МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение высшего образования

Санкт-Петербургский национальный 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

Выполнил студент группы №M3101:

Пантелеев Ярослав Кириллович

Преподаватель:

Скаков Павел Сергеевич

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.

Теоретическая часть:

Заголовок файла

В начале файла содержится следующая структура данных, представляющая 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)

Далее идут 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 него записываются подряд 3 байта яркости цветов в формате RGB.

Преобразования

Всего в лабораторной работе выполнено 5 преобразований изображений.

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.
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

Экспериментальная часть

Язык выполнения работы: C++17, компилятор Microsoft Visual C++.

Этапы работы программы:

1) Чтение файла картинки (хедера и информации о цвете каждого пикселя) с обработкой ошибок чтения
2) Выполнение заданного аргументом командной строки преобразования изображения
3) Запись полученного изображения в выходной файл

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 исходного кода:

main.cpp

```
#include <iostream>
#include "Image.h"
using std::cout;
using std::cin;

int main(int argc, char** argv) {
    Image img1;
    if (argc != 4) {
        cout << "Unsupported count of arguments\n";
               exit(1);
           }
           img1.set_vars(argv[1], argv[2]);
           img1.read();
           if (argv[3][0] == '0')
               img1.inversion();
           else if (argv[3][0] == '1')
               img1.horizontal_display();
           else if (argv[3][0] == '2')
               img1.vertical_display();
           else if (argv[3][0] == '3')
               img1.rotate(0);
           else if (argv[3][0] == '4')
               img1.rotate(1);
           else {
               cout << "Unsupported type of operation\n";
               exit(1);
           }
           img1.write();
           return 0;
       }
```

Image.h

```
#pragma once
#include <iostream>
#include <vector>
#include <string>

using std::cout;
using std::cin;
using byte = unsigned char;
using big = unsigned long long int;

       struct Pixel {
           byte r;
           byte g;
           byte b;
       };

       int to_int(std::vector <char>& v);

       class Image {
       public:
           void set_vars(char* in, char* out);
           void read(char* in);
           void read();
           void write(char* out);
           void write();
           void inversion();
```

```
27.           void vertical_display();
28.           void horizontal_display();
29.           void rotate(bool type);
30.           ~Image();
31.       protected:
32.           void _read(char* in);
33.           void _write(char* out);
34.       private:
35.           char* in;
36.           char* out;
37.           FILE* file_in;
38.           FILE* file_out;
39.           char mode[3]; // P5 or P6
40.           size_t w, h, d; // width, heigth and color depth
41.           std::vector <std::vector <byte>> bnw; // for P5
42.           std::vector <std::vector <Pixel>> rgb; // for P6
43.       };
```

Image.cpp

```
1. #define _CRT_SECURE_NO_WARNINGS
2. #include "Image.h"
3.
4. int to_int(std::vector <char>& v) {
5.     int result = 0;
6.     size_t size = v.size();
7.     for (size_t i = 0; i < size; i++) {
8.         result += (int)std::pow(10.0, (double)i) * (int)(v[size - i - 1] - '0');
9.     }
10.           return result;
11.       };
12.
13.       void Image::set_vars(char* in, char* out) {
14.           this->in = in;
15.           this->out = out;
16.       };
17.
18.       void Image::read(char* in) {
19.           return _read(in);
20.       };
21.
22.       void Image::read() {
23.           if (!this->in) {
24.               cout << "Input file is not defined\n";
25.               exit(1);
26.           }
27.           else return _read(this->in);
28.       };
29.
30.       void Image::_read(char* in) {
31.           this->file_in = fopen(in, "rb");
32.           if (!this->file_in) {
33.               cout << "Unable to open input file\n";
34.               exit(1);
35.           }
36.           fseek(this->file_in, 0, SEEK_END); // end of file
37.           big current_size = ftell(this->file_in); // size of file
38.           fseek(this->file_in, 0, SEEK_SET); // start of file
39.
40.           // Reading Header
41.           std::vector <char> _w, _h, _d;
42.           fread(this->mode, 1, 2, this->file_in); // reading format (P5 or P6)
43.           fseek(this->file_in, 1, SEEK_CUR); // passing '\n'
44.           if (this->mode[0] != 'P' || (this->mode[1] != '5' && this->mode[1] != '6')) {
45.               cout << "Unsupported format\n"; // if format is not P5 or P6
```

```
46.             fclose(this->file_in);
47.             exit(1);
48.         }
49.
50.         char buffer = fgetc(this->file_in);
51.         while (buffer != ' ') { // reading W
52.             _w.push_back(buffer);
53.             buffer = fgetc(this->file_in);
54.         }
55.         buffer = fgetc(this->file_in);
56.         while (buffer != '\n') { // reading H
57.             _h.push_back(buffer);
58.             buffer = fgetc(this->file_in);
59.         }
60.         buffer = fgetc(this->file_in);
61.         while (buffer != '\n') { // reading D
62.             _d.push_back(buffer);
63.             buffer = fgetc(this->file_in);
64.         }
65.         this->w = to_int(_w);
66.         this->h = to_int(_h);
67.         this->d = to_int(_d);
68.
69.         big required_size;
70.         if (this->mode[0] == 'P' && this->mode[1] == '5')
71.             required_size = (big)(this->w * this->h);
72.         else
73.             required_size = (big)(3 * this->w * this->h);
74.
75.         required_size += ftell(this->file_in);
76.         if (current_size < required_size) {
77.             cout << "File size is too small\n";
78.             exit(1);
79.         }
80.         if (this->mode[1] == '5') {
81.             this->bnw.assign(h, std::vector<byte>(w)); // setting size of vector
82.             for (size_t i = 0; i < this->h; i++) {
83.                 for (size_t j = 0; j < this->w; j++) {
84.                     this->bnw[i][j] = fgetc(this->file_in);
85.                 }
86.             }
87.         }
88.         else {
89.             this->rgb.assign(h, std::vector<Pixel>(w)); // setting size of vector
90.             for (size_t i = 0; i < this->h; i++) {
91.                 for (size_t j = 0; j < this->w; j++) {
92.                     this->rgb[i][j].r = fgetc(this->file_in);
93.                     this->rgb[i][j].g = fgetc(this->file_in);
94.                     this->rgb[i][j].b = fgetc(this->file_in);
95.                 }
96.             }
97.         }
98.         fclose(this->file_in);
99.     };
100.
101.    void Image::write(char* out) {
102.        return _write(out);
103.    };
104.
105.    void Image::write() {
106.        if (!this->out) {
107.            cout << "Output file is not defined\n";
108.            exit(1);
109.        }
110.        else return _write(this->out);
111.    };
112.
113.    void Image::_write(char* out) {
```

```
114.            this->file_out = fopen(out, "wb");
115.            if (!this->file_out) {
116.                cout << "Unable to open output file\n";
117.                exit(1);
118.            }
119.            fprintf(this->file_out, "%c%c\n%i %i\n%i\n", mode[0], mode[1], w, h, d); // writing
header
120.            // writing pixels
121.            if (this->mode[1] == '6') {
122.                for (size_t i = 0; i < this->h; i++) {
123.                    for (size_t j = 0; j < this->w; j++) {
124.                        fprintf(this->file_out, "%c%c%c", this->rgb[i][j].r, this->rgb[i][j].g,
this->rgb[i][j].b);
125.                    }
126.                }
127.            }
128.            else {
129.                for (size_t i = 0; i < this->h; i++) {
130.                    for (size_t j = 0; j < this->w; j++) {
131.                        fprintf(this->file_out, "%c", this->bnw[i][j]);
132.                    }
133.                }
134.            }
135.            fclose(this->file_out);
136.        };
137.
138.        Image::~Image() {
139.            if (this->file_in) fclose(this->file_in);
140.            if (this->file_out) fclose(this->file_out);
141.        };
142.
143.        void Image::inversion() {
144.            if (this->mode[1] == '5')
145.                for (size_t i = 0; i < this->h; ++i)
146.                    for (size_t j = 0; j < this->w; ++j)
147.                        this->bnw[i][j] = byte(this->d - this->bnw[i][j]);
148.            else
149.                for (size_t i = 0; i < this->h; ++i)
150.                    for (size_t j = 0; j < this->w; ++j) {
151.                        this->rgb[i][j].r = byte(this->d - this->rgb[i][j].r);
152.                        this->rgb[i][j].g = byte(this->d - this->rgb[i][j].g);
153.                        this->rgb[i][j].b = byte(this->d - this->rgb[i][j].b);
154.                    }
155.        };
156.
157.        void Image::horizontal_display() {
158.            if (this->mode[1] == '5')
159.                for (size_t i = 0; i < this->h; i++)
160.                    for (size_t j = 0; j < this->w / 2; j++)
161.                        std::swap(this->bnw[i][j], this->bnw[i][w - j - 1]);
162.            else
163.                for (size_t i = 0; i < this->h; i++)
164.                    for (size_t j = 0; j < this->w / 2; j++) {
165.                        std::swap(this->rgb[i][j].r, this->rgb[i][w - j - 1].r);
166.                        std::swap(this->rgb[i][j].g, this->rgb[i][w - j - 1].g);
167.                        std::swap(this->rgb[i][j].b, this->rgb[i][w - j - 1].b);
168.                    }
169.        };
170.
171.        void Image::vertical_display() {
172.            if (this->mode[1] == '5')
173.                for (size_t i = 0; i < this->h / 2; i++)
174.                    for (size_t j = 0; j < this->w; j++)
175.                        std::swap(this->bnw[i][j], this->bnw[h - i - 1][j]);
176.            else
177.                for (size_t i = 0; i < this->h / 2; i++)
178.                    for (size_t j = 0; j < this->w; j++) {
179.                        std::swap(this->rgb[i][j].r, this->rgb[h - i - 1][j].r);
```

```
                    std::swap(this->rgb[i][j].g, this->rgb[h - i - 1][j].g);
                    std::swap(this->rgb[i][j].b, this->rgb[h - i - 1][j].b);
                }
    };

    void Image::rotate(bool type) {
        // 0 - clock wise
        // 1 - counter clock wise
        if (this->mode[1] == '5') {
            std::vector <std::vector <byte>> new_bnw;
            int new_h = this->w;
            int new_w = this->h;
            new_bnw.assign(this->w, std::vector<byte>(this->h));
            if (!type)
                for (size_t i = 0; i < this->w; i++)
                    for (size_t j = 0; j < this->h; j++)
                        new_bnw[i][j] = this->bnw[new_w - j - 1][i];
            else
                for (size_t i = 0; i < this->w; i++)
                    for (size_t j = 0; j < this->h; j++)
                        new_bnw[i][j] = this->bnw[j][new_h - i - 1];
            this->w = new_w;
            this->h = new_h;
            this->bnw = new_bnw;
        }
        else {
            std::vector <std::vector <Pixel>> new_rgb;
            int new_h = this->w;
            int new_w = this->h;
            new_rgb.assign(this->w, std::vector<Pixel>(this->h));
            if (!type)
                for (size_t i = 0; i < this->w; i++)
                    for (size_t j = 0; j < this->h; j++) {
                        new_rgb[i][j].r = this->rgb[new_w - j - 1][i].r;
                        new_rgb[i][j].g = this->rgb[new_w - j - 1][i].g;
                        new_rgb[i][j].b = this->rgb[new_w - j - 1][i].b;
                    }
            else
                for (size_t i = 0; i < this->w; i++)
                    for (size_t j = 0; j < this->h; j++) {
                        new_rgb[i][j].r = this->rgb[j][new_h - i - 1].r;
                        new_rgb[i][j].g = this->rgb[j][new_h - i - 1].g;
                        new_rgb[i][j].b = this->rgb[j][new_h - i - 1].b;
                    }
            this->w = new_w;
            this->h = new_h;
            this->rgb = new_rgb;
        }
    };
```