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 № 3**

**По дисциплине «Компьютерная графика и геометрия»**

**Изучение алгоритмов псевдотонирования изображений**

**Выполнил студент группы M3101  
*Дудко Матвей Владимирович***

**Проверил:  
Скаков Павел Сергеевич**

***САНКТ-ПЕТЕРБУРГ***

***2020***

# Цель работы

Изучение алгоритмов и реализация программы, применяющей алгоритмы дизеринга к изображению в формате PGM (P5) с учетом гамма-коррекции.

# Описание работы

Программа должна быть написана на C/C++ и не использовать внешние библиотеки.

Аргументы передаются через командную строку:

**program.exe <имя\_входного\_файла> <имя\_выходного\_файла> <градиент> <дизеринг> <битность> <гамма>**

где

* <имя\_входного\_файла>, <имя\_выходного\_файла>: формат файлов: PGM P5; ширина и высота берутся из <имя\_входного\_файла>;
* <градиент>: 0 - используем входную картинку, 1 - рисуем горизонтальный градиент (0-255) (ширина и высота берутся из <имя\_входного\_файла>);
* <дизеринг> - алгоритм дизеринга:
  + 0 – Нет дизеринга;
  + 1 – Ordered (8x8);
  + 2 – Random;
  + 3 – Floyd–Steinberg;
  + 4 – Jarvis, Judice, Ninke;
  + 5 - Sierra (Sierra-3);
  + 6 - Atkinson;
  + 7 - Halftone (4x4, orthogonal);
* <битность> - битность результата дизеринга (1..8);
* <гамма>: 0 - sRGB гамма, иначе - обычная гамма с указанным значением.

**Частичное решение**:

* <градиент> = 1;
* <дизеринг> = 0..3;
* <битность> = 1..8;
* <гамма> = 1 (аналогично отсутствию гамма-коррекции)

+ корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок.

**Полное решение**: все остальное

Если программе передано значение, которое не поддерживается – следует сообщить об ошибке.

Коды возврата:

0 - ошибок нет

1 - произошла ошибка

В поток вывода ничего не выводится (printf, cout).

Сообщения об ошибках выводятся в поток вывода ошибок:

С: fprintf(stderr, "Error\n");

C++: std::cerr

Следующие параметры гарантировано не будут выходить за обусловленные значения:

* <градиент> = 0 или 1;
* <битность> = 1..8;
* width и height в файле - положительные целые значения;
* яркостных данных в файле ровно width \* height;
* <гамма> - вещественная неотрицательная

# Теоретическая часть

## Определение и применение дизеринга:

*Дизеринг (англ. dither), псевдотонирование* — при обработке цифровых сигналов представляет собой подмешивание в первичный сигнал псевдослучайного шума со специально подобранным спектром. Применяется при обработке цифрового звука, видео и графической информации для уменьшения негативного эффекта от квантования.

В компьютерной графике дизеринг используется для создания иллюзии глубины цвета для изображений с относительно небольшим количеством цветов в палитре. Отсутствующие цвета составляются из имеющихся путем их «перемешивания».

При оптимизации изображений путём уменьшения количества цветов, применение дизеринга приводит к визуальному улучшению изображения.

## Виды дизеринга:

Все алгоритмы дизеринга условно можно поделить на 2 вида: алгоритмы с рассеиванием ошибки (Error diffusion) и упорядоченные алгоритмы (Ordered).

Для определения пороговых (threshold) цветов для разных битностей воспользуемся следующим алгоритмом: для округления текущего значения цвета до ближайшего, который можно отобразить в задаваемой битности B, из целочисленного значения цвета берутся B старших бит и дублируются сдвигами по B бит в текущее значение цвета.

## Алгоритмы дизеринга, использующиеся в лабораторной работе:

1. **Нет дизеринга (no dithering)**

Данный метод подразумевает лишь округление всех цветов до пороговых.

### Алгоритмы с упорядоченным распределением ошибки (Ordered):

1. **Ordered (8x8)**

Алгоритм уменьшает количество цветов, применяя карту порогов M (другое обозначение: Bayer matrix) к отображаемым пикселям, в результате чего некоторые пиксели меняют цвет в зависимости от расстояния исходного цвета от доступных записей цветов в уменьшенной палитре.

Для каждого пикселя производится смещение его значения цвета на соответствующее значение из карты порогов M в соответствии с его местоположением, в результате чего значение пикселя квантуется на другой цвет, если оно превышает пороговое значение.

Для большинства случаев сглаживания достаточно просто добавить пороговое значение к каждому пикселю или эквивалентно сравнить значение этого пикселя с порогом: если значение пикселя меньше, чем число в соответствующей ячейке матрицы, записать в пиксель черный цвет, в противном случае, белый в случае битности 1.

Поскольку алгоритм работает с одиночными пикселями и не имеет условных операторов, он очень быстрый и подходит для преобразований в реальном времени. Кроме того, расположение шаблонов сглаживания всегда остается одинаковым относительно кадра дисплея, что способствует улучшению сжатия изображения. Упорядоченное сглаживание больше подходит для линейной графики, так как приводит к более прямым линиям и меньшему количеству аномалий. Однако, результат, получаемый после работы данного метода, получаются хуже, чем после применения алгоритмов с рассеянием ошибок, о которых будет изложено далее.

1. **Halftone (4x4, orthogonal)**

Halftone, полутонирование – создание изображения со многими уровнями серого или цвета (т.е. слитный тон) на аппарате с меньшим количеством тонов, обычно чёрно-белый принтер.

В случае обработки цифрового изображения Halftone представляет собой матрицы порогов, позволяющие воспроизводить “точки” как при печати изображения.

1. **Random**

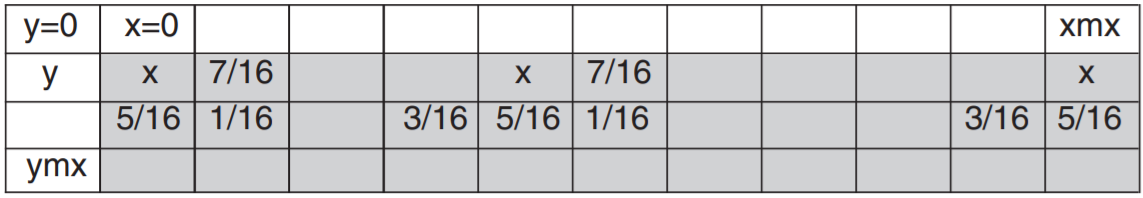
Данный алгоритм можно также отнести к типу ordered с тем условием, что для определения добавки к значению текущего пикселя берется не элемент матрицы, а случайное число.

### Алгоритмы с рассеиванием ошибки (Error diffusion):

1. **Floyd–Steinberg**

Первая и возможно самая известная формула рассеивания ошибок была опубликована Робертом Флойдом и Луисом Стейнбергом в 1976 году. Рассеивание ошибок происходит по следующей схеме:

* **х** - текущий пиксель, от которого распространяется ошибка
* **y,x** -строка/столбец изображения
* **ymx, xmx** - номер последние строки и столбца



|  |  |  |
| --- | --- | --- |
|  | Х | +7 \* 0,6875 |
| +3 \* 0,6875 | +5 \* 0,6875 | +1 \* 0,6875 |

Пример преобразования пиксельного значения 96: при окрашивании пикселя в темно-серый мы получаем ошибку 11. Мы распространяем эту ошибку окружающим пикселям, поделив 11 на 16 (= 0,6875), затем умножаем её на соответствующие значения, например:

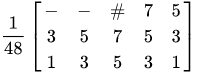
Если наш пиксель округляется в большее значение (например, 129 к 170), то ошибка будет отрицательной соответственно.

Алгоритм даёт достаточно хорошее качество, а также требует только один передний массив (одномерный массив шириной в изображение, где хранятся значения ошибок, распространяемые к следующей строке). Кроме того, поскольку его делитель 16, вместо деления можно использовать битовые сдвиги. Так алгоритм достигает высокой скорости работы даже на старом оборудовании.

Что касается значений 1/3/5/7, используемых для распространения ошибки – они были выбраны специально, потому что они создают равномерный клетчатый узор для серого изображения.

1. **Jarvis, Judice, Ninke**

В год, когда Флойд и Стейнберг опубликовали свой знаменитый алгоритм дизеринга, был издан менее известный, но гораздо более мощный алгоритм. Фильтр Джарвиса, Джудиса и Нинке значительно сложнее, чем Флойда-Стейнберга:

Алгоритм Джарвиса, Джудиса и Нинке аналогичен алгоритму Флойда-Стейнберга с точностью до матрицы распределения ошибки. При данном алгоритме ошибка распределяется на в три раза больше пикселей, чем у Флойда-Стейнберга, что приводит к более гладкому и более тонкому результату.

1. **Sierra (Sierra-3)**

Аналогично алгоритму Флойда-Стейнберга, но с другой матрицей:

1. **Atkinson**

Аналогично алгоритму Флойда-Стейнберга, но с другой матрицей:

# Экспериментальная часть

Лабораторная работа выполнена на языке C++. Стандарт языка C++14.

Для удобства все основные функции для работы с изображением были вынесены в отдельную библиотеку.

## Хранение данных изображения и заголовка при чтении из файла

Данные заголовка файла хранятся в переменных:

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

Чтение производится при помощи функции fscanf:

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

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

int k\_bytes = height \* width;

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

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

И считываются при помощи функции fread:

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

Где:

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

Также исходно производится гамма-преобразование исходных цветов:

decode\_gamma\_from\_file(input\_pix\_data, width \* height, gamma);

## Хранение данных нового изображения

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

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

И исходно инициализируются нулями:

// Initialization  
for (int i = 0; i < k\_bytes; ++i) {  
 pix\_data[i] = 0;  
}

Далее алгоритмы записывают свои преобразования в данный массив.

## Алгоритмы дизеринга

Чтобы объединить считывание цветов из входного файла и горизонтального градиента используется функция get\_pix\_color:

unsigned char get\_pix\_color(int x, int y, int width, const unsigned char \*pix\_data) {  
 if (pix\_data == nullptr) {  
 // Горизонтальный градиент  
 // x in [0 .. width - 1]  
 return (unsigned char) (255.0 \* x / (width - 1));  
 } else {  
 return \*(pix\_data + y \* width + x);  
 }  
}

### Нет дизеринга (no dithering)

Для отрисовки горизонтального градиента: используется функция no\_dithering, которая проходит один раз по горизонтали и заполняет цветом соответствующие вертикали при помощи функции fill\_vertical\_line. Дополнительная память не используется.

Для входного изображения: используется функция no\_dithering\_file, которая округляет все цвета пикселей в соответствии с битностью при помощи функции change\_bitness. Дополнительная память не используется.

### Ordered (8x8)

Для данного алгоритма используется функция ordered\_dithering, которая проходит по всем пикселям, изменяя их яркость в соответствии со значением из таблицы Bayer\_Matrix\_double, зависящим от положения пикселя, и затем округляет до ближайшего доступного цвета из палитры функцией change\_bitness. Дополнительная память не используется.

### Random

Для данного алгоритма используется функция random\_dithering, которая проходит по всем пикселям, изменяя их яркость случайным целым числом из диапазона [-128, 128], и затем округляет до ближайшего доступного цвета из палитры функцией change\_bitness. Дополнительная память не используется.

### Floyd- Steinberg

Для данного алгоритма используется функция Floyd\_Steinberg\_dithering.

Значения ошибок при кодировании пикселей хранятся дополнительно выделяемой памяти – в векторе errors, размерами height \* width, типа double.

Алгоритм проходит по всем пикселям и кодирует их в ближайший цвет с учетом ошибки кодирования предыдущих пикселей.

Распределение ошибки происходит в соответствии с таблицей, приведенной в теоретической части отчета.

### Jarvis, Judice, Ninke

### Sierra (Sierra-3)

### Atkinson

Данные три алгоритма полностью аналогичны алгоритму Флойда-Стейнберга с точностью до матрицы распределения ошибок при кодировании пикселей, которые приведены в теоретической части отчета.

Для запуска данных алгоритмов используются следующие функции соответственно:

* + - Jarvis\_Judice\_Ninke\_dithering
    - Sierra\_3\_dithering
    - Atkinson\_dithering

Аналогично используется вектор errors размерами height \* width типа double для хранения ошибок.

### Halftone (4x4, orthogonal)

Для данного алгоритма используется функция Halftone\_dithering.

Алгоритм проходит по всем значениям пикселей и меняет их при помощи функции find\_nearest\_palette\_color, которая возвращает 0, если текущий цвет пикселя меньше, чем цвет из таблицы Halftone\_Matrix\_double, или возвращает цвет из палитры цветов, если текущий цвет больше цвета из таблицы Halftone\_Matrix\_double.

# Вывод

Выполнение данной лабораторной работы позволило изучить алгоритмы псевдотонирования изображений как с упорядоченным распределением ошибок, так и с рассеиванием ошибок. Были реализованы следующие алгоритмы для псевдотонирования изображений:

1. Ordered (8x8)
2. Random
3. Floyd–Steinberg
4. Jarvis, Judice, Ninke
5. Sierra (Sierra-3)
6. Atkinson
7. Halftone (4x4, orthogonal)

При реализации чтения и записи изображения была изучена гамма-коррекция значений пикселей изображения. Реализована гамма-коррекция для вещественного значения гаммы, а также sRGB гамма-коррекция.

# Листинг кода

Исходный код можно посмотреть на ресурсе GitHub по следующей ссылке: <https://github.com/DudkoMatt/GeometryAndGraphics/blob/master/Lab_03>

Содержание проекта:

./GeometryAndGraphics/Lab\_03/

main.cpp

write\_data\_pnm.cpp

write\_data\_pnm.h

Название файла: ./GeometryAndGraphics/Lab\_03/main.cpp

Исходный код:

#include <iostream>  
#include <cmath>  
#include <vector>  
#include "write\_data\_pnm.h"  
#include <algorithm>  
#include <string>  
  
//#define FILE\_OUTPUT  
#define ENABLE\_FILE\_INPUT  
  
// Debug output  
#ifdef FILE\_OUTPUT  
#include <fstream>  
#endif  
  
const double Bayer\_Matrix\_double[8][8] = {  
 {-0.5, 0.25, -0.3125, 0.4375, -0.453125, 0.296875, -0.265625, 0.484375},  
 {0.0, -0.25, 0.1875, -0.0625, 0.046875, -0.203125, 0.234375, -0.015625},  
 {-0.375, 0.375, -0.4375, 0.3125, -0.328125, 0.421875, -0.390625, 0.359375},  
 {0.125, -0.125, 0.0625, -0.1875, 0.171875, -0.078125, 0.109375, -0.140625},  
 {-0.46875, 0.28125, -0.28125, 0.46875, -0.484375, 0.265625, -0.296875, 0.453125},  
 {0.03125, -0.21875, 0.21875, -0.03125, 0.015625, -0.234375, 0.203125, -0.046875},  
 {-0.34375, 0.40625, -0.40625, 0.34375, -0.359375, 0.390625, -0.421875, 0.328125},  
 {0.15625, -0.09375, 0.09375, -0.15625, 0.140625, -0.109375, 0.078125, -0.171875}  
};  
  
const int MATRIX\_SIZE = 8;  
const double Halftone\_Matrix\_double[4][4] = {  
 {0.375, 0.75, 0.625, 0.1875},  
 {0.6875, 0.9375, 0.8125, 0.4375},  
 {0.5625, 0.875, 0.3125, 0.0625},  
 {0.25, 0.5, 0.125, 0.0}  
};  
  
// Вывод в 0..255  
unsigned char get\_pix\_color(int x, int y, int width, const unsigned char \*pix\_data) {  
 if (pix\_data == nullptr) {  
 // Горизонтальный градиент  
 // x in [0 .. width - 1]  
 return (unsigned char) (255.0 \* x / (width - 1));  
 } else {  
 return \*(pix\_data + y \* width + x);  
 }  
}  
  
// aka. find\_nearest\_color, округление  
unsigned char change\_bitness(unsigned bitness, unsigned char data) {  
 unsigned char tmp = data & (((1u << bitness) - 1) << (8 - bitness));  
 data = 0;  
  
 for (unsigned i = 0; i < 8 / bitness + 1; ++i) {  
 data = data | ((unsigned char) (tmp >> bitness \* i));  
 }  
  
 return data;  
}  
  
void no\_dithering\_file(unsigned bitness, size\_t k, unsigned char \*pix\_data) {  
 if (bitness == 8) return;  
 for (int i = 0; i < k; ++i) {  
 \*(pix\_data + i) = change\_bitness(bitness, \*(pix\_data + i));  
 }  
}  
  
void fill\_vertical\_line(int x, int width, int height, unsigned char color, double gamma, unsigned char \*pix\_data) {  
 for (int i = 0; i < height; ++i) {  
 draw\_pix(pix\_data, width, x, i, color, gamma);  
 }  
}  
  
void no\_dithering(int width, int height, unsigned char \*pix\_data, double gamma, unsigned bitness) {  
 for (int i = 0; i < width; ++i) {  
 fill\_vertical\_line(  
 i, width, height,  
 change\_bitness(  
 bitness,  
 get\_pix\_color(i, 0, width, nullptr)  
 ),  
 gamma, pix\_data);  
 }  
}  
  
  
void ordered\_dithering(int width, int height, unsigned char \*pix\_data, double gamma, unsigned bitness,  
 unsigned char \*pix\_data\_input = nullptr) {  
 for (int y = 0; y < height; ++y) {  
 for (int x = 0; x < width; ++x) {  
  
 double barrier\_brightness = Bayer\_Matrix\_double[y % MATRIX\_SIZE][x % MATRIX\_SIZE];  
  
 draw\_pix(pix\_data, width, x, y,  
 change\_bitness(bitness, (unsigned char)  
  
 limit\_brightness(get\_pix\_color(x, y, width, pix\_data\_input) + barrier\_brightness \* 255)  
  
 ),  
 gamma);  
  
  
 }  
 }  
}  
  
void random\_dithering(int width, int height, unsigned char \*pix\_data, double gamma, unsigned bitness,  
 unsigned char \*pix\_data\_input = nullptr) {  
 for (int y = 0; y < height; ++y) {  
 for (int x = 0; x < width; ++x) {  
  
 unsigned char curr\_brightness\_char = get\_pix\_color(x, y, width, pix\_data\_input);  
 unsigned char nearest\_palette\_color;  
  
 nearest\_palette\_color = change\_bitness(bitness,  
 limit\_brightness(curr\_brightness\_char +  
 (((double) std::rand() / (RAND\_MAX)) \* 255 - 128)));  
  
 draw\_pix(pix\_data, width, x, y,  
 nearest\_palette\_color,  
 gamma);  
  
 }  
 }  
}  
  
void Floyd\_Steinberg\_dithering(int width, int height, unsigned char \*pix\_data, double gamma, unsigned bitness,  
 unsigned char \*pix\_data\_input = nullptr) {  
 std::vector<std::vector<double>> errors = std::vector<std::vector<double>>(height, std::vector<double>(width, 0));  
  
 for (int y = 0; y < height; ++y) {  
 for (int x = 0; x < width; ++x) {  
  
 unsigned char curr\_brightness\_char = limit\_brightness(  
 get\_pix\_color(x, y, width, pix\_data\_input) + errors[y][x]);  
 unsigned char nearest\_palette\_color = change\_bitness(bitness, curr\_brightness\_char);  
 int err = curr\_brightness\_char - nearest\_palette\_color;  
  
 // Вправо на данной строке  
 if (x != width - 1) {  
 errors[y][x + 1] += 7 \* err / 16.0;  
 }  
  
 // Вниз на строку  
 if (y != height - 1) {  
 // Влево  
 if (x != 0)  
 errors[y + 1][x - 1] += 3 \* err / 16.0;  
  
 // Центр  
 errors[y + 1][x] += 5 \* err / 16.0;  
  
 // Вправо  
 if (x != width - 1)  
 errors[y + 1][x + 1] += err / 16.0;  
 }  
  
 draw\_pix(pix\_data, width, x, y,  
 nearest\_palette\_color,  
 gamma);  
  
  
 }  
 }  
  
}  
  
void Jarvis\_Judice\_Ninke\_dithering(int width, int height, unsigned char \*pix\_data, double gamma, unsigned bitness,  
 unsigned char \*pix\_data\_input = nullptr) {  
 std::vector<std::vector<double>> errors = std::vector<std::vector<double>>(height, std::vector<double>(width, 0));  
  
 for (int y = 0; y < height; ++y) {  
 for (int x = 0; x < width; ++x) {  
  
 unsigned char curr\_brightness\_char = limit\_brightness(  
 get\_pix\_color(x, y, width, pix\_data\_input) + errors[y][x]);  
 unsigned char nearest\_palette\_color = change\_bitness(bitness, curr\_brightness\_char);  
 int err = curr\_brightness\_char - nearest\_palette\_color;  
  
 const double k = 48.0;  
  
 // Вправо на данной строке  
 if (x < width - 1) {  
 errors[y][x + 1] += 7 \* err / k;  
 if (x < width - 2)  
 errors[y][x + 2] += 5 \* err / k;  
 }  
  
 // Вниз на строку  
 if (y < height - 1) {  
 // Влево на 2  
 if (x >= 2)  
 errors[y + 1][x - 2] += 3 \* err / k;  
  
 // Влево на 1  
 if (x != 0)  
 errors[y + 1][x - 1] += 5 \* err / k;  
  
 // Центр  
 errors[y + 1][x] += 7 \* err / k;  
  
 // Вправо на 1  
 if (x < width - 1)  
 errors[y + 1][x + 1] += 5 \* err / k;  
  
 // Вправо на 2  
 if (x < width - 2)  
 errors[y + 1][x + 2] += 3 \* err / k;  
  
 // Если есть строка на 2 ниже  
 if (y < height - 2) {  
 // Влево на 2  
 if (x >= 2)  
 errors[y + 2][x - 2] += 1 \* err / k;  
  
 // Влево на 1  
 if (x != 0)  
 errors[y + 2][x - 1] += 3 \* err / k;  
  
 // Центр  
 errors[y + 2][x] += 5 \* err / k;  
  
 // Вправо на 1  
 if (x < width - 1)  
 errors[y + 2][x + 1] += 3 \* err / k;  
  
 // Вправо на 2  
 if (x < width - 2)  
 errors[y + 2][x + 2] += 1 \* err / k;  
 }  
 }  
  
 draw\_pix(pix\_data, width, x, y,  
 nearest\_palette\_color,  
 gamma);  
  
  
 }  
 }  
}  
  
void Sierra\_3\_dithering(int width, int height, unsigned char \*pix\_data, double gamma, unsigned bitness,  
 unsigned char \*pix\_data\_input = nullptr) {  
 std::vector<std::vector<double>> errors = std::vector<std::vector<double>>(height, std::vector<double>(width, 0));  
  
 for (int y = 0; y < height; ++y) {  
 for (int x = 0; x < width; ++x) {  
  
 unsigned char curr\_brightness\_char = limit\_brightness(  
 get\_pix\_color(x, y, width, pix\_data\_input) + errors[y][x]);  
 unsigned char nearest\_palette\_color = change\_bitness(bitness, curr\_brightness\_char);  
 int err = curr\_brightness\_char - nearest\_palette\_color;  
  
 const double k = 32.0;  
 const double delta\_err = err / k;  
  
 // Вправо на данной строке  
 if (x < width - 1) {  
 errors[y][x + 1] += 5 \* delta\_err;  
 if (x < width - 2)  
 errors[y][x + 2] += 3 \* delta\_err;  
 }  
  
 // Вниз на строку  
 if (y < height - 1) {  
 // Влево на 2  
 if (x >= 2)  
 errors[y + 1][x - 2] += 2 \* delta\_err;  
  
 // Влево на 1  
 if (x != 0)  
 errors[y + 1][x - 1] += 4 \* delta\_err;  
  
 // Центр  
 errors[y + 1][x] += 5 \* delta\_err;  
  
 // Вправо на 1  
 if (x < width - 1)  
 errors[y + 1][x + 1] += 4 \* delta\_err;  
  
 // Вправо на 2  
 if (x < width - 2)  
 errors[y + 1][x + 2] += 2 \* delta\_err;  
  
 // Если есть строка на 2 ниже  
 if (y < height - 2) {  
 // Влево на 1  
 if (x != 0)  
 errors[y + 2][x - 1] += 2 \* delta\_err;  
  
 // Центр  
 errors[y + 2][x] += 3 \* delta\_err;  
  
 // Вправо на 1  
 if (x < width - 1)  
 errors[y + 2][x + 1] += 2 \* delta\_err;  
 }  
 }  
  
 draw\_pix(pix\_data, width, x, y,  
 nearest\_palette\_color,  
 gamma);  
  
 }  
 }  
}  
  
void Atkinson\_dithering(int width, int height, unsigned char \*pix\_data, double gamma, unsigned bitness,  
 unsigned char \*pix\_data\_input = nullptr) {  
 std::vector<std::vector<double>> errors = std::vector<std::vector<double>>(height, std::vector<double>(width, 0));  
  
 for (int y = 0; y < height; ++y) {  
 for (int x = 0; x < width; ++x) {  
  
 unsigned char curr\_brightness\_char = limit\_brightness(  
 get\_pix\_color(x, y, width, pix\_data\_input) + errors[y][x]);  
 unsigned char nearest\_palette\_color = change\_bitness(bitness, curr\_brightness\_char);  
 int err = curr\_brightness\_char - nearest\_palette\_color;  
  
 const double k = 8.0;  
 const double delta\_err = err / k;  
  
 // Вправо на данной строке  
 if (x < width - 1) {  
 errors[y][x + 1] += delta\_err;  
 if (x < width - 2)  
 errors[y][x + 2] += delta\_err;  
 }  
  
 // Вниз на строку  
 if (y < height - 1) {  
 // Влево на 1  
 if (x != 0)  
 errors[y + 1][x - 1] += delta\_err;  
  
 // Центр  
 errors[y + 1][x] += delta\_err;  
  
 // Вправо на 1  
 if (x < width - 1)  
 errors[y + 1][x + 1] += delta\_err;  
  
 // Если есть строка на 2 ниже  
 if (y < height - 2) {  
 // Центр  
 errors[y + 2][x] += delta\_err;  
 }  
 }  
  
 draw\_pix(pix\_data, width, x, y,  
 nearest\_palette\_color,  
 gamma);  
  
  
 }  
 }  
}  
  
unsigned char calc\_value(unsigned char pattern, unsigned bitness) {  
 return change\_bitness(bitness, pattern << (8 - bitness));  
}  
  
unsigned char  
find\_nearest\_palette\_color(unsigned bitness, unsigned char current\_color, unsigned char barrier\_brightness) {  
 // Return color in [0..255]  
  
 if (current\_color <= barrier\_brightness)  
 return 0;  
 else {  
 unsigned char \_pattern = (unsigned) current\_color >> (8 - bitness);  
 if (calc\_value(\_pattern, bitness) <= barrier\_brightness)  
 \_pattern++;  
  
 return calc\_value(\_pattern, bitness);  
 }  
}  
  
void Halftone\_dithering(int width, int height, unsigned char \*pix\_data, double gamma, unsigned bitness,  
 unsigned char \*pix\_data\_input = nullptr) {  
 for (int y = 0; y < height; ++y) {  
 for (int x = 0; x < width; ++x) {  
  
 draw\_pix(pix\_data, width, x, y,  
 find\_nearest\_palette\_color(bitness,  
 get\_pix\_color(x, y, width, pix\_data\_input),  
 (unsigned char) (255 \* Halftone\_Matrix\_double[y % 4][x % 4])),  
 gamma);  
  
  
 }  
 }  
}  
  
int main(int argc, char \*argv[]) {  
  
 if (argc != 7 && argc != 6) {  
 std::cerr  
 << "Wrong number of arguments. Syntax:\n<lab>.exe file\_in file\_out gradient dithering bitness [gamma]\n";  
 return 1;  
 }  
  
 const char \*file\_in\_name = argv[1];  
 const char \*file\_out\_name = argv[2];  
  
 unsigned gradient = argv[3][0] - '0';  
 unsigned dithering = argv[4][0] - '0';  
 unsigned bitness = argv[5][0] - '0';  
  
 double gamma = 0; // if gamma = 0, sRGB will be used  
  
 if (argc == 7)  
 try {  
 gamma = std::stod(argv[6]);  
 } catch (std::invalid\_argument &e) {  
 std::cerr << "Cannot convert gamma from string to double\n";  
 return 1;  
 }  
  
 // Если что-то пошло не так:  
 if (gradient > 1 || dithering > 7 || bitness == 0 || bitness > 8 || gamma < 0) {  
 std::cerr << "Wrong arguments\n";  
 return 1;  
 }  
  
 // Считывание размеров из файла  
 char char\_header;  
 int width, height;  
 unsigned int max\_value;  
  
 FILE \*file\_in = fopen(file\_in\_name, "rb");  
  
 if (file\_in == nullptr) {  
 std::cerr << "Cannot open file to read: " << file\_in\_name << "\n";  
 return 1;  
 }  
  
 fscanf(file\_in, "P%c\n%i %i\n%i\n", &char\_header, &width, &height, &max\_value);  
  
 if (width <= 0 || height <= 0) {  
 std::cerr << "Wrong file attributes\n";  
 fclose(file\_in);  
 return 1;  
 }  
  
  
 // Открытие файла на выход и выделение памяти для данных  
  
 FILE \*file\_out = nullptr;  
 int k\_bytes = height \* width;  
  
 auto pix\_data = (unsigned char \*) calloc(k\_bytes, 1);  
 if (!pix\_data) {  
 std::cerr << "Cannot allocate " << k\_bytes << " bytes of memory\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 // Initialization  
 for (int i = 0; i < k\_bytes; ++i) {  
 pix\_data[i] = 0;  
 }  
  
 file\_out = fopen(file\_out\_name, "wb");  
 if (file\_out == nullptr) {  
 std::cerr << "Cannot open file to write: " << file\_out\_name << "\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 // Основная часть программы  
  
  
#ifndef ENABLE\_FILE\_INPUT  
 if (gradient == 0) {  
 std::cout << "File input disabled\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
#endif  
  
#ifdef ENABLE\_FILE\_INPUT  
  
 if (gradient == 0) {  
 // Читаем из файла  
  
 auto input\_pix\_data = (unsigned char \*) calloc(k\_bytes, 1);  
 if (!input\_pix\_data) {  
 std::cerr << "Cannot allocate " << k\_bytes << " bytes of memory\n";  
 free\_data(file\_in, file\_out, pix\_data);  
 return 1;  
 }  
  
 int bytes\_read = fread(input\_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, input\_pix\_data);  
 free(input\_pix\_data);  
 return 1;  
 }  
  
 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, input\_pix\_data);  
 free(input\_pix\_data);  
 return 1;  
 }  
  
 decode\_gamma\_from\_file(input\_pix\_data, width \* height, gamma);  
  
 if (dithering == 0) {  
 // no\_dithering  
  
 no\_dithering\_file(bitness, width \* height, input\_pix\_data);  
  
 for (int y = 0; y < height; ++y) {  
 for (int x = 0; x < width; ++x) {  
 draw\_pix(pix\_data, width, x, y,  
 change\_pix\_gamma\_to\_print(\*(input\_pix\_data + y \* width + x), gamma),  
 gamma);  
 }  
 }  
  
  
 } else if (dithering == 1) {  
 ordered\_dithering(width, height, pix\_data, gamma, bitness, input\_pix\_data);  
 } else if (dithering == 2) {  
 random\_dithering(width, height, pix\_data, gamma, bitness, input\_pix\_data);  
 } else if (dithering == 3) {  
 Floyd\_Steinberg\_dithering(width, height, pix\_data, gamma, bitness, input\_pix\_data);  
 } else if (dithering == 4) {  
 Jarvis\_Judice\_Ninke\_dithering(width, height, pix\_data, gamma, bitness, input\_pix\_data);  
 } else if (dithering == 5) {  
 Sierra\_3\_dithering(width, height, pix\_data, gamma, bitness, input\_pix\_data);  
 } else if (dithering == 6) {  
 Atkinson\_dithering(width, height, pix\_data, gamma, bitness, input\_pix\_data);  
 } else if (dithering == 7) {  
 Halftone\_dithering(width, height, pix\_data, gamma, bitness, input\_pix\_data);  
 }  
  
 free(input\_pix\_data);  
  
 } else {  
#endif  
 // Горизонтальный градиент  
 if (dithering == 0) {  
 no\_dithering(width, height, pix\_data, gamma, bitness);  
 } else if (dithering == 1) {  
 ordered\_dithering(width, height, pix\_data, gamma, bitness);  
 } else if (dithering == 2) {  
 random\_dithering(width, height, pix\_data, gamma, bitness);  
 } else if (dithering == 3) {  
 Floyd\_Steinberg\_dithering(width, height, pix\_data, gamma, bitness);  
 } else if (dithering == 4) {  
 Jarvis\_Judice\_Ninke\_dithering(width, height, pix\_data, gamma, bitness);  
 } else if (dithering == 5) {  
 Sierra\_3\_dithering(width, height, pix\_data, gamma, bitness);  
 } else if (dithering == 6) {  
 Atkinson\_dithering(width, height, pix\_data, gamma, bitness);  
 } else if (dithering == 7) {  
 Halftone\_dithering(width, height, pix\_data, gamma, bitness);  
 }  
  
#ifdef ENABLE\_FILE\_INPUT  
 }  
#endif  
  
 write\_to\_file(file\_out, char\_header, width, height, max\_value, pix\_data);  
  
  
#ifdef FILE\_OUTPUT  
 // Debug output  
 std::ofstream out("debug.txt");  
  
  
 for (int i = 0; i < width \* 1; ++i) {  
 out << (int) \*(pix\_data + i) << " ";  
 }  
#endif  
  
 free\_data(file\_in, file\_out, pix\_data);  
 return 0;  
}

Название файла: ./GeometryAndGraphics/Lab\_03/write\_data\_pnm.cpp

Исходный код:

//  
// Created by dudko on 11.03.2020.  
//  
  
#include "write\_data\_pnm.h"  
#include <cmath>  
  
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 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);  
}  
  
void free\_data(FILE \*file, unsigned char \*pix\_data) {  
 free\_data(nullptr, file, pix\_data);  
}  
  
void free\_data(FILE \*file) {  
 free\_data(file, nullptr);  
}  
  
void free\_data(unsigned char \*pix\_data) {  
 free\_data(nullptr, pix\_data);  
}  
  
void write\_to\_file(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value,  
 unsigned char \*pix\_data) {  
 write\_header(file\_out, char\_header, width, height, max\_value);  
 write\_data(file\_out, width \* height, pix\_data);  
}  
  
// brightness in [0..255] scale  
unsigned char limit\_brightness(double brightness) {  
 return (unsigned char) std::min(255.0, std::max(0.0, brightness));  
}  
  
double from\_sRGB(double \_brightness) {  
 if (\_brightness <= 0.0031308) {  
 return 323.0 \* \_brightness / 25.0;  
 } else {  
 return (211 \* pow(\_brightness, 5.0 / 12.0) - 11) / 200.0;  
 }  
}  
  
double from\_sRGB(int brightness) {  
 double \_brightness = brightness / 255.0;  
 return from\_sRGB(\_brightness);  
}  
  
double to\_sRGB(double \_brightness) {  
 if (\_brightness <= 0.04045) {  
 return 25.0 \* \_brightness / 323;  
 } else {  
 return pow((200 \* \_brightness + 11) / 211.0, 12.0 / 5.0);  
 }  
}  
  
double to\_sRGB(int brightness) {  
 double \_brightness = brightness / 255.0;  
 return to\_sRGB(\_brightness);  
}  
  
double change\_pix\_gamma\_to\_print(double \_brightness, double gamma) {  
 // Гамма коррекция:  
 if (gamma > 0) {  
 \_brightness = std::pow(\_brightness, 1.0 / gamma);  
 } else {  
 \_brightness = from\_sRGB(\_brightness);  
 }  
  
 return \_brightness;  
}  
  
double change\_pix\_gamma\_to\_print(unsigned char pix\_data, double gamma) {  
 double \_brightness = pix\_data / 255.0;  
 return change\_pix\_gamma\_to\_print(\_brightness, gamma);  
}  
  
double change\_pix\_gamma\_from\_file(double \_brightness, double gamma) {  
 if (gamma > 0) {  
 \_brightness = std::pow(\_brightness, gamma);  
 } else {  
 \_brightness = to\_sRGB(\_brightness);  
 }  
  
 return \_brightness;  
}  
  
double change\_pix\_gamma\_from\_file(unsigned char pix\_data, double gamma) {  
 double \_brightness = pix\_data / 255.0;  
 return change\_pix\_gamma\_from\_file(\_brightness, gamma);  
}  
  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, int brightness, double gamma) {  
 double \_brightness = brightness / 255.0;  
  
 // Гамма коррекция:  
 change\_pix\_gamma\_to\_print(\_brightness, gamma);  
  
 pix\_data[width \* y + x] = limit\_brightness(255 \* \_brightness);  
}  
  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, double \_brightness, double gamma) {  
 change\_pix\_gamma\_to\_print(\_brightness, gamma);  
 pix\_data[width \* y + x] = limit\_brightness(255 \* \_brightness);  
}  
  
void decode\_gamma\_from\_file(unsigned char \*pix\_data, int k, double gamma) {  
 for (int i = 0; i < k; ++i) {  
 \*(pix\_data + i) = limit\_brightness(255 \* change\_pix\_gamma\_from\_file(\*(pix\_data + i), gamma));  
 }  
}

Название файла: ./GeometryAndGraphics/Lab\_03/write\_data\_pnm.h

Исходный код:

//  
// Created by dudko on 11.03.2020.  
//  
  
#ifndef LAB\_03\_WRITE\_DATA\_PNM\_H  
#define LAB\_03\_WRITE\_DATA\_PNM\_H  
  
#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 free\_data(FILE \*file\_in, FILE \*file\_out, unsigned char \*pix\_data);  
void free\_data(FILE \*file, unsigned char \*pix\_data);  
void free\_data(FILE \*file);  
void free\_data(unsigned char \*pix\_data);  
void write\_to\_file(FILE \*file\_out, char char\_header, int width, int height, unsigned int max\_value, unsigned char \*pix\_data);  
  
unsigned char limit\_brightness(double brightness);  
  
double to\_sRGB(double \_brightness);  
double to\_sRGB(int brightness);  
double from\_sRGB(double \_brightness);  
double from\_sRGB(int brightness);  
  
double change\_pix\_gamma\_to\_print(double \_brightness, double gamma);  
double change\_pix\_gamma\_to\_print(unsigned char pix\_data, double gamma);  
  
double change\_pix\_gamma\_from\_file(double \_brightness, double gamma);  
double change\_pix\_gamma\_from\_file(unsigned char pix\_data, double gamma);  
  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, int brightness, double gamma);  
void draw\_pix(unsigned char \*pix\_data, int width, int x, int y, double \_brightness, double gamma);  
  
void decode\_gamma\_from\_file(unsigned char \*pix\_data, int k, double gamma);  
  
#endif //LAB\_03\_WRITE\_DATA\_PNM\_H