**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.**

Выполнил: студент гр. M32342

Ларионов А.Н.

Преподаватель: Скаков П.С.

Санкт-Петербург

2020

**Цель работы**: изучить алгоритмы и реализовать программу выполняющую 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**

Для того, чтобы выполнять преобразования максимально просто, я создаю массив данных размеров width \* height**,** который содержит информацию о каждом пикселе.

Чтобы вывести картинку без преобразований, достаточно пройтись по ячейкам матрицы в следующем порядке:

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

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

**Экспериментальная часть**

Используемый язык программирования: C++11

На входе мы можем получить один из двух форматов, поэтому память я выделяю динамически. Для файла формата PGM я выделяю width \* height байт памяти, а для файла формата PPM я выделяю 3 \* width \* height байт памяти. Независимо от формата файла, при завершении программы память будет корректно освобождена.

0) Для инверсии порядок обхода ячеек не меняется, а к каждой ячейке применяется операция логического отрицания.

1) Для зеркального отражения по горизонтали меняется порядок обхода во внутреннем цикле на обратный.

2) Для зеркального отражения по горизонтали меняется порядок обхода во внешнем цикле на обратный.

3) Для поворота на 90° по часовой стрелке измеренияwidth и height меняются местами, а также меняется порядок обхода во внутреннем цикле на обратный.

4) Для поворота на 90° против часовой стрелке измерения width и height меняются местами, а также меняется порядок обхода во внешнем цикле на обратный.

**Выводы**

В ходе выполнения работы была получена программа, которая выполняет простые преобразования серых и цветных изображений в формате PNM.

**Листинг**

hw1.cpp

#include <cstdio>

#include <iostream>

typedef unsigned char uchar;

struct RGB {

uchar r, g, b;

};

RGB \*my\_data1;

FILE \*my\_file;

uchar \*my\_data;

int clearData(int cur\_type) {

delete [] my\_data;

delete [] my\_data1;

fclose(my\_file);

return cur\_type;

}

void write(int index, int cur\_type) {

if (cur\_type == 5) {

fwrite(&my\_data[index], sizeof(uchar), 1, my\_file);

} else {

fwrite(&my\_data1[index], sizeof(RGB), 1, my\_file);

}

}

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

int type;

int max\_color;

int width, height;

int count = 0;

if (argc != 4) {

std::cerr << "Wrong count of arguments\n";

return 1;

}

char operation = argv[3][0];

if (operation < '0' || operation > '4') {

std::cerr << "Operation doesn't exist\n";

return 1;

}

my\_file = fopen(argv[1], "rb");

if (my\_file == nullptr) {

std::cerr << "Can't open input file\n";

return 1;

}

fscanf(my\_file, "P%d%d%d%d\n", &type, &width, &height, &max\_color);

my\_data = new uchar[height \* width + 1];

my\_data1= new RGB[height \* width + 1];

if (width <= 0 || height <= 0 || type < 5 || type > 6) {

std::cerr << "Wrong format of input file\n";

return clearData(1);

}

if (type == 5) {

count = fread(my\_data, sizeof(uchar), width \* height + 1, my\_file);

} else {

count = fread(my\_data1, sizeof(RGB), width \* height + 1, my\_file);

}

if (count != height \* width) {

std::cerr << "Wrong amount of data\n";

return clearData(1);

}

my\_file = fopen(argv[2], "wb");

if (my\_file == nullptr) {

std::cerr << "Can't open output file\n";

return clearData(1);

}

if (operation == '0') {

fprintf(my\_file, "P%d\n%d %d\n%d\n", type, width, height, max\_color);

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

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

int index = i \* width + j;

if (type == 5) {

my\_data[index] = ~my\_data[index];

fwrite(&my\_data[index], sizeof(uchar), 1, my\_file);

} else {

my\_data1[index].r = ~my\_data1[index].r;

my\_data1[index].g = ~my\_data1[index].g;

my\_data1[index].b = ~my\_data1[index].b;

fwrite(&my\_data1[index], sizeof(RGB), 1, my\_file);

}

}

}

}

if (operation == '1') {

fprintf(my\_file, "P%d\n%d %d\n%d\n", type, width, height, max\_color);

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

for (int j = width - 1; j >= 0; j--) {

write(i \* width + j, type);

}

}

}

if (operation == '2') {

fprintf(my\_file, "P%d\n%d %d\n%d\n", type, width, height, max\_color);

for (int i = height - 1; i >= 0; i--) {

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

write(i \* width + j, type);

}

}

}

if (operation == '3') {

fprintf(my\_file, "P%d\n%d %d\n%d\n", type, height, width, max\_color);

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

for (int j = height - 1; j >= 0; j--) {

write(j \* width + i, type);

}

}

}

if (operation == '4') {

fprintf(my\_file, "P%d\n%d %d\n%d\n", type, height, width, max\_color);

for (int i = width - 1; i >= 0; i--) {

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

write(j \* width + i, type);

}

}

}

return clearData(0);

}