

Техническое задание

Разработать программу для преобразования высококачественного 16,32 битного изображения к 8 битному rgba изображению. Изображение для конвертации задается пользователем. Применяя полученные наработки, реализовать рисование градиента с устранением ступенчатости и быструю компьютерную кисть. Исследовать и выбрать оптимальные для этого алгоритмы, учитывая качество получаемого изображения и скорость работы.

Содержание

Введение	6
1 Аналитический раздел	7
1.1 Случайный дизеринг	7
1.2 Шаблонный дизеринг	7
1.3 Упорядоченный дизеринг	8
1.4 Дизеринг при помощи диффузии ошибок	9
1.5 Вариации алгоритма дизеринга при помощи диффузии ошибок	9
1.5.1 Фильтр Флойда-Стейнберга	9
1.5.2 "Ложный" фильтр Флойда-Стейнберга	9
1.5.3 Фильтр Джарвиса, Джунка и Нинка	10
1.5.4 Фильтр Стаки	10
1.5.5 Фильтр Бурка	10
1.6 Выбор оптимального класса алгоритма	10
2 Конструкторский раздел	12
2.1 Оценка качества изображений	12
2.1.1 PSNR	12
2.1.2 SSIM	12
2.1.3 Алгоритм случайного распределения	13
2.2 Виды случайных распределений	13
2.2.1 Белый шум	13
2.2.2 Коричневый шум	14
2.2.3 Гауссовский шум	15
2.2.4 Фиолетовый шум	15
2.2.5 Розовый и синий шумы	16
2.3 Алгоритм Флойда-Стейнберга	17
2.4 Комбинация алгоритмов Флойда-Стейнберга и алгоритма случайного распределения	17
3 Технологический раздел	19
3.1 Выбор языка программирования	19
3.2 Выбор вспомогательных библиотек	19
3.2.1 Диаграмма классов	19
Заключение	21
Список использованных источников	22

Глоссарий

Артефакт — аномалия, возникающая во время визуального представления изображения[?]

Цифровой шум — дефект изображения, вносимый фотосенсорами и электроникой устройств, которые их используют вследствие несовершенства технологий. Цифровой шум заметен на изображении в виде наложенной маски из пикселей случайного цвета и яркости [?]

Фиксированная пороговая обработка (fixed thresholding) — обработка пикселя на основе какого-то фиксированного порогового значения. В случае если текущее значения пикселя больше этого порогового значения, пиксель закрашивается одним фиксированным цветом, иначе - другим. [?]

Интенсивность цвета — степень отличия хроматического цвета от равного ему по светлоте ахроматического, «глубина» цвета. Два оттенка одного тона могут различаться степенью блёклости. При уменьшении насыщенности каждый хроматический цвет приближается к серому.[?]

Ахроматические цвета — оттенки серого (в диапазоне белый — чёрный).[?]

Монохромное изображение — изображение, содержащее свет одного цвета (длины волны), воспринимаемый, как один оттенок (в отличие от цветного изображения, содержащего различные цвета).[?]

Цвета шума — система терминов, приписывающая некоторым видам стационарных шумовых сигналов определённые цвета исходя из аналогии между спектром сигнала произвольной природы.[?]

Спектральная плотность $S(w)$ стационарного случайного процесса $x(t)$ — это частотная функция, характеризующая спектральный (частотный) состав процесса, и представляет собой частотную характеристику для средних значений квадратов амплитуд гармоник, на которые может быть разложен случайный процесс.[?]

Светлый пиксель — пиксель, код цвета которого более или равен 128 для одноцветной палитры 0-255.

Темный пиксель — пиксель, код цвета которого менее 128 для одноцветной палитры 0-255.

Обозначения и сокращения

- ⊠ — Искомый пиксель
- — Пустой пиксель
- — Закрашенный пиксель
- ▣ — Произвольный пиксель

Введение

Изображения, хранимые в цифровом виде, представляются как массив из значений атрибутов; при этом для представления полноцветных фотографий используется диапазон из нескольких миллионов значений на каждый атрибут. Но количество выводимых отображающим устройством оттенков ограничено. Если графическое устройство не способно воссоздавать достаточное количество цветов, тогда используют растривание — независимо от того, растровое это устройство или нерастровое. Такие способы используют особенности человеческого зрения и в первую очередь — пространственную интеграцию. Если достаточно близко расположить маленькие точки разных цветов, то они будут восприниматься как одна точка с некоторым усредненным цветом. Если на плоскости густо расположить много маленьких разноцветных точек, то будет создана визуальная иллюзия закрашивания плоскости определенным усредненным цветом. Однако, если увеличивать размеры точек и (или) расстояние между ними, то иллюзия сплошного закрашивания исчезает — включается другая система человеческого зрения, которая обеспечивает способность различать объекты, подчеркивать контуры. В компьютерных графических системах часто используют эти методы. Они позволяют увеличить количество оттенков цветов за счет снижения пространственного разрешения растрового изображения (иначе говоря — это обмен разрешающей способности на количество цветов). В компьютерной графике такие методы растривания получили название дизеринг (eng.dithering - разрежение, дрожание). В последнее время, в связи с распространением высококачественного видео (4К видео, 360° видео), высококачественных фотографий, проблема сохранения качества изображения при уменьшении его размера становится все более актуальной.

1 Аналитический раздел

В компьютерной графике дизеринг используется для создания иллюзии глубины цвета для изображений с относительно небольшим количеством цветов в палитре. Отсутствующие цвета составляются из имеющихся путём их «перемешивания». Например, если необходимо получить отсутствующий в палитре фиолетовый цвет, его можно получить, разместив красные и синие пиксели в шахматном порядке; оранжевый цвет может быть составлен из красных и желтых точек.

При оптимизации изображений путём уменьшения количества цветов, применение дизеринга приводит к визуальному улучшению изображения, однако для отдельных сжатых форматов (например, PNG), увеличивает его размер.

Алгоритмы дизеринга подразделяются на следующие категории:

- Случайный дизеринг(Random dither);
- Шаблонный дизеринг(Patterning);
- Упорядоченный дизеринг(Ordered);
- Дизеринг рассеивания ошибок(Error-diffusion).

Нижеприведенные алгоритмы описываются для черно-белых изображений. Для цветных изображений алгоритмы аналогичны.

1.1 Случайный дизеринг

Этот алгоритм тривиален. По сравнению с остальными алгоритмами, его качество слишком низко, поэтому он применяется лишь там, где необходима высокая скорость работы в ущерб качеству.[?] Для каждого пикселя в нашем черно-белом изображении мы генерируем случайное число в диапазоне 0-255: если случайное число больше, чем значение в данной точке, то отображаем белый пиксель, иначе отображаем черн пиксель. Это создает изображение с большим количеством шумов. Хотя изображение выглядит неточным и зернистым, оно не содержит артефактов[?]]. Этот метод дизеринга полезен при воспроизведении низкокачественных изображений, где отсутствие артефактов более важно, чем наличие шумов. Например, изображение содержит градиент всех уровней от черного до белого. Это изображение не будет иметь артефактов после того, как к нему применят случайный дизеринг, а остальные методы дизеринга приведут к возникновению артефактов [?].

1.2 Шаблонный дизеринг

Шаблонный дизеринг подразумевает то, что мы увеличиваем разрешение изображения. Так же, как и случайный дизеринг, это тривиальный алгоритм, но он гораздо более эффективен.[?] Для каждой точки изображения мы генерируем «шаблон» пикселей, который аппроксимирует эту точку. Тем самым имитируется больший

Основным недостатком данного метода считается то, что в результате его работы формируется большое количество артефактов[?].

1.4 Дизеринг при помощи диффузии ошибок

Метод, обладающий наилучшим качеством среди представленных, - метод рассеивания ошибок. Но так же он, к сожалению, самый медленный.[?] Существуют несколько вариантов этого алгоритма, причем скорость алгоритма обратно пропорционально качеству изображения.[?] Суть алгоритма: для каждой точки изображения находим ближайший возможный цвет. Затем мы рассчитываем разницу между текущим значением и ближайшим возможным. Эта разница и будет нашим значением ошибки. Это значение ошибки мы распределяем между соседними элементами, которые мы ещё не посещали. Для последних точек ошибка распределяется между уже посещенными точками.

1.5 Вариации алгоритма дизернга при помощи диффузии ошибок

Линия сканирование движется слева-направо. Когда линия сканирования доходит до конца горизонтальной строки пикселей, переходим к первому пикселю следующей строки и повторяем необходимые действия.

*Примечание: числа на схемах - это доли от значения ошибки. Например, $7/16$ на схеме выглядит как 7. То есть 7 обозначает некую величину, равную значению ошибки $*7/16$*

1.5.1 Фильтр Флойда-Стейнберга

Каждый пиксель распределяет свою ошибку на соседние с ним пиксели. Коэффициенты были подобраны таким образом, что в районах с интенсивностью $1/2$ от общего количество оттенков, изображение выглядело похожим на шахматную доску.

$$\left| \begin{array}{cc|c} \boxminus & \boxtimes & 7 \\ 3 & 5 & 1 \end{array} \right| (1/16)$$

1.5.2 "Ложный"фильтр Флойда-Стейнберга

В случае сканирования слева-направо этот фильтр порождает большое количество артефактов. Чтобы получить изображение с меньшим количеством артефактов, нужно чётные строки сканировать справа-налево, а нечетные строки сканировать слева-направо.

$$\left| \begin{array}{cc|c} \boxtimes & 3 \\ 3 & 2 \end{array} \right| (1/8)$$

1.5.3 Фильтр Джарвиса, Джунка и Нинка

В случае когда фильтры Флойда-Стейдберга дают недостаточно хороший результат, применяются фильтры с более широким распределением ошибки. Фильтр Джарвиса, Джунка и Нинка требует связи с 12 соседями, что очевидно ведет в большим затратам памяти и времени[?]:

$$\begin{vmatrix} \boxminus & \boxminus & \boxtimes & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{vmatrix} (1/48)$$

1.5.4 Фильтр Стаки

Фильтр разработан на основе фильтра Джарвиса, Джунка и Нинка. После такого как мы вычислим 8/42 ошибки, остальные значения можно получить при помощи побитовых сдвигов, тем самым сокращая время работы алгоритма.

$$\begin{vmatrix} \boxminus & \boxminus & \boxtimes & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{vmatrix} (1/42)$$

1.5.5 Фильтр Бурка

Стаки. Результат можно получить чуть быстрее за счет использования побитовых операций.

$$\begin{vmatrix} \boxminus & \boxminus & \boxtimes & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \end{vmatrix} (1/32)$$

Существует много различных вариантов фильтро дизеринга при помощи диффузии ошибок, здесь приведены наиболее популярные алгоритмы.[?]

1.6 Выбор оптимального класса алгоритма

Вышеприведенные методы упорядочены по качеству получаемого на выходе изображения, однако, такие соображения как время, экономия памяти и прочие являются определяющими при выборе алгоритма[?]. На основе вышеприведенных данных сравним классы алгоритмов дизеринга.

Характеристика Вид алгоритма	Скорость	Качество	Доп память
Случайный	+	-	-
Шаблонный	+-	-+	+
Упорядоченный	-+	+-	-
Диффузия ошибок	-	+	+

Самым быстрым классом алгоритмов дизеринга является случайный дизеринг. Однако, качество получаемых при помощи изображений низко. Выдвинем

гипотезу, что существуют алгоритмы случайного дизеринга не сильно уступающие по качеству алгоритму диффузии ошибок Флойда-Стейнберга. Так же выдвинем гипотезу, что сочетание алгоритма Флойда-Стейнберга и алгоритмов случайного дизеринга позволяет получить изображение более высокого качества, чем после применения каждого из этих алгоритмов по отдельности.

2 Конструкторский раздел

2.1 Оценка качества изображений

2.1.1 PSNR

Пиковое отношение сигнала к шуму (англ. peak signal-to-noise ratio) - соотношение между максимумом возможного значения сигнала и мощностью шума, искажающего значения сигнала.[?]

$$PSNR = 20 \log_{10} \frac{MAX_i}{\sqrt{MSE}} \quad (2.1)$$

Где MAX_i - это максимальное значение, принимаемое пикселем изображения, MSE - среднеквадратичное отклонение. Для двух монохромных изображений I и K размера $m \times n$, одно из которых считается зашумленным приближением другого, вычисляется так:

$$MSE = \frac{1}{m * n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |I(i,j) - K(i,j)|^2 \quad (2.2)$$

2.1.2 SSIM

Индекс структурного сходства (SSIM от англ. structure similarity) — метод измерения схожести между двумя изображениями путем полного сопоставления. SSIM-индекс является развитием традиционных методов, таких как PSNR (peak signal-to-noise ratio) и метод среднеквадратичной ошибки MSE, которые оказались несовместимы с физиологией человеческого восприятия.

Отличительной особенностью метода, в отличие от MSE и PSNR, является то, что он учитывает «восприятие ошибки» благодаря учёту структурного изменения информации. Идея заключается в том, что пиксели имеют сильную взаимосвязь, особенно когда они близки пространственно. Данные зависимости несут важную информацию о структуре объектов и о сцене в целом. Особенностью является, что SSIM всегда лежит в промежутке от -1 до 1, причем при его значении равном 1, означает, что мы имеем две одинаковые картинки. Общая формула имеет вид

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (2.3)$$

Тут μ_x среднее значение для первой картинки, μ_y для второй, σ_x среднеквадратичное отклонение для первой картинки, и соответственно σ_y для второй, σ_{xy} это уже ковариация. Она находится следующим образом:

$$\sigma_{xy} = \mu_{xy} - \mu_x\mu_y \quad (2.4)$$

c_1 и c_2 - поправочные коэффициенты, которые нужны вследствие малости знаменателя.

$$c_1 = (0,01 * d)^2 \quad (2.5)$$

$$c_2 = (0,03 * d)^2 \quad (2.6)$$

d - количество цветов, соответствующих данной битности изображения. Для подтверждения или опровержения вышеописанных гипотезы реализуются несколько алгоритмов случайного распределения, алгоритм Флойда-Стейнберга, модификации на основе алгоритма Флойда-Стейнберга. Результаты работы алгоритмов сравниваются по времени работы, по количеству затрачиваемой памяти, а так же по SSIM и PSNR.

2.1.3 Алгоритм случайного распределения

$P(x,y)$ - цвет конкретного пикселя

Листинг 2.1 — Алгоритм случайного распределения

```

1 for x in range(height):
2     for y in range(weight):
3         if P(x,y) > 127:
4             P(x,y) = 255
5         else:
6             P(x,y) = 0

```

2.2 Виды случайных распределений

2.2.1 Белый шум

Белый шумом называют сигнал с равномерной спектральной плотностью на всех частотах и дисперсией, равной бесконечности. Является стационарным случайным процессом. В качестве сигнала в задаче дизеринга рассматривается последовательность чисел, получаемых от генератора случайных чисел.

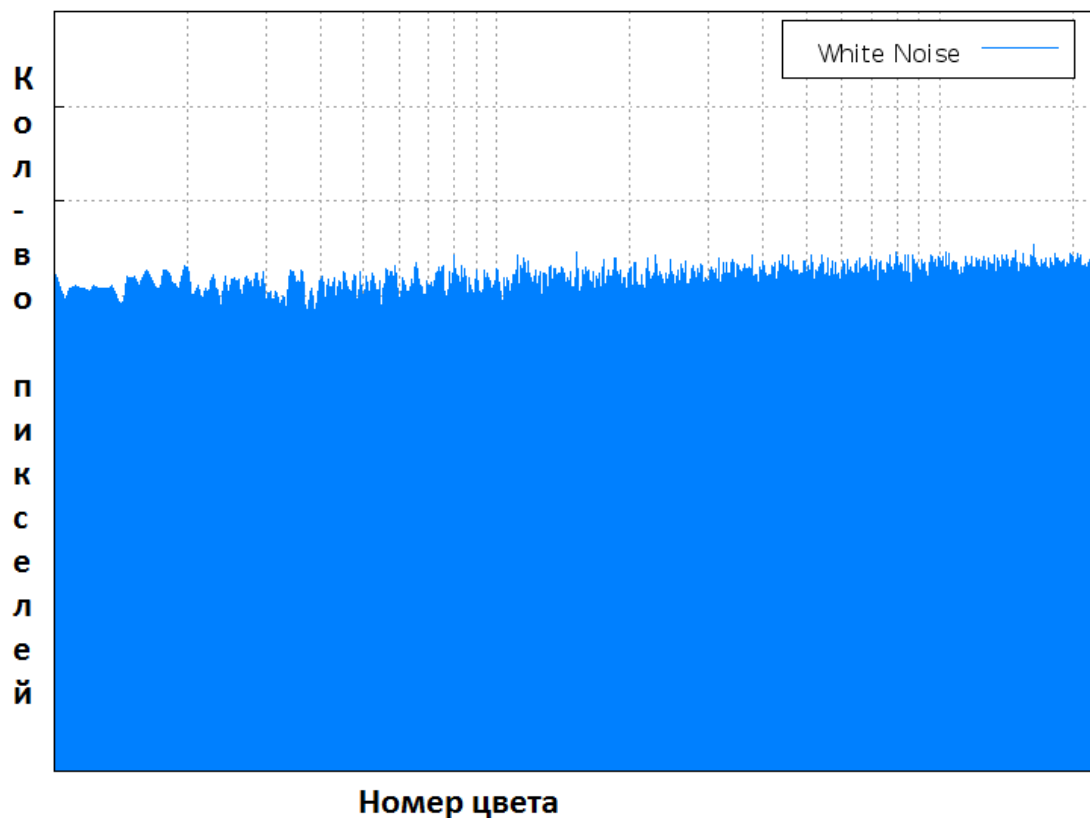


Рисунок 2.1 — Диаграмма белого шума//почему-то очень много места сверху

2.2.2 Коричневый шум

Спектральная плотность коричневого шума пропорциональна $1/f^2$, где f — частота. Это означает, что на низких частотах шум имеет больше энергии, чем на высоких. То есть пикселей темных цветов большей, чем пикселей светлого цвета. Применение фильтра коричневого шума в целом затемняет получаемое изображение.

Листинг 2.2 — Получение коричневого шума

```

1 def smoother(noise):
2     output = []
3     for i in range(len(noise) - 1):
4         output.append(0.5 * (noise[i] + noise[i+1]))
5 return output

```

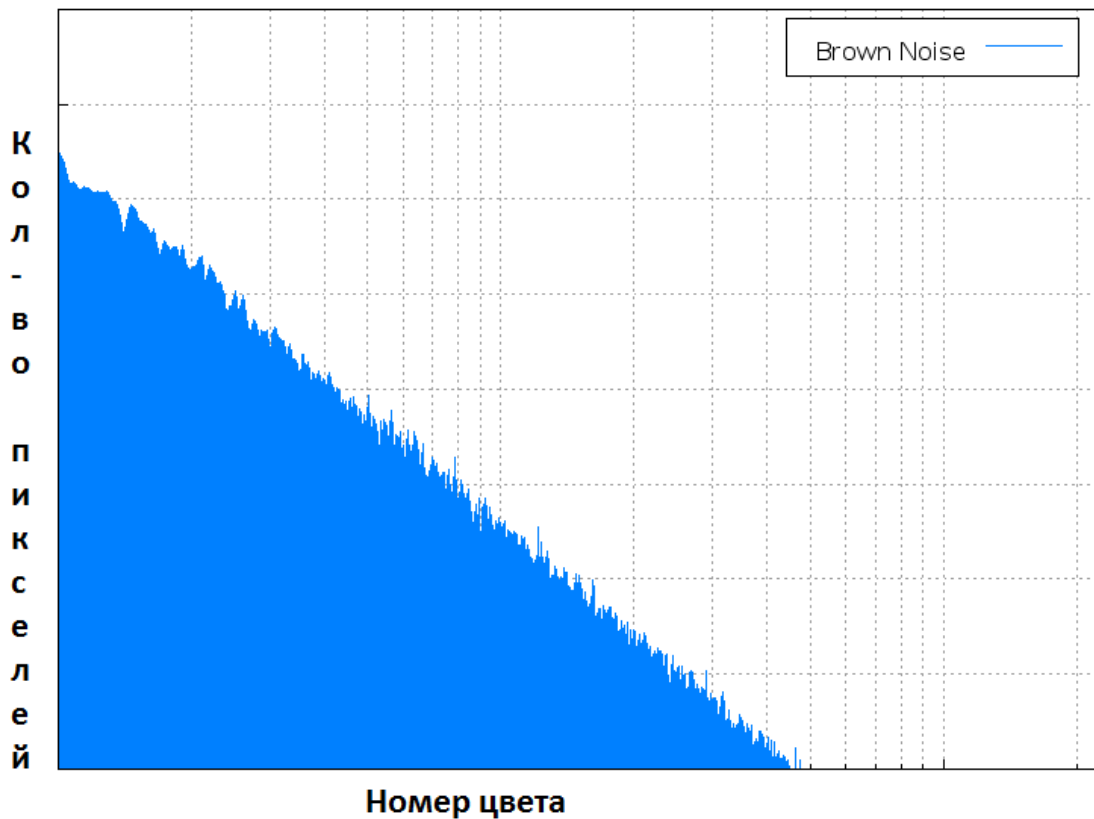


Рисунок 2.2 — Диаграмма красного шума

2.2.3 Гауссовский шум

Гауссовский шум - шум, имеющий функцию плотности вероятности (PDF), равную нормальному распределению, которое также известно как гауссово распределение.

$$p_g(z) = \frac{1}{\sigma\sqrt{2 * \pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \quad (2.7)$$

z - количество цветов, μ -среднее значение, σ - стандартное отклонение.

2.2.4 Фиолетовый шум

Фиолетовый шум представляет собой противоположность между коричневому шуму. Получение его аналогично получению коричневого шума. Применение фильтра фиолетового шума в целом засветляет получаемое изображение.

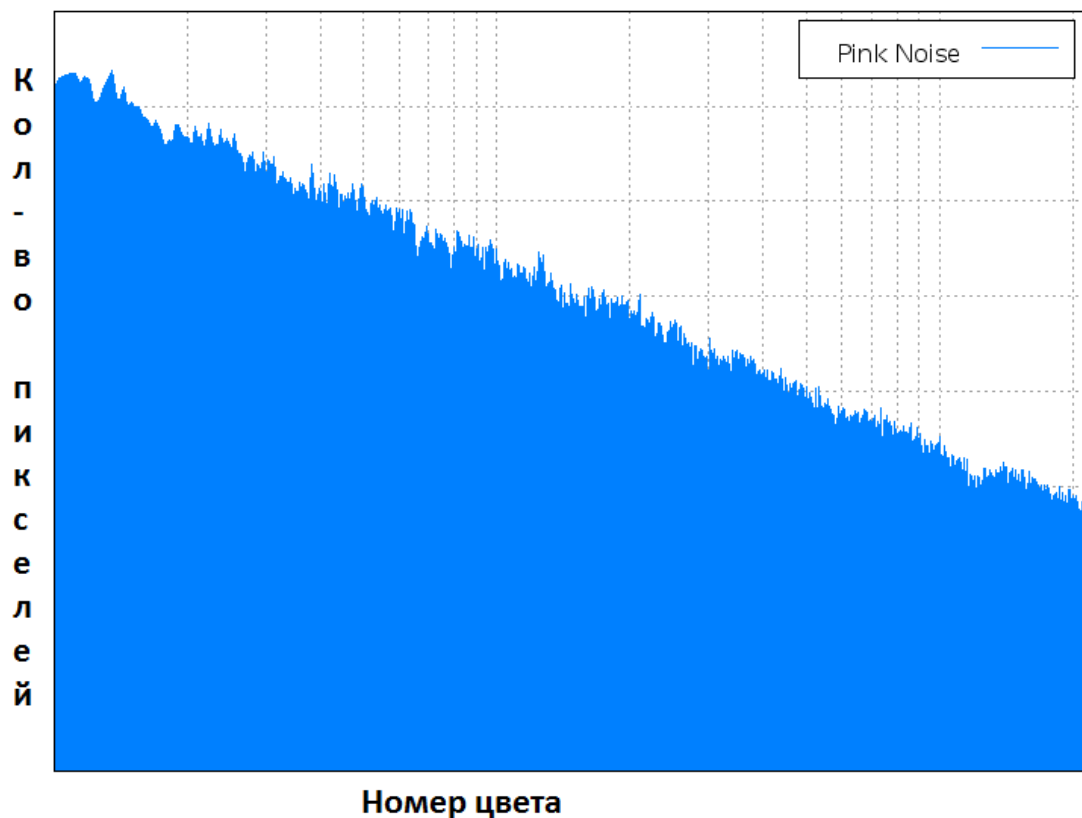


Рисунок 2.3 — Диаграмма розового шума

Листинг 2.3 — Получение розового шума

```

1 def rougher(noise):
2     output = []
3     for i in range(len(noise) - 1):
4         output.append(0.5 * (noise[i] - noise[i+1]))
5     return output

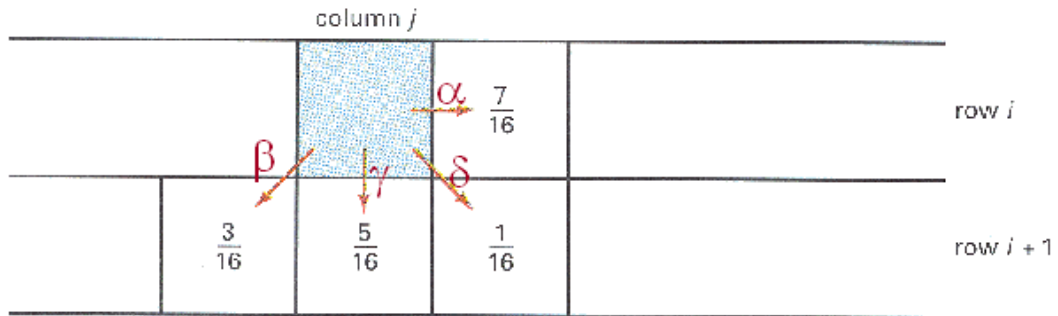
```

2.2.5 Розовый и синий шумы

Розовый и синий шумы представляют собой "промежуточные" шумы. Изображение с розовым шумом темнее изображения с белым шумом, но светлее изображения с коричневым шумом. Изображение с синим шумом светлее изображения с белым шумом, но темнее чем изображение с фиолетовым шумом. Их получение аналогично получению со коричневого и фиолетового шумов.

2.3 Алгоритм Флойда-Стейнберга

Рассмотрим более детально алгоритм Флойда-Стейнберга. $P(x,y)$ - цвет пикселя в точке x,y $I(x,y)$ - предполагаемый цвет пикселя с учетом ошибки (действительное число)



$$\alpha + \beta + \gamma + \delta = 1.0$$

Рисунок 2.4 — Схема рассеивания ошибки

Листинг 2.4 — Алгоритм Флойда-Стейнберга

```

1 for x in range(width):
2     for y in range(height):
3         P(x,y) = trunc(I(x,y)+0.5)
4         e = I(x,y) - P(x,y)
5         I(x,y+1) += alpha*e
6         I(x+1, y-1) += beta*e
7         I(x+1, y) += gamma*e
8         I(x+1, y+1) += sigma*e

```

2.4 Комбинация алгоритмов Флойда-Стейнберга и алгоритма случайного распределения

Комбинируя алгоритмы Флойда-Стейнберга и алгоритм случайного распределения, мы, возможно, получим алгоритм, превосходящий их по метрикам SSIM и PSNR. Идея совмещения заключается в том, что коэффициенты диффузии ошибки вычисляются при помощи генерации элемента последовательности шума. Например, для коэффициента $3/16$ и алгоритма белого шума, можно будет равновероятности получить вместо этого коэффициент $0/16, 1/16, 2/16, 3/16$.

Алгоритм комбинации для белого шума, комбинации остальных шумов и алгоритма Флойда-Стейнберга тривиальны и аналогичны.

Листинг 2.5 — Алгоритм Флойда-Стейнберга и белый шум

```
1 for x in range(width):
2     for y in range(height):
3         P(x,y) = trunc(I(x,y)+0.5)
4         e = I(x,y) - P(x,y)
5         I(x,y+1) += random.randint(1,alpha*16)/16*e
6         I(x+1, y-1) += random.randint(1,beta*16)/16*e
7         I(x+1, y) += random.randint(1,gamma*16)/16*e
8         I(x+1, y+1) += random.randint(1,sigma*16)/16*e
```

3 Технологический раздел

3.1 Выбор языка программирования

Выбранный язык – C++. Причины:

- а) Компилируемый язык со статической типизацией.
- б) Сочетание высокоуровневых и низкоуровневых средств.
- в) Реализация ООП.
- г) Наличие удобной стандартной библиотеки шаблоны

3.2 Выбор вспомогательных библиотек

Для реализации программы была выбрана библиотека Qt.

- а) Широкие возможности работы с изображениями, в том числе и попиксельно
- б) Наличии более функциональных аналогов стандартной библиотеки шаблонов в том числе для разнообразных структур данных

3.2.1 Диаграмма классов

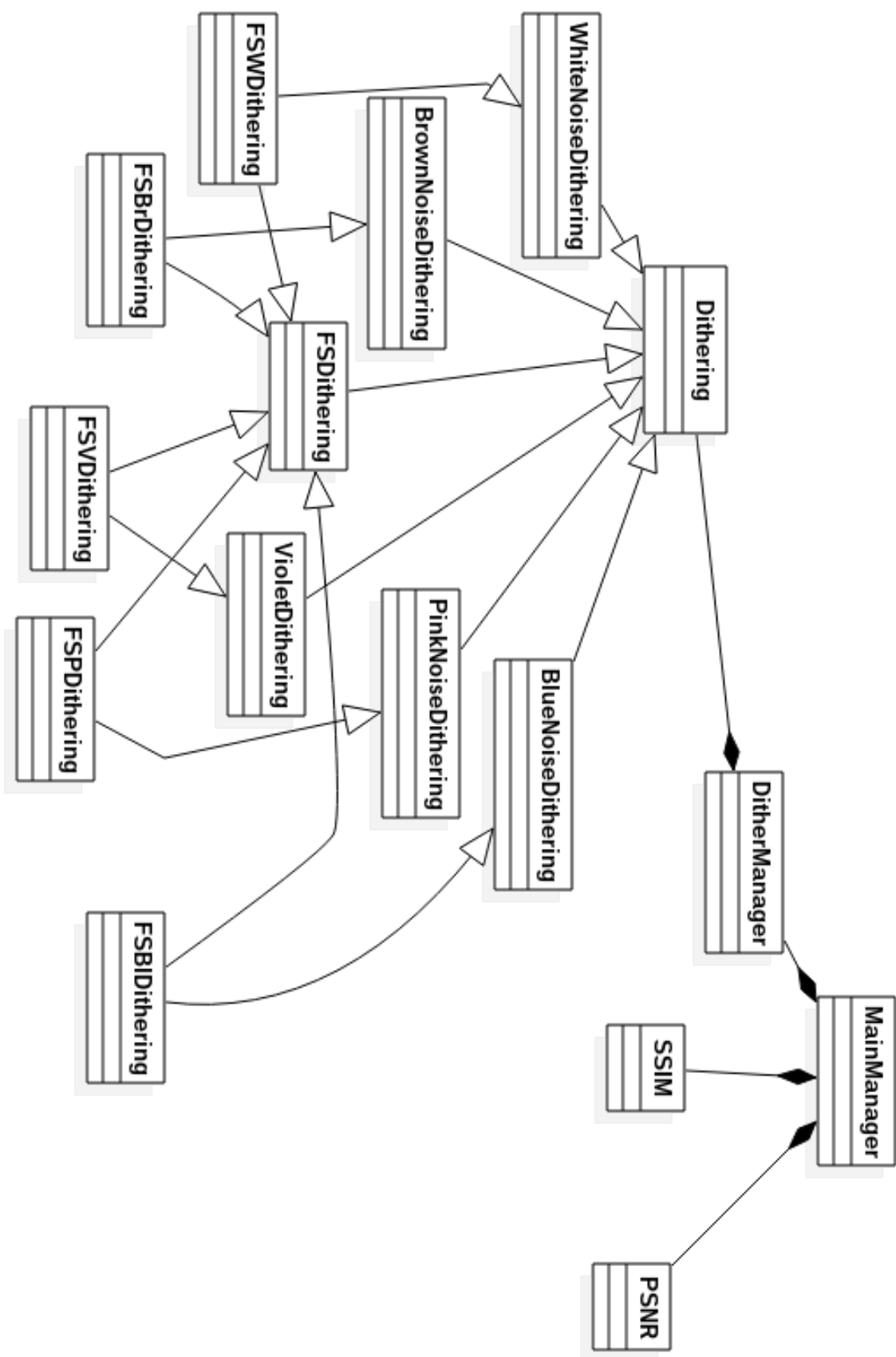


Рисунок 3.1 — Диаграмма классов

Заключение

За практику я смог рассмотреть и попробовать полный процесс подготовки к разработке законченного программного продукта, а именно составить техническое задание, провести анализ алгоритмов и методов решения поставленной задачи, составить диаграмму классов и примерный каркас программы. Так же были получены первые результаты работы программы- а именно "зашумленные"при помощи дизеринга картинки.

Список использованных источников

1. *Пупкин, Василий*. L^AT_EX для «чайников» / Василий Пупкин, А. Эйнштейн.
— М., 2009.