МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа № 1

По дисциплине «Компьютерная графика и геометрия»

Изучение простых преобразований изображений

Выполнил студент группы М3101

Кузьмук Павел Юрьевич

Проверил:

Скаков Павел Сергеевич

САНКТ-ПЕТЕРБУРГ

2020

ЦЕЛЬ РАБОТЫ

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

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Описание хранения данных изображения в формате PNM

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 лабораторной работе мы работаем с максимально возможным значением 255 для каждого пикселя, поэтому данные каждого пикселя хранятся в виде одного байта. Перевод строки.

Данные изображения:

Для форматов P5 и P6 после заголовка следуют данные 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

Всего в лабораторной работе выполнено 5 преобразований изображений.

1. *Инверсия*

Инверсия представляет собой изменение значения яркости каждого пикселя на противоположное или, другими словами, на значение, получаемое путем вычитания из максимально возможной яркости пикселя яркости самого пикселя (или яркости канала в случае цветного изображения).

2. *Зеркальное отражение по горизонтали*

Зеркальное отражение по горизонтали представляет собой перестановку пикселей в каждой строке относительно середины строки.

3. *Зеркальное отражение по вертикали*

Зеркальное отражение по вертикали представляет собой перестановку пикселей в каждом столбце относительно середины столбца.

4. *Поворот на 90 градусов по часовой стрелке*

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

5. *Поворот на 90 градусов против часовой стрелки*

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

ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

Для хранения изображения используем структуру, которая хранит ширину, высоту и глубину цвета

Преобразования:

1. *Инверсия*

Инверсия цветов изображения работает за один проход по массиву значений пикселей, изменяя значения яркости на противоположные. Не требует выделения дополнительной памяти.

2. *Зеркальное отражение по горизонтали*

Зеркальное отображение работает как классическое отражение матрицы

3. *Зеркальное отражение по вертикали*

Аналогично п.2

4. *Поворот на 90 градусов по часовой стрелке*

Любой поворот меняет местами высоту и ширину изображения, так что в общем случае не избежать перевыделения памяти, в остальном работает как поворот матрицы

5. *Поворот на 90 градусов против часовой стрелки*

Аналогично п.4

ВЫВОД

Выполнение данной лабораторной работы позволило изучить формат PNM, а также позволило освоить и применить простые алгоритмы для обработки изображений: инверсия цветов, зеркальное отражение изображения по вертикали и горизонтали, поворот изображения на 90 градусов.

ЛИСТИНГ КОДА

main.cpp:

```
#include <iostream>

#include <string>
#include "pgm_image.h"
#include "ppm_image.h"
#include "pnm_image.h"

using namespace std;

int main(int argc, char *argv[]) {
if(argc != 4) {
cout << "command line arguments are invalid" << endl;
exit(0);
}
string fin = string(argv[1]);
string fout = string(argv[2]);
string operations = string(argv[3]);

PNM_Image* image;

ifstream inFile(fin, ios::binary);
if(!inFile.is_open()) {
cout << "file isn't opened. Check permissions / file existance" <<
endl;
exit(0);
}
unsigned char format[2];
for(int i = 0; i < 2; i ++)
if(!inFile.eof())
inFile >> format[i];
else {
inFile.close();
cout << "file has no two chars at start" << endl;
exit(0);
}
inFile.close();
if(!(format[0] == 'P' && (format[1] == '5' || format[1] == '6'))) {
cout << "file is not P5/P6 file" << endl;
exit(0);
}

try {
if(format[1] == '5')
image = new PGM_Image(fin);
```

```
else
image = new PPM_Image(fin);
}
catch (const exception& e) {
cout << e.what() << endl;
exit(0);
}
try {
for(auto i : operations)
if(i == '0')
image -> inverse_colors();
else if(i == '1')
image -> reflect_horizontal();
else if(i == '2')
image -> reflect_verical();
else if(i == '3')
image -> rotate_right();
else if(i == '4')
image -> rotate_left();
else
throw runtime_error("operation not supported");
}
catch (const exception& e) {
cout << e.what() << endl;
exit(0);
}
try { image -> drop(fout); }
catch (const exception& e) {
cout << e.what() << endl;
exit(0);
}
}
```

pnm_image.h:

```
#pragma once


#include <string>
#include <fstream>
using namespace std;

class PNM_Image{
public:
virtual void inverse_colors() = 0;
virtual void rotate_right() = 0;
virtual void rotate_left() = 0;
```

```
virtual void reflect_verical() = 0;
virtual void reflect_horizontal() = 0;
virtual void drop(string) = 0;
};
```

pgm_image.h:

```
#pragma once

#include <vector>
#include <algorithm>
#include <stdexcept>
#include <fstream>
#include "pnm_image.h"

using namespace std;

class PGM_Image: public PNM_Image{
private:
int width, height, color_depth;
vector<vector<unsigned char>> image;
public:
PGM_Image(string);
void inverse_colors();
void rotate_right();
void rotate_left();
void reflect_verical();
void reflect_horizontal();
void drop(string);
};
```

ppm_image.h:

```
#pragma once

#include <string>
#include <vector>
#include <algorithm>
#include <stdexcept>
#include <tuple>
#include <fstream>
#include "pnm_image.h"
typedef tuple<unsigned char, unsigned char, unsigned char> color;

using namespace std;

class PPM_Image: public PNM_Image{
private:
```

```
int width, height, color_depth;
vector<vector<color>> image;
public:
PPM_Image(string);
void inverse_colors();
void rotate_right();
void rotate_left();
void reflect_verical();
void reflect_horizontal();
void drop(string);
};
```

pgm_image.cpp:

```
#include "pgm_image.h"


PGM_Image::PGM_Image(string filename) {
ifstream fin(filename, ios::binary);
char cc;
fin >> cc >> cc;
fin >> width >> height >> color_depth;
image.assign(height, vector<unsigned char>(width));
fin.read(&cc, 1);
for(int i = 0; i < height; i ++)
for(int j = 0; j < width; j ++)
{
if(fin.eof()) {
fin.close();
throw runtime_error("some pixels not found");
}
fin.read(&cc, sizeof(unsigned char));
image[i][j] = cc;
}
fin.close();
}

void PGM_Image::drop(string filename) {
ofstream fout(filename, ios::binary);
if(!fout.is_open()) {
throw runtime_error("cannot open output file");
}
fout << "P5\n" << width << ' ' << height << '\n' << color_depth << '\
n';
for(int i = 0; i < height; i ++)
for(int j = 0; j < width; j ++)
fout << (unsigned char)(image[i][j]);
fout.flush();
```

```
fout.close();
}

void PGM_Image::inverse_colors() {
for(int i = 0; i < height; i ++)
for(int j = 0; j < width; j ++)
image[i][j] = color_depth - image[i][j];
}

void PGM_Image::rotate_right() {
vector<vector<unsigned char>> new_image(width, vector<unsigned
char>(height, 0));
for(int i = 0; i < height; i ++)
for(int j = 0; j < width; j ++)
new_image[j][height - i - 1] = image[i][j];
image = new_image;
swap(height, width);
}

void PGM_Image::rotate_left() {
vector<vector<unsigned char>> new_image(width, vector<unsigned
char>(height, 0));
for(int i = 0; i < height; i ++)
for(int j = 0; j < width; j ++)
new_image[width - j - 1][i] = image[i][j];
image = new_image;
swap(height, width);
}

void PGM_Image::reflect_horizontal() {
for(int i = 0; i < height; i ++)
reverse(image[i].begin(), image[i].end());
}

void PGM_Image::reflect_verical() {
for(int i = 0; i < width; i ++)
for(int j = 0; j < height / 2; j ++)
swap(image[j][i], image[height - j - 1][i]);
}
```

ppm_image.cpp:

```
#include "ppm_image.h"


PPM_Image::PPM_Image(string filename) {
ifstream fin(filename, ios::binary);
```

```
        char cc;
        fin >> cc >> cc;
        fin >> width >> height >> color_depth;
        fin.read(&cc, 1);
        image.assign(height, vector<color>(width));

        char col[3];
        for(int i = 0; i < height; i ++)
        for(int j = 0; j < width; j ++)
        {
        if(fin.eof()) {
        fin.close();
        throw runtime_error("some pixels not found");
        }
        fin.read(col, 3);
        image[i][j] = {col[0], col[1], col[2]};
        }
        fin.close();
        }

        void PPM_Image::drop(string filename) {
        ofstream fout(filename, ios::binary);
        if(!fout.is_open()) {
        throw runtime_error("cannot open output file");
        }
        fout << "P6\n" << width << ' ' << height << '\n' << color_depth << '\
n';
        for(int i = 0; i < height; i ++)
        for(int j = 0; j < width; j ++)
        fout  <<  char(get<0>(image[i][j]))  <<  char(get<1>(image[i][j]))  <<
char(get<2>(image[i][j]));
        fout.flush();
        fout.close();
        }

        void PPM_Image::inverse_colors() {
        for(int i = 0; i < height; i ++)
        for(int j = 0; j < width; j ++)
        image[i][j]  =  {color_depth  -  get<0>(image[i][j]),  color_depth  -
get<1>(image[i][j]), color_depth - get<2>(image[i][j])};
        }

        void PPM_Image::rotate_right() {
        vector<vector<color>> new_image(width, vector<color>(height));
        for(int i = 0; i < height; i ++)
        for(int j = 0; j < width; j ++)
        new_image[j][height - i - 1] = image[i][j];
```

```
image = new_image;
swap(height, width);
}

void PPM_Image::rotate_left() {
vector<vector<color>> new_image(width, vector<color>(height));
for(int i = 0; i < height; i ++)
for(int j = 0; j < width; j ++)
new_image[width - j - 1][i] = image[i][j];
image = new_image;
swap(height, width);
}

void PPM_Image::reflect_horizontal() {
for(int i = 0; i < height; i ++)
reverse(image[i].begin(), image[i].end());
}

void PPM_Image::reflect_verical() {
for(int i = 0; i < width; i ++)
for(int j = 0; j < height / 2; j ++)
swap(image[j][i], image[height - j - 1][i]);
}
```