**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное

учреждение высшего образования

**Санкт-Петербургский исследовательский университет**

**Информационных технологий, механики и оптики**

Факультет информационных технологий и программирования

Дисциплина: Компьютерная геометрия и графика

**Отчет**

по лабораторной работе № 3

Изучение алгоритмов псевдотонирования изображений

Выполнила: студент гр. M3102

Муслимов О. С.

Преподаватель: Скаков П.С.

Санкт-Петербург

2020

**Цель работы:** изучить алгоритмы и реализовать программу, применяющий алгоритм 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

Примеры преобразования 2 (random) при разных значениях гаммы и битностей: [dither\_random](https://drive.google.com/drive/u/0/folders/1W9LYSeSapdiuFvu2atWtaJ5YvWRLFiHX)

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**

**Дизеринг (англ. *dither*), псевдотонирование** — при [обработке цифровых сигналов](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB%D0%BE%D0%B2) 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](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB%D0%BE%D0%B2)). В компьютерной графике дизеринг используется для создания иллюзии [глубины цвета](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D1%83%D0%B1%D0%B8%D0%BD%D0%B0_%D1%86%D0%B2%D0%B5%D1%82%D0%B0) для изображений с относительно небольшим количеством цветов в палитре. Отсутствующие цвета составляются из имеющихся путем их «перемешивания». Но перед описанием самих алгоритм определим, как именно будут определяться наши пороговые (threshold) цвета для наших битностей: 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 с распространением (рассеиванием) ошибок распределяют остаток квантования по соседним пикселям, которые еще не были обработаны. Данные алгоритмы работают с некоторой окрестностью пикселя: то, что алгоритм делает в одном месте, влияет на то, что происходит в других местах. Это означает, что требуется буферизация и усложняет параллельную обработку. Таким образом, алгоритмы с рассеянием ошибки действуют следующим образом:

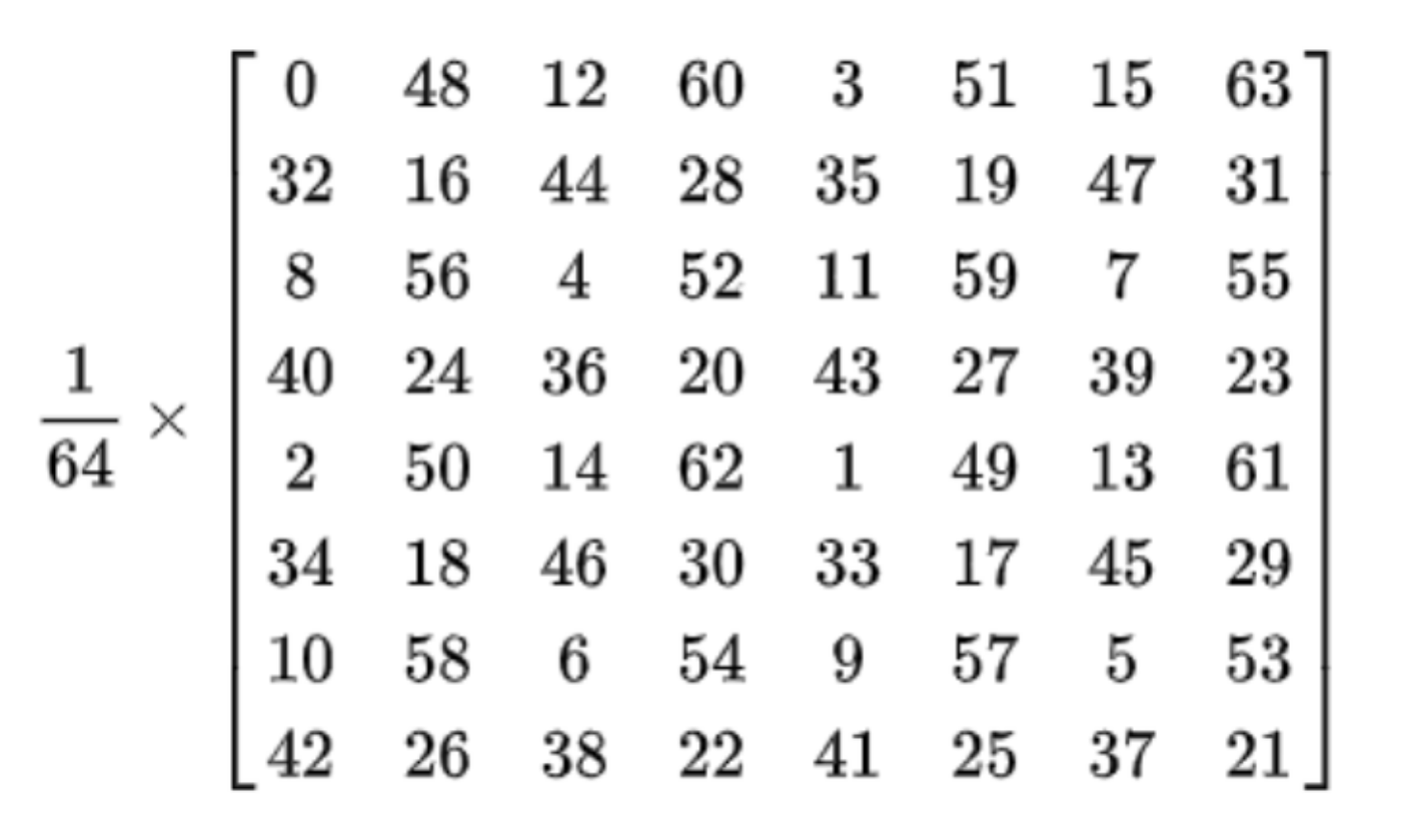
1. Обход по пикселям совершается слева направо, сверху вниз
2. При анализе текущего пикселя (color):
   1. определяется его новый цвет color' через findNearestPaletteColor
   2. рассчитывается ошибка как color-color'
   3. Затем ошибка распространяется согласно используемому паттерну/алгоритму (см. далее)(увеличиваем значение пикселя на ошибка \* значение в матрице)

Алгоритмы Дизеринга:

**Ordered:**

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.

Карта M:

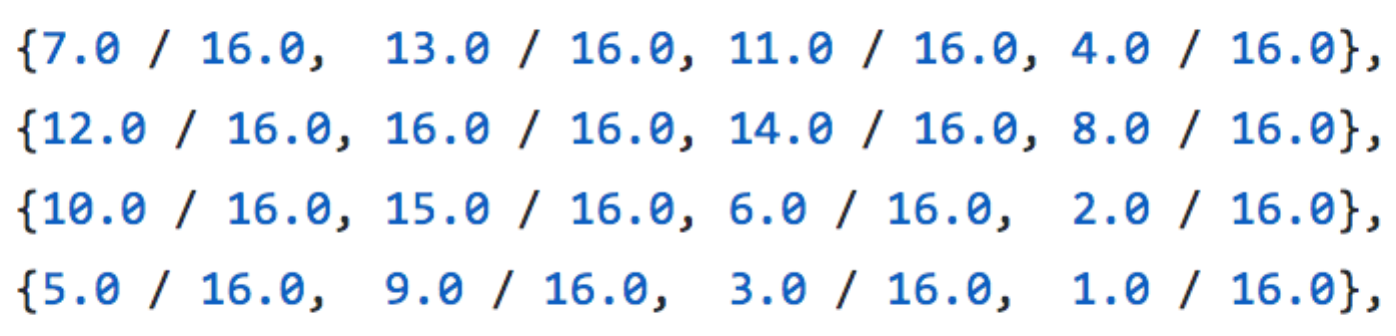


Алгоритм выполняет следующее преобразование для каждого цвета c каждого пикселя:

- функция, возвращающая ближайший цвет к подаваемому, который можно [отобразить на текущей палитре](https://docs.google.com/document/d/1h8c5CUP9d0wpbYDe9jGUukHes-LLGX-ZyAL7WXz43G8/edit#heading=h.rj4j1g3h32y8)

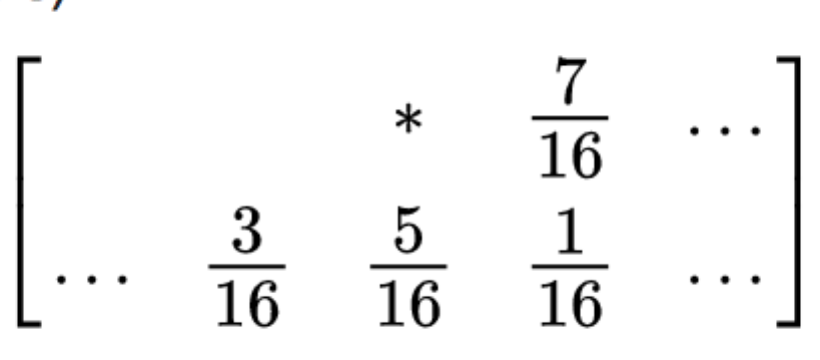
**Halftone 4x4 Ortogonal:**

Так же, как и Ordered, но с другой матрицей:



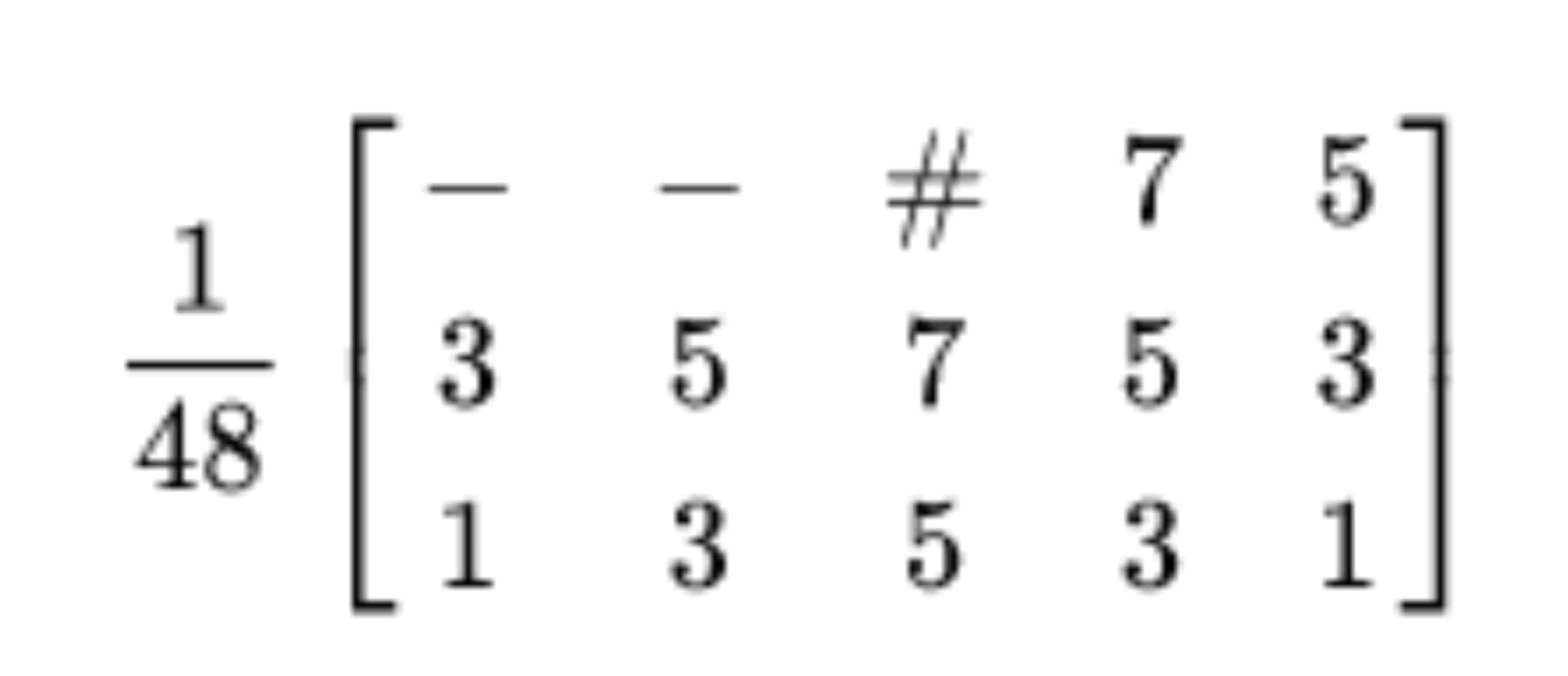
(Далее идут алгоритмы с распространением ошибок)

**Floyd-Steinberg:**

****

\* - текущая позиция

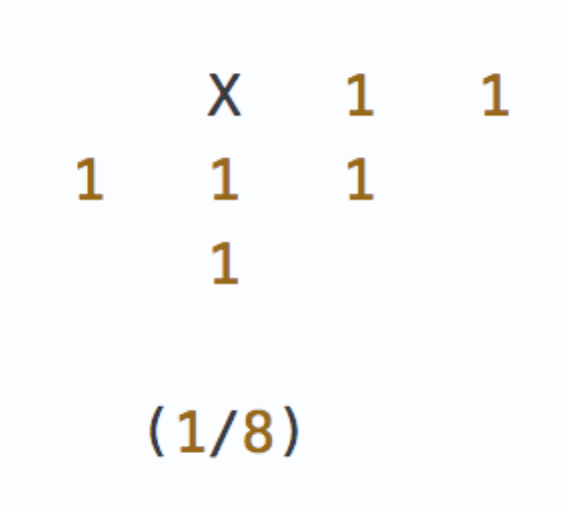
**Jarvis, Judice, Ninke:**



# - текущая позиция

При таком алгоритме ошибка распределяется в три раза больше пикселей, чем у Флойда-Стейнберга, что приводит к более гладкому и более тонкому результату.

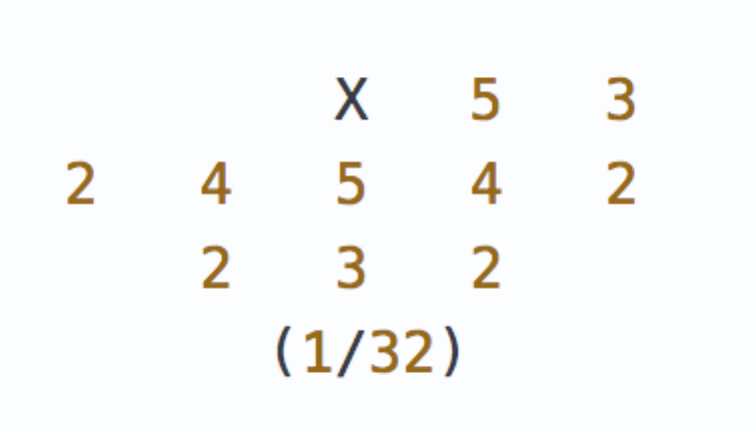
**Atkinson:**



X – текущий пиксель.  
**Random:**

Алгоритм такой же, как и у Ordered, но вместо значения матрицы берется рандомное число из диапазона (0;1]

**Sierra-3:**



Х – текущий пиксель

**Экспериментальная часть**

Язык программирования: С++20

Создаем палитру нужной нам битности. Если нам нужно преоброзование без дизеринга, то мы просто представляем значение пикселя в нужной нам битности. В алгоритмах без распространения ошибок мы действуем так, как указано выше, в теориотической части. В слуачае алгоритмов с распростронением ошибок мы увеличиваем все пиксели рядом (опираясь на матрицу) на <ошибка>\*<значение в матрице>. Также, есть класс CException, который фиксирует ошибки аргументов в консоли и ошибки, которые могут быть в файле.

**Выводы**

В ходе проделанной работы были изучены алгоритмы псведнотонирования изображения, а так же их применение в разных областях. Также получена программа, которая умеет псевдотонировать изображение, используя вышеперечисленные алгоритмы.

**Листинг**

**main.cpp**

|  |
| --- |
| #include <iostream> |
|  | #include "CExpension.h" |
|  | #include <string.h> |
|  | #include "CImage.h" |
|  |  |
|  | SInput input; |
|  |  |
|  | void getAnsw(const char \*fileName, CImage &output) { |
|  | FILE \*new\_f = fopen(fileName, "wb"); |
|  | if (!new\_f) { |
|  | throw CExpension("Output file didn't open", new\_f); |
|  | } |
|  | fprintf(new\_f, "P%i\n%i %i\n%i\n", 5, output.width, output.height, output.max\_val); |
|  | unsigned char \*buffer = new unsigned char[output.size]; |
|  | for (int i = 0; i < output.size; i++) { |
|  |  |
|  | buffer[i] = (unsigned char) output.pix[i]; |
|  | } |
|  | fwrite(buffer, sizeof(unsigned char), output.size, new\_f); |
|  | delete[] buffer; |
|  | fclose(new\_f); |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) { |
|  | try { |
|  | FILE \*f; |
|  | if (argc != 7) { |
|  | throw CExpension("Wrong amount of arguments"); |
|  | } |
|  | for (int i = 1; i < argc; i++) { |
|  | switch (i) { |
|  | case 1: |
|  | input.inputName = argv[i]; |
|  | f = fopen(input.inputName, "rb"); |
|  | if (!f) { |
|  | throw CExpension("Input File didn't open", f); |
|  | } |
|  | break; |
|  | case 2: |
|  | input.outputName = argv[i]; |
|  | break; |
|  | case 3: { |
|  | input.gradient = argv[i][0]; |
|  | break; |
|  | } |
|  | case 4: |
|  | input.dith = atoi(argv[i]); |
|  | if (input.dith < 0 || input.dith > 7) { |
|  | throw CExpension("Dither number is < 0 or > 7", f); |
|  | } |
|  | break; |
|  | case 5: |
|  | input.bit = atoi(argv[i]); |
|  | break; |
|  | case 6: |
|  | input.gamma = atof(argv[i]); |
|  | break; |
|  | } |
|  | } |
|  | CImage image(f, input); |
|  | image.ditherIt(input); |
|  | getAnsw(input.outputName, image); |
|  | return 0; |
|  | } catch (CExpension &expension) { |
|  | cerr << expension.getError(); |
|  | if (expension.getFile()) { |
|  | fclose(expension.getFile()); |
|  | } |
|  | exit(1); |
|  | } |
|  | } |

**Matrix.h**

|  |
| --- |
| #ifndef GRAPH3\_MATRIX\_H |
|  | #define GRAPH3\_MATRIX\_H |
|  | const double Bayer[8][8] = { |
|  | {0.0 / 64.0, 48.0 / 64.0, 12.0 / 64.0, 60.0 / 64.0, 3.0 / 64.0, 51.0 / 64.0, 15.0 / 64.0, 63.0 / 64.0}, |
|  | {32.0 / 64.0, 16.0 / 64.0, 44.0 / 64.0, 28.0 / 64.0, 35.0 / 64.0, 19.0 / 64.0, 47.0 / 64.0, 31.0 / 64.0}, |
|  | {8.0 / 64.0, 56.0 / 64.0, 4.0 / 64.0, 52.0 / 64.0, 11.0 / 64.0, 59.0 / 64.0, 7.0 / 64.0, 55.0 / 64.0}, |
|  | {40.0 / 64.0, 24.0 / 64.0, 36.0 / 64.0, 20.0 / 64.0, 43.0 / 64.0, 27.0 / 64.0, 39.0 / 64.0, 23.0 / 64.0}, |
|  | {2.0 / 64.0, 50.0 / 64.0, 14.0 / 64.0, 62.0 / 64.0, 1.0 / 64.0, 49.0 / 64.0, 13.0 / 64.0, 61.0 / 64.0}, |
|  | {34.0 / 64.0, 18.0 / 64.0, 46.0 / 64.0, 30.0 / 64.0, 33.0 / 64.0, 17.0 / 64.0, 45.0 / 64.0, 29.0 / 64.0}, |
|  | {10.0 / 64.0, 58.0 / 64.0, 6.0 / 64.0, 54.0 / 64.0, 9.0 / 64.0, 57.0 / 64.0, 5.0 / 64.0, 53.0 / 64.0}, |
|  | {42.0 / 64.0, 26.0 / 64.0, 38.0 / 64.0, 22.0 / 64.0, 41.0 / 64.0, 25.0 / 64.0, 37.0 / 64.0, 21.0 / 64.0}, |
|  | }; |
|  | const double Halftone[4][4] = { |
|  | {7.0 / 16.0, 13.0 / 16.0, 11.0 / 16.0, 4.0 / 16.0}, |
|  | {12.0 / 16.0, 16.0 / 16.0, 14.0 / 16.0, 8.0 / 16.0}, |
|  | {10.0 / 16.0, 15.0 / 16.0, 6.0 / 16.0, 2.0 / 16.0}, |
|  | {5.0 / 16.0, 9.0 / 16.0, 3.0 / 16.0, 1.0 / 16.0}, |
|  | }; |
|  | #endif //GRAPH3\_MATRIX\_H |

**CImage.h**

|  |
| --- |
| #ifndef GRAPH3\_CIMAGE\_H |
|  | #define GRAPH3\_CIMAGE\_H |
|  |  |
|  | #include <iostream> |
|  | #include "CExpension.h" |
|  | #include <cmath> |
|  | #include <string> |
|  | #include "Matrix.h" |
|  |  |
|  | struct SInput { |
|  | const char \*inputName; |
|  | const char \*outputName; |
|  | char gradient; |
|  | int dith; |
|  | int bit; |
|  | double gamma; |
|  | }; |
|  |  |
|  |  |
|  | class CImage { |
|  | public: |
|  | CImage(FILE \*f, SInput config); |
|  |  |
|  | void ditherIt(SInput config); |
|  |  |
|  | friend void getAnsw(const char \*fileName, CImage &output); |
|  |  |
|  |  |
|  | private: |
|  | int pallete[256]; |
|  | FILE \*file; |
|  | int version; |
|  | int width; |
|  | int height; |
|  | int max\_val; |
|  | int size; |
|  | double \*pix; |
|  |  |
|  | double Gamma(double value, double gamma); |
|  |  |
|  | double reverseGamma(double value, double gamma); |
|  |  |
|  | void WithoutDith(SInput config); |
|  |  |
|  | void Ordered8x8(SInput config); |
|  |  |
|  | void Random(SInput config); |
|  |  |
|  | void FloydSteinberg(SInput config); |
|  |  |
|  | void JJN(SInput config); |
|  |  |
|  | void Sierra(SInput config); |
|  |  |
|  | void Atkinson(SInput config); |
|  |  |
|  | void Halftone4x4orthogonal(SInput config); |
|  |  |
|  | void makeGradient(SInput config); |
|  |  |
|  | int newPix(int n, int bit); |
|  |  |
|  | int index(int x, int y); |
|  |  |
|  | int findNearestPalleteCollor(int value); |
|  |  |
|  | }; |
|  |  |
|  |  |
|  | #endif //GRAPH3\_CIMAGE\_H |

**CImage.cpp**

|  |
| --- |
| #include "CImage.h" |
|  |  |
|  | CImage::CImage(FILE \*f, SInput config) { |
|  | file = f; |
|  | if (fscanf(f, "P%i%i%i%i\n", &this->version, &this->width, &this->height, &max\_val) != 4) { |
|  | throw CExpension("Wrong amount data in file", f); |
|  | } |
|  | if (version != 5) { |
|  | throw CExpension("Wrong version"); |
|  | } |
|  | size = width \* height; |
|  | unsigned char \*buffer = new unsigned char[size]; |
|  | pix = new double[size]; |
|  | fread(buffer, sizeof(unsigned char), size, f); |
|  | for (int i = 0; i < size; i++) { |
|  | pix[i] = (double) buffer[i]; |
|  | } |
|  | for (int i = 0; i < 256; i++) { |
|  | pallete[i] = newPix(i, config.bit); |
|  | } |
|  | delete[] buffer; |
|  | fclose(f); |
|  | } |
|  |  |
|  | void CImage::ditherIt(SInput config) { |
|  | if (config.gradient == '1') { |
|  | makeGradient(config); |
|  | } |
|  | for (int i = 0; i < size; i++) { |
|  | pix[i] = Gamma(pix[i], config.gamma); |
|  | } |
|  | switch (config.dith) { |
|  | case 0: |
|  | WithoutDith(config); |
|  | break; |
|  | case 1: |
|  | Ordered8x8(config); |
|  | break; |
|  | case 2: |
|  | Random(config); |
|  | break; |
|  | case 3: |
|  | FloydSteinberg(config); |
|  | break; |
|  | case 4: |
|  | JJN(config); |
|  | break; |
|  | case 5: |
|  | Sierra(config); |
|  | break; |
|  | case 6: |
|  | Atkinson(config); |
|  | break; |
|  | case 7: |
|  | Halftone4x4orthogonal(config); |
|  | break; |
|  | } |
|  | for (int i = 0; i < size; i++) { |
|  | pix[i] = reverseGamma(pix[i], config.gamma); |
|  | } |
|  | } |
|  |  |
|  | void CImage::makeGradient(SInput config) { |
|  | for (int i = 0; i < width; i++) { |
|  | double newColor = 255.0 \* (double) i / (width - 1); |
|  | for (int j = 0; j < height; j++) { |
|  | pix[j \* width + i] = newColor; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | double CImage::Gamma(double value, double gamma) { |
|  | value /= 255.0; |
|  | if (gamma == 0.0) { |
|  | double a = 0.055; |
|  | if (value <= 0.04045) { |
|  | return 255.0 \* value / 12.92; |
|  | } |
|  | return 255.0 \* pow((value + a) / (1.0 + a), 2.4); |
|  | } |
|  | return 255.0 \* pow(value, gamma); |
|  | } |
|  |  |
|  | double CImage::reverseGamma(double value, double gamma) { |
|  | value /= 255.0; |
|  | if (gamma == 0.0) { |
|  | double a = 0.055; |
|  | if (value <= 0.0031308) { |
|  | return 255.0 \* value \* 12.92; |
|  | } |
|  | return 255.0 \* ((1.0 + a) \* pow(value, (double) (1 / 2.4)) - a); |
|  | } |
|  | return 255.0 \* pow(value, pow(gamma, -1)); |
|  | } |
|  |  |
|  | void CImage::WithoutDith(SInput config) { |
|  | for (int i = 0; i < size; i++) { |
|  | pix[i] = newPix((int) pix[i], config.bit); |
|  | } |
|  | } |
|  |  |
|  | void CImage::Ordered8x8(SInput config) { |
|  | for (int i = 0; i < width; i++) { |
|  | for (int j = 0; j < height; j++) { |
|  | int clr = (int) pix[j \* width + i]; |
|  | clr = clr + 255.0 \* (Bayer[i % 8][j % 8] - 0.5); |
|  | pix[j \* width + i] = findNearestPalleteCollor(clr); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void CImage::Random(SInput config) { |
|  | for (int i = 0; i < width; i++) { |
|  | for (int j = 0; j < height; j++) { |
|  | double rndm = (double)rand() / RAND\_MAX; |
|  | int clr = pix[index(i, j)]; |
|  | clr = clr + 255.0 \* (double) (rndm - 0.5); |
|  | pix[j \* width + i] = findNearestPalleteCollor(clr); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void CImage::FloydSteinberg(SInput config) { |
|  | for (int j = 0; j < height; j++) { |
|  | for (int i = 0; i < width; i++) { |
|  | double oldPixel = pix[index(i, j)]; |
|  | double newPixel = findNearestPalleteCollor((int) oldPixel); |
|  | double error = oldPixel - newPixel; |
|  | pix[index(i, j)] = newPixel; |
|  | if (i + 1 < width) { |
|  | pix[index(i + 1, j)] += error \* (7.0 / 16.0); |
|  | } |
|  | if (i - 1 >= 0 && j + 1 < height) { |
|  | pix[index(i - 1, j + 1)] += error \* (3.0 / 16.0); |
|  | } |
|  | if (j + 1 < height) { |
|  | pix[index(i, j + 1)] += error \* (5.0 / 16.0); |
|  | } |
|  | if (i + 1 < width && j + 1 < height) { |
|  | pix[index(i + 1, j + 1)] += error \* (1.0 / 16.0); |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void CImage::JJN(SInput config) { |
|  | for (int j = 0; j < height; j++) { |
|  | for (int i = 0; i < width; i++) { |
|  | double oldPixel = pix[index(i, j)]; |
|  | double newPixel = findNearestPalleteCollor((int) oldPixel); |
|  | double error = oldPixel - newPixel; |
|  | pix[index(i, j)] = newPixel; |
|  | if (i + 1 < width) { |
|  | pix[index(i + 1, j)] += error \* (7.0 / 48.0); |
|  | } |
|  | if (i + 2 < width) { |
|  | pix[index(i + 2, j)] += error \* (5.0 / 48.0); |
|  | } |
|  | if (j + 1 < height) { |
|  | if (i - 2 >= 0) { |
|  | pix[index(i - 2, j + 1)] += error \* (3.0 / 48.0); |
|  | } |
|  | if (i - 1 >= 0) { |
|  | pix[index(i - 1, j + 1)] += error \* (5.0 / 48.0); |
|  | } |
|  | pix[index(i, j + 1)] += error \* (7.0 / 48.0); |
|  | if (i + 1 < width) { |
|  | pix[index(i + 1, j + 1)] += error \* (5.0 / 48.0); |
|  | } |
|  | if (i + 2 < width) { |
|  | pix[index(i + 2, j + 1)] += error \* (3.0 / 48.0); |
|  | } |
|  | } |
|  | if (j + 2 < height) { |
|  | if (i - 2 >= 0) { |
|  | pix[index(i - 2, j + 2)] += error \* (1.0 / 48.0); |
|  | } |
|  | if (i - 1 >= 0) { |
|  | pix[index(i - 1, j + 2)] += error \* (3.0 / 48.0); |
|  | } |
|  | pix[index(i, j + 2)] += error \* (5.0 / 48.0); |
|  | if (i + 1 < width) { |
|  | pix[index(i + 1, j + 2)] += error \* (3.0 / 48.0); |
|  | } |
|  | if (i + 2 < width) { |
|  | pix[index(i + 2, j + 2)] += error \* (1.0 / 48.0); |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void CImage::Sierra(SInput config) { |
|  | for (int j = 0; j < height; j++) { |
|  | for (int i = 0; i < width; i++) { |
|  | double oldPixel = pix[index(i, j)]; |
|  | double newPixel = findNearestPalleteCollor((int) oldPixel); |
|  | double error = oldPixel - newPixel; |
|  | pix[index(i, j)] = newPixel; |
|  | if (i + 1 < width) { |
|  | pix[index(i + 1, j)] += error \* (5.0 / 32.0); |
|  | } |
|  | if (i + 2 < width) { |
|  | pix[index(i + 2, j)] += error \* (3.0 / 32.0); |
|  | } |
|  | if (j + 1 < height) { |
|  | if (i - 2 >= 0) { |
|  | pix[index(i - 2, j + 1)] += error \* (2.0 / 32.0); |
|  | } |
|  | if (i - 1 >= 0) { |
|  | pix[index(i - 1, j + 1)] += error \* (4.0 / 32.0); |
|  | } |
|  | pix[index(i, j + 1)] += error \* (5.0 / 32.0); |
|  | if (i + 1 < width) { |
|  | pix[index(i + 1, j + 1)] += error \* (4.0 / 32.0); |
|  | } |
|  | if (i + 2 < width) { |
|  | pix[index(i + 2, j + 1)] += error \* (2.0 / 32.0); |
|  | } |
|  | } |
|  | if (j + 2 < height) { |
|  | if (i - 1 >= 0) { |
|  | pix[index(i - 1, j + 2)] += error \* (2.0 / 32.0); |
|  | } |
|  | pix[index(i, j + 2)] += error \* (3.0 / 32.0); |
|  | if (i + 1 < width) { |
|  | pix[index(i + 1, j + 2)] += error \* (2.0 / 32.0); |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void CImage::Atkinson(SInput config) { |
|  | for (int j = 0; j < height; j++) { |
|  | for (int i = 0; i < width; i++) { |
|  | double oldPixel = pix[index(i, j)]; |
|  | double newPixel = findNearestPalleteCollor((int) oldPixel); |
|  | double error = oldPixel - newPixel; |
|  | pix[index(i, j)] = newPixel; |
|  | if (i + 1 < width) { |
|  | pix[index(i + 1, j)] += error \* (1.0 / 8.0); |
|  | } |
|  | if (i + 2 < width) { |
|  | pix[index(i + 2, j)] += error \* (1.0 / 8.0); |
|  | } |
|  | if (j + 1 < height) { |
|  | if (i - 1 >= 0) { |
|  | pix[index(i - 1, j + 1)] += error \* (1.0 / 8.0); |
|  | } |
|  | pix[index(i, j + 1)] += error \* (1.0 / 8.0); |
|  | if (i + 1 < width) { |
|  | pix[index(i + 1, j + 1)] += error \* (1.0 / 8.0); |
|  | } |
|  | } |
|  | if (j + 2 < height) { |
|  | pix[index(i, j + 2)] += error \* (1.0 / 8.0); |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void CImage::Halftone4x4orthogonal(SInput config) { |
|  | for (int i = 0; i < width; i++) { |
|  | for (int j = 0; j < height; j++) { |
|  | int clr = (int) pix[j \* width + i]; |
|  | clr = clr + 255.0 \* (Halftone[i % 4][j % 4] - 0.5); |
|  | pix[j \* width + i] = findNearestPalleteCollor(clr); |
|  | } |
|  | } |
|  |  |
|  | } |
|  |  |
|  | int CImage::newPix(int n, int bit) { |
|  | if (bit == 8) { |
|  | return n; |
|  | } |
|  | int tmp = n >> (8 - bit); |
|  | int result = 0; |
|  | switch (bit) { |
|  | case 1: |
|  | for (int i = 0; i < 7; i++) { |
|  | result |= tmp; |
|  | result = result << 1; |
|  | } |
|  | return result | tmp; |
|  | case 2: |
|  | for (int i = 0; i < 3; i++) { |
|  | result |= tmp; |
|  | result = result << 2; |
|  | } |
|  | return result | tmp; |
|  | case 3: |
|  | for (int i = 0; i < 2; i++) { |
|  | result |= tmp; |
|  | result = result << 3; |
|  | } |
|  | result >>= 3; |
|  | result = result << 2; |
|  | tmp = tmp >> 1; |
|  | return result | tmp; |
|  | case 4: |
|  | for (int i = 0; i < 2; i++) { |
|  | result |= tmp; |
|  | result = result << 4; |
|  | } |
|  | return result >> 4; |
|  | case 5: |
|  | result |= tmp; |
|  | result = result << 3; |
|  | tmp = tmp >> 2; |
|  | return result | tmp; |
|  | case 6: |
|  | result |= tmp; |
|  | result = result << 2; |
|  | tmp = tmp >> 4; |
|  | return result | tmp; |
|  | case 7: |
|  | result |= tmp; |
|  | result = result << 1; |
|  | tmp = tmp >> 6; |
|  | return result | tmp; |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  | int CImage::findNearestPalleteCollor(int value) { |
|  | int min = 256; |
|  | int result; |
|  | for (int i = 0; i < 256; i++) { |
|  | if (min > abs(value - pallete[i])) { |
|  | min = abs(value - pallete[i]); |
|  | result = pallete[i]; |
|  | } |
|  | } |
|  | return result; |
|  | } |
|  |  |
|  | int CImage::index(int x, int y) { |
|  | return y \* width + x; |
|  | } |

**CException.h**

|  |
| --- |
| #ifndef GRAPH3\_CEXPENSION\_H |
|  | #define GRAPH3\_CEXPENSION\_H |
|  |  |
|  | #include <string> |
|  |  |
|  | using namespace std; |
|  |  |
|  | class CExpension { |
|  | public: |
|  | CExpension(string error); |
|  |  |
|  | CExpension(string error, FILE \*file); |
|  |  |
|  | string getError(); |
|  |  |
|  | FILE \*getFile(); |
|  |  |
|  | private: |
|  | string error\_; |
|  | FILE \*file\_ = nullptr; |
|  | }; |
|  |  |
|  |  |
|  | #endif //GRAPH3\_CEXPENSION\_H |

**CException.cpp**

|  |
| --- |
| #include "CExpension.h" |
|  |  |
|  | CExpension::CExpension(string error) { |
|  | this->error\_ = error; |
|  | } |
|  |  |
|  | CExpension::CExpension(string error, FILE \*file) { |
|  | this->error\_ = error; |
|  | this->file\_ = file; |
|  | } |
|  |  |
|  | string CExpension::getError() { |
|  | return this->error\_; |
|  | } |
|  |  |
|  | FILE \*CExpension::getFile() { |
|  | return this->file\_; |
|  | } |