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

Компьютерная графика и геометрия

Отчет

по лабораторной работе №1

Изучение простых преобразований изображений

Выполнил: студент гр. М3101
Ершов Михаил Юрьевич
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

1. **Формат файлов PNM**

В начале файла находится заголовок, которые имеет следующую структуру:

a. Тип файла – “P5” или “P6”
b. Перевод строки в формате LF – ‘\n’
c. Ширина и высота изображения в десятичном виде через пробел
d. Ещё один перевод строки
e. Максимальное значение пикселя – в данной лабораторной равно 255
f. Данные изображения в двоичном виде – для PNM P5 значение каждого пикселя хранится 1 байтом(так как максимальное значения пикселя – 255), для P6 – 3 байтами(3 канала RGB)

2. **Виды преобразований**
   a. Инверсия – каждому пикселю присваивается значение равное максимальному значению пикселя минус текущее значение пикселя, в данной лабораторной работе – x = 255 - x
   b. Зеркальное отражение по горизонтали – в каждой строке меняем местами пиксели, которые расположены симметрично относительно середины строки
   c. Зеркальное отражение по горизонтали – в каждом столбце меняем местами пиксели, которые расположены симметрично относительно середины столбца
   d. Поворот на 90 градусов по часовой стрелке – меняем местами ширину и высоту, а матрица пикселей нового изображения равна матрице исходного изображения, повернутой на 90 градусов по часовой стрелке: newMatrix[j][oldHeight – i – 1] = Matrix[i][j]
   e. Поворот на 90 градусов против часовой стрелки – меняем местами ширину и высоту, а матрица пикселей нового изображения равна матрице исходного изображения, повернутой на 90 градусов против часовой стрелки: newMatrix[oldWidth – j – 1][i] = Matrix[i][j]

Экспериментальная часть

Язык программирования: C

Считываем изображение из файла, определяем его тип, P5 или P6. Выполняем требуемую операцию и осуществляем вывод в указанный файл. Для инверсии требуется пройти по матрице пикселей и инвертировать каждый пиксель в соответствии с формулой. Для зеркального отражения надо пройти либо построчно, либо по столбцам и и “развернуть” столбец/строку соответственно. Для поворота требуется создать новую матрицу у которой ширина равна исходной высоте, а высота – исходной ширине, затем пройти и в соответствии с формулой установить значения новой матрицы, которой по итогу станет основной матрице, а ширина и высота поменяются местами.

Выводы

Выполнение данной работы позволило изучить форматы изображений PNM P5 и P6, а также освоить алгоритмы простых преобразований изображений: инверсии, зеркального отражения по горизонтали и вертикали, поворота изображения на 90 градусов по часовой и против часовой стрелки.

Листинг

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "manager.h"

int main(int argc, char* argv[]) {
    if (argc < 4) {
        printf("Not enough arguments\n");
        exit(404);
    }
    if (argc > 4) {
        printf("Too many arguments\n");
        exit(404);
    }
    manageFile(argv[1], argv[2], atoi(argv[3]));
    return 0;
}
```

manager.h

```
#ifndef MANAGER_H
#define MANAGER_H

void manageFile(char* inputFileName, char* outputFileName, int action);

#endif
```

manager.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "ppm.h"
#include "pgm.h"

void manageFile(char* inputFileName, char* outputFileName, int action) {
    FILE* fin = fopen(inputFileName, "rb");
    if (fin == NULL) {
        printf("File can't be opened\n");
        return;
    }
    char header[3];
    fread(header, 1, 2, fin);
    header[2] = '\0';
    fclose(fin);
    if (strcmp(header, "P5") == 0) {
        PGM* pgm = readPGM(inputFileName);
        switch (action) {
        case 0: inversePGM(pgm); writePGM(outputFileName, pgm);  break;
        case 1: reflectPGM_H(pgm); writePGM(outputFileName, pgm); break;
        case 2: reflectPGM_V(pgm); writePGM(outputFileName, pgm); break;
        case 3: rotatePGM_R(pgm); writePGM(outputFileName, pgm); break;
        case 4: rotatePGM_L(pgm); writePGM(outputFileName, pgm); break;
        default: printf("Incorrect action\n"); break;
        }
        if (pgm != NULL) {
            free(pgm->data);
            free(pgm);
        }
        return;
    }
    if (strcmp(header, "P6") == 0) {
        PPM* ppm = readPPM(inputFileName);
        switch (action) {
        case 0: inversePPM(ppm); writePPM(outputFileName, ppm); break;
        case 1: reflectPPM_H(ppm); writePPM(outputFileName, ppm); break;
        case 2: reflectPPM_V(ppm); writePPM(outputFileName, ppm); break;
        case 3: rotatePPM_R(ppm); writePPM(outputFileName, ppm); break;
        case 4: rotatePPM_L(ppm); writePPM(outputFileName, ppm); break;
        default: printf("Incorrect action\n"); break;
        }
        if (ppm != NULL) {
            free(ppm->data);
            free(ppm);
        }
        return;
    }
    printf("File is not a pbm file\n");
}
```

pgm.h

```
#ifndef PGM_H
#define PGM_H

typedef unsigned char uchar;

typedef struct PGM {
	char header[2];
	int width, height;
	uchar maxValue;
	uchar* data;
}PGM;

PGM* readPGM(char* fileName);
void rotatePGM_L(PGM* pgm);
void rotatePGM_R(PGM* pgm);
void inversePGM(PGM* pgm);
void reflectPGM_V(PGM* pgm);
void reflectPGM_H(PGM* pgm);
void writePGM(char* fileName, PGM* pgm);

#endif
```

pgm.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "pgm.h"

PGM* readPGM(char* fileName) {
	FILE* fin = fopen(fileName, "rb");
	if (fin == NULL) {
		printf("File can't be opened\n");
		return NULL;
	}
	PGM* pgm = (PGM*)malloc(sizeof(PGM));
	if (pgm == NULL) {
		printf("Malloc error happened\n");
		return NULL;
	}
	int tmp;
	if (fscanf(fin, "%s%d%d%d", pgm->header, &(pgm->width), &(pgm->height), &tmp) < 4) {
		printf("Header reading error happened\n");
		free(pgm);
		return NULL;
	}
	if (tmp > 255) {
		printf("Unsupported colours\n");
		free(pgm);
		return NULL;
	}
	pgm->maxValue = (uchar)tmp;
	if (pgm->width < 0 || pgm->height < 0) {
		printf("Invalid header\n");
		free(pgm);
		return NULL;
	}
	if (fgetc(fin) == EOF) {
		printf("Data reading error happened\n");
	}
	pgm->data = (uchar*)malloc(pgm->height * pgm->width);
	if (pgm->data == NULL) {
		printf("Malloc error happened\n");
		free(pgm);
		return NULL;
	}
	if (fread(pgm->data, 1, pgm->width * pgm->height, fin) < pgm->width * pgm->height) {
		printf("Data reading error happened\n");
		free(pgm->data);
		free(pgm);
		return NULL;
	}
	if (fclose(fin) != 0) {
		printf("Can't close file\n");
		free(pgm->data);
		free(pgm);
		return NULL;
	}
	return pgm;
}

void inversePGM(PGM* pgm) {
	if (pgm == NULL) {
		return;
	}
	for (int i = 0; i < pgm->height * pgm->width; i++) {
		pgm->data[i] = pgm->maxValue - pgm->data[i];
	}
```

```
}

void reflectPGM_V(PGM* pgm) {
	if (pgm == NULL) {
		return;
	}
	for (int i = 0; i < pgm->height / 2; i++) {
		for (int j = 0; j < pgm->width; j++) {
			uchar tmp = pgm->data[i * pgm->width + j];
			pgm->data[i * pgm->width + j] = pgm->data[(pgm->height - i - 1) * pgm->width
+ j];
			pgm->data[(pgm->height - i - 1) * pgm->width + j] = tmp;
		}
	}
}

void reflectPGM_H(PGM* pgm) {
	if (pgm == NULL) {
		return;
	}
	for (int i = 0; i < pgm->height; i++) {
		for (int j = 0; j < pgm->width / 2; j++) {
			uchar tmp = pgm->data[i * pgm->width + j];
			pgm->data[i * pgm->width + j] = pgm->data[i * pgm->width + pgm->width - j -
1];
			pgm->data[i * pgm->width + pgm->width - j - 1] = tmp;
		}
	}
}

void rotatePGM_L(PGM* pgm) {
	if (pgm == NULL) {
		return;
	}
	int tmp = pgm->width;
	pgm->width = pgm->height;
	pgm->height = tmp;
	uchar* buffer = (uchar*)malloc(pgm->width * pgm->height);
	if (buffer == NULL) {
		printf("Malloc error happened\n");
		free(pgm->data);
		free(pgm);
		pgm = NULL;
		return;
	}
	for (int i = 0; i < pgm->height; i++) {
		for (int j = 0; j < pgm->width; j++) {
			buffer[i * pgm->width + j] = pgm->data[j * pgm->height + pgm->height - i -
1];
		}
	}
	free(pgm->data);
	pgm->data = buffer;
}

void rotatePGM_R(PGM* pgm) {
	if (pgm == NULL) {
		return;
	}
	int tmp = pgm->width;
	pgm->width = pgm->height;
	pgm->height = tmp;
	uchar* buffer = (uchar*)malloc(pgm->width * pgm->height);
	if (buffer == NULL) {
		printf("Malloc error happened\n");
```

```
			free(pgm->data);
			free(pgm);
			pgm = NULL;
			return;
		}
		for (int i = 0; i < pgm->height; i++) {
			for (int j = 0; j < pgm->width; j++) {
				buffer[i * pgm->width + j] = pgm->data[(pgm->width - j - 1) * pgm->height +
i];
			}
		}
		free(pgm->data);
		pgm->data = buffer;
}

void writePGM(char* fileName, PGM* pgm) {
		if (pgm == NULL) {
			return;
		}
		FILE* fout = fopen(fileName, "wb");
		if (fout == NULL) {
			printf("Output file can't be created/rewritten\n");
			return;
		}
		fprintf(fout, "%s\n%i %i\n%i\n", pgm->header, pgm->width, pgm->height, pgm->maxValue);
		fwrite(pgm->data, 1, pgm->width * pgm->height, fout);
		fclose(fout);
}
```

ppm.h

```
#ifndef PPM_H
#define PPM_H

typedef unsigned char uchar;

typedef struct PIXEL {
	uchar r, g, b;
}PIXEL;

typedef struct PPM {
	char header[2];
	int width, height;
	uchar maxValue;
	PIXEL* data;
}PPM;

PPM* readPPM(char* fileName);
void rotatePPM_L(PPM* ppm);
void rotatePPM_R(PPM* ppm);
void inversePPM(PPM* ppm);
void reflectPPM_V(PPM* ppm);
void reflectPPM_H(PPM* ppm);
void writePPM(char* fileName, PPM* ppm);

#endif
```

ppm.c

```
#include <stdio.h>
#include <stdlib.h>
#include "ppm.h"

PPM* readPPM(char* fileName) {
    FILE* fin = fopen(fileName, "rb");
    if (fin == NULL) {
        printf("File can't be opened\n");
        return NULL;
    }
    PPM* ppm = (PPM*)malloc(sizeof(PPM));
    if (ppm == NULL) {
        printf("Malloc error happened\n");
        return NULL;
    }
    int tmp;
    if (fscanf(fin, "%s%d%d%d", ppm->header, &(ppm->width), &(ppm->height), &tmp) < 4) {
        printf("Header reading error happened\n");
        free(ppm);
        return NULL;
    }
    if (tmp > 255) {
        printf("Unsupported colours\n");
        free(ppm);
        return NULL;
    }
    ppm->maxValue = (uchar)tmp;
    if (ppm->width < 0 || ppm->height < 0) {
        printf("Invalid header\n");
        free(ppm);
        return NULL;
    }
    if (fgetc(fin) == EOF) {
        printf("Data reading error happened\n");
        free(ppm);
        return NULL;
    }
    ppm->data = (PIXEL*)malloc(sizeof(PIXEL) * ppm->height * ppm->width);
    if (ppm->data == NULL) {
        printf("Malloc error happened\n");
        free(ppm);
        return NULL;
    }
    if (fread(ppm->data, sizeof(PIXEL), ppm->height * ppm->width, fin) < ppm->height * ppm-
>width) {
        printf("Data reading error happened\n");
        free(ppm->data);
        free(ppm);
        return NULL;
    }
    if (fclose(fin) != 0) {
        printf("File can't be close\n");
    }
    return ppm;
}

void inversePPM(PPM* ppm) {
    if (ppm == NULL) {
        return;
    }
    for (int i = 0; i < ppm->height * ppm->width; i++) {
        ppm->data[i].r = ppm->maxValue - ppm->data[i].r;
```

```
        ppm->data[i].g = ppm->maxValue - ppm->data[i].g;
        ppm->data[i].b = ppm->maxValue - ppm->data[i].b;
    }
}

void reflectPPM_V(PPM* ppm) {
    if (ppm == NULL) {
        return;
    }
    for (int i = 0; i < ppm->height / 2; i++) {
        for (int j = 0; j < ppm->width; j++) {
            PIXEL tmp = ppm->data[i * ppm->width + j];
            ppm->data[i * ppm->width + j] = ppm->data[(ppm->height - i - 1) * ppm->width + j];
            ppm->data[(ppm->height - i - 1) * ppm->width + j] = tmp;
        }
    }
}

void reflectPPM_H(PPM* ppm) {
    if (ppm == NULL) {
        return;
    }
    for (int i = 0; i < ppm->height; i++) {
        for (int j = 0; j < ppm->width / 2; j++) {
            PIXEL tmp = ppm->data[i * ppm->width + j];
            ppm->data[i * ppm->width + j] = ppm->data[i * ppm->width + ppm->width - j - 1];
            ppm->data[i * ppm->width + ppm->width - j - 1] = tmp;
        }
    }
}

void rotatePPM_L(PPM* ppm) {
    if (ppm == NULL) {
        return;
    }
    int tmp = ppm->height;
    ppm->height = ppm->width;
    ppm->width = tmp;
    PIXEL* buffer = (PIXEL*)malloc(sizeof(PIXEL) * ppm->width * ppm->height);
    if (buffer == NULL) {
        printf("Malloc error happened\n");
        free(ppm->data);
        free(ppm);
        ppm = NULL;
        return;
    }
    for (int i = 0; i < ppm->height; i++) {
        for (int j = 0; j < ppm->width; j++) {
            buffer[i * ppm->width + j] = ppm->data[j * ppm->height + ppm->height - i - 1];
        }
    }
    free(ppm->data);
    ppm->data = buffer;
}

void rotatePPM_R(PPM* ppm) {
    if (ppm == NULL) {
        return;
    }
    int tmp = ppm->height;
    ppm->height = ppm->width;
    ppm->width = tmp;
    PIXEL* buffer = (PIXEL*)malloc(sizeof(PIXEL) * ppm->width * ppm->height);
    if (buffer == NULL) {
        printf("Malloc error happened\n");
```

```
        free(ppm->data);
        free(ppm);
        ppm = NULL;
        return;
    }
    for (int i = 0; i < ppm->height; i++) {
        for (int j = 0; j < ppm->width; j++) {
            buffer[i * ppm->width + j] = ppm->data[(ppm->width - j - 1) * ppm->height + i];
        }
    }
    free(ppm->data);
    ppm->data = buffer;
}

void writePPM(char* fileName, PPM* ppm) {
    if (ppm == NULL) {
        return;
    }
    FILE* fout = fopen(fileName, "wb");
    if (fout == NULL) {
        printf("Output file can't be created/rewritten\n");
        return;
    }
    fprintf(fout, "%s\n%i %i\n%i\n", ppm->header, ppm->width, ppm->height, ppm->maxValue);
    fwrite(ppm->data, sizeof(PIXEL), ppm->width * ppm->height, fout);
    fclose(fout);
}
```