Мегафакультет Трансляционных информационных технологий

Факультет Информационных технологий и программирования

**Отчет**

по лабораторной работе № 5

По дисциплине «Компьютерная геометрия и графика»

Изучение алгоритма настройки автояркости изображения

Выполнила: студент гр. M3101

Тарасов Денис Евгеньевич

Преподаватель: Скаков П.С.

Санкт-Петербург

2020

**Цель работы**

Реализовать программу, 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).

**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>".

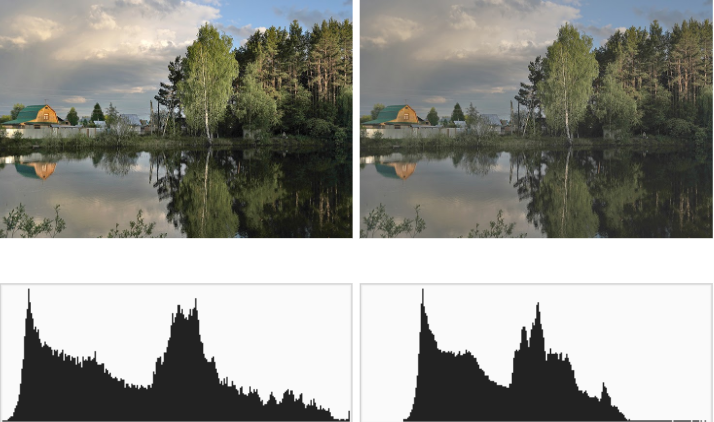
Сообщения об ошибках выводятся в поток вывода ошибок:

С: fprintf(stderr, "Error\n");

C++: std::cerr

**Теоретическая часть**

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

Преобразование

|  |  |  |
| --- | --- | --- |
|  | RGB-> | ->RGB |
| YCbCr.601 |  |  |

**Экспериментальная часть**

Язык программирования: С++ 17

Этапы выполнения:

1) Читаем картинку

2) Если нужно выполнить преобразования 1,3,5, то переводим картинку в YCrCb.601, а потом выполняем преобразования. Опять, если это преобразования 1,3,5, то переводим из YCrCb.601 в RGB

3)Записываем картинку

**Выводы**

Выполнение работы позволило узнать об алгоритмах корректировки яркости изображения. Были реализованы:

* Корректировка на определенные значения
* Корректировка с помощью поиска максимума и минимума
* Корректировка с помощью К-го максимума и К-го минимума

Все было сделано в пространствах

**Листинг**

Source.cpp

#include "PPM.h"

#include <iostream>

#include <vector>

#include <fstream>

#include <string>

using namespace std;

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

if (argc != 11) {

cerr << "Too few arguments";

return 1;

}

string inpFileName;

string outFileName;

int count1;

int count2;

string From;

string To;

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

if (string(argv[i]) == "-f" && i < argc - 1) {

From = string(argv[i + 1]);

}

if (string(argv[i]) == "-t" && i < argc - 1) {

To = string(argv[i + 1]);

}

if (string(argv[i]) == "-i" && i < argc - 2) {

count1 = atoi(argv[i + 1]);

inpFileName = string(argv[i + 2]);

}

if (string(argv[i]) == "-o" && i < argc - 2) {

count2 = atoi(argv[i + 1]);

outFileName = string(argv[i + 2]);

}

}

PPM\* pict;

try {

pict = new PPM(count1, inpFileName);

pict->ColorSpace\_changes(From, To);

pict->write(count2, outFileName);

}

catch (exception& ex) {

cerr << ex.what();

return 1;

}

delete pict;

return 0;

}

PPM.h

#pragma once

#include <vector>

#include <iostream>

#include <fstream>

#include <cmath>

#include <algorithm>

#include <string>

using namespace std;

struct Pixel

{

unsigned char r, g, b;

};

class PPM {

private:

char vers[2];

int width;

int height;

int colorDepth;

vector<vector<Pixel>> pict;

int rround(int a) {

if (a > colorDepth) {

return colorDepth;

}

else if (a < 0) {

return 0;

}

else {

return a;

}

}

public:

PPM(int count, string inpFileName) {

if (count == 1) {

ifstream inpFile(inpFileName, ios::binary);

if (!inpFile.is\_open())

throw runtime\_error("Cant open File");

inpFile >> vers[0] >> vers[1];

if (vers[0] != 'P' || vers[1] != '6')

throw runtime\_error("Wrong Format");

inpFile >> width >> height >> colorDepth;

pict.assign(height, vector<Pixel>(width));

char p;

inpFile.read(&p, 1);

for (int i = 0; i < height; i++)

for (int j = 0; j < width; j++)

{

Pixel cur;

inpFile.read(&p, sizeof(unsigned char));

cur.r = p;

inpFile.read(&p, sizeof(unsigned char));

cur.g = p;

inpFile.read(&p, sizeof(unsigned char));

cur.b = p;

pict[i][j] = cur;

}

inpFile.close();

}

else {

int dotNumb = -1;

for (int i = inpFileName.size() - 1; i >= 0; i--) {

if (inpFileName[i] == '.') {

dotNumb = i;

break;

}

}

if (dotNumb == -1) {

throw runtime\_error("Cant Open File");

}

string beforeDot = inpFileName.substr(0, dotNumb);

string AfterDot = inpFileName.substr(dotNumb);

vector<string> files;

files.resize(3);

files[0] = beforeDot + "\_1"+AfterDot;

files[1] = beforeDot + "\_2" + AfterDot;

files[2] = beforeDot + "\_3" + AfterDot;

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

{

ifstream inpFile(files[k], ios::binary);

if (!inpFile.is\_open()) {

throw runtime\_error("Cant Open File ");

}

inpFile >> vers[0] >> vers[1];

if (vers[0] != 'P' || vers[1] != '5') {

throw runtime\_error("Wrong Format");

}

inpFile >> width >> height >> colorDepth;

if (pict.size() == 0)

pict.assign(height, vector<Pixel>(width));

char p;

inpFile.read(&p, 1);

for (int i = 0; i < height; i++)

for (int j = 0; j < width; j++)

{

inpFile.read(&p, sizeof(unsigned char));

if (k == 0) {

pict[i][j].r = p;

}

else if (k == 1) {

pict[i][j].g = p;

}

else {

pict[i][j].b = p;

}

}

inpFile.close();

}

}

}

void write(int count, string outFileName) {

if (count == 1) {

ofstream outFile(outFileName, ios::binary);

if (!outFile.is\_open()) {

throw runtime\_error("cannot open output file");

}

outFile << "P6**\n**" << width << ' ' << height << '**\n**' << colorDepth << '**\n**';

for (int i = 0; i < height; i++)

for (int j = 0; j < width; j++)

{

outFile << pict[i][j].r;

outFile << pict[i][j].g;

outFile << pict[i][j].b;

}

outFile.close();

}

else {

int dotNumb = -1;

for (int i = outFileName.size()-1; i >= 0; i--) {

if (outFileName[i] == '.') {

dotNumb = i;

break;

}

}

if (dotNumb == -1)

throw runtime\_error("Cant open File");

string beforeDot = outFileName.substr(0, dotNumb);

string AfterDot = outFileName.substr(dotNumb);

vector<string> files;

files.resize(3);

files[0] = beforeDot + "\_1" + AfterDot;

files[1] = beforeDot + "\_2" + AfterDot;

files[2] = beforeDot + "\_3" + AfterDot;

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

{

ofstream outFile(files[k], ios::binary);

if (!outFile.is\_open()) {

throw runtime\_error("Cant open output file ");

}

outFile << "P5**\n**" << width << ' ' << height << '**\n**' << colorDepth << '**\n**';

for (int i = 0; i < height; i++)

for (int j = 0; j < width; j++)

{

if (k == 0) {

outFile << pict[i][j].r;

}

else if (k == 1) {

outFile << pict[i][j].g;

}

else {

outFile << pict[i][j].b;

}

}

outFile.close();

}

}

}

void From\_RGB\_to\_HSL() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

Pixel curr = pict[i][j];

double r = curr.r \* 1.0 / double(colorDepth);

double g = curr.g \* 1.0 / double(colorDepth);

double b = curr.b \* 1.0 / double(colorDepth);

double maxi = max(r, max(g, b));

double mini = min(r, min(g, b));

double H;

double L = (maxi + mini) / 2;

double S = (maxi - mini) / (1 - abs(1 - maxi - mini));

if (maxi == mini) {

H = 0;

}

else if (r == maxi && g >= b) {

H = ((g - b) / (maxi - mini)) \* 60;

}

else if (r == maxi && g < b) {

H = (60 \* ((b - r) / (maxi - mini))) + 360;

}

else if (maxi == g) {

H = (60 \* (b - r) / (maxi - mini)) + 120;

}

else {

H = (60 \* ((r - g) / (maxi - mini))) + 240;

}

pict[i][j] = { unsigned char(round(H \* colorDepth / 360)),unsigned char(round(S \* colorDepth)),unsigned char(round(L \* colorDepth)) };

}

}

}

void From\_HSL\_to\_RGB() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

double H = pict[i][j].r / double(colorDepth);

double S = pict[i][j].g / double(colorDepth);

double L = pict[i][j].b / double(colorDepth);

double Q, P;

if (L < 0.5) {

Q = L \* (1.0 + S);

}

else {

Q = L + S - (L \* S);

}

P = 2 \* L - Q;

double Tr = H + (1.0 / 3);

double Tg = H;

double Tb = H - (1.0 / 3);

if (Tr < 0)Tr += 1.0;

if (Tr > 1.0)Tr -= 1.0;

if (Tg < 0)Tg += 1.0;

if (Tg > 1.0)Tg -= 1.0;

if (Tb < 0)Tb += 1.0;

if (Tb > 1.0)Tb -= 1.0;

double R, G, B;

if (Tr < 1.0 / 6) {

R = P + ((Q - P) \* 6 \* Tr);

}

else if (Tr >= 1.0 / 6 && Tr < 1.0 / 2) {

R = Q;

}

else if (Tr >= 1.0 / 2 && Tr < 2.0/ 3) {

R = P + ((Q - P) \* ((2.0/ 3) - Tr) \* 6);

}

else {

R = P;

}

if (Tg < 1.0 / 6) {

G = P + ((Q - P) \* 6 \* Tg);

}

else if (Tg >= 1.0 / 6 && Tg < 1.0 / 2) {

G = Q;

}

else if (Tg >= 1.0 / 2 && Tg < 2.0/ 3) {

G = P + ((Q - P) \* ((2.0/ 3) - Tg) \* 6);

}

else {

G = P;

}

if (Tb < 1.0 / 6) {

B = P + ((Q - P) \* 6 \* Tb);

}

else if (Tb >= 1.0 / 6 && Tb < 1.0 / 2) {

B = Q;

}

else if (Tb >= 1.0 / 2 && Tb < 2.0/ 3) {

B = P + ((Q - P) \* ((2.0/ 3) - Tb) \* 6);

}

else {

B = P;

}

R = R \* colorDepth;

G = G \* colorDepth;

B = B \* colorDepth;

pict[i][j] = { unsigned char(rround(round(R))), unsigned char(rround(round(G))),unsigned char(rround(round(B))) };

}

}

}

void From\_RGB\_to\_HSV() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

double R = pict[i][j].r \* 1.0 / double(colorDepth);

double G = pict[i][j].g \* 1.0 / double(colorDepth);

double B = pict[i][j].b \* 1.0 / double(colorDepth);

double maxi = max(R, max(G, B));

double mini = min(R, min(G, B));

double delta = maxi - mini;

double H, S, V;

if (delta == 0) {

H = 0;

}

else if (R == maxi) {

H = 60 \* ((G - B) / delta);

}

else if (G == maxi) {

H = 60 \* (((B - R) / delta) + 2);

}

else {

H = 60 \* (((R - G) / delta) + 4);

}

if (maxi == 0) {

S = 0;

}

else {

S = delta / maxi;

}

V = maxi;

if (H < 0) {

H += 360;

}

pict[i][j] = { unsigned char(round(H \* double(colorDepth) / 360.0)),unsigned char(rround(round(S \* double(colorDepth)))),unsigned char(rround(round(V \* double(colorDepth)))) };

}

}

}

void From\_HSV\_to\_RGB() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

double H = pict[i][j].r / 255.0 \* 360.0;

double S = pict[i][j].g / 255.0;

double V = pict[i][j].b / 255.0;

double C = V \* S;

double Hi = H / 60;

double X = C \* (1 - abs(((int)Hi) % 2 + (Hi - (int)Hi) - 1));

double m = V - C;

double R, G, B;

if (H >= 0 && H <= 60)

R = C, G = X, B = 0;

else if (H >= 60 && H <= 120)

R = X, G = C, B = 0;

else if (H >= 120 && H <= 180)

R = 0, G = C, B = X;

else if (H >= 180 && H <= 240)

R = 0, G = X, B = C;

else if (H >= 240 && H <= 300)

R = X, G = 0, B = C;

else

R = C, G = 0, B = X;

int Rr = (int)(round((R + m) \* 255));

int Gg = (int)(round((G + m) \* 255));

int Bb = (int)(round((B + m) \* 255));

if (Rr < 0) Rr = 0;

if (Rr > 255) Rr = 255;

if (Bb < 0) Bb = 0;

if (Bb > 255) Bb = 255;

if (Gg < 0) Gg = 0;

if (Gg > 255) Gg = 255;

pict[i][j]={ (unsigned char)Rr, (unsigned char)Gg, (unsigned char)Bb };

}

}

}

void From\_RGB\_to\_YCbCr601() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

double Kr = 0.299;

double Kg = 0.576;

double Kb = 0.114;

double R = pict[i][j].r / double(colorDepth);

double G = pict[i][j].g / double(colorDepth);

double B = pict[i][j].b / double(colorDepth);

double Y = Kr \* R + Kg \* G + Kb \* B;

double Cb = (B / 2) - (Kr \* R / (2 \* (1 - Kb))) - (Kg \* G / (2 \* (1 - Kb)));

double Cr = (R / 2) - (G \* Kg / (2 \* (1 - Kr))) - (B \* Kb / (2 \* (1 - Kr)));

pict[i][j] = { unsigned char(rround(int(round(Y \* colorDepth)))),unsigned char(rround(int(round((Cb + 0.5) \* colorDepth)))),unsigned char(rround(int(round((Cr + 0.5) \* colorDepth)))) };

}

}

}

void From\_YCbCr601\_to\_RGB() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

double Kr = 0.299;

double Kg = 0.576;

double Kb = 0.114;

double Y = pict[i][j].r / double(colorDepth);

double Cb = (pict[i][j].g / double(colorDepth)) - 0.5;

double Cr = (pict[i][j].b / double(colorDepth)) - 0.5;

double R = Y + Cr \* (2 - 2 \* Kr);

double G = Y - (Cb\*(Kb \* (2 - 2 \* Kb) / Kg)) - (Cr\*(Kr \* (2-2 \* Kr) / Kg));

double B = Y + Cb \* (2 - 2 \* Kb);

pict[i][j] = { unsigned char(rround(int(round(R \* colorDepth)))),unsigned char(rround(int(round(G \* colorDepth)))),unsigned char(rround(int(round(B \* colorDepth)))) };

}

}

}

void From\_RGB\_to\_YCbCr709(){

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

double Kr = 0.2126;

double Kg = 0.7152;

double Kb = 0.0722;

double R = double(pict[i][j].r) / double(colorDepth);

double G = double(pict[i][j].g) / double(colorDepth);

double B = double(pict[i][j].b) / double(colorDepth);

//cout << Kb << " " << Kr << " " << Kg << " "<<R<<" "<<G<<" "<<B<<"\n";

double Y = Kr \* R + Kg \* G + Kb \* B;

double Cb = (B - Y) / (2.0 \* (1.0 - Kb));

double Cr = (R - Y) / (2.0 \* (1.0 - Kr));

pict[i][j] = { unsigned char(rround(int(round(Y \* colorDepth)))),unsigned char(rround(int(round((Cb + 0.5) \* colorDepth)))),unsigned char(rround(int(round((Cr + 0.5) \* colorDepth)))) };

}

}

}

void From\_YCbCr709\_to\_RGB() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

double Kb = 0.0722;

double Kr = 0.2126;

double Kg = 0.7152;

double Y = double(pict[i][j].r) / double(colorDepth);

double Cb = (double(pict[i][j].g )/ double(colorDepth)) - 0.5;

double Cr = (double(pict[i][j].b) / double(colorDepth)) - 0.5;

double R = Y + Cr \* (2.0 - 2.0\* Kr);

double G = Y - (Kb \* (2.0 - 2.0 \* Kb) / Kg)\*Cb - Cr\*(Kr \* (2.0-2.0 \* Kr) / Kg);

double B = Y + Cb \* (2.0 - 2.0 \* Kb);

pict[i][j] = { unsigned char(rround(int(round(R \* colorDepth)))),unsigned char(rround(int(round(G \* colorDepth)))),unsigned char(rround(int(round(B \* colorDepth)))) };

}

}

}

void From\_RGB\_to\_YCoCg() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

double R = pict[i][j].r \* 1.0;

double G = pict[i][j].g \* 1.0;

double B = pict[i][j].b \* 1.0;

double Y = (R / 4) + (G / 2) + (B / 4);

double Co = (R / 2) - (B / 2) + 0.5 \* colorDepth;

double Cg = (G / 2) - (R / 4) - (B / 4) + 0.5 \* colorDepth;

pict[i][j] = { unsigned char(rround(int(Y))),unsigned char(rround(int(Co))),unsigned char(rround(int(Cg))) };

}

}

}

void From\_YCoCg\_to\_RGB() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

double Y = pict[i][j].r;

double Co = pict[i][j].g \* 1.0 - 0.5 \* colorDepth;

double Cg = pict[i][j].b \* 1.0 - 0.5 \* colorDepth;

double R = Y + Co - Cg;

double G = Y + Cg;

double B = Y - Co - Cg;

pict[i][j] = { unsigned char(rround(int(R))),unsigned char(rround(int(G))),unsigned char(rround(int(B))) };

}

}

}

void From\_RGB\_to\_CMY() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

double R = pict[i][j].r \* 1.0;

double G = pict[i][j].g \* 1.0;

double B = pict[i][j].b \* 1.0;

pict[i][j] = { unsigned char(colorDepth - R),unsigned char(colorDepth - G),unsigned char(colorDepth - B) };

}

}

}

void From\_CMY\_to\_RGB() {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

double C = pict[i][j].r \* 1.0;

double M = pict[i][j].g \* 1.0;

double Y = pict[i][j].b \* 1.0;

pict[i][j] = { unsigned char(colorDepth - C),unsigned char(colorDepth - M),unsigned char(colorDepth - Y) };

}

}

}

void ColorSpace\_changes(string From,string To) {

if (From != "RGB") {

if (From == "HSL") {

From\_HSL\_to\_RGB();

}

else if (From == "HSV") {

From\_HSV\_to\_RGB();

}

else if (From == "YCbCr.601") {

From\_YCbCr601\_to\_RGB();

}

else if (From == "YCbCr.709") {

From\_YCbCr709\_to\_RGB();

}

else if (From == "YCoCg") {

From\_YCoCg\_to\_RGB();

}

else if (From == "CMY") {

From\_CMY\_to\_RGB();

}

else {

throw exception("There is no such Color Space");

}

}

if (To != "RGB") {

if (To == "HSL") {

From\_RGB\_to\_HSL();

}

else if (To == "HSV") {

From\_RGB\_to\_HSV();

}

else if (To == "YCbCr.601") {

From\_RGB\_to\_YCbCr601();

}

else if (To == "YCbCr.709") {

From\_RGB\_to\_YCbCr709();

}

else if (To == "YCoCg") {

From\_RGB\_to\_YCoCg();

}

else if (To == "CMY") {

From\_RGB\_to\_CMY();

}

else {

throw exception("There is no such Color Space");

}

}

}

};