Компьютерная графика

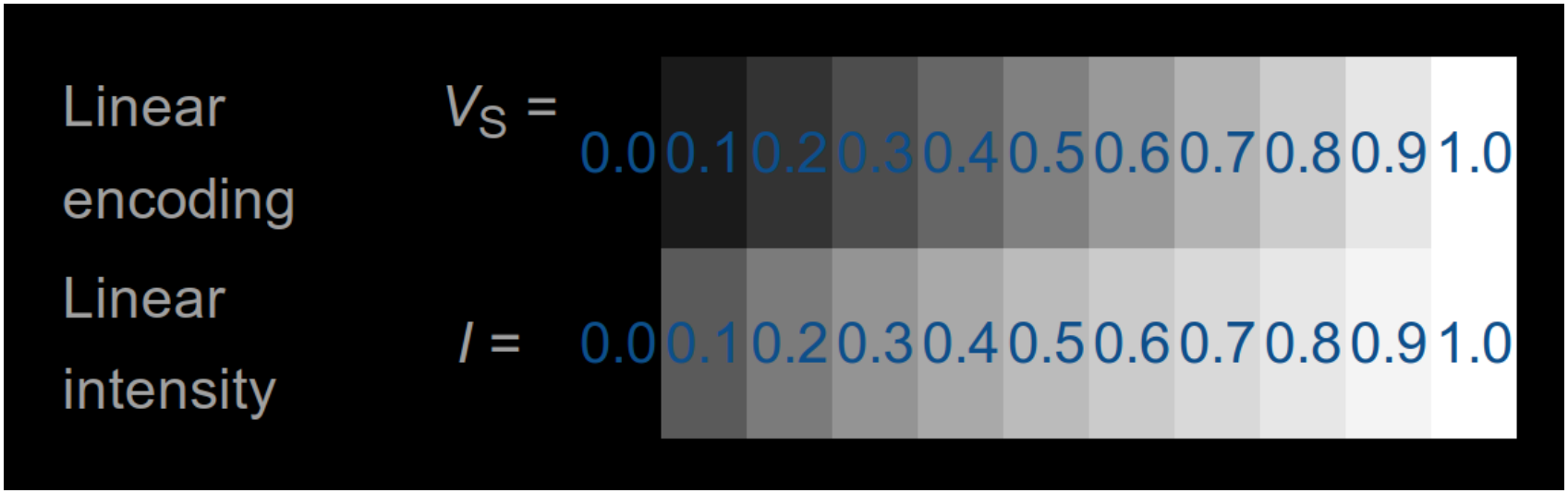
Лекция 10: Gamma correction, sRGB, color banding, dithering

2021

Гамма

- Обычно, интенсивность света I, излучаемого монитором, нелинейно зависит от значения V, записанного в пикселе

Линейное значение пикселя vs линейная интенсивность излучения



Гамма

- Обычно, интенсивность света I, излучаемого монитором, нелинейно зависит от значения V, записанного в пикселе
- Это лучше соответствует восприятию света человеком

Гамма

- Обычно, интенсивность света I, излучаемого монитором, нелинейно зависит от значения V, записанного в пикселе
- Это лучше соответствует восприятию света человеком
- Почти всегда используется показательная функция:

$$I \sim V^{\gamma} \tag{1}$$

Гамма

- Обычно, интенсивность света I, излучаемого монитором, нелинейно зависит от значения V, записанного в пикселе
- Это лучше соответствует восприятию света человеком
- Почти всегда используется показательная функция:

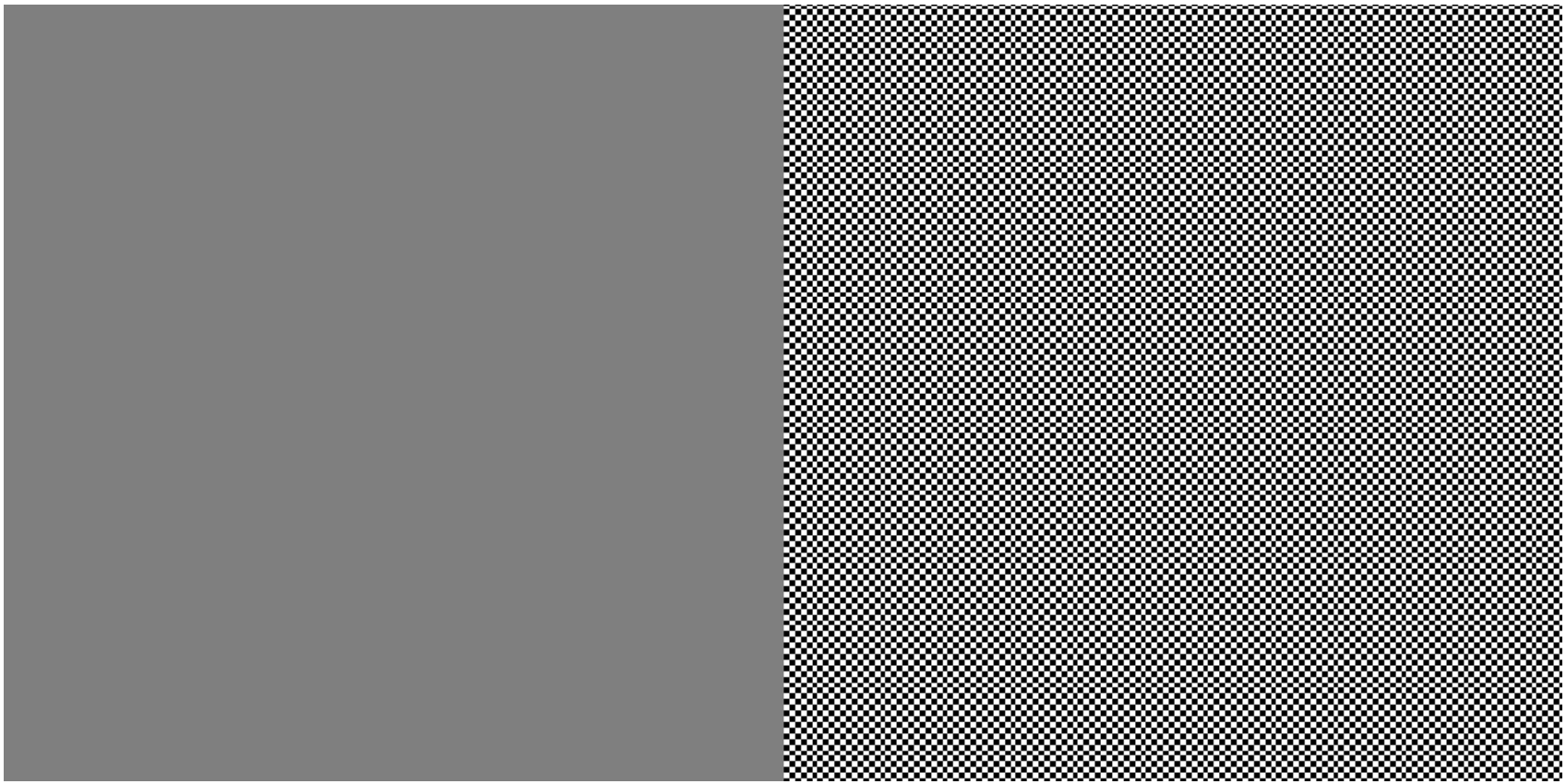
$$I \sim V^{\gamma} \tag{1}$$

- γ обычно равна 2.2 (некоторые компьютеры Macintosh использовали 1.8)

Проблемы гаммы

- Картинка может издалека выглядеть ярче, чем её усреднённый вариант (e.g. mipmap)

Серый (цвет=0.5) квадрат и квадрат с мелкой шахматной раскраской



Проблемы гаммы

- Картинка может издалека выглядеть ярче, чем её усреднённый вариант (e.g. mipmap)
- Искажается восприятие относительных яркостей, особенно при реалистичном рендеринге (e.g. объект в два раза ярче не будет выглядеть в два раза ярче)

Проблемы гаммы

- Картинка может издалека выглядеть ярче, чем её усреднённый вариант (e.g. mipmap)
- Искажается восприятие относительных яркостей, особенно при реалистичном рендеринге (e.g. объект в два раза ярче не будет выглядеть в два раза ярче)
- Неправильно выглядит освещение, наложение источников света, и т.д.

Коррекция гаммы (gamma-correction)

- Коррекция гаммы - общий термин для применения любых нелинейных преобразований над интенсивностью пикселя

Коррекция гаммы (gamma-correction)

- Коррекция гаммы - общий термин для применения любых нелинейных преобразований над интенсивностью пикселя
- В рендеринге под гамма-коррекцией обычно подразумевают применение обратного к $V^{2.2}$ преобразования, чтобы получить линейную зависимость выходящего излучения от значения пикселя

Коррекция гаммы (gamma-correction)

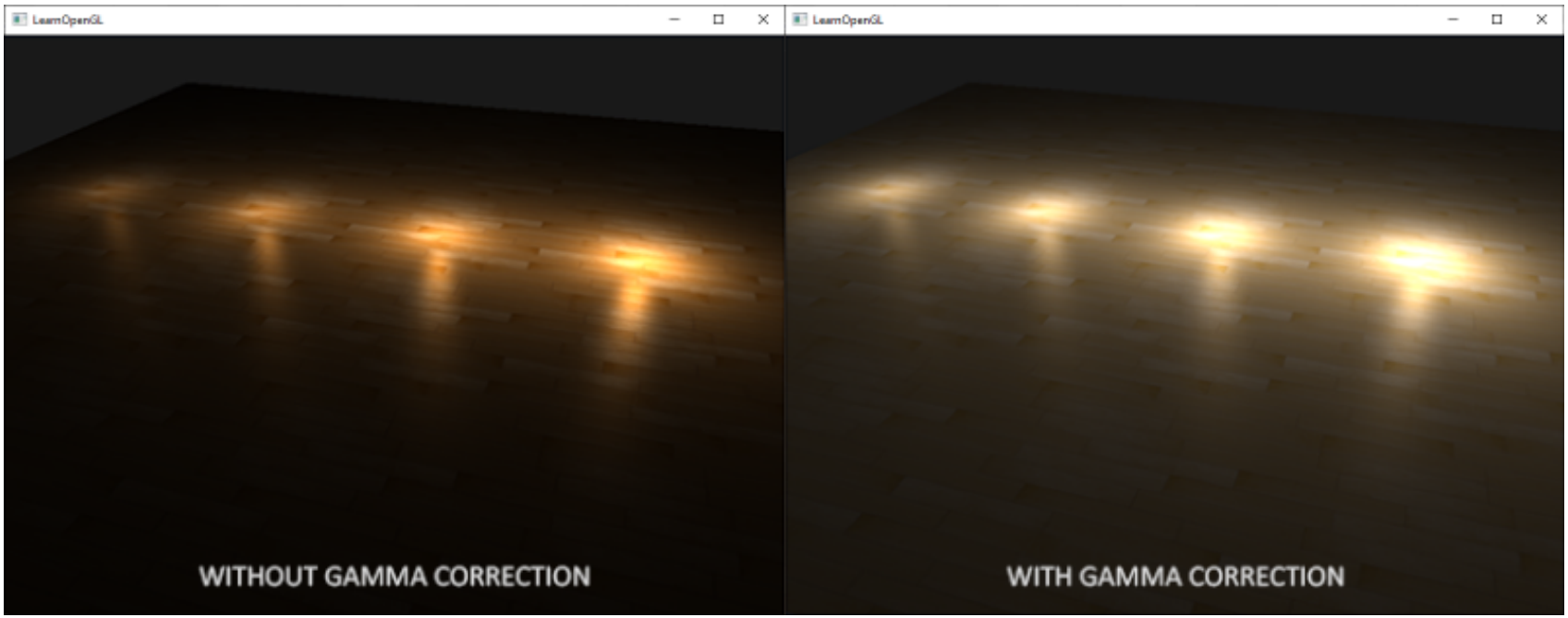
- Коррекция гаммы - общий термин для применения любых нелинейных преобразований над интенсивностью пикселя
- В рендеринге под гамма-коррекцией обычно подразумевают применение обратного к $V^{2.2}$ преобразования, чтобы получить линейную зависимость выходящего излучения от значения пикселя
- Делает картинку ярче и часто более реалистичной

Коррекция гаммы (gamma-correction)

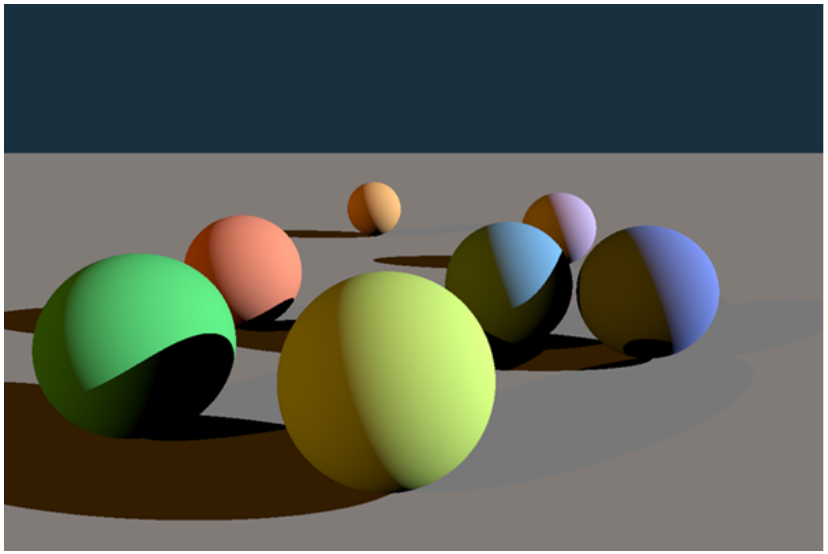
```
// Вычислили цвет пикселя
// с учётом освещения
vec3 color = ...;

color = pow(color, vec3(1.0 / 2.2));
```

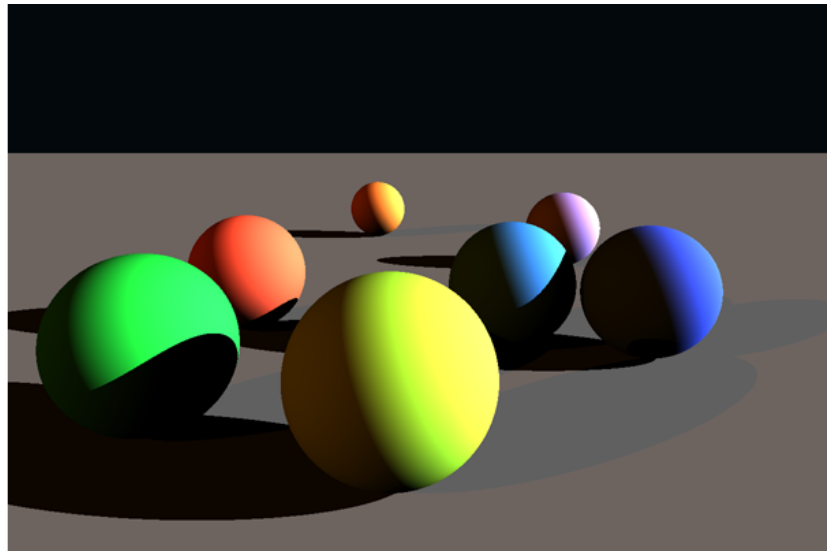
Эффекты коррекции гаммы

Эффекты коррекции гаммы



Эффекты коррекции гаммы





Коррекция гаммы: ссылки

- en.wikipedia.org/wiki/Gamma_correction
- What every coder should know about gamma
- Linear-space lighting (i.e. gamma)
- Туториал на learnopengl.com
- Пример с гамма-коррекцией

sRGB

- По тем же причинам, по которым существует нелинейная гамма у мониторов (отдать больше бит под тёмные цвета), хочется хранить текстуры не в сыром формате, а в гамма-преобразованном: значение пикселя V зависит от желаемой интенсивности I как $V \sim I^{\frac{1}{\gamma}}$

sRGB

- По тем же причинам, по которым существует нелинейная гамма у мониторов (отдать больше бит под тёмные цвета), хочется хранить текстуры не в сыром формате, а в гамма-преобразованном: значение пикселя V зависит от желаемой интенсивности I как $V \sim I^{\frac{1}{\gamma}}$
- Такой формат хранения цвета называется sRGB
  - N.B. Точная формула преобразования sRGB отличается от $I^{\frac{1}{2.2}}$, но очень близка к ней

sRGB

- По тем же причинам, по которым существует нелинейная гамма у мониторов (отдать больше бит под тёмные цвета), хочется хранить текстуры не в сыром формате, а в гамма-преобразованном: значение пикселя V зависит от желаемой интенсивности I как $V \sim I^{\frac{1}{\gamma}}$
- Такой формат хранения цвета называется sRGB
  - N.B. Точная формула преобразования sRGB отличается от $I^{\frac{1}{2.2}}$, но очень близка к ней
- Обычно изображения хранятся именно в таком формате

sRGB

- По тем же причинам, по которым существует нелинейная гамма у мониторов (отдать больше бит под тёмные цвета), хочется хранить текстуры не в сыром формате, а в гамма-преобразованном: значение пикселя V зависит от желаемой интенсивности I как $V \sim I^{\frac{1}{\gamma}}$
- Такой формат хранения цвета называется sRGB
  - N.B. Точная формула преобразования sRGB отличается от $I^{\frac{1}{2.2}}$, но очень близка к ней
- Обычно изображения хранятся именно в таком формате
- Проверить формат можно любым редактором изображений или программой `identify` пакета `imagemagick`

sRGB

- При чтении из sRGB-текстуры нужно возводить прочитанное значение в степень 2.2

sRGB

- При чтении из sRGB-текстуры нужно возводить прочитанное значение в степень 2.2
- При записи в sRGB-текстуру нужно возводить записываемое значение в степень 1/2.2

sRGB

- При чтении из sRGB-текстуры нужно возводить прочитанное значение в степень 2.2
- При записи в sRGB-текстуру нужно возводить записываемое значение в степень 1/2.2
- В OpenGL есть поддержка sRGB для текстур и фреймбуферов

sRGB-текстуры

- Специальное значение internal format для текстуры (`GL_SRGB8` или `GL_SRGB8_ALPHA8`) означает, что текстура хранит sRGB-значения - они будут *автоматически* переведены в линейные значения (т.е. возведены в степень 2.2) при чтении из шейдера

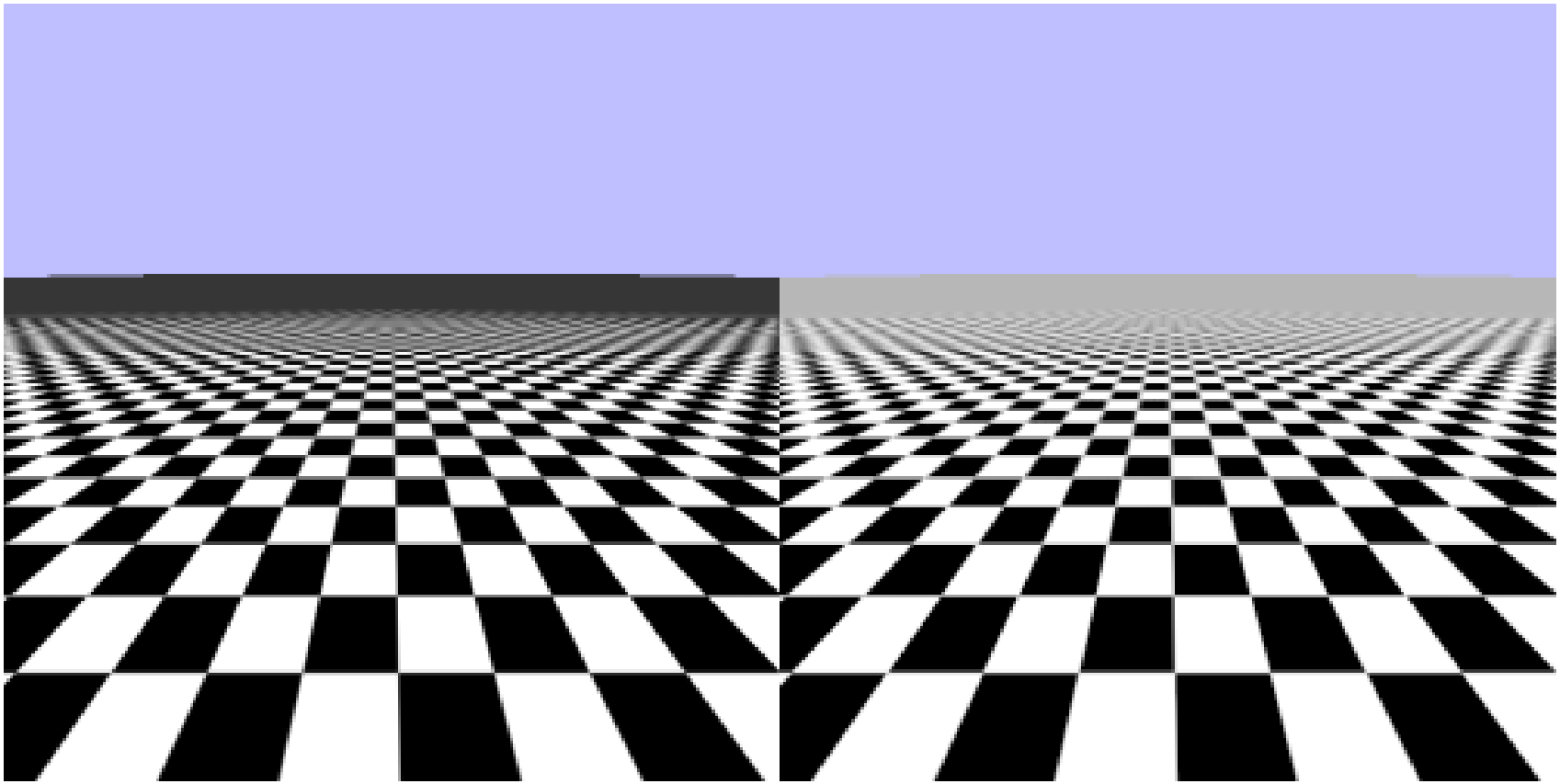
sRGB-текстуры

- Специальное значение internal format для текстуры (`GL_SRGB8` или `GL_SRGB8_ALPHA8`) означает, что текстура хранит sRGB-значения - они будут *автоматически* переведены в линейные значения (т.е. возведены в степень 2.2) при чтении из шейдера
- `glEnable(GL_FRAMEBUFFER_SRGB)` включит автоматическое обратное преобразование (возведение в степень 1/2.2) при рисовании в sRGB-текстуру

sRGB-текстуры

- Специальное значение internal format для текстуры (`GL_SRGB8` или `GL_SRGB8_ALPHA8`) означает, что текстура хранит sRGB-значения - они будут *автоматически* переведены в линейные значения (т.е. возведены в степень 2.2) при чтении из шейдера
- `glEnable(GL_FRAMEBUFFER_SRGB)` включит автоматическое обратное преобразование (возведение в степень 1/2.2) при рисовании в sRGB-текстуру
- N.B.: `glGenerateMipmap` *не обязан* правильно обрабатывать sRGB-текстуры!

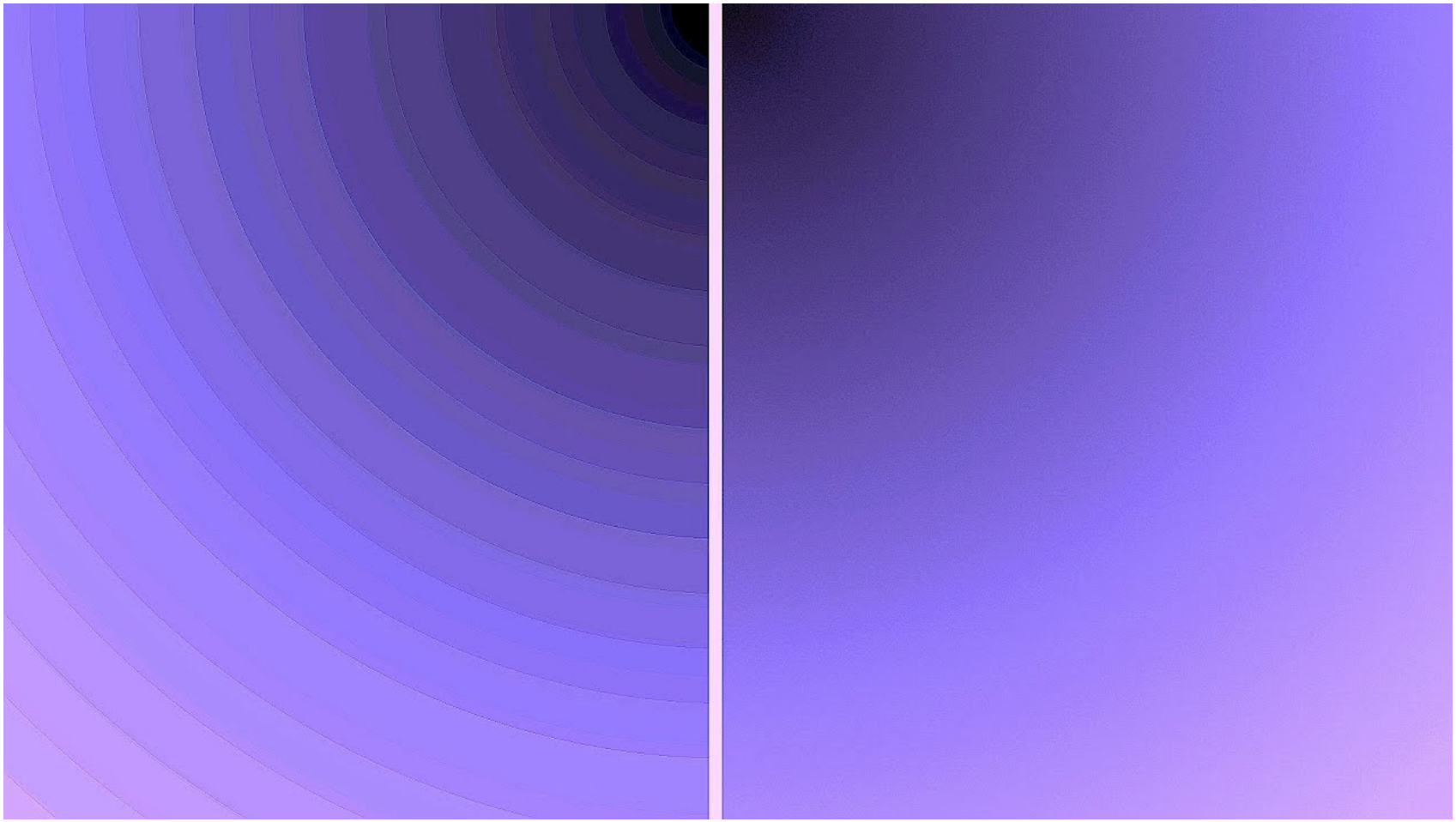
sRGB-correct mipmap



sRGB: ссылки

- en.wikipedia.org/wiki/SRGB
- Подробный текст про sRGB на stackoverflow
- sRGB framebuffers

Color banding



Color banding

- Артефакт, когда чётко видна граница между областями, заполненными одним цветом

Color banding

- Артефакт, когда чётко видна граница между областями, заполненными одним цветом
- Обычно не сильно заметен, особенно после наложения текстур и пост-обработки

Color banding

- Артефакт, когда чётко видна граница между областями, заполненными одним цветом
- Обычно не сильно заметен, особенно после наложения текстур и пост-обработки
- Хорошо заметен на монотонных поверхностях (напр. небо)

Color banding

- Артефакт, когда чётко видна граница между областями, заполненными одним цветом
- Обычно не сильно заметен, особенно после наложения текстур и пост-обработки
- Хорошо заметен на монотонных поверхностях (напр. небо)
- Может усиливаться при неаккуратной работе с форматами хранения и нелинейными преобразованиями

Color banding: пример возникновения

- Рисуем сцену, записываем линейную интенсивность в текстуру в формате `GL_RGB8`

Color banding: пример возникновения

- Рисуем сцену, записываем линейную интенсивность в текстуру в формате `GL_RGB8`
- Выводим сцену на экран с гамма-коррекцией: значения из текстуры возводятся в степень $1/2.2$

Color banding: пример возникновения

- Рисуем сцену, записываем линейную интенсивность в текстуру в формате `GL_RGB8`
- Выводим сцену на экран с гамма-коррекцией: значения из текстуры возводятся в степень 1/2.2
- Значение текстуры 0 переходит в значение 0 на экране

Color banding: пример возникновения

- Рисуем сцену, записываем линейную интенсивность в текстуру в формате `GL_RGB8`
- Выводим сцену на экран с гамма-коррекцией: значения из текстуры возводятся в степень 1/2.2
- Значение текстуры 0 переходит в значение 0 на экране
- Значение текстуры 1 переходит в значение `255 * (1/255)^(1/2.2) = 21` на экране

Color banding: пример возникновения

- Рисуем сцену, записываем линейную интенсивность в текстуру в формате `GL_RGB8`
- Выводим сцену на экран с гамма-коррекцией: значения из текстуры возводятся в степень 1/2.2
- Значение текстуры 0 переходит в значение 0 на экране
- Значение текстуры 1 переходит в значение `255 * (1/255)^(1/2.2) = 21` на экране
- Колоссальная потеря точности: значения от 1 до 20 не используются!

Color banding: пример возникновения

- Рисуем сцену, записываем линейную интенсивность в текстуру в формате `GL_RGB8`
- Выводим сцену на экран с гамма-коррекцией: значения из текстуры возводятся в степень 1/2.2
- Значение текстуры 0 переходит в значение 0 на экране
- Значение текстуры 1 переходит в значение `255 * (1/255)^(1/2.2) = 21` на экране
- Колоссальная потеря точности: значения от 1 до 20 не используются!
- Для избавления можно увеличить битность промежуточной текстуры до `GL_RGB16`

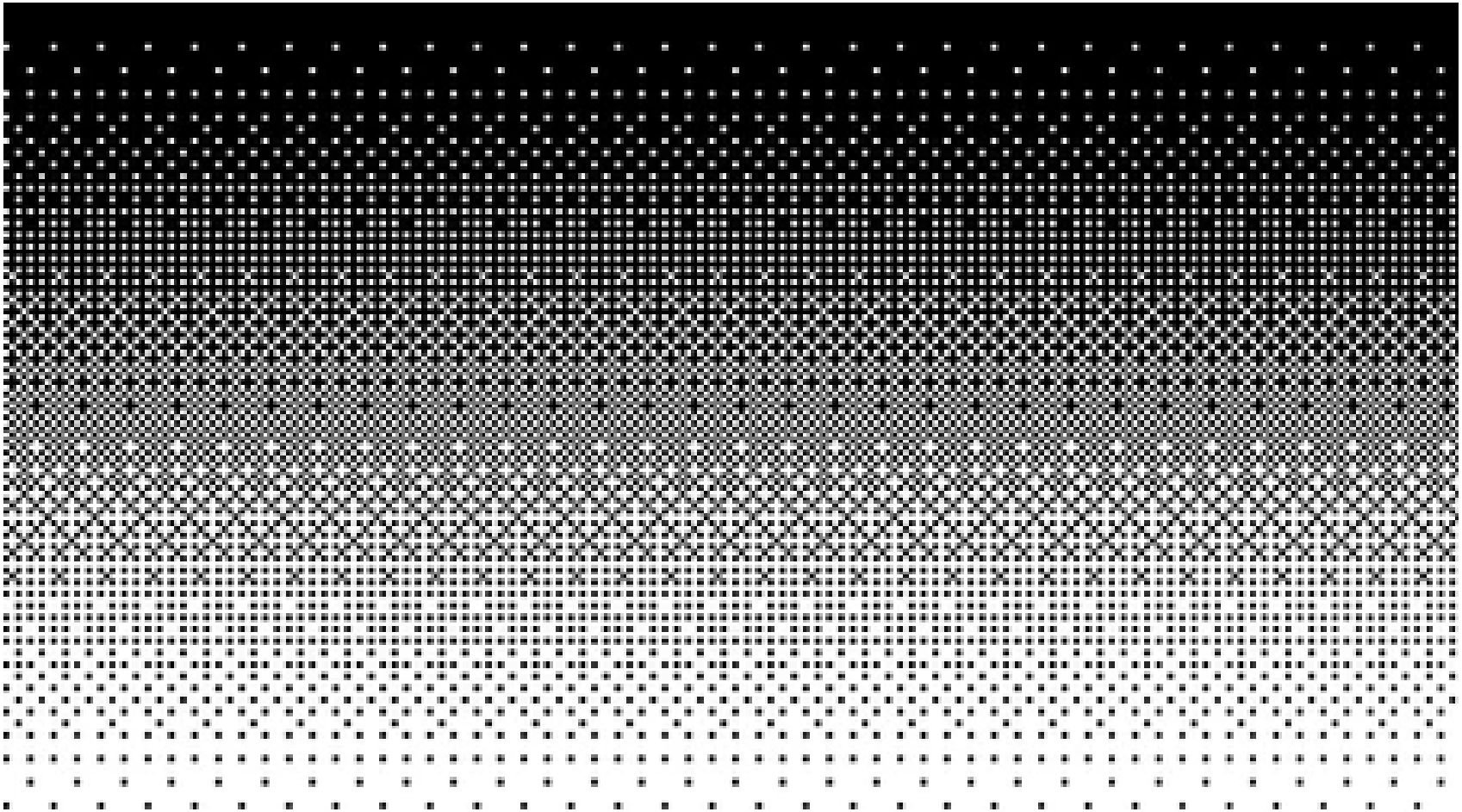
Color banding

- Banding, связанный с потерей точности, обычно решают увеличением точности

Color banding

- Banding, связанный с потерей точности, обычно решают увеличением точности
- Banding, связанный с ограниченной точностью самого экрана, обычно решают с помощью *дизеринга*

Dithering



Dithering

- Способ рисования градиентов (переходов между цветами) за счёт варьирования количества пикселей двух цветов

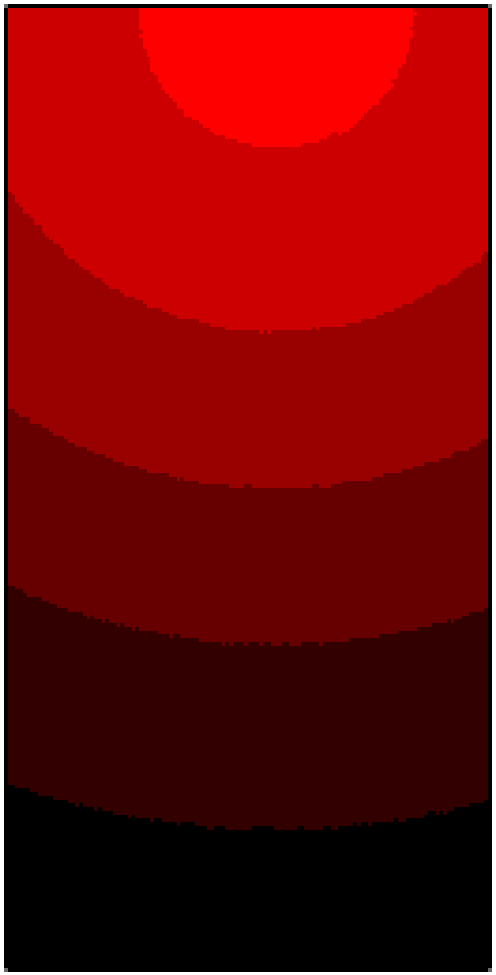
Dithering

- Способ рисования градиентов (переходов между цветами) за счёт варьирования количества пикселей двух цветов
- Работает только для больших (по размеру) градиентов

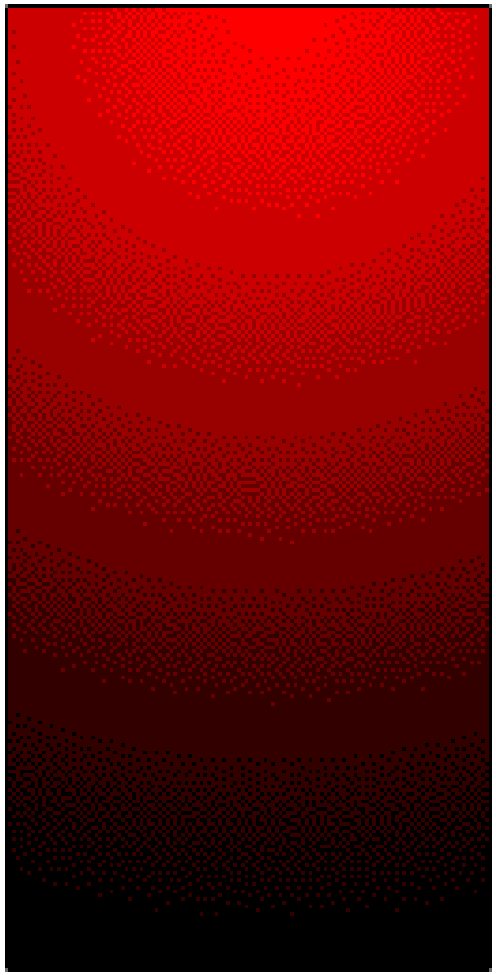
Dithering

- Способ рисования градиентов (переходов между цветами) за счёт варьирования количества пикселей двух цветов
- Работает только для больших (по размеру) градиентов
- Не зависит от точности

Dithering



8-bit gradient



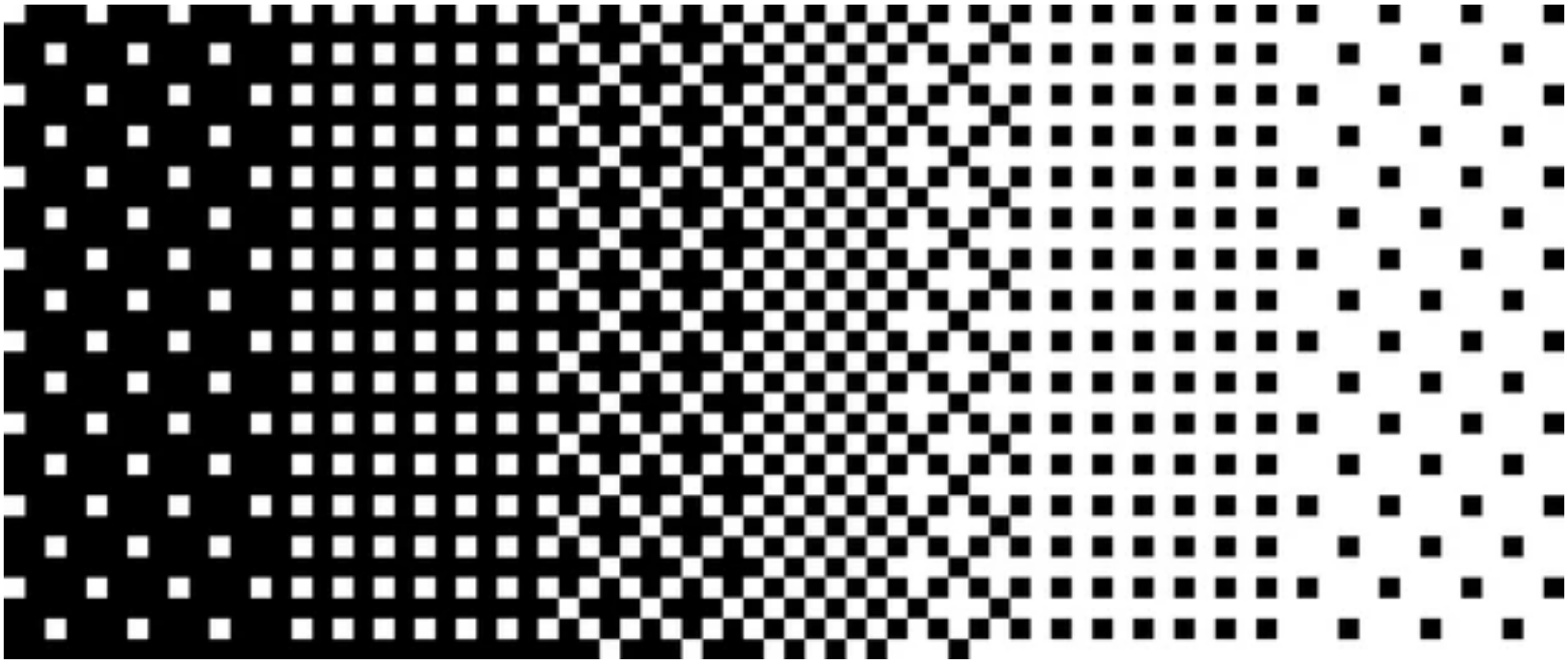
8-bit gradient, dithered

24-bit gradient

Dithering: реализация

- Допустим, мы рисуем в однобитное (чёрно-белое) изображение, и хотим добиться серого цвета с помощью дизеринга

Dithering



Dithering: реализация

- Допустим, мы рисуем в однобитное (чёрно-белое) изображение, и хотим добиться серого цвета с помощью дизеринга
- Нам нужен способ определить, какой из пикселей будет чёрным, а какой - белым

Dithering: реализация

- Допустим, мы рисуем в однобитное (чёрно-белое) изображение, и хотим добиться серого цвета с помощью дизеринга
- Нам нужен способ определить, какой из пикселей будет чёрным, а какой - белым
- В real-time графике это обычно делается с помощью т.н. dither mask / dither matrix (ordered dithering)

Dithering: реализация

- Dither mask - одноканальная текстура, хранящая для каждого пикселя пороговое значение, начиная с которого его надо рисовать белым

Dithering: реализация

- Dither mask - одноканальная текстура, хранящая для каждого пикселя пороговое значение, начиная с которого его надо рисовать белым
- Например, для серого цвета с яркостью 0.25 мы нарисуем белыми те пиксели, для которых значение в dither map ≤ 0.25

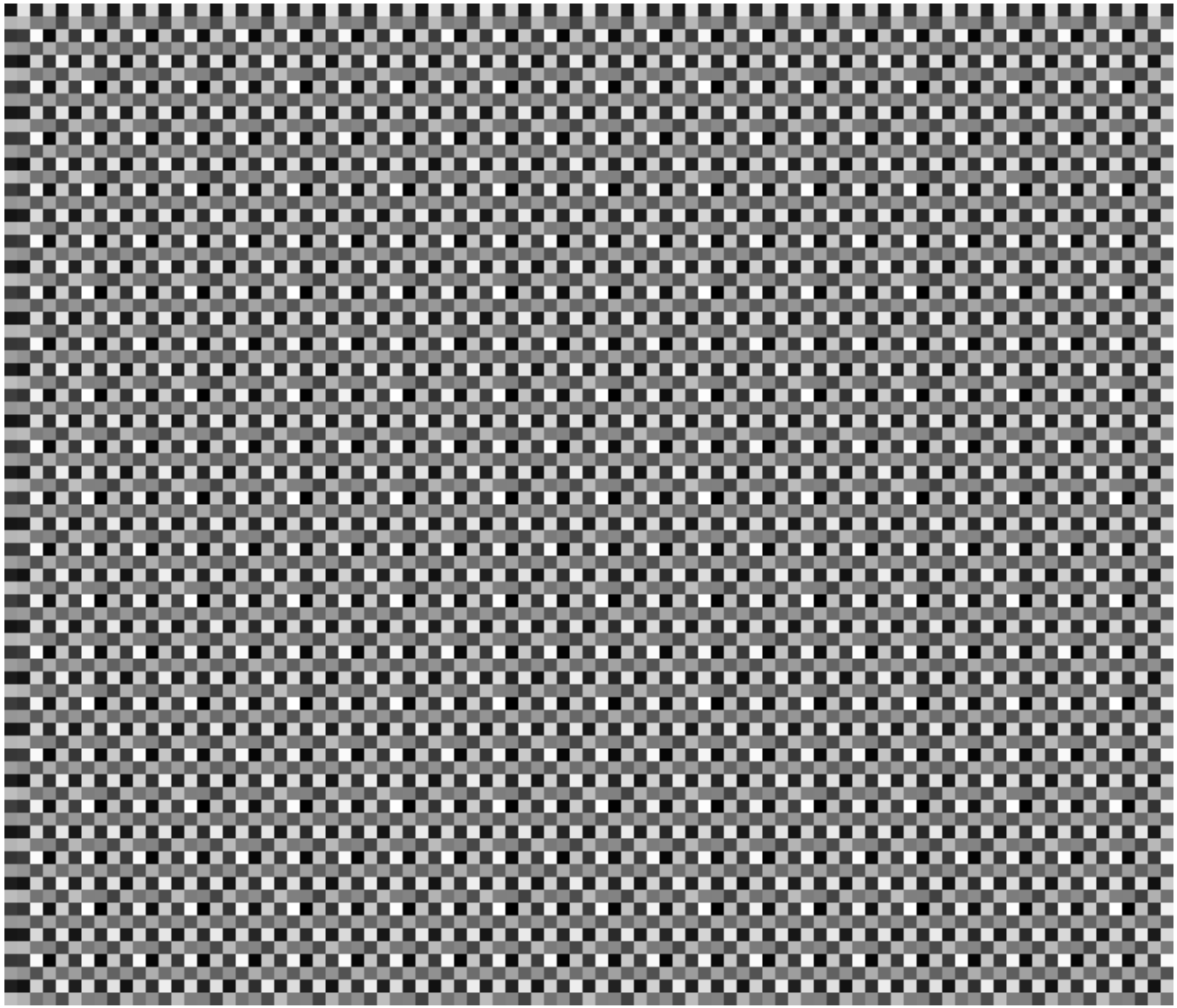
Dithering: реализация

- Dither mask - одноканальная текстура, хранящая для каждого пикселя пороговое значение, начиная с которого его надо рисовать белым
- Например, для серого цвета с яркостью 0.25 мы нарисуем белыми те пиксели, для которых значение в dither map ≤ 0.25
- Например, для серого цвета с яркостью 0.5 мы нарисуем белыми те пиксели, для которых значение в dither map ≤ 0.5

Dithering: реализация

- Dither mask - одноканальная текстура, хранящая для каждого пикселя пороговое значение, начиная с которого его надо рисовать белым
- Например, для серого цвета с яркостью 0.25 мы нарисуем белыми те пиксели, для которых значение в dither map ≤ 0.25
- Например, для серого цвета с яркостью 0.5 мы нарисуем белыми те пиксели, для которых значение в dither map ≤ 0.5
- Например, для серого цвета с яркостью 0.75 мы нарисуем белыми те пиксели, для которых значение в dither map ≤ 0.75

Dither mask



Dithering: реализация

- Для дизеринга при рисовании в 8-битный буфер:
  - Хотим записать значение $v \in [0, 1]$
  - Вычисляем $v' = 255 \cdot v$
  - Вычисляем $v_0 = floor(v')$
  - Вычисляем $v_1 = v_0 + 1$
  - Вычисляем $dv = v' - v0$
  - Если значение в dither mask меньше dv, рисуем значение $v_1/255$, иначе $v_0/255$

Dithering: реализация

- Для дизеринга при рисовании в 8-битный буфер:
  - Хотим записать значение $v \in [0, 1]$
  - Вычисляем $v' = 255 \cdot v$
  - Вычисляем $v_0 = floor(v')$
  - Вычисляем $v_1 = v_0 + 1$
  - Вычисляем $dv = v' - v0$
  - Если значение в dither mask меньше dv, рисуем значение $v_1/255$, иначе $v_0/255$
- Дизеринг нужно делать для каждого канала (RGB) по отдельности

Dithering: реализация

- Для дизеринга при рисовании в 8-битный буфер:
  - Хотим записать значение $v \in [0, 1]$
  - Вычисляем $v' = 255 \cdot v$
  - Вычисляем $v_0 = floor(v')$
  - Вычисляем $v_1 = v_0 + 1$
  - Вычисляем $dv = v' - v0$
  - Если значение в dither mask меньше dv, рисуем значение $v_1/255$, иначе $v_0/255$
- Дизеринг нужно делать для каждого канала (RGB) по отдельности
- Обычно dither mask имеет маленький размер (напр. 16x16) и циклически (`GL_REPEAT`) повторяется на весь экран

Dithering: реализация

```
vec3 color = ...;

vec3 c = color * 255.0;
vec3 c0 = floor(c);
vec3 c1 = c0 + vec3(1.0);
vec3 dc = c - c0;

float threshold = texture(dither_mask, pixel_coord);

color = c0 / 255.0;
if (dc.r > threshold) color.r = c1.r / 255.0;
if (dc.g > threshold) color.g = c1.g / 255.0;
if (dc.b > threshold) color.b = c1.b / 255.0;
```

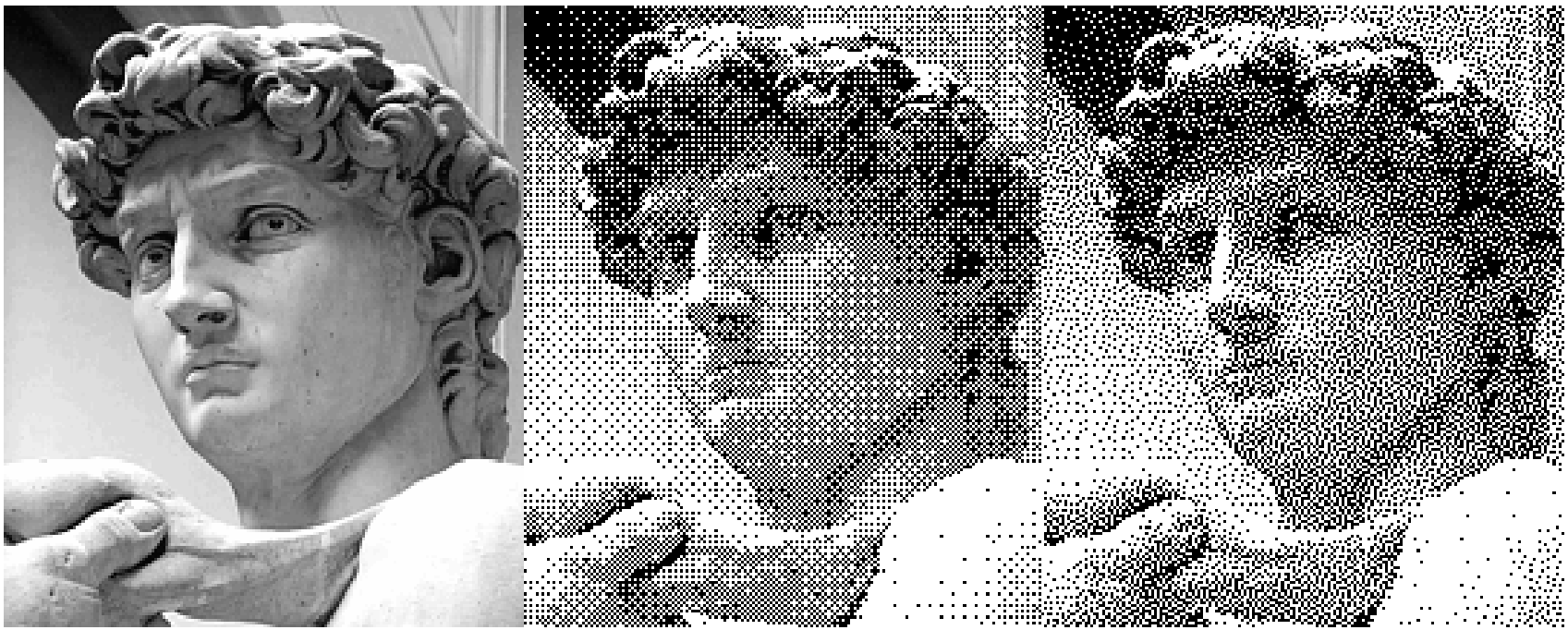
Генерация dither mask

- Bayer dithering - очень быстро генерируется, могут быть заметны артефакты из-за регулярности паттерна

Генерация dither mask

- Bayer dithering - очень быстро генерируется, могут быть заметны артефакты из-за регулярности паттерна
- Void-and-cluster - использует медленный blue noise для генерации маски, лучше распределение пикселей

Сравнение bayer и void-and-cluster



Screen-door transparency

Screen-door transparency

- Дизеринг часто применяют как дешёвый аналог прозрачности

Screen-door transparency

- Дизеринг часто применяют как дешёвый аналог прозрачности
- Если значение в альфа-канале больше значения из dither mask, то рисуем пиксель, иначе - не рисуем (`discard`)

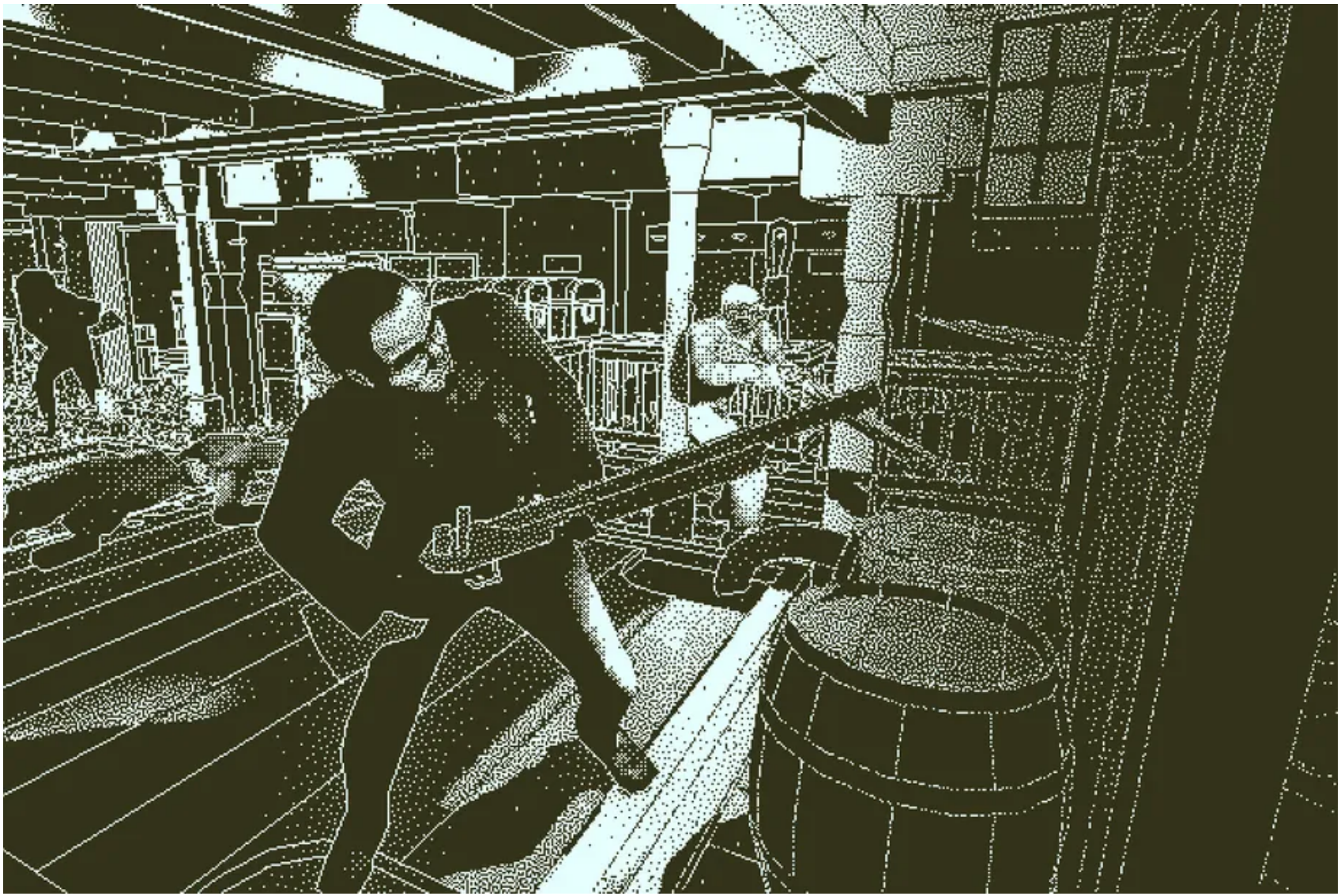
Screen-door transparency

- Дизеринг часто применяют как дешёвый аналог прозрачности
- Если значение в альфа-канале больше значения из dither mask, то рисуем пиксель, иначе - не рисуем (`discard`)
- Не требует сортировки объектов по расстоянию

Screen-door transparency

- Дизеринг часто применяют как дешёвый аналог прозрачности
- Если значение в альфа-канале больше значения из dither mask, то рисуем пиксель, иначе - не рисуем (`discard`)
- Не требует сортировки объектов по расстоянию
- Выглядит заметно хуже, часто применяется для временно прозрачных объектов (появляющихся/исчезающих)

Return of the Obra Dinn (2018)



Dithering: ссылки

- en.wikipedia.org/wiki/Dither
- Статья про дизеринг
- Ещё одна статья про дизеринг
- Использование дизеринга в Return of the Obra Dinn