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 № 1
Изучение простых преобразований изображений

Выполнила: студент гр. M3101
Шмарина Л.С.
Преподаватель: Скаков П.С.

Санкт-Петербург
2020

Цель работы: Изучение алгоритмов и реализация программы, выполняющей простые преобразования серых и цветных изображений в формате PNM.

Описание работы

Программа должна поддерживать серые и цветные изображения (варианты PNM P5 и P6), самостоятельно определяя формат по содержимому.

Аргументы программе передаются через командную строку:

lab#.exe <имя_входного_файла> <имя_выходного_файла> <преобразование>

где <преобразование>:

0 - инверсия,

1 - зеркальное отражение по горизонтали,

2 - зеркальное отражение по вертикали,

3 - поворот на 90 градусов по часовой стрелке,

4 - поворот на 90 градусов против часовой стрелки.

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

Частичное решение: работают преобразования 0-2; имена файлов и преобразование, возможно, написаны в исходном коде или читаются с консоли, а не берутся из командной строки.

Полное решение: всё работает + корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок: не удалось открыть файл, формат файла не поддерживается, не удалось выделить память.

Теоретическая часть

Структура файла.

Заголовок файла: В начале файла содержится следующая структура данных, представляющих собой текст: Для изображений в градациях серого указывается: P5 Для изображений в формате RGB: P6 Далее следует одинарный перевод строки в формате LF (‘\n’ или 0x0A). Ширина и высота

изображения в десятичном виде через пробел. Перевод строки. Максимально возможное значение яркости (в данной работе гарантируется значение 255). Перевод строки. Данные изображения (в работе используются форматы P5 и P6, где все данные представлены в двоичном виде) Для формата P5: каждый байт представляет собой яркость пикселя. Для формата P6: каждый пиксель цветной, поэтому для каждого пикселя записываются подряд 3 байта яркости цветов в формате RGB.

Преобразования.

0 - инверсия - значение яркости каждого пикселя (если картинка цветная, то всех трех каналов по отдельности) заменяется на противоположное, то есть 255 минус исходной значение.

1 - зеркальное отражение по горизонтали - перестановка пикселей каждой строки симметрично относительно середины

2 - зеркальное отражение по вертикали - перестановка пикселей каждого столбца симметрично относительно середины

3 - поворот на 90 градусов по часовой стрелке - создание нового изображения путем записи столбцов изображения в строки нового изображения, начиная с крайнего правого столбца, заканчивая крайним левым, идя сверху вниз.

4 - поворот на 90 градусов против часовой стрелки - создание нового изображения путем записи столбцов изображения в строки нового изображения, начиная с крайнего левого столбца, заканчивая крайним правым, идя снизу вверх.

Экспериментальная часть

Язык программирования: C++ 14

Были созданы классы PBM и PPM для работы над цветными и черно-белыми изображениями соответственно, наследующие интерфейс NetPBM. Данные хедера записываются в поля классов. Для работы с цветными пикселями создана структура данных.

Преобразования:

0 - инверсия - отнимаем от 255 значение яркости пикселя и записываем результат в пиксель (для цветных изображений для трех каналов по отдельности), не используем дополнительную память

1 - зеркальное отражение по горизонтали - считываем строки по одной и в каждой строке меняем два крайних пикселя между собой в цикле от края к центру

2 - аналогично п.1

3 - поворот на 90 градусов по часовой стрелке - создание нового изображения путем записи столбцов изображения в строки нового изображения, начиная с крайнего правого столбца, заканчивая крайним левым, идя сверху вниз. Выделяется дополнительная память на хранение дополнительного изображение

4 - поворот на 90 градусов против часовой стрелки - создание нового изображения путем записи столбцов изображения в строки нового изображения, начиная с крайнего левого столбца, заканчивая крайним правым, идя снизу вверх. Выделяется дополнительная память на хранение дополнительного изображение

Выводы

В ходе выполнения работы я изучила формат PNM, его структуру и работу с ним, реализовала несколько алгоритмов. Не все алгоритмы были реализованы максимально эффективно: алгоритмы поворота на 90 градусов можно было реализовать без выделения такого объема дополнительной памяти, двигаясь от краев картинки к центру и циклично меняя пиксели с четырех сторон.

Листинг

Main.cpp

```
#include <iostream>

#include <cstdlib>
```

```cpp
#include <set>
#include "PPM.h"
#include "PBM.h"
#include "NetPBM.h"
using namespace std;

int main(int argc, char **argv) {
    try {
        if (argc == 666) {
            throw logic_error("Invalid user");
        }
        if (argc != 4) {
            throw logic_error("Invalid argument count");
        }
        char *infilename = argv[1];
        char *outfilename = argv[2];
        int conversion = atoi(argv[3]);
        auto *file = new ifstream(infilename, std::ios::binary);
        (*file).unsetf(ios_base::skipws);
        if (!(*file).is_open()) {
            delete file;
            throw logic_error("Can't open file to read");
        }
        unsigned char buf;
        *file >> buf;
        if (buf != 'P') {
            throw logic_error("Bad file");
        }
        *file >> buf;
```

```
        if (buf == '1' || buf == '2' || buf == '3' || buf == '4'
|| buf == '7') {
            throw logic_error("Not supported type of NetPCM");
        } else if (buf != '5' && buf != '6') {
            throw logic_error("Bad file");
        }
        NetPBM *picture;
        if (buf == '5') {
            picture = new PBM(file);
        }
        if (buf == '6') {
            picture = new PPM(file);
        }
        switch (conversion) {
            case 0:
                picture->inversion();
                break;
            case 1:
                picture->horizontal_mirror();
                break;
            case 2:
                picture->vertical_mirror();
                break;
            case 3:
                picture->rotate_90(true);
                break;
            case 4:
                picture->rotate_90(false);
                break;
```

```
        }


                    auto   *outfile   =   new   ofstream(outfilename,
std::ios::binary);
        if (!outfile->is_open()) {
            file->close();
            delete file;
            delete outfile;
            delete picture;
            throw logic_error("Can't open file to write");
        }


        picture->write_to_file(outfile);
        outfile->close();
        file->close();
        delete file;
        delete outfile;
        delete picture;
        return 0;
    }
    catch (const logic_error &error) {
        cerr << error.what() << endl;
        return 1;
    }
}
```

NetPBM.h

```
#include <iostream>
#include <fstream>
```

```cpp
#include <cstdint>
#include <cstdlib>
#include <cmath>
#include <set>

#ifndef LAB3_NETPBM_H
#define LAB3_NETPBM_H

using namespace std;

class NetPBM {
public:
//    virtual int16_t getType() const;
   virtual void write_to_file(ofstream *outfile) = 0;

   virtual void inversion() = 0;

   virtual void horizontal_mirror() = 0;

   virtual void vertical_mirror() = 0;

   virtual void rotate_90(bool isclock) = 0;

   virtual ~NetPBM() = default;

};

#endif //LAB3_NETPBM_H
```

PPM.h

```
#ifndef LAB4_PPM_H
#define LAB4_PPM_H

#include <iostream>
#include <fstream>
#include <cstdint>
#include <cstdlib>
#include <cmath>
#include "NetPBM.h"

using namespace std;
struct pixel_t {
    unsigned char R;
    unsigned char G;
    unsigned char B;

};

class PPM : public NetPBM {
private:
    ifstream *file;
    uint16_t type = -1;
    uint16_t width = -1;
    uint16_t height = -1;
    uint16_t depth = -1;
    pixel_t **array;

    void read_header() {
```

```
        unsigned char buf;
        *this->file >> buf;
        *this->file >> this->width;
        *this->file >> buf;
        *this->file >> this->height;
        *this->file >> buf;
        *this->file >> this->depth;
        if (this->depth != 255) {
                throw logic_error("Not supported non 255 colors count");
        }
    }

    void read_data() {
        for (int i = 0; i < this->height; i++) {
            for (int j = 0; j < this->width; j++) {
                unsigned char tmp;
                *this->file >> tmp;
                this->array[i][j].R = tmp;
                *this->file >> tmp;
                this->array[i][j].G = tmp;
                *this->file >> tmp;
                this->array[i][j].B = tmp;
            }
        }
    }

    void write_to_file(ofstream *outfile) override {
         *outfile << "P6" << (unsigned char) (10) << width << " " << height << (unsigned char) (10) << depth
```

```
                << (unsigned char) (10);
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                *outfile << array[i][j].R;
                *outfile << array[i][j].G;
                *outfile << array[i][j].B;
            }
        }
    }

public:
    void inversion() override {
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                array[i][j].R = 255 - array[i][j].R;
                array[i][j].G = 255 - array[i][j].G;
                array[i][j].B = 255 - array[i][j].B;
            }
        }
    }

    void horizontal_mirror() override {
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < (width - 1) / 2; j++) {
                pixel_t temp = array[i][j];
                array[i][j] = array[i][width - j - 1];
                array[i][width - j - 1] = temp;
            }
        }
```

```
}

void rotate_90(bool isclock) override {
    int tmp = this->width;
    this->width = this->height;
    this->height = tmp;
    pixel_t **arr;
    arr = new pixel_t *[height];
    for (int i = 0; i < this->height; i++) {
        arr[i] = new pixel_t[width];
        for (int j = 0; j < width; j++) {
            if (!isclock)
                arr[i][j] = this->array[j][height - i - 1];
            else
                arr[i][j] = this->array[width - 1 - j][i];
        }
    }
    for (int i = 0; i < width; i++) {
        delete[] array[i];
    }
    delete[] array;

    array = arr;
};

void vertical_mirror() override {
    for (int i = 0; i < (height - 1) / 2; i++) {
        for (int j = 0; j < width; j++) {
            int R, G, B;
```

```
                pixel_t temp = array[i][j];
                array[i][j] = array[height - i - 1][j];
                array[height - i - 1][j] = temp;
            }
        }
    }

    explicit
    PPM(ifstream *file) {
        this->file = file;
        this->read_header();
        file->ignore(1);
        this->array = new pixel_t *[height];
        for (int i = 0; i < this->height; i++) {
            this->array[i] = new pixel_t[width];
        }
        read_data();
    }

    ~PPM() override {
        for (int i = 0; i < height; i++) {
            delete[] array[i];
        }
        delete[] array;
    }
};

#endif //LAB4_PPM_H
```

PBM.h

```
#ifndef LAB4_PBM_H
#define LAB4_PBM_H

#include <iostream>
#include <fstream>
#include <cstdint>
#include <cstdlib>
#include <cmath>
#include "NetPBM.h"

using namespace std;

class PBM : public NetPBM {
private:
    ifstream *file;
    uint16_t type = -1;
    uint16_t width = -1;
    uint16_t height = -1;
    uint16_t depth = -1;
    unsigned char **array;

    void read_header() {
        unsigned char buf;
        *this->file >> buf;
        *this->file >> this->width;
        *this->file >> buf;
        *this->file >> this->height;
        *this->file >> buf;
```

```
        *this->file >> this->depth;
        if (this->depth != 255) {
                throw logic_error("Not supported non 255 colors
count");
        }
    }

    void read_data() {
        for (int i = 0; i < this->height; i++) {
            for (int j = 0; j < this->width; j++) {
                unsigned char tmp;
                *this->file >> tmp;
                this->array[i][j] = tmp;
            }
        }
    }

    void write_to_file(ofstream *outfile) override {
         *outfile << "P5" << (unsigned char) (10) << width << " "
<< height << (unsigned char) (10) << depth
                 << (unsigned char) (10);
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                *outfile << array[i][j];
            }
        }
    }

public:
    void inversion() override {
```

```
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            array[i][j] = 255 - array[i][j];
        }
    }
}

void horizontal_mirror() override {
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < (width - 1) / 2; j++) {
            int R, G, B;
            int color = array[i][j];
            array[i][j] = array[i][width - j - 1];
            array[i][width - j - 1] = color;
        }
    }
}

void vertical_mirror() override {
    for (int i = 0; i < (height - 1) / 2; i++) {
        for (int j = 0; j < width; j++) {
            int color = array[i][j];
            array[i][j] = array[height - i - 1][j];
            array[height - i - 1][j] = color;
        }
    }
}

void rotate_90(bool isclock) override {
```

```
    int tmp = this->width;
    this->width = this->height;
    this->height = tmp;
    unsigned char **arr;
    arr = new unsigned char *[height];
    for (int i = 0; i < this->height; i++) {
        arr[i] = new unsigned char[width];
        for (int j = 0; j < width; j++) {
            if (!isclock)
                arr[i][j] = this->array[j][height - i - 1];
            else
                arr[i][j] = this->array[width - 1 - j][i];
        }
    }
    for (int i = 0; i < width; i++) {
        delete[] array[i];
    }
    delete[] array;

    array = arr;
};

explicit
PBM(ifstream
    *file) {
    this->file = file;
    this->read_header();
    file->ignore(1);
    this->array = new unsigned char *[height];
```

```
        for (int i = 0; i < this->height; i++) {
            this->array[i] = new unsigned char[width];
        }
        read_data();
    }

    ~PBM() override {
        for (int i = 0; i < height; i++) {
            delete[] array[i];
        }
        delete[] array;
    }
};

#endif //LAB4_PBM_H
```