**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 № 1

По дисциплине «Компьютерная геометрия и графика»

Изучение простых преобразований изображений

Выполнила: студент гр. M3101

Тарасов Денис Евгеньевич

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.

**Теоретическая часть**

Описание хранения данных изображения в формате PNM

Заголовок файла:

Первые два байта в заголовке показывают версию:

* P5- изображение состоит из градаций серого
* P6- изображение в формате RGB

После этого идет перевод строки

После этого идет ширина и высота изображения в десятичном формате. Перевод строки.

После этого идет максимально возможное значение цвета. В данной лабораторной работе нам гарантировано, что это значение будет равно 255. Следовательно каждое значение яркости мы можем хранить в 1 байте. Перевод строки.

После этого идет яркостная информация:

* Для P5: представляет яркость пикселя
* Для P6: Все пиксели цветные и имеют 3 канала. Следовательно для пикселя подряд записывается 3 байта яркости для всех каналов

Преобразования

1. Инверсия

Представляет собой изменение значения яркости каждого пикселя на значение, получаемое путем вычитания из глубины цвета данного пикселя (для P6 для каждого канала).

1. Зеркальное отражение по горизонтали

Представляет собой перестановку пикселей в каждой строке относительно центра.

1. Зеркальное отражение по вертикали

То же самое, только вместо строк столбцы.

1. Поворот на 90 градусов по часовой стрелке.

Представляет собой создание нового изображения, в котором столбцы являются старыми строчками, причем запись из старого изображения в новое начинается с крайнего правого столбца, идя сверху вниз.

1. Поворот на 90 градусов против часовой стрелки

Почти то же самое, что поворот на 90 градусов по часовой, но обход начинается с левого столбца, причем идем снизу вверх

**Экспериментальная часть**

Язык программирования: С++ 17, Полное решение

Для начала создал три класса: PNM- виртуальный класс, от которого наследуются классы PGM и PPM. Класс представляет из себя двумерный массив, а также содержит дополнительную информацию: Высоту и ширину массива, версию изображения.

Этапы исполнения программы:

1. Чтение картинки
2. Выполнение одного из алгоритмов
3. Запись Картинки в файл

**Выводы**

Выполнение данной работы помогло изучить формат PNM, а также позволило применить знания, полученные на других дисциплинах.

**Листинг**

Lab1.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <fstream>

#include "PGM.h";

#include "PPM.h";

using namespace std;

struct pixel

{

unsigned char red;

unsigned char green;

unsigned char blue;

};

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

if (argc != 4) {

cout << "command is invalid**\n**";

exit(0);

}

string finName = string(argv[1]);

string foutName = string(argv[2]);

string command = string(argv[3]);

ifstream fin;

try

{

fin.open(finName,ios::binary);

}

catch (const std::exception& ex)

{

cout << ex.what();

}

char vers[2];

try

{

fin >> vers[0] >> vers[1];

}

catch (const std::exception& ex)

{

cout << ex.what();

}

if (!(vers[0] == 'P' && (vers[1] == '5' || vers[1] == '6'))) {

cout << "Version is invalid**\n**";

exit(0);

}

PNM\* picture;

try

{

if (vers[1] == '5') {

picture = new PGM(finName);

}

else {

picture = new PPM(finName);

}

}

catch (const std::exception& ex)

{

cout << ex.what();

exit(0);

}

try

{

for (int i = 0; i < command.length(); i++) {

if (command[i] == '0') {

picture->ColorInversion();

}

else if (command[i] == '1') {

picture->HorizontalReflect();

}

else if (command[i] == '2') {

picture->VerticalReflect();

}

else if (command[i] == '3') {

picture->RotateRight();

}

else if (command[i] == '4') {

picture->RotateLeft();

}

else {

throw exception("Invalid command");

}

}

}

catch (const std::exception& ex)

{

cout << ex.what();

}

try

{

picture->write(foutName);

}

catch (const std::exception& ex)

{

ex.what();

}

fin.close();

return 0;

}

PNM.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class PNM {

public:

virtual void write(string outFileName) = 0;

virtual void ColorInversion() = 0;

virtual void HorizontalReflect() = 0;

virtual void VerticalReflect() = 0;

virtual void RotateRight() = 0;

virtual void RotateLeft() = 0;

};

PGM.h

#pragma once

#include "PNM.h"

#include <vector>

#include <iostream>

#include <fstream>

using namespace std;

class PGM:public PNM{

private:

char vers[2];

int width;

int height;

int colorDepth;

vector<vector<unsigned char>> pict;

public:

PGM(string inpFileName) {

ifstream inpFile(inpFileName,ios::binary);

inpFile >> vers[0] >> vers[1]>>width>>height>>colorDepth;

char read\_char[1];

inpFile.read(read\_char, 1);

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

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

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

if (inpFile.eof()) {

throw exception("Too few pixels");

}

inpFile.read(read\_char, 1);

pict[i][j] = read\_char[0];

}

}

inpFile.close();

}

void write(string outFileName) {

ofstream outFile(outFileName, ios::binary);

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

throw exception("Cant open output file");

}

outFile << vers[0] << vers[1] << '**\n**' << width << ' ' << height << '**\n**' << colorDepth << '**\n**';

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

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

outFile << pict[i][j];

}

}

outFile.close();

}

void ColorInversion() {

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

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

pict[i][j] = colorDepth - pict[i][j];

}

}

}

void HorizontalReflect() {

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

for (int j = 0; j < width / 2; j++) {

swap(pict[i][j], pict[i][width - j - 1]);

}

}

}

void VerticalReflect() {

for (int i = 0; i < height / 2; i++) {

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

swap(pict[i][j], pict[height - i - 1][j]);

}

}

}

void RotateRight() {

vector<vector<unsigned char>> new\_pict(width, vector<unsigned char>(height));

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

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

new\_pict[j][height - i - 1] = pict[i][j];

}

}

swap(width, height);

pict = new\_pict;

}

void RotateLeft() {

vector<vector<unsigned char>> new\_pict(width, vector<unsigned char>(height));

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

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

new\_pict[width-j-1][i] = pict[i][j];

}

}

swap(width, height);

pict = new\_pict;

}

};

PPM.h

#pragma once

#include <vector>

#include <iostream>

#include <fstream>

using namespace std;

struct Pixel

{

unsigned char r, g, b;

};

class PPM:public PNM {

private:

char vers[2];

int width;

int height;

int colorDepth;

vector<vector<Pixel>> pict;

public:

PPM(string inpFileName) {

ifstream inpFile(inpFileName, ios::binary);

inpFile >> vers[0] >> vers[1] >> width >> height >> colorDepth;

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

char pixel\_read[3];

char read\_char[1];

inpFile.read(read\_char, 1);

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

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

if (inpFile.eof()) {

inpFile.close();

cout << i << " " << j << "**\n**";

throw exception("Too few pixels");

}

inpFile.read(pixel\_read, 3);

pict[i][j].r = pixel\_read[0];

pict[i][j].g = pixel\_read[1];

pict[i][j].b = pixel\_read[2];

}

}

inpFile.close();

}

void write(string outFileName) {

ofstream outFile(outFileName, ios::binary);

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

throw exception("Cant open output file");

}

outFile << vers[0] << vers[1] << '**\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<<pict[i][j].g<<pict[i][j].b;

}

}

outFile.close();

}

void ColorInversion() {

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

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

pict[i][j].r = colorDepth - pict[i][j].r;

pict[i][j].g = colorDepth - pict[i][j].g;

pict[i][j].b = colorDepth - pict[i][j].b;

}

}

}

void HorizontalReflect() {

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

for (int j = 0; j < width / 2; j++) {

swap(pict[i][j], pict[i][width - j - 1]);

}

}

}

void VerticalReflect() {

for (int i = 0; i < height / 2; i++) {

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

swap(pict[i][j], pict[height - i - 1][j]);

}

}

}

void RotateRight() {

vector<vector<Pixel>> new\_pict(width, vector<Pixel>(height));

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

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

new\_pict[j][height - i - 1] = pict[i][j];

}

}

swap(width, height);

pict = new\_pict;

}

void RotateLeft() {

vector<vector<Pixel>> new\_pict(width, vector<Pixel>(height));

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

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

new\_pict[width - j - 1][i] = pict[i][j];

}

}

swap(width, height);

pict = new\_pict;

}

};