

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»

Институт компьютерных наук
Кафедра инженерной кибернетики

Курсовая работа

по дисциплине «Методы и средства обработки изображений»
на тему
«Дизеринг черно-белых изображений»

Выполнила:
студент(ка) 3-го курса,
гр. БПМ-21-3 Ишкуватова М.И.

Проверил:
доцент, к.т.н. Полевой Д.В.

Москва, 2024

СОДЕРЖАНИЕ

1 Техническое задание	2
2 Математическое описание	3
2.1 Метод Байера	3
2.2 Метод случайного дизеринга	3
2.3 Метод Флойда-Стейнберга	4
2.4 Метод дизеринга с синим шумом	4
3 Пользовательская инструкция	5
4 Техническая документация	7
4.1 Инструкция по сборке	7
4.2 Формат данных	8
5 Документация	8
6 Пример работы программы	11
7 Источники	14

1 Техническое задание

Получить функцию для дизеринга черно-белых фотографий различными методами, с реализацией на C++, для того чтобы пользователь смог обработать фотографию.

Функция должна включать в себя:

- Метод упорядоченного дизеринга (Метод Байера)
- Метод случайного дизеринга
- Метод Флойда-Стейнберга
- Метод дизеринга с синим шумом

2 Математическое описание

2.1 Метод Байера

Алгоритм уменьшает количество цветов, применяя к отображаемым пикселям матрицу Байера, в результате чего некоторые пиксели меняют цвет в зависимости от удаленности исходного цвета от доступных цветовых записей в уменьшенной палитре.

В нашем случае нормализованная матрица Байера имеет вид:

$$\mathbf{M} = \frac{1}{16} \times \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$$

Она нормализованная, поскольку мы поделили каждый элемент на n^2 , где n - количество строк/столбцов.

Далее мы рассчитываем новые значения пикселей по данной формуле:

$$pixel' = 255, \text{ если } pixel > M(x \bmod n, y \bmod n)$$

$$pixel' = 0, \text{ если } pixel \leq M(x \bmod n, y \bmod n)$$

2.2 Метод случайного дизеринга

Вместо того чтобы квантовать каждый пиксель напрямую, мы добавляем к пикселям шум, значения которого находятся между -0.5 и 0.5. Идея тут в том, что некоторые пиксели теперь будут квантоваться к «неправильным» цветам, но то, как часто это происходит, зависит от изначальной светлоты пикселя. Чёрные пиксели всегда остаются чёрными, белые всегда остаются белыми, а средне-серые будут, примерно в 50% случаев, оказываться чёрными.

Новые значения считаются по формуле:

$$pixel' = 255, \text{ если } \frac{pixel}{255} + noise > 0.5$$

$$pixel' = 0, \text{ если } \frac{pixel}{255} + noise \leq 0.5$$

Шум при этом равен:

$$noise = \frac{random}{RAND_MAX} - 0.5$$

2.3 Метод Флойда-Стейнберга

Алгоритм Флойда-Стейнберга использует матрицу рассеяния ошибок. Числа в этой матрице тщательно подобраны для того чтобы как можно сильнее уменьшить возможность образования повторяющихся паттернов.

Матрица рассеяния ошибки Роберта У. Флойда и Луиса Стейнберга:

$$\frac{1}{16} \cdot \begin{pmatrix} & * & 7 \\ 3 & 5 & 1 \end{pmatrix}$$

Далее мы пересчитываем новое значение в текущем пикселе:

$$pixel' = 255, \text{ если } pixel > 128$$

$$pixel' = 0, \text{ если } pixel \leq 128$$

Считаем ошибку в пикселе:

$$error = pixel - pixel'$$

И в соседних пикселях. Для этого прибавляем к соседним пикселям значения из матрицы рассеяния, умноженные на *error*.

$$pixel(y, x + 1) = pixel(y, x + 1) + error * \frac{7}{16}$$

$$pixel(y + 1, x - 1) = pixel(y + 1, x - 1) + error * \frac{3}{16}$$

$$pixel(y + 1, x) = pixel(y + 1, x) + error * \frac{5}{16}$$

$$pixel(y + 1, x + 1) = pixel(y + 1, x + 1) + error * \frac{1}{16}$$

2.4 Метод дизеринга с синим шумом

В данном методе мы используем сгенерированный синий шум (blueNoise), размером 64*64. (матрица заполняется случайными числами, после чего происходит размытие по Гауссу и нормализация)

Далее считаем яркость каждого пикселя:

$$brightness = \frac{pixel}{255}$$

Далее мы считаем шум в пикселе:

$$noise = blueNoise(y \bmod blueNoise.rows, x \bmod blueNoise.cols)$$

Ну и наконец, пересчитываем значение в пикселе:

$pixel' = 255$, если $(brightness + noise - 0.5) > 0.5$

$pixel' = 0$, если $(brightness + noise - 0.5) \leq 0.5$

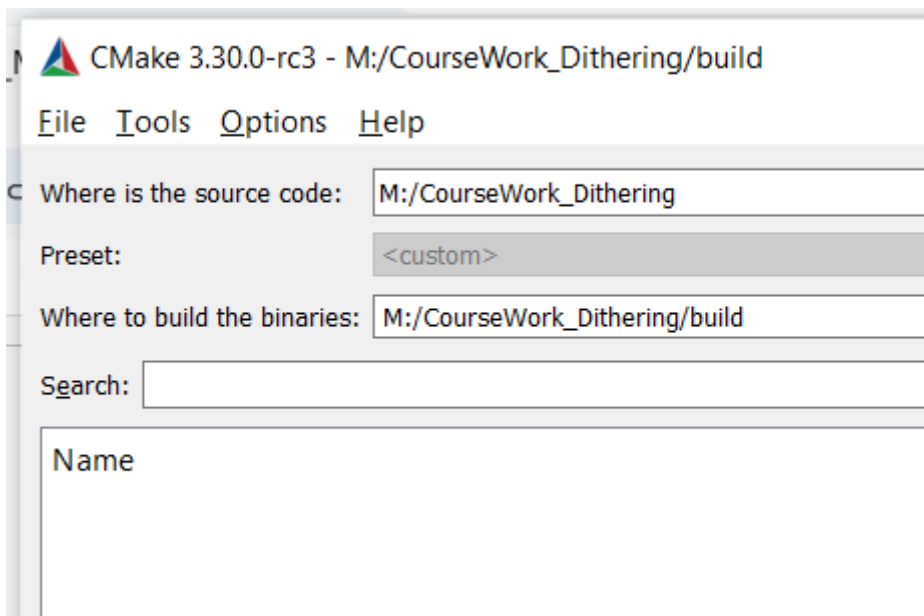
3 Пользовательская инструкция

Требования для сборки и использования проекта:

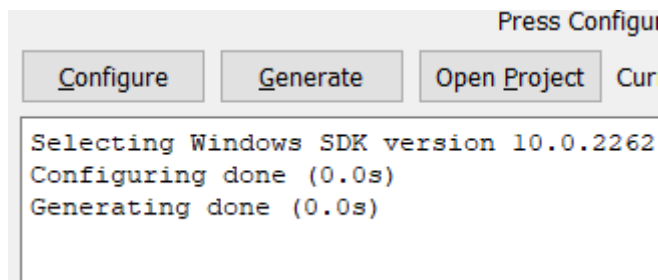
1. Cmake минимальной версии 3.30.
2. Язык C++ стандарта 17.
3. OpenCV минимальной версии 4.10.
4. Visual Studio 17.

Установка и запуск программы:

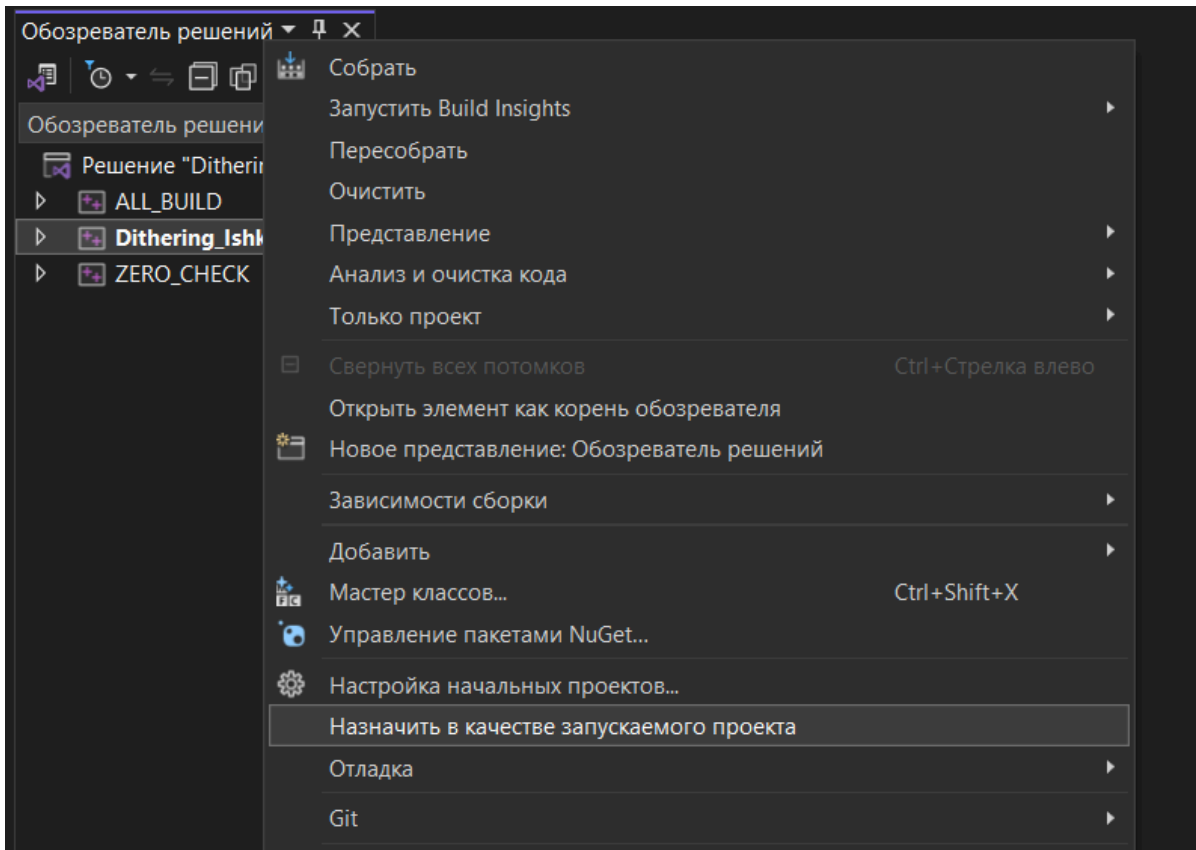
1. Выгрузите репозиторий: https://github.com/mellonii/CourseWork_Dithering
2. Соберите и установите программу с помощью Cmake, используя Visual Studio 17 2022 в качестве генератора



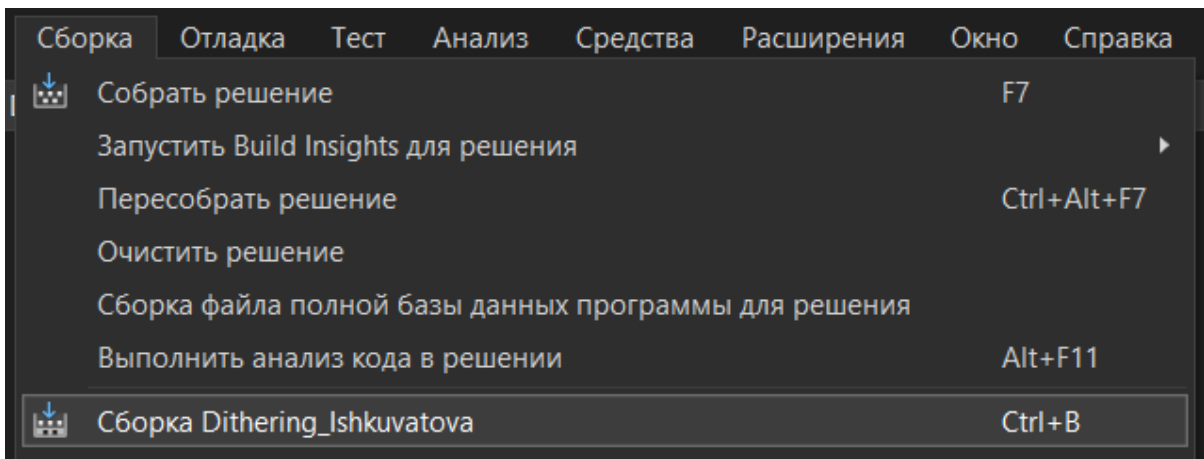
3. Нажмите на кнопки Configure, Generate, Open Project по очереди



4. Далее нажмите правой кнопкой мышки на проект Dithering_Ishkuvatova, и установите его в качестве запускаемого



5. Соберите проект



6. Запустите проект, нажав на зеленую стрелочку

Использование функции

Укажите путь к вашей фотографии. (Предварительно загрузите ее в папку с кодом)

```
std::string imagePath = "example.jpg";
```

Выберите необходимый метод.

- 1: "Bayer Dithering"
- 2: "Random Dithering"
- 3: "Floyd Steinberg Dithering"
- 4: "Blue Noise Dithering"
- 5: все методы

```
int method = 4;
```

Передайте параметры в функцию.

```
Dithering(imagePath, method);
```

Запустите код.

4 Техническая документация

4.1 Инструкция по сборке

Требования для сборки и использования проекта:

1. Сmake минимальной версии 3.30.
2. Язык C++ стандарта 17.
3. OpenCV минимальной версии 4.10.

Репозиторий проекта на GitHub:

https://github.com/mellonii/CourseWork_Dithering

1. Клонировать репозиторий в любую директорию

2. Соберите программу с помощью Stake

4.2 Формат данных

Входящие фотографии должны быть определенного формата:

- JPEG: .jpg, .jpeg
- PNG: .png
- BMP: .bmp
- TIFF: .tiff, .tif
- PPM, PGM, PBM: .ppm, .pgm, .pbm
- JPEG 2000: .jp2
- WebP: .webp
- GIF: .gif (возьмется первое изображение в GIF-анимации)

5 Документация

Функция Dithering

Главная функция, принимающая на себя роль навигации.

```
void Dithering(const std::string imagePath, const int method)
```

Входные данные

- const std::string imagePath - путь до тестовой картинки
- const int method - номер метода
 - 1: "Bayer Dithering"
 - 2: "Random Dithering"
 - 3: "Floyd Steinberg Dithering"
 - 4: "Blue Noise Dithering"
 - 5: все методы

Используемые переменные

result	cv::Mat	Картинка, полученная после дизеринга
--------	---------	--------------------------------------

Функция **Bayer_dithering**

Функция, реализующая метод Байера

cv::Mat Bayer_dithering(cv::Mat image)

Входные данные

- cv::Mat image - тестовая картинка

Выходные данные

- cv::Mat result - картинка, после дизеринга

Используемые переменные

bw_image	cv::Mat	Черно-белая версия изображения
bayerMatrix	int[][]	Матрица Байера размером 4x4
ditherMatrix	cv::Mat	Нормализованная матрица Байера размером 4x4
pixelValue	uchar	Значение в пикселе (y, x) входной фотографии
ditherValue	uchar	Значение нормализованной матрицы Байера, соответствующее пикселю (y, x)

Функция **Random_dithering**

Функция, реализующая метод случайного дизеринга

cv::Mat Random_dithering(cv::Mat image)

Входные данные

- cv::Mat image - тестовая картинка

Выходные данные

- cv::Mat result - картинка, после дизеринга

Используемые переменные

bw_image	cv::Mat	Черно-белая версия изображения
brightness	uchar	Яркость пикселя
noise	float	Случайный шум в диапазоне [-0.5, 0.5]
newPixel	uchar	Новое значение в пикселе

Функция FloydSteinberg_method

Функция, реализующая дизеринг методом Флойда-Стейнберга

cv::Mat FloydSteinberg_method(cv::Mat image)

Входные данные

- cv::Mat image - тестовая картинка

Выходные данные

- cv::Mat result - картинка, после дизеринга

Используемые переменные

bw_image	cv::Mat	Черно-белая версия изображения
oldPixel	uchar	Значение в пикселе (y, x) входной фотографии
newPixel	uchar	Новое значение в пикселе (y, x)
error	int	Ошибка в пикселе

Функция BlueNoise_dithering

Функция, реализующая метод дизеринга с синим шумом

cv::Mat BlueNoise_dithering(cv::Mat image)

Входные данные

- cv::Mat image - тестовая картинка

Выходные данные

- `cv::Mat result` - картинка, после дизеринга

Используемые переменные

<code>bw_image</code>	<code>cv::Mat</code>	Черно-белая версия изображения
<code>blueNoise</code>	<code>cv::Mat</code>	Карта синего шума размером 64x64
<code>brightness</code>	<code>uchar</code>	Яркость пикселя
<code>noise</code>	<code>float</code>	Значение шума из карты синего шума
<code>newPixel</code>	<code>uchar</code>	Новое значение в пикселе

Функция `generateBlueNoise`

Функция, генерирующая карту синего шума

`cv::Mat generateBlueNoise(int width, int height)`

Входные данные

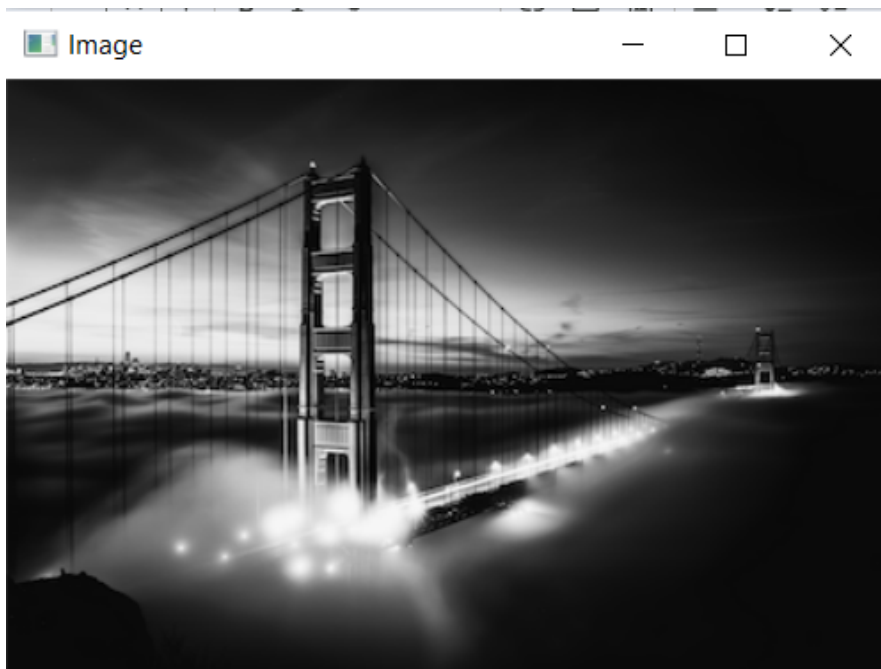
- `int width` - ширина карты
- `int height` - высота карты

Выходные данные

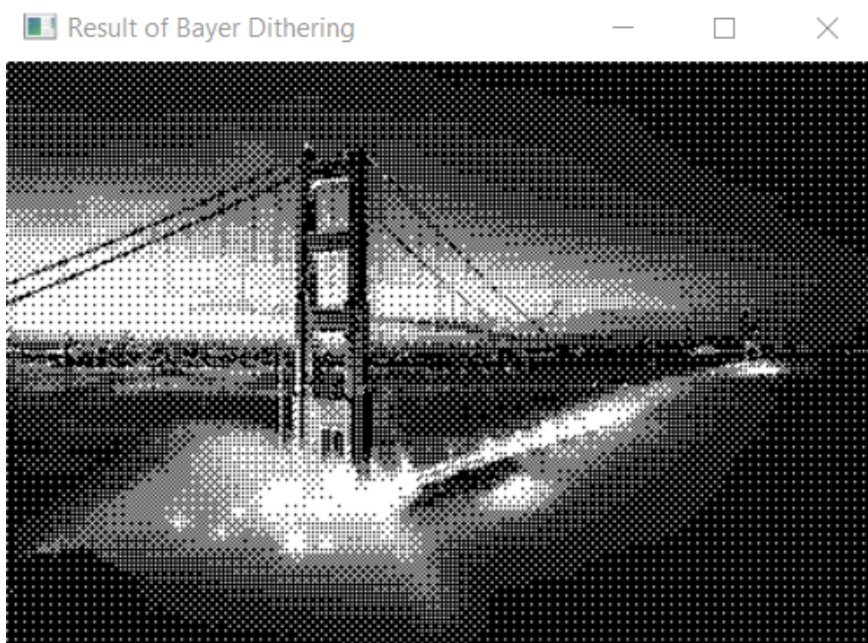
- `cv::Mat blueNoise` - карта синего шума

6 Пример работы программы

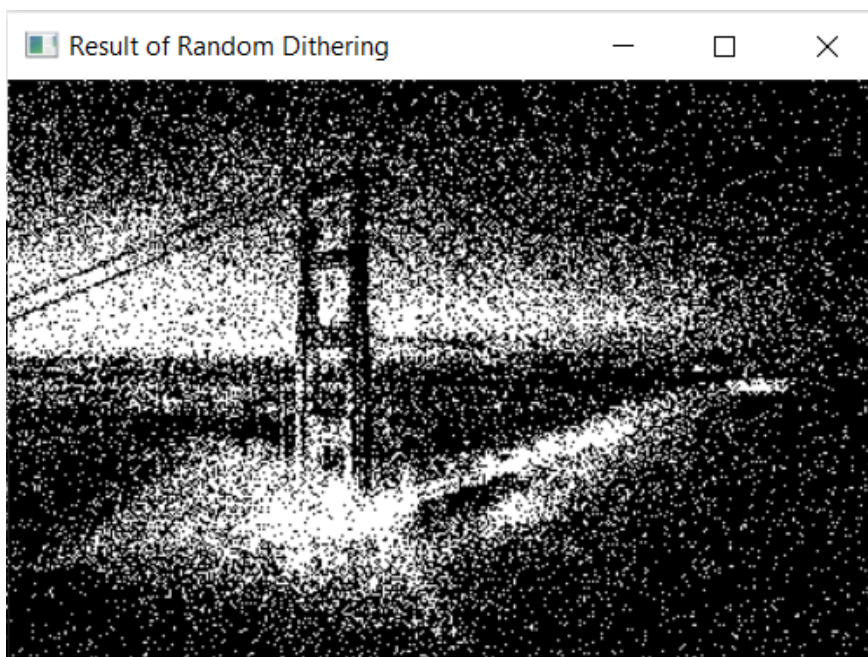
Оригинальное изображение:



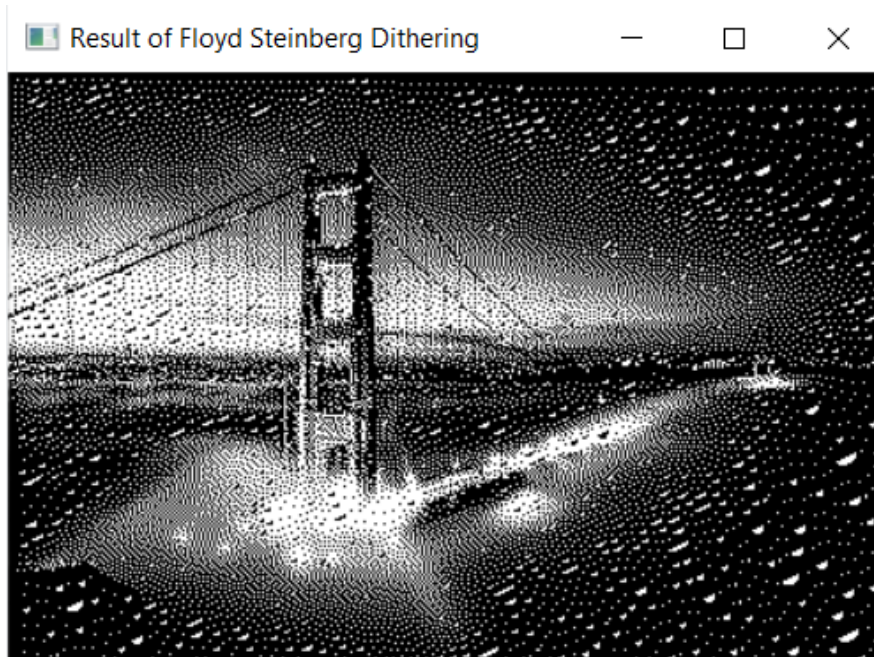
Метод упорядоченного дизеринга (Метода Байера)



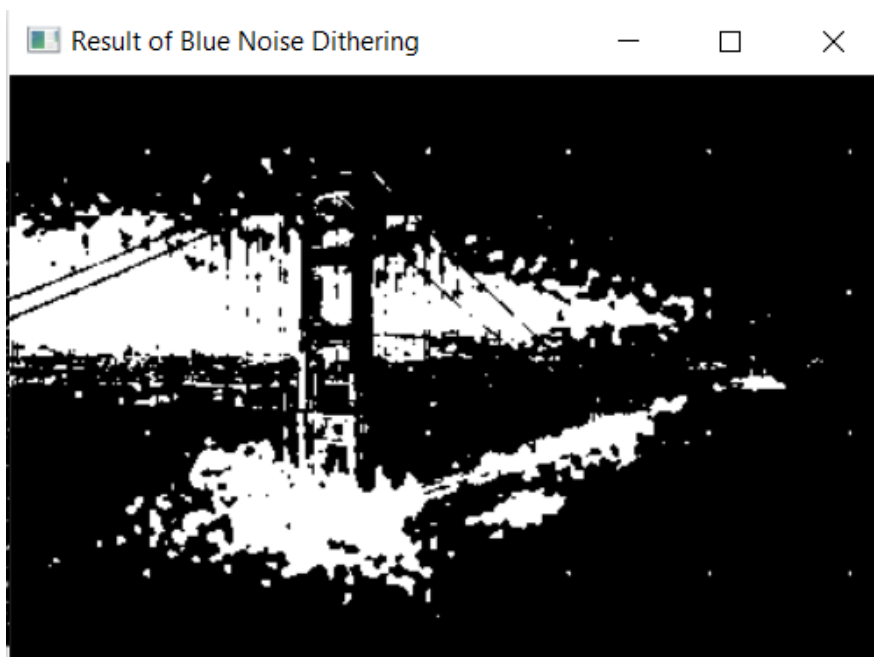
Метод случайного дизеринга



Метод Флойда-Стейнберга



Метод дизеринга с синим шумом



7 Источники

- <https://habr.com/ru/companies/wunderfund/articles/680154/>
- <https://forums.tigsource.com/index.php?topic=40832.msg1363742#msg1363742>
- <https://surma.dev/things/ditherpunk/>