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

Компьютерная графика и геометрия

Отчет
по лабораторной работе №5
Изучение алгоритма настройки автояркости изображения

Выполнил: студент гр. М3101
Ершов Михаил Юрьевич
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).

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:
C: fprintf(stderr, "Error\n");
C++: std::cerr

Теоретическая часть

1. Модель зрения человека

Представление и обработка графической информации в вычислительных системах основаны на наших знаниях о модели зрения человека.
Не только регистрация и отображение изображений стараются соответствовать системе зрения человека, но и алгоритмы кодирования и сжатия данных становятся намного эффективнее при учёте того, что видит и не видит человек.

Согласно современным представлениям, система зрения человека имеет 4 вида рецепторов:

- 3 вида “колбочек”: S (short), M (medium), L (long), отвечающих за цветное зрение. Работают только при высокой освещённости.
- 1 вид “палочек”: R (rods), позволяющих регистрировать яркость. Работают только при низкой освещённости.

Система цветного зрения человека трёхкомпонентная: воспринимаемый цвет описывается тремя значениями. Любые спектры излучения, приводящие к одинаковым этим трём значениям, неразличимы для человека.

Регистрация спектра L, M и S рецепторами лежит в основе цветовой модели RGB, описывающей цвет как комбинацию красного, синего и зелёного.
SML сигнал (что условно соответствует RGB) преобразуется следующим образом:
Y = **S** + **M** + **L**
A = **L** - **M**
B = (**L** + **M**) - **S**

То есть представление красный-зелёный-синий превращается в яркость (**Y**) и две цветоразницы: красно-зелёную (**A**) и жёлто-синюю (**B**).

Всё это послужило основой для различных цветоразностных систем представления цвета, например, YUV (альтернативное название: YCbCr), широко используемых при эффективном кодировании и сжатии графической информации.

2. Цветовые пространства

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.

Цветовые пространства бывают аддитивные (например, RGB) и субтрактивные (например, CMY). В аддитивных пространствах 0 соответствует чёрному цвету, а 100% всех компонент – белому. Это отражает работу источников света, например, отображение информации на мониторе. В субтрактивных наоборот: отсутствие компонент – это белый, а полное присутствие – чёрный. Это соответствует смешению красок на бумаге.

3. RGB

Пространство RGB – это самое широко используемое цветовое пространство. Его компоненты примерно соответствуют трём видам наших цветовых рецепторов: L, M, S.

R (Red) – красный
G (Green) – зелёный
B (Blue) – синий

Типичный диапазон значений: 0..255 для каждой компоненты, но возможны и другие значения, например, 0..1023 для 10-битных данных.

4. YUV/YCbCr

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

Y – яркость

U/Cb – цветоразность “хроматический синий”

V/Cr – цветоразность “хроматический красный”

В пространстве YUV традиционно существует два диапазона значений.

Для 8-битных данных:

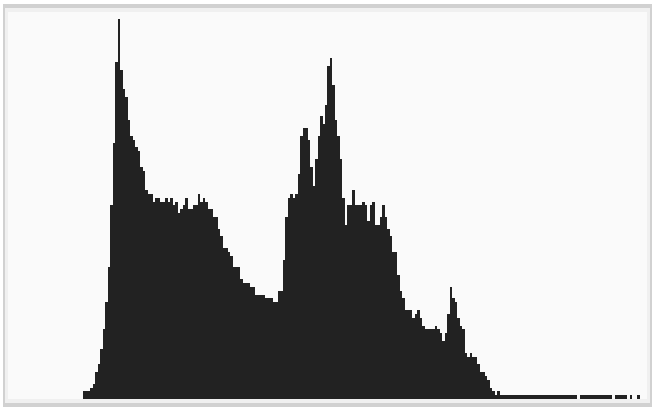
PC уровни	TV уровни
Y: 0..255	Y: 16..235
U: 0..255	U: 16..240
V: 0..255	V: 16..240

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

5. Коррекция яркости и контрастности изображений

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.

Экспериментальная часть

Язык программирования: C++14

Частичное решение.

Считываем изображение из PPM файла и либо применяем к каждому пикселю указанные в вводе значения смещения и множителя в пространствах RGB или YCbCr.601, либо же определяем эти значения самостоятельно на основе максимального и минимального значений пикселей в пространствах RGB илиYCbCr.601, то есть применяем алгоритм автояркости.

1. Формула перехода для RGB/YCbCr.601
   Первая тройка формул – переход из RGB в YCbCr.601, вторая, под таблицей, – переход из YCbCr.601 в RGB. Коэффициенты записаны в таблице, в столбце BT.601.

```
Y  = a * R + b * G + c * B
Cb = (B - Y) / d
Cr = (R - Y) / e
```

	BT.601	BT.709	BT.2020
a	0.299	0.2126	0.2627
b	0.587	0.7152	0.6780
c	0.114	0.0722	0.0593
d	1.772	1.8556	1.8814
e	1.402	1.5748	1.4747

```
R = Y + e * Cr
G = Y - (a * e / b) * Cr - (c * d / b) * Cb
B = Y + d * Cb
```

2. Применение коэффициентов смещения и множителя для пространств RGB/YCbCr.601 и определение коэффициентов для автояркости в пространствах RGB/YCbCr.601.

Так как в пространстве RGB необходимо производить изменение значения пикселя по каждому каналу, и коэффициенты должны быть одинаковы для всех каналов, то при применении автояркости максимум и минимум ищутся среди всех каналов, после чего к каждому каналу каждого пикселя применяется формула:

$$y = (x - offset)*mult$$

где offset – значение смещения, либо значение минимума для случая автояркости, mult – значение множителя, либо значение 255 / (max – min) для случая автояркости.

В случае YCbCr пространства яркость зависит только от канала Y, поэтому максимум и минимум для автояркости ищутся только по этому каналу, и изменяется только значение этого канала по той же, вышеприведённой формуле.

Выводы

Выполнение данной работы позволило изучить алгоритм изменения яркости изображения, когда значения смещения и множителя известны, и алгоритм автояркости изображения, когда значения смещения и множителя требуется определить самостоятельно, в цветовых пространствах RGB и YCbCr.601.

Листинг

main.cpp

```
#include <iostream>
#include <string>
#include <cstdlib>
#include "pgm.h"

using namespace std;

int main(int argc, char* argv[]) {
    if (argc != 4 && argc != 6) {
        cerr << "Invalid number of arguments\n";
        return 1;
    }
    try {
        PPM image(argv[1]);
        int it = atoi(argv[3]);
        pair<int, double> res;
        switch (it) {
        case 0:
            image.changeRGB(atoi(argv[4]), stod(argv[5]));
            break;
        case 1:
            image.changeYCbCr_601(atoi(argv[4]), stod(argv[5]));
            break;
        case 2:
            res = image.autoRGB();
            cout << res.first << " " << res.second << "\n";
            break;
        case 3:
            res = image.autoYCbCr_601();
            cout << res.first << " " << res.second << "\n";
            break;
        default:
            cerr << "Invalid transformation\n";
            return 1;
        }
        image.print(argv[2]);
    }
    catch (const exception& e) {
        cerr << e.what() << "\n";
        return 1;
    }
    return 0;
}
```

pgm.h

```
#ifndef LAB5_PGM_H
#define LAB5_PGM_H

#include <vector>
#include <cstdio>
#include <cstring>

typedef unsigned char uchar;

struct Pixel {
    uchar first, second, third;
    Pixel(uchar first, uchar second, uchar third) : first(first), second(second), third(third) {}
    Pixel() = default;
};

class PPM {
private:
    char header[2];
    int width, height;
    uchar maxValue;
    Pixel* data;
    static Pixel changeRGBPixel(const Pixel& pixel, const int& offset, const double& mult);
    static Pixel changeYCbCr_601Pixel(const Pixel& pixel, const int& offset, const double& mult);
    static Pixel convertRGBtoYCbCr_601(const Pixel& pixel);
    static Pixel convertYCbCr_601toRGB(const Pixel& pixel);
public:
    explicit PPM(char* fileName);
    ~PPM();
    void print(char* fileName);
    void changeRGB(int offset, double mult);
    void changeYCbCr_601(int offset, double mult);
    std::pair<int, double> autoRGB();
    std::pair<int, double> autoYCbCr_601();
};
#endif //LAB5_PGM_H
```

pgm.cpp

```
#include <stdexcept>
#include <iostream>
#include <algorithm>
#include "pgm.h"

PPM::PPM(char* fileName) {
    FILE* fin = fopen(fileName, "rb");
    if (fin == nullptr) {
        throw std::runtime_error("File can't be opened\n");
    }
    int tmp;
    if (fscanf(fin, "%s%d%d%d", header, &width, &height, &tmp) < 4) {
        throw std::runtime_error("Invalid header\n");
    }
    if (strcmp(header, "P6") != 0 && strcmp(header, "P5") != 0) {
        throw std::runtime_error("File is not a PGM image\n");
    }
    if (tmp > 255) {
        throw std::runtime_error("Unsupported colours\n");
    }
    maxValue = (uchar)tmp;
    if (width < 0 || height < 0) {
        throw std::runtime_error("Invalid header\n");
    }
    if (fgetc(fin) == EOF) {
        throw std::runtime_error("Invalid header\n");
    }
    data = new Pixel[width * height];
    if (strcmp(header, "P5") == 0) {
        uchar* dataP5 = new uchar[width * height];
        fread(dataP5, 1, width * height, fin);
        for (int i = 0; i < width * height; i++) {
            data[i] = Pixel(dataP5[i], dataP5[i], dataP5[i]);
        }
        delete[] dataP5;
    }
    else {
        fread(data, sizeof(Pixel), width * height, fin);
    }
    if (fclose(fin) != 0) {
        throw std::runtime_error("File can't be closed\n");
    }
}

void PPM::print(char* fileName) {
    FILE* fout = fopen(fileName, "wb");
    if (fout == nullptr) {
        throw std::runtime_error("File can't be opened or created\n");
    }
    fprintf(fout, "%s\n%i %i\n%i\n", header, width, height, maxValue);
    if (strcmp(header, "P5") == 0) {
        uchar* dataP5 = new uchar[width * height];
        for (int i = 0; i < width * height; i++) {
            dataP5[i] = data[i].first;
        }
        fwrite(dataP5, 1, width * height, fout);
        delete[] dataP5;
    }
    else {
        fwrite(data, sizeof(Pixel), width * height, fout);
    }
    fclose(fout);
}
```

```
PPM::~PPM() {
    delete[] data;
}

Pixel PPM::changeRGBPixel(const Pixel& pixel, const int& offset, const double& mult) {
    double r = std::max(std::min(((double)pixel.first - offset) * mult, 255.0), 0.0);
    double g = std::max(std::min(((double)pixel.second - offset) * mult, 255.0), 0.0);
    double b = std::max(std::min(((double)pixel.third - offset) * mult, 255.0), 0.0);
    return Pixel(r, g, b);
}

Pixel PPM::changeYCbCr_601Pixel(const Pixel& pixel, const int& offset, const double& mult) {
    double Y = std::max(std::min(((double)pixel.first - offset) * mult, 255.0), 0.0);
    return Pixel(Y, pixel.second, pixel.third);
}

Pixel PPM::convertYCbCr_601toRGB(const Pixel& pixel) {
    double Y = (double)pixel.first / 255.0;
    double Cb = (double)pixel.second / 255.0 - 0.5;
    double Cr = (double)pixel.third / 255.0 - 0.5;
    double R = Y + 1.402 * Cr;
    double G = Y - (0.299 * 1.402 / 0.587) * Cr - (0.114 * 1.772 / 0.587) * Cb;
    double B = Y + 1.772 * Cb;
    uchar r = std::max(std::min(255.0, 255 * R), 0.0);
    uchar g = std::max(std::min(255.0, 255 * G), 0.0);
    uchar b = std::max(std::min(255.0, 255 * B), 0.0);
    return Pixel(r, g, b);
}

Pixel PPM::convertRGBtoYCbCr_601(const Pixel& pixel) {
    double R = (double)pixel.first / 255.0;
    double G = (double)pixel.second / 255.0;
    double B = (double)pixel.third / 255.0;
    double Y = 0.299 * R + 0.587 * G + 0.114 * B;
    double Cb = (B - Y) / 1.772 + 0.5;
    double Cr = (R - Y) / 1.402 + 0.5;
    Y = std::max(std::min(Y, 1.0), 0.0);
    Cb = std::max(std::min(Cb, 1.0), 0.0);
    Cr = std::max(std::min(Cr, 1.0), 0.0);

    return Pixel(255 * Y, 255 * Cb, 255 * Cr);
}

void PPM::changeRGB(int offset, double mult) {
    for (int i = 0; i < width * height; i++) {
        data[i] = changeRGBPixel(data[i], offset, mult);
    }
}

void PPM::changeYCbCr_601(int offset, double mult) {
    for (int i = 0; i < width * height; i++) {
        data[i] = convertYCbCr_601toRGB(changeYCbCr_601Pixel(convertRGBtoYCbCr_601(data[i]),
            offset, mult));
    }
}

std::pair<int, double> PPM::autoRGB() {
    int minVal = 1000, maxVal = -1;
    for (int i = 0; i < width * height; i++) {
        minVal = std::min(minVal, (int)data[i].first);
        minVal = std::min(minVal, (int)data[i].second);
        minVal = std::min(minVal, (int)data[i].third);
        maxVal = std::max(maxVal, (int)data[i].first);
        maxVal = std::max(maxVal, (int)data[i].second);
```

```
            maxVal = std::max(maxVal, (int)data[i].third);
        }
        int offset = minVal;
        double mult = maxVal == minVal ? 255.0 : 255.0 / (maxVal - minVal);
        for (int i = 0; i < width * height; i++) {
            data[i] = changeRGBPixel(data[i], offset, mult);
        }
        return { offset, mult };
    }

    std::pair<int, double> PPM::autoYCbCr_601() {
        for (int i = 0; i < width * height; i++) {
            data[i] = convertRGBtoYCbCr_601(data[i]);
        }
        int minVal = 1000, maxVal = -1;
        for (int i = 0; i < width * height; i++) {
            minVal = std::min(minVal, (int)data[i].first);
            maxVal = std::max(maxVal, (int)data[i].first);
        }
        int offset = minVal;
        double mult = maxVal == minVal ? 255.0 : 255.0 / (maxVal - minVal);
        for (int i = 0; i < width * height; i++) {
            data[i] = convertYCbCr_601toRGB(changeYCbCr_601Pixel(data[i], offset, mult));
        }
        return { offset, mult };
    }
```