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 геометрия и графика»

Изучение простых преобразований изображений

Выполнил студент группы №M3101
Семенов Георгий Витальевич

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:

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

PNM-файл представляет изображение и последовательно содержит следующие параметры:

1. Название формата (P5 или P6)
2. Размер в пикселях (два целых числа)
3. Максимальную глубину цвета (в работе рассматриваем значение 255)
4. Последовательности байт, передающие цвета (в P5 — черно-белый байт, в P6 — 3 RGB байта)

Требуемые операции заключаются в следующем:

1. Инверсия. Заменить параметры RGB цвета каждого пикселя на противоположные (сумма исходного и противоположного цвета даёт глубину цвета)
2. Горизонтальное отражение. Необходимо отразить, т.е. поменять местами части изображения, разделённые центральной вертикалью.
3. Вертикальное отражение. Необходимо отразить, т.е. поменять местами части изображения, разделённые центральным горизонтом.

4. Поворот по часовой стрелке. Поменять местами ширину и высоту изображения, последовательно установить изоморфизм между старыми и новыми пикселями, меняя направляющие координатных осей так, чтобы изображение “легло на правый бок”.
5. Поворот против часовой стрелки. Аналогично, но “на левый бок”.

Экспериментальная часть

Язык программирования: C++.

Изображение описано в классе pnmImage, хранящее в двумерном массиве (vector) указатели на экземпляры класса pnmColor, ширину и высоту изображения.

В pnmImage реализованы конструкторы, считывающие изображение из файла, метод, записывающий изображение в файл, и процедуры соответствующих преобразований. Для преобразований не используется дополнительная память.

Вывод

Алгоритмы элементарных преобразований по времени зависят линейно от размера изображения и не требуют дополнительной памяти.

Листинг

https://github.com/MrGeorgeous/ComputerGeometryAndGraphics/blob/master/README.md

Файл: Main.cpp

```
#define _CRT_SECURE_NO_WARNINGS

#include<stdio.h>
#include <iostream>
#include <fstream>
#include <istream>
#include <iomanip>

#include<string>
#include<vector>


using namespace std;

class pnmColor {

public:

	unsigned char red = 0;
	unsigned char green = 0;
	unsigned char blue = 0;
```

```
	pnmColor() {

	}



	pnmColor(const pnmColor& c) {
		red = c.red;
		green = c.green;
		blue = c.blue;
	}

	pnmColor(unsigned char black) {
		red = green = blue = black;
	}

	pnmColor(unsigned char r, unsigned char g, unsigned char b) {
		red = r;
		green = g;
		blue = b;
	}

	~pnmColor() {

	}

	void inverseColor(unsigned char depth) {
		red = depth - red;
		green = depth - green;
		blue = depth - blue;
	}

};



void progresser(size_t & previous_step, size_t & percentage, size_t & step) {

	if (percentage >= 100) { return; }

	previous_step = (previous_step + 1) % step;
	if (previous_step == 0) {
		cerr << "\b\b\b\b\b\b\b";
		for (int mmm = 0; mmm < 3; mmm++) {
			if (mmm < percentage % 3) {
				cerr << ".";
			}
			else {
				cerr << " ";
			}
		}
		percentage++;
		cerr << " " << ((percentage < 10) ? " " : "") << percentage << "%";
	}

	if (percentage > 99) {
		cerr << "\n";
	}

}

typedef vector<vector<pnmColor *>> pnmMatrix;
typedef enum {P5, P6} pnmFormat;
typedef vector<char> chars;
```

```
class pnmImage {

public:

	size_t width = 0; // y
	size_t height = 0; // x
	size_t depth = 255; // d

	pnmFormat f = P6;
	pnmMatrix m;
	chars errorEncounter;

	pnmImage(size_t w, size_t h, pnmColor color = pnmColor(255)) {

		m.resize(h);
		for (int i = 0; i < h; i++) {
			m[i].resize(w);
			pnmColor* t = new pnmColor(color);
			for (int j = 0; j < w; j++) {
				m[i].push_back(t);
			}
		}

	}

	~pnmImage() {

		for (int i = 0; i < height; i++) {
			for (int j = 0; j < width; j++) {
				delete m[i][j];
			}
		}

	}

	pnmImage(string filename) {

		if (!errorEncounter.empty()) {
			return;
		}

		cerr << "READING FILE " << filename << "\n";

		FILE* file = fopen(filename.c_str(), "rb");
		if ((file != NULL)) {} else {
			cerr << "failed\n";
			errorEncounter.push_back(1);
			return;
		}

		char p1, p2 = ' ';
		size_t w = 0, h = 0, d = 0;

		fscanf(file, "%c%c\n%i %i\n%i\n", &p1, &p2, &w, &h, &d);

		width = w;
		height = h;
		depth = d;

		m = pnmMatrix(h, vector<pnmColor *>(w, nullptr));

		cerr << "consistency check\n";
		if (p1 != 'P') {
			cerr << "Format Error.";
			errorEncounter.push_back(1);
```

```
        return;
    }
    if ( !((p2 == '5') || (p2 == '6')) ) {
        cerr << "Only P5 and P6 are supported.";
        errorEncounter.push_back(1);
        return;
    }
    if (((w == 0) || (h == 0))) {
        cerr << "Empty image.";
        errorEncounter.push_back(1);
        return;
    }
    if (!(d == 255)) {
        cerr << "Depth is not 255.";
        errorEncounter.push_back(1);
        return;
    }
    cerr << "ok\n";


    size_t percentage = 0;
    size_t step = width * height / 100;
    size_t previous_step = 0;
    cerr << "processing...  0%";

    if (p2 == '5') {
        f = P5;

        unsigned char t;
        for (int j = 0; j < h; j++) {
            for (int i = 0; i < w; i++) {

                progresser(previous_step, percentage, step);

                fread(&t, sizeof(unsigned char), 1, file);
                m[j][i] = new pnmColor(t);
            }
        }

    }

    if (p2 == '6') {
        f = P6;

        unsigned char r,g,b;
        for (int j = 0; j < h; j++) {
            m[j].resize(w);
            for (int i = 0; i < w; i++) {

                progresser(previous_step, percentage, step);

                fread(&r, sizeof(unsigned char), 1, file);
                fread(&g, sizeof(unsigned char), 1, file);
                fread(&b, sizeof(unsigned char), 1, file);

                pnmColor* t = new pnmColor(r,g,b);

                m[j][i] = t;

            }
        }

    }

    fclose(file);
```

```
	}

	void print(string filename) {

		if (!errorEncounter.empty()) {
			return;
		}

		cerr << "WRITING FILE " << filename << "\n";

		FILE* file = fopen(filename.c_str(), "wb");
		if ((file != NULL)) {} else {
			cerr << "failed\n";
			errorEncounter.push_back(1);
			return;
		}

		switch (f) {
		case P5:
			fprintf(file, "P5\n");
			break;
		case P6:
			fprintf(file, "P6\n");
			break;
		}

		fprintf(file, "%i %i\n%i\n", width, height, depth);

		size_t percentage = 0;
		size_t step = width * height / 100;
		size_t previous_step = 0;
		cerr << "processing...  0%";

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

				progresser(previous_step, percentage, step);

				if (f == P5) {
					unsigned char t = (m[j][i]->red + m[j][i]->green + m[j][i]-
>blue) / 3;
					fwrite(&(t), sizeof(unsigned char), 1, file);
				}

				if (f == P6) {
					fwrite(&(m[j][i]->red), sizeof(unsigned char), 1, file);
					fwrite(&(m[j][i]->green), sizeof(unsigned char), 1, file);
					fwrite(&(m[j][i]->blue), sizeof(unsigned char), 1, file);
				}

			}

		}

		fclose(file);

	}

	void printToConsole() {
		switch (f) {
		case P5:
			cerr << "P5" << "\n";
			break;
		case P6:
```

```
                        cerr << "P6" << "\n";
                        break;
                }

                cerr << width << " " << height << "\n" << depth << "\n";

                for (int j = 0; j < height; j++) {
                        for (int i = 0; i < width; i++) {
                                cerr << (unsigned char)m[j][i]->red << (unsigned char)m[j][i]->green
<< (unsigned char)m[j][i]->blue;
                        }
                }
        }

        void inverseColor() {

                if (!errorEncounter.empty()) {
                        return;
                }

                size_t percentage = 0;
                size_t step = width * height / 100;
                size_t previous_step = 0;
                cerr << "INVERSING COLORS...  0%";

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

                                progresser(previous_step, percentage, step);

                                m[i][j]->inverseColor(depth);
                        }
                }

        }

        void reflectHorizontal() {

                if (!errorEncounter.empty()) {
                        return;
                }

                size_t percentage = 0;
                size_t step = width * height / 100 / 2;
                size_t previous_step = 0;
                cerr << "REFLECTING HORIZONTALLY...  0%";

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

                                progresser(previous_step, percentage, step);

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

        }

        void reflectVertical() {

                if (!errorEncounter.empty()) {
                        return;
                }
```

```
        size_t percentage = 0;
        size_t step = width * height / 100 /2;
        size_t previous_step = 0;
        cerr << "REFLECTING VERTICALLY...  0%";

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

                        progresser(previous_step, percentage, step);

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

}

void clockwise90() {

        if (!errorEncounter.empty()) {
                return;
        }

        pnmMatrix newImage = pnmMatrix(width, vector<pnmColor*>(height, nullptr));

        size_t percentage = 0;
        size_t step = width * height / 100 ;
        size_t previous_step = 0;
        cerr << "ROTATING -> 90...  0%";

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

                        progresser(previous_step, percentage, step);

                        newImage[j][height - i - 1] = m[i][j];

                }
        }

        m = newImage;
        swap(width, height);

}

void counterclockwise90() {

        if (!errorEncounter.empty()) {
                return;
        }

        pnmMatrix newImage = pnmMatrix(width, vector<pnmColor*>(height, nullptr));

        size_t percentage = 0;
        size_t step = width * height / 100;
        size_t previous_step = 0;
        cerr << "ROTATING <- 90...  0%";

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

                        progresser(previous_step, percentage, step);
                        newImage[width - j - 1][i] = m[i][j];

                }
```

```
                }

                m = newImage;
                swap(width, height);

        }


};

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

        string fn, in, out;
        int mode = -1;

        for (int i = 0; i < argc; i++) {
                if (i == 0) {
                        fn = argv[i];
                }
                if (i == 1) {
                        in = argv[i];
                }
                if (i == 2) {
                        out = argv[i];
                }
                if (i == 3) {
                        mode = atoi(argv[i]);
                }
        }

        pnmImage im(in);

        switch (mode) {
        case 0:
                im.inverseColor();
                break;
        case 1:
                im.reflectHorizontal();
                break;
        case 2:
                im.reflectVertical();
                break;
        case 3:
                im.clockwise90();
                break;
        case 4:
                im.counterclockwise90();
                break;
        default:
                //im.inverseColor();
                //im.reflectHorizontal();
                //im.reflectVertical();
                //im.clockwise90();
                //im.counterclockwise90();
                break;
        }

        im.print(out);

        cerr << "\nCLEANING MEMORY\n";

        if (im.errorEncounter.empty()) {
                return 0;
        } else {
                return 1;
```

```
        }
}
```