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 M3101  
*Дудко Матвей Владимирович***

**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.

### 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).

### Зеркальное отражение по горизонтали

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

### Зеркальное отражение по вертикали

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

### 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.

### 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

Лабораторная работа выполнена на языке C++.

## Хранение данных изображения и заголовка

Данные заголовка файла хранятся в переменных:

char char\_header;  
int width, height;  
unsigned int max\_value;

Чтение производится при помощи функции fscanf:

int scanned = fscanf(file\_in, "P%c\n%i %i\n%i\n", &char\_header, &width, &height, &max\_value);

Где:

* char\_header – номер версии файла (возможные варианты 5 или 6)
* width – ширина изображения
* height – высота изображения
* max\_value – максимально возможное значение яркости
* scanned – переменная для обработки ошибок при чтении файла

Общее количество байт для считывания вычисляется и записывается в переменную k\_bytes:

int k\_bytes = height \* width;  
  
bool is\_color = false;  
  
if (char\_header == '6') {  
 k\_bytes \*= 3;  
 is\_color = true;  
} else if (char\_header != '5') {  
 std::cout << "Unsupported file format\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
}

* is\_colour – переменная, обозначающая цветное изображение или нет

Данные пикселей хранятся в одномерном динамическом массиве, выделяющемся в куче:

pix\_data = (unsigned char \*) calloc(k\_bytes, 1);

И считываются при помощи функции fread:

int bytes\_read = fread(pix\_data, 1, k\_bytes, file\_in);

Где:

* bytes\_read – переменная для обработки ошибок при чтении файла

## Преобразования

### Инверсия

Инверсия цветов изображения работает за один проход по массиву значений пикселей, изменяя значения яркости на противоположные. Не требует выделения дополнительной памяти. Для инверсии и записи результата в файл используется функция inversion.

### Зеркальное отражение по горизонтали

Зеркальное отражение по горизонтали работает за один проход по массиву значений пикселей. Используется дополнительная память для создания нового изображения:

auto new\_pix\_data = (unsigned char \*) calloc(k\_bytes, 1);

Для преобразования и записи результата в файл используется функция horizontal\_reflection.

### Зеркальное отражение по вертикали

Зеркальное отражение по вертикали работает за один проход по массиву значений пикселей. Используется дополнительная память для создания нового изображения:

auto new\_pix\_data = (unsigned char \*) calloc(k\_bytes, 1);

Для преобразования и записи результата в файл используется функция vertical\_reflection.

### Поворот на 90 градусов по часовой стрелке

Поворот на 90 градусов по часовой стрелке работает за один проход по массиву значений пикселей. Используется дополнительная память для создания нового изображения:

auto new\_pix\_data = (unsigned char \*) calloc(k\_bytes, 1);

Для преобразования и записи результата в файл используется функция rotate\_clockwise\_90.

### Поворот на 90 градусов против часовой стрелки

Поворот на 90 градусов против часовой стрелки работает за один проход по массиву значений пикселей. Используется дополнительная память для создания нового изображения:

auto new\_pix\_data = (unsigned char \*) calloc(k\_bytes, 1);

Для преобразования и записи результата в файл используется функция rotate\_counter\_clockwise\_90.

# Вывод

Выполнение данной лабораторной работы позволило изучить такой формат хранения изображений, как PNM, а также позволило освоить и применить простые алгоритмы для обработки изображений: инвертирование цветов, зеркальное отражение изображения по вертикали и горизонтали, поворот изображения на 90 градусов как по часовой стрелке, так и против.

# Листинг кода

Исходный код можно посмотреть на ресурсе GitHub по следующей ссылке: <https://github.com/DudkoMatt/GeometryAndGraphics/blob/master/Lab_01/main.cpp>

Название файла: ./GeometryAndGraphics/Lab\_01/main.cpp

Исходный код:

#include <iostream>  
  
void write\_header(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value);  
  
void write\_data(FILE \*file\_out, int k\_bytes, unsigned char \*pix\_data);  
  
void inversion(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value, int k\_bytes, unsigned char \*pix\_data);  
  
void vertical\_reflection(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value, int k\_bytes, unsigned char \*pix\_data, bool is\_color);  
  
void horizontal\_reflection(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value, int k\_bytes, unsigned char \*pix\_data, bool is\_color);  
  
void rotate\_clockwise\_90(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value, int k\_bytes, unsigned char \*pix\_data, bool is\_color);  
  
void rotate\_counter\_clockwise\_90(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value, int k\_bytes, unsigned char \*pix\_data, bool is\_color);  
  
void free\_data(FILE \*file\_in, FILE \*file\_out, unsigned char \*pix\_data);  
  
  
int main(int argc, char \*argv[]) {  
  
 // Часть 1: обработка агрументов командной строки  
  
 if (argc != 4) {  
 std::cout << "Wrong number of arguments. Syntax:\n<lab>.exe file\_in file\_out conversion\n";  
 return 0;  
 }  
  
 const char \*file\_in\_name = argv[1];  
 const char \*file\_out\_name = argv[2];  
 int conversion = (int) (\*argv[3] - '0');  
  
 if (!(0 <= conversion && conversion <= 4)) {  
 std::cout << "Wrong conversion parameter: " << conversion << std::endl;  
 return 0;  
 }  
  
  
  
  
  
  
  
  
 // Часть 2: проверка на существование файла. Попытка открыть файл на запись  
 // Note: предполагаем, что в картинках нет комментариев  
  
 unsigned char \*pix\_data = nullptr;  
 FILE \*file\_in = nullptr, \*file\_out = nullptr;  
  
 file\_in = fopen(file\_in\_name, "rb");  
  
 if (file\_in == nullptr) {  
 std::cout << "Cannot open file to read: " << file\_in\_name << "\n";  
 return 0;  
 }  
  
 // Часть 3: чтение файла  
  
 // Чтение заголовка  
 char char\_header;  
 int width, height;  
 unsigned int max\_value;  
  
 int scanned = fscanf(file\_in, "P%c\n%i %i\n%i\n", &char\_header, &width, &height, &max\_value);  
  
 if (scanned != 4) {  
 std::cout << "Something went wrong. Error during scanning header\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
 }  
  
 if (width <= 0 || height <= 0 || max\_value <= 0 || max\_value > 255) {  
 std::cout << "Something went wrong. At least one of the numbers is wrong. Maybe there is a comment in a file\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
 }  
  
 int k\_bytes = height \* width;  
  
 bool is\_color = false;  
 if (char\_header == '6') {  
 k\_bytes \*= 3;  
 is\_color = true;  
 } else if (char\_header != '5') {  
 std::cout << "Unsupported file format\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
 }  
  
 pix\_data = (unsigned char \*) calloc(k\_bytes, 1);  
 if (!pix\_data) {  
 std::cout << "Cannot allocate " << k\_bytes << " bytes of memory\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
 }  
  
 int bytes\_read = fread(pix\_data, 1, k\_bytes, file\_in);  
  
  
  
  
  
  
 if (bytes\_read < k\_bytes) {  
 std::cout << "Can't read all data:\n";  
 std::cout << "Expected " << k\_bytes << " bytes, but only " << bytes\_read << " were read\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
 }  
  
 long current\_pos\_in\_file = ftell(file\_in);  
 fseek(file\_in, 0, SEEK\_END);  
 if (current\_pos\_in\_file != ftell(file\_in)) {  
 std::cout << "Error: file contains more data than expected\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
 }  
  
 fclose(file\_in);  
 file\_in = nullptr;  
  
 // Часть 4: обработка действий  
  
 file\_out = fopen(file\_out\_name, "wb");  
  
 if (file\_out == nullptr) {  
 std::cout << "Cannot open file to write: " << file\_out\_name << "\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
 }  
  
 if (conversion == 0) {  
 // Инверсия  
 inversion(file\_out, char\_header, width, height, max\_value, k\_bytes, pix\_data);  
 } else if (conversion == 1) {  
 // Зеркальное отображение по горизонтали  
 horizontal\_reflection(file\_out, char\_header, width, height, max\_value, k\_bytes, pix\_data, is\_color);  
 } else if (conversion == 2) {  
 // Зеркальное отражение по вертикали  
 vertical\_reflection(file\_out, char\_header, width, height, max\_value, k\_bytes, pix\_data, is\_color);  
 } else if (conversion == 3) {  
 // Поворот на 90 градусов по часовой стрелке  
 rotate\_clockwise\_90(file\_out, char\_header, width, height, max\_value, k\_bytes, pix\_data, is\_color);  
 } else if (conversion == 4) {  
 // Поворот на 90 градусов против часовой стрелки  
 rotate\_counter\_clockwise\_90(file\_out, char\_header, width, height, max\_value, k\_bytes, pix\_data, is\_color);  
 }  
  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
}  
  
void write\_header(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value) {  
 fseek(file\_out, 0, SEEK\_SET);  
 fprintf(file\_out, "P%c\n%i %i\n%i\n", char\_header, width, height, max\_value);  
}  
  
  
  
void write\_data(FILE \*file\_out, int k\_bytes, unsigned char \*pix\_data) {  
 fwrite(pix\_data, k\_bytes, 1, file\_out);  
}  
  
void inversion(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value, int k\_bytes, unsigned char \*pix\_data) {  
 write\_header(file\_out, char\_header, width, height, max\_value);  
 for (int i = 0; i < k\_bytes; ++i) {  
 pix\_data[i] = max\_value - pix\_data[i];  
 }  
 write\_data(file\_out, k\_bytes, pix\_data);  
}  
  
void vertical\_reflection(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value, int k\_bytes, unsigned char \*pix\_data, bool is\_color) {  
 write\_header(file\_out, char\_header, width, height, max\_value);  
  
 auto new\_pix\_data = (unsigned char \*) calloc(k\_bytes, 1);  
 if (!new\_pix\_data) {  
 std::cout << "Cannot allocate " << k\_bytes << " bytes of memory\n";  
 free\_data(nullptr, file\_out, pix\_data);  
 exit(0);  
 }  
  
 if (is\_color) {  
 width \*= 3;  
 }  
  
 for (int i = 0; i < height; ++i) {  
 for (int j = 0; j < width; ++j) {  
 new\_pix\_data[(height - i - 1) \* width + j] = pix\_data[i \* width + j];  
 }  
 }  
  
 write\_data(file\_out, k\_bytes, new\_pix\_data);  
 free(new\_pix\_data);  
}  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
void horizontal\_reflection(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value, int k\_bytes, unsigned char \*pix\_data, bool is\_color) {  
 write\_header(file\_out, char\_header, width, height, max\_value);  
  
 auto new\_pix\_data = (unsigned char \*) calloc(k\_bytes, 1);  
 if (!new\_pix\_data) {  
 std::cout << "Cannot allocate " << k\_bytes << " bytes of memory\n";  
 free\_data(nullptr, file\_out, pix\_data);  
 exit(0);  
 }  
  
 if (is\_color) {  
 width \*= 3;  
 for (int i = 0; i < height; ++i) {  
 for (int j = 0; j < width; j += 3) {  
 new\_pix\_data[i \* width + j] = pix\_data[i \* width + (width - j - 3)];  
 new\_pix\_data[i \* width + j + 1] = pix\_data[i \* width + (width - j - 2)];  
 new\_pix\_data[i \* width + j + 2] = pix\_data[i \* width + (width - j - 1)];  
 }  
 }  
 } else {  
 for (int i = 0; i < height; ++i) {  
 for (int j = 0; j < width; ++j) {  
 new\_pix\_data[i \* width + j] = pix\_data[i \* width + width - j - 1];  
 }  
 }  
 }  
  
 write\_data(file\_out, k\_bytes, new\_pix\_data);  
 free(new\_pix\_data);  
}  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
void rotate\_counter\_clockwise\_90(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value, int k\_bytes, unsigned char \*pix\_data, bool is\_color){  
 // Swap height and width  
 write\_header(file\_out, char\_header, height, width, max\_value);  
  
 auto new\_pix\_data = (unsigned char \*) calloc(k\_bytes, 1);  
 if (!new\_pix\_data) {  
 std::cout << "Cannot allocate " << k\_bytes << " bytes of memory\n";  
 free\_data(nullptr, file\_out, pix\_data);  
 exit(0);  
 }  
  
 if (is\_color) {  
 width \*= 3;  
  
 int counter = 0;  
 for (int j = width - 3; j >= 0; j -= 3) {  
 for (int i = 0; i < height; ++i) {  
 new\_pix\_data[counter++] = pix\_data[i \* width + j];  
 new\_pix\_data[counter++] = pix\_data[i \* width + j + 1];  
 new\_pix\_data[counter++] = pix\_data[i \* width + j + 2];  
 }  
 }  
  
 } else {  
 int counter = 0;  
 for (int j = width - 1; j >= 0; --j) {  
 for (int i = 0; i < height; ++i) {  
 new\_pix\_data[counter++] = pix\_data[i \* width + j];  
 }  
 }  
 }  
  
 write\_data(file\_out, k\_bytes, new\_pix\_data);  
 free(new\_pix\_data);  
}  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
void rotate\_clockwise\_90(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value, int k\_bytes, unsigned char \*pix\_data, bool is\_color){  
 // Swap height and width  
 write\_header(file\_out, char\_header, height, width, max\_value);  
  
 auto new\_pix\_data = (unsigned char \*) calloc(k\_bytes, 1);  
 if (!new\_pix\_data) {  
 std::cout << "Cannot allocate " << k\_bytes << " bytes of memory\n";  
 free\_data(nullptr, file\_out, pix\_data);  
 exit(0);  
 }  
  
 if (is\_color) {  
 width \*= 3;  
  
 int counter = 0;  
 for (int j = 0; j < width; j += 3) {  
 for (int i = height - 1; i >= 0; --i) {  
 new\_pix\_data[counter++] = pix\_data[i \* width + j];  
 new\_pix\_data[counter++] = pix\_data[i \* width + j + 1];  
 new\_pix\_data[counter++] = pix\_data[i \* width + j + 2];  
 }  
 }  
  
 } else {  
 int counter = 0;  
 for (int j = 0; j < width; ++j) {  
 for (int i = height - 1; i >= 0; --i) {  
 new\_pix\_data[counter++] = pix\_data[i \* width + j];  
 }  
 }  
 }  
  
 write\_data(file\_out, k\_bytes, new\_pix\_data);  
 free(new\_pix\_data);  
}  
  
void free\_data(FILE \*file\_in, FILE \*file\_out, unsigned char \*pix\_data) {  
 if (file\_in) fclose(file\_in);  
 if (file\_out) fclose(file\_out);  
 if (pix\_data) free(pix\_data);  
}