

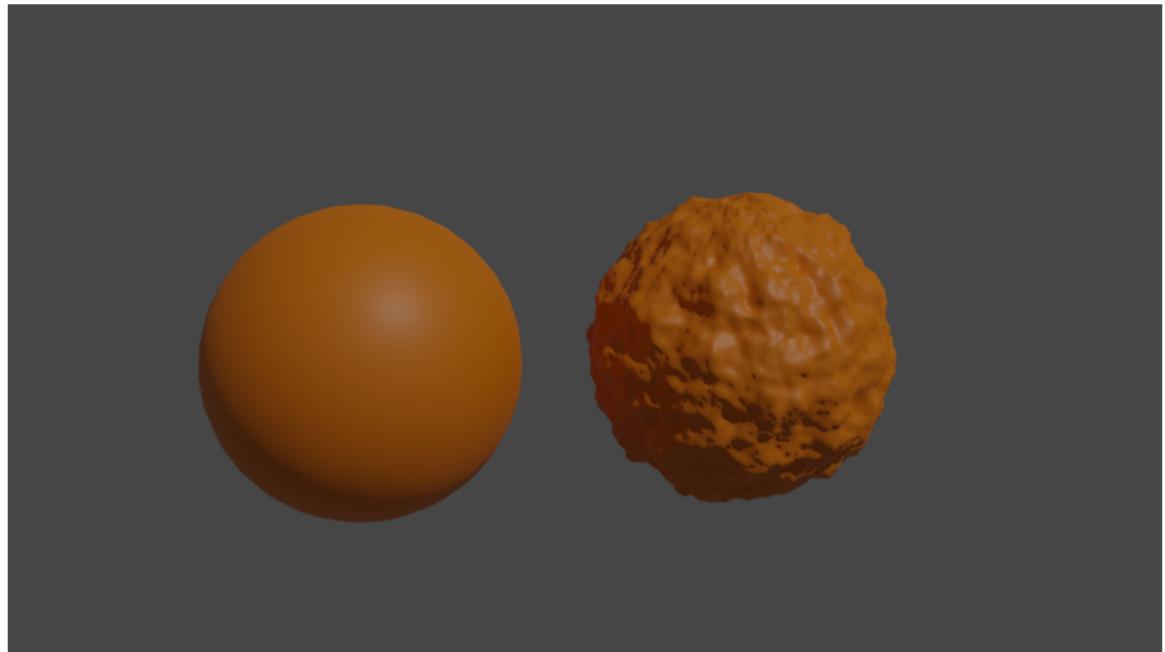
Компьютерная графика

Лекция 10: Normal mapping, material mapping, environment mapping, отражения, несколько источников света

2022

Normal mapping

- ▶ Хотим высокую детализацию поверхности объекта

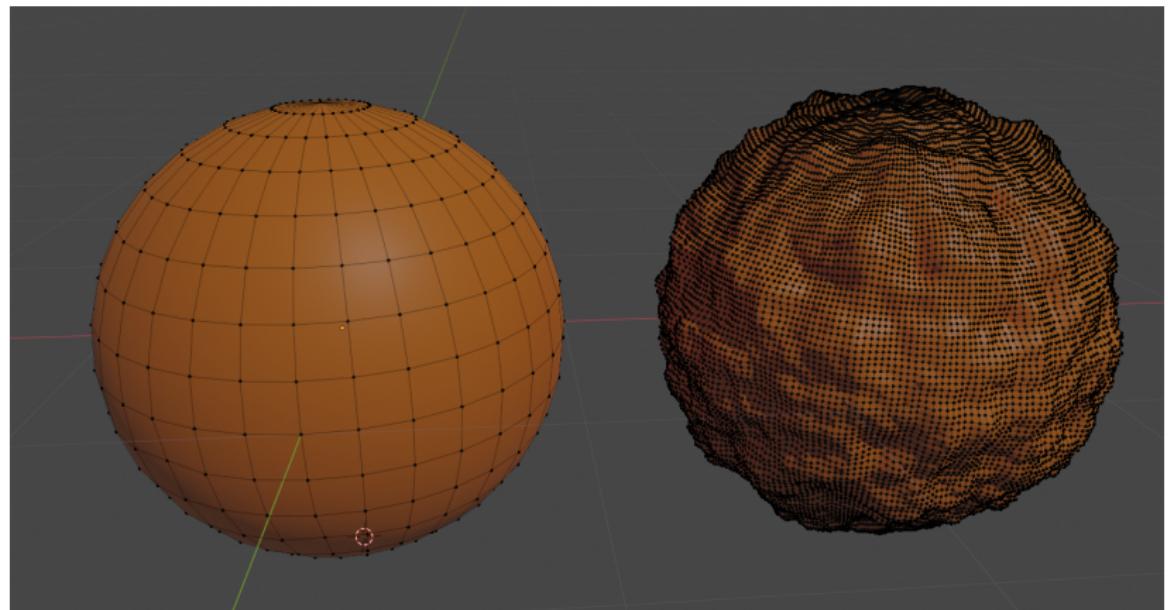


Normal mapping

- Можно взять очень много вершин

Normal mapping

- ▶ Можно взять очень много вершин
- ▶ Очень дорого, особенно когда объектов много



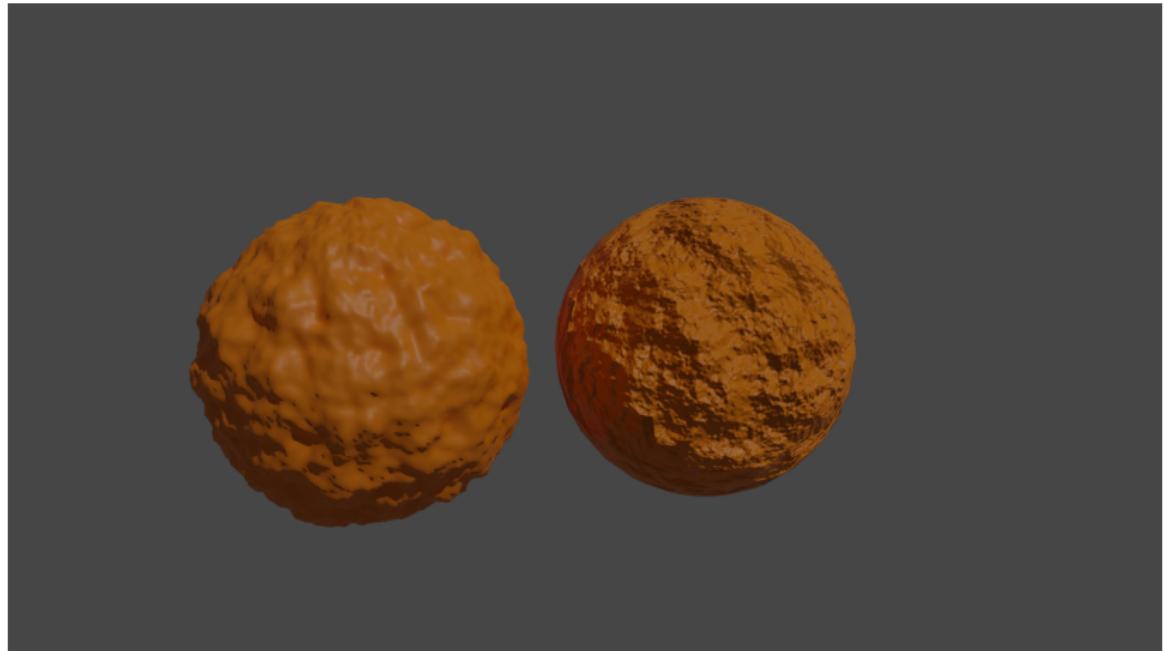
Normal mapping

- ▶ Наблюдение: высокая детализация даёт 2 вещи
 - ▶ Детализацию координат вершин – слабо заметна
 - ▶ Детализацию нормалей вершин – сильно заметна, так как влияет на освещение

Normal mapping

- ▶ Наблюдение: высокая детализация даёт 2 вещи
 - ▶ Детализацию координат вершин – слабо заметна
 - ▶ Детализацию нормалей вершин – сильно заметна, так как влияет на освещение
- ▶ Давайте откажемся от детализации координат!

Normal mapping



Normal mapping

- ▶ Normal mapping: используем текстуру, в которой закодированы нормали в точках объекта

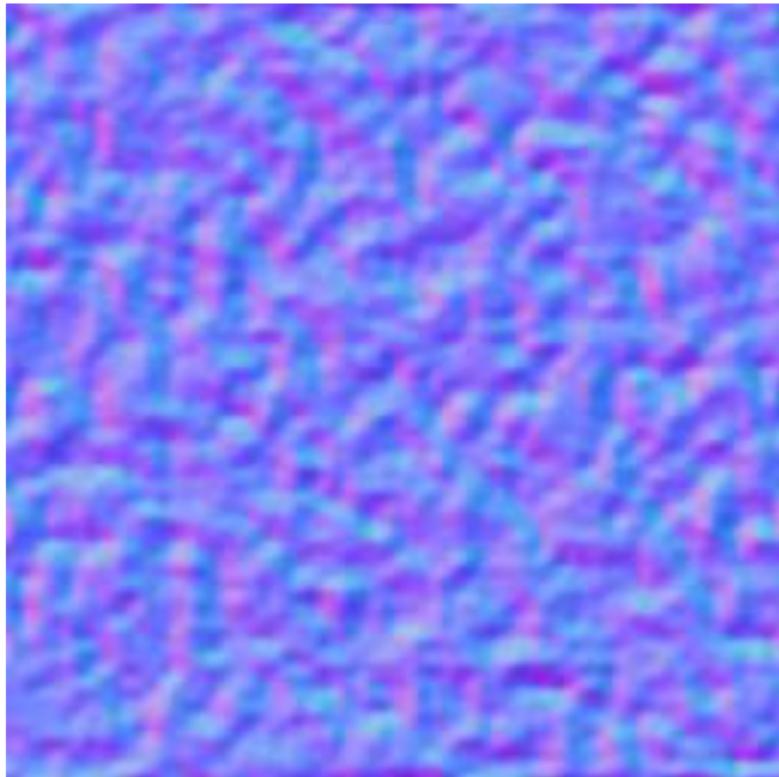
Normal mapping

- ▶ Normal mapping: используем текстуру, в которой закодированы нормали в точках объекта
- ▶ Вместо нормалей, приходящих в вершинах, достаём нормали из текстуры во фрагментном шейдере (используя те же текстурные координаты, что и для цвета)

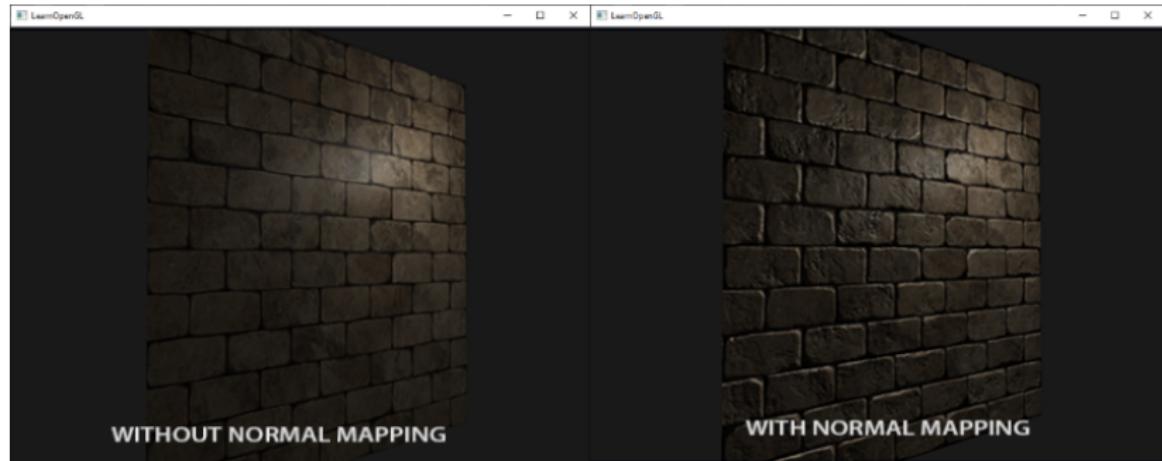
Normal mapping

- ▶ Normal mapping: используем текстуру, в которой закодированы нормали в точках объекта
- ▶ Вместо нормалей, приходящих в вершинах, достаём нормали из текстуры во фрагментном шейдере (используя те же текстурные координаты, что и для цвета)
- ▶ При этом можно использовать слабо детализированную геометрию

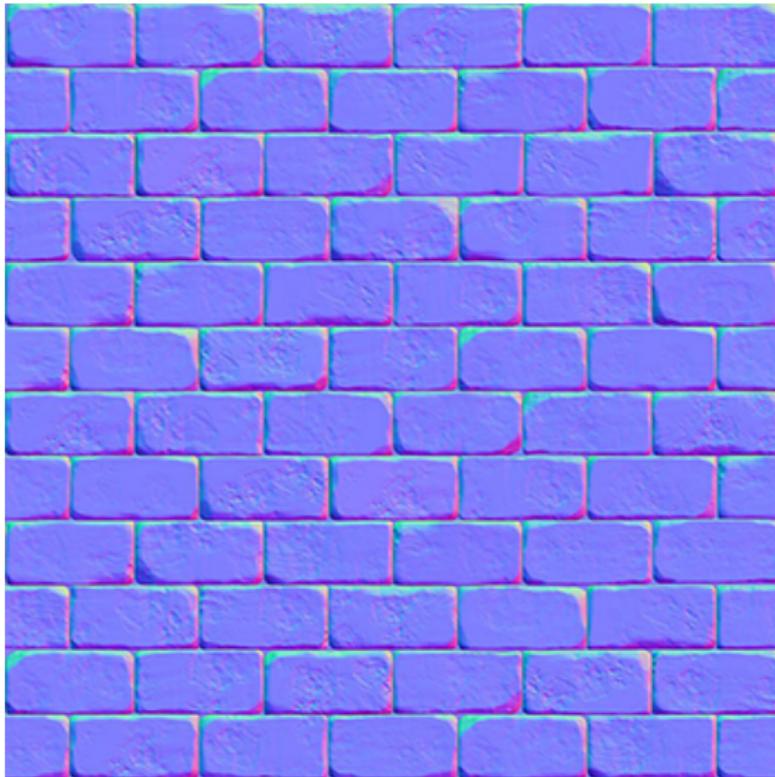
Normal mapping



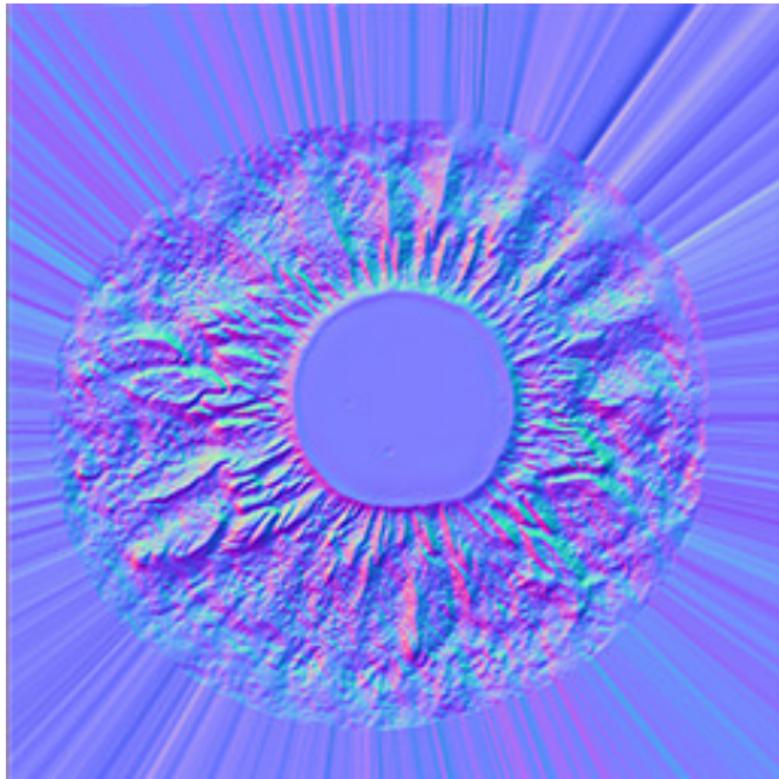
Normal mapping



Normal mapping



Normal mapping



Normal mapping: система координат

- ▶ В какой системе координат хранить нормали в текстуре?

Normal mapping: система координат

- ▶ В какой системе координат хранить нормали в текстуре?
- ▶ В системе координат объекта
 - ▶ + Нет лишних операций, просто читаем нормаль из текстуры (и, возможно, применяем `model` матрицу)
 - ▶ - Неудобно при подготовке данных: любое изменение модели (напр. слегка повернули часть объекта) нужно пересчитывать текстуру нормалей

Normal mapping: система координат

- ▶ В какой системе координат хранить нормали в текстуре?
- ▶ В системе координат объекта
 - ▶ + Нет лишних операций, просто читаем нормаль из текстуры (и, возможно, применяем *model* матрицу)
 - ▶ - Неудобно при подготовке данных: любое изменение модели (напр. слегка повернули часть объекта) нужно пересчитывать текстуру нормалей
- ▶ В локальной системе координат пикселя, учитывая геометрию объекта
 - ▶ - Нужно восстанавливать локальную систему координат в шейдере (используя атрибуты вершины)
 - ▶ + Можно сэкономить и хранить только две координаты нормали
 - ▶ + Обычно такая текстура с нормалями легче воспринимается на глаз
 - ▶ Наиболее распространённый способ
 - ▶ Такие текстуры часто выглядят синими, потому что направление Z считается нормалью вершины

Normal mapping: TBN

- ▶ Для второго способа (локальная система координат пикселя) нужно восстановить эту систему координат

Normal mapping: TBN

- ▶ Для второго способа (локальная система координат пикселя) нужно восстановить эту систему координат
- ▶ Она задаётся ортонормированным базисом из трёх векторов – tangent, bitangent, normal (TBN):

Normal mapping: TBN

- ▶ Для второго способа (локальная система координат пикселя) нужно восстановить эту систему координат
- ▶ Она задаётся ортонормированным базисом из трёх векторов – tangent, bitangent, normal (TBN):
 - ▶ Tangent – смотрит в направлении роста первой текстурной координаты вдоль поверхности
 - ▶ Bitangent – смотрит в направлении роста второй текстурной координаты вдоль поверхности
 - ▶ Normal – перпендикуляр к поверхности

Normal mapping: TBN

- ▶ Для второго способа (локальная система координат пикселя) нужно восстановить эту систему координат
- ▶ Она задаётся ортонормированным базисом из трёх векторов – tangent, bitangent, normal (TBN):
 - ▶ Tangent – смотрит в направлении роста первой текстурной координаты вдоль поверхности
 - ▶ Bitangent – смотрит в направлении роста второй текстурной координаты вдоль поверхности
 - ▶ Normal – перпендикуляр к поверхности
- ▶ Обычно их хранят в атрибутах вершин
- ▶ Достаточно любых двух векторов (обычно – normal + tangent), так как третий восстанавливается через векторное произведение

Normal mapping: формат хранения

- ▶ Как хранить нормали в текстуре?

Normal mapping: формат хранения

- ▶ Как хранить нормали в текстуре?
- ▶ Нормаль – 3х-компонентный нормированный вектор

Normal mapping: формат хранения

- ▶ Как хранить нормали в текстуре?
- ▶ Нормаль – 3х-компонентный нормированный вектор
- ▶ Floating-point текстуры (GL_RGB32F или GL_RGB16F)
 - ▶ – Занимают много места ($3 \times 16\text{-bit} = 6$ байт на пиксель)
 - ▶ – Излишняя точность
 - ▶ – Значительная часть возможных значений (координаты больше 1 по модулю) вообще не используется

Normal mapping: формат хранения

- ▶ Как хранить нормали в текстуре?
- ▶ Нормаль – 3х-компонентный нормированный вектор
- ▶ Floating-point текстуры (GL_RGB32F или GL_RGB16F)
 - ▶ – Занимают много места ($3 \times 16\text{-bit} = 6$ байт на пиксель)
 - ▶ – Излишняя точность
 - ▶ – Значительная часть возможных значений (координаты больше 1 по модулю) вообще не используется
- ▶ Signed normalized текстуры (GL_RGB8_SNORM)
 - ▶ + Компактные
 - ▶ + Достаточно точности
 - ▶ – Форматы изображений обычно не умеют работать с signed пикселями

Normal mapping: формат хранения

- ▶ Как хранить нормали в текстуре?
- ▶ Нормаль – 3х-компонентный нормированный вектор
- ▶ Floating-point текстуры (GL_RGB32F или GL_RGB16F)
 - ▶ – Занимают много места ($3 \times 16\text{-bit} = 6$ байт на пиксель)
 - ▶ – Излишняя точность
 - ▶ – Значительная часть возможных значений (координаты больше 1 по модулю) вообще не используется
- ▶ Signed normalized текстуры (GL_RGB8_SNORM)
 - ▶ + Компактные
 - ▶ + Достаточно точности
 - ▶ – Форматы изображений обычно не умеют работать с signed пикселями
- ▶ Обычные unsigned normalized текстуры (GL_RGB8, GL_RGB10_A2, GL_RGB565, GL_RGB5_A1, GL_R3_G3_B2)
 - ▶ + Компактные
 - ▶ + Достаточно точности
 - ▶ + Можно использовать обычные форматы изображений
 - ▶ Координаты хранятся в виде, отмасштабированном в диапазон [0, 1], т.е. $0.5 * (x + 1)$

Normal mapping: формат хранения

- ▶ Если нормаль всегда ориентирована локально ‘вверх’ (как в варианте с TBN), можно не хранить Z-компоненту нормали: её можно восстановить по XY как $\sqrt{1 - x^2 - y^2}$

Normal mapping: формат хранения

- ▶ Если нормаль всегда ориентирована локально ‘вверх’ (как в варианте с TBN), можно не хранить Z-компоненту нормали: её можно восстановить по XY как $\sqrt{1 - x^2 - y^2}$
- ▶ Для текстуры нормалей обычно имеет смысл включать линейную интерполяцию и мипмапы, но мипмапы могут работать неправильно

Normal mapping: формат хранения

- ▶ Если нормаль всегда ориентирована локально ‘вверх’ (как в варианте с TBN), можно не хранить Z-компоненту нормали: её можно восстановить по XY как $\sqrt{1 - x^2 - y^2}$
- ▶ Для текстуры нормалей обычно имеет смысл включать линейную интерполяцию и мипмапы, но мипмапы могут работать неправильно
- ▶ Нормаль, прочитанную из текстуры, лучше явно нормировать в шейдере (функцией `normalize`), так как она может быть не нормированной из-за неточностей формата хранения и линейной интерполяции, что приведёт к артефактам при расчёте `specular` освещения

Normal mapping: ссылки

- ▶ learnopengl.com/Advanced-Lighting/Normal-Mapping
- ▶ opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping
- ▶ sudonull.com/post/14500-Learn-OpenGL-Lesson-55-Normal-Mapping
- ▶ habr.com/ru/post/415579

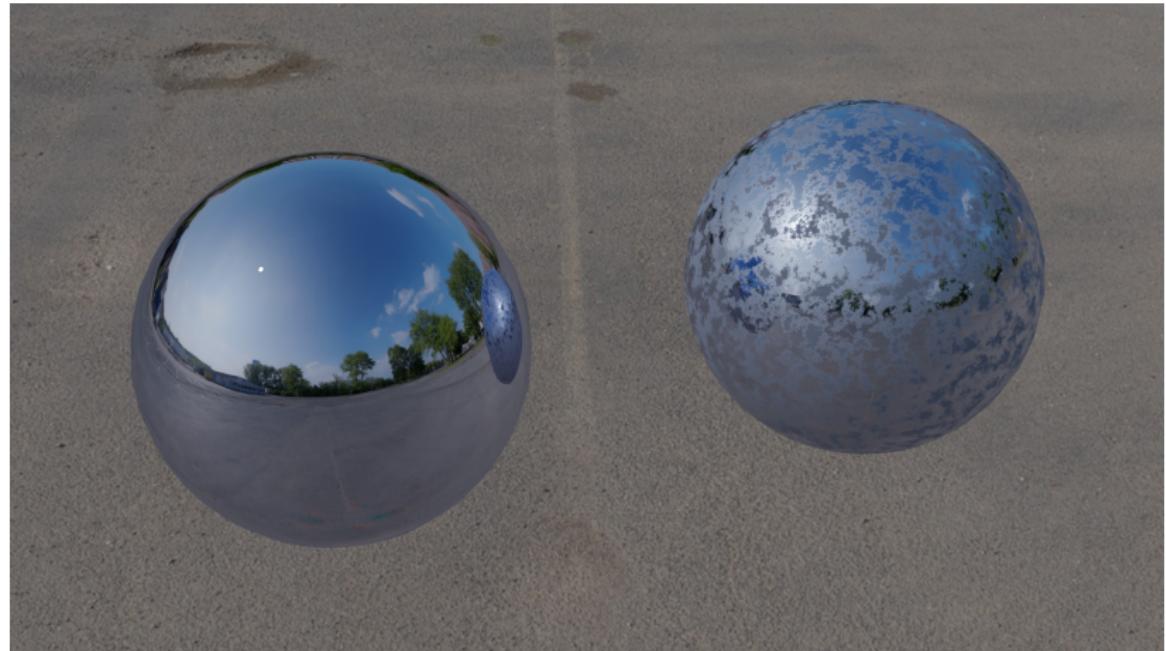
Material mapping

- ▶ В обработке освещения часто встречаются настраиваемые параметры, помимо цвета (альбедо) объекта (например, glossiness и roughness для specular освещения)

Material mapping

- ▶ В обработке освещения часто встречаются настраиваемые параметры, помимо цвета (альбедо) объекта (например, `glossiness` и `roughness` для `specular` освещения)
- ▶ Можно их тоже варьировать, читая из текстуры!

Material mapping



Environment mapping

- ▶ Одноцветный фон сцены – скучно, хочется нарисовать что-нибудь ‘вдалеке’

Environment mapping

- ▶ Одноцветный фон сцены – скучно, хочется нарисовать что-нибудь ‘в далёке’
- ▶ Тратить ресурсы на то, чтобы рисовать что-то очень далёкое, не хочется

Environment mapping

- ▶ Одноцветный фон сцены – скучно, хочется нарисовать что-нибудь ‘в далёке’
- ▶ Тратить ресурсы на то, чтобы рисовать что-то очень далёкое, не хочется
- ▶ Решение: статическая (обычно), заранее подготовленная картинка ‘environment map’ (облака, горы, etc.)

Environment mapping: как хранить?

- ▶ Environment map задаёт цвет ‘фона’ в каждом возможном направлении

Environment mapping: как хранить?

- ▶ Environment map задаёт цвет ‘фона’ в каждом возможном направлении
- ▶ Варианты:
 - ▶ Cubemap-текстура

Environment mapping: как хранить?

- ▶ Environment map задаёт цвет ‘фона’ в каждом возможном направлении
- ▶ Варианты:
 - ▶ Cubemap-текстура
 - ▶ Обычная 2D текстура с равнопромежуточной проекцией (текстурные координаты соответствуют ‘долготе’ и ‘широте’ направления взгляда)

Environment mapping: как хранить?

- ▶ Environment map задаёт цвет ‘фона’ в каждом возможном направлении
- ▶ Варианты:
 - ▶ Сиветар-текстура
 - ▶ Обычная 2D текстура с равнопромежуточной проекцией (текстурные координаты соответствуют ‘долготе’ и ‘широте’ направления взгляда)
 - ▶ Две 2D текстуры с параболической/fisheye проекцией

Environment map



Environment mapping: как рисовать?

- ▶ Первый вариант:

Environment mapping: как рисовать?

- ▶ Первый вариант:
 - ▶ Нарисовать куб вокруг камеры

Environment mapping: как рисовать?

- ▶ Первый вариант:
 - ▶ Нарисовать куб вокруг камеры
 - ▶ По координате точки и координате камеры получить направление взгляда

Environment mapping: как рисовать?

- ▶ Первый вариант:
 - ▶ Нарисовать куб вокруг камеры
 - ▶ По координате точки и координате камеры получить направление взгляда
 - ▶ Использовать направление для обращения к environment map

Environment mapping: как рисовать?

- ▶ Первый вариант:
 - ▶ Нарисовать куб вокруг камеры
 - ▶ По координате точки и координате камеры получить направление взгляда
 - ▶ Использовать направление для обращения к environment map
- ▶ Второй вариант:
 - ▶ Нарисовать полноэкранный прямоугольник

Environment mapping: как рисовать?

- ▶ Первый вариант:
 - ▶ Нарисовать куб вокруг камеры
 - ▶ По координате точки и координате камеры получить направление взгляда
 - ▶ Использовать направление для обращения к environment map
- ▶ Второй вариант:
 - ▶ Нарисовать полноэкранный прямоугольник
 - ▶ Используя обратную view+projection матрицу получить координаты этого прямоугольника в пространстве

Environment mapping: как рисовать?

- ▶ Первый вариант:
 - ▶ Нарисовать куб вокруг камеры
 - ▶ По координате точки и координате камеры получить направление взгляда
 - ▶ Использовать направление для обращения к environment map
- ▶ Второй вариант:
 - ▶ Нарисовать полноэкранный прямоугольник
 - ▶ Используя обратную view+projection матрицу получить координаты этого прямоугольника в пространстве
 - ▶ По координате точки и координате камеры получить направление взгляда

Environment mapping: как рисовать?

- ▶ Первый вариант:
 - ▶ Нарисовать куб вокруг камеры
 - ▶ По координате точки и координате камеры получить направление взгляда
 - ▶ Использовать направление для обращения к environment map
- ▶ Второй вариант:
 - ▶ Нарисовать полноэкранный прямоугольник
 - ▶ Используя обратную view+projection матрицу получить координаты этого прямоугольника в пространстве
 - ▶ По координате точки и координате камеры получить направление взгляда
 - ▶ Использовать направление для обращения к environment map

Environment mapping: как рисовать?

- ▶ Первый вариант:
 - ▶ Нарисовать куб вокруг камеры
 - ▶ По координате точки и координате камеры получить направление взгляда
 - ▶ Использовать направление для обращения к environment map
- ▶ Второй вариант:
 - ▶ Нарисовать полноэкранный прямоугольник
 - ▶ Используя обратную view+projection матрицу получить координаты этого прямоугольника в пространстве
 - ▶ По координате точки и координате камеры получить направление взгляда
 - ▶ Использовать направление для обращения к environment map
- ▶ Environment map нужно рисовать первым объектом сцены, без теста глубины

Environment/reflection mapping

- ▶ Рисовать честные отражения в real-time очень сложно

Environment/reflection mapping

- ▶ Рисовать честные отражения в real-time очень сложно
- ▶ Дешёвая аппроксимация: будем учитывать в отражениях только environment map

Environment/reflection mapping

- ▶ Рисовать честные отражения в real-time очень сложно
- ▶ Дешёвая аппроксимация: будем учитывать в отражениях только environment map
- ▶ Такая техника известна под названиями reflection mapping или environment mapping

Environment/reflection mapping

- ▶ Рисовать честные отражения в real-time очень сложно
- ▶ Дешёвая аппроксимация: будем учитывать в отражениях только environment map
- ▶ Такая техника известна под названиями reflection mapping или environment mapping
- ▶ + Просто в реализации
- ▶ + Быстро работает
- ▶ - Не позволяет учесть динамические объекты

Environment/reflection mapping

- ▶ Рисовать честные отражения в real-time очень сложно
- ▶ Дешёвая аппроксимация: будем учитывать в отражениях только environment map
- ▶ Такая техника известна под названиями reflection mapping или environment mapping
- ▶ + Просто в реализации
- ▶ + Быстро работает
- ▶ - Не позволяет учесть динамические объекты
- ▶ Хорошо работает для плохо заметных отражений: дверные ручки, гильзы от пуль, ржавые трубы, etc

Reflection mapping: реализация

- ▶ Во фрагментном шейдере вычисляем направление луча из камеры в точку

Reflection mapping: реализация

- ▶ Во фрагментном шейдере вычисляем направление луча из камеры в точку
- ▶ Вычисляем направление отражённого луча

Reflection mapping: реализация

- ▶ Во фрагментном шейдере вычисляем направление луча из камеры в точку
- ▶ Вычисляем направление отражённого луча
- ▶ Используем для чтения из environment map

Reflection mapping: реализация

- ▶ Во фрагментном шейдере вычисляем направление луча из камеры в точку
- ▶ Вычисляем направление отражённого луча
- ▶ Используем для чтения из environment map
- ▶ Можно явно использовать тирмап'ы environment map'a чтобы аппроксимировать размытое отражение (roughness)

Reflection mapping



Environment/reflection mapping: ссылки

- ▶ learnopengl.com/Advanced-OpenGL/Cubemaps

Отражения

- ▶ Расчёт отражений в real-time – очень сложная задача

Отражения

- ▶ Расчёт отражений в real-time – очень сложная задача
- ▶ Для плоских зеркал можно нарисовать сцену целиком, применив отражающее преобразование и нарисовав зеркало предварительно в stencil буфер, чтобы перевёрнутая сцена нарисовалась только там, где находится зеркало

Отражения

- ▶ Расчёт отражений в real-time – очень сложная задача
- ▶ Для плоских зеркал можно нарисовать сцену целиком, применив отражающее преобразование и нарисовав зеркало предварительно в stencil буфер, чтобы перевёрнутая сцена нарисовалась только там, где находится зеркало
 - ▶ — Рисовать заново всю сцену – дорого
 - ▶ — Не работает для неплоских зеркал

Отражения

- ▶ Расчёт отражений в real-time – очень сложная задача
- ▶ Для плоских зеркал можно нарисовать сцену целиком, применив отражающее преобразование и нарисовав зеркало предварительно в stencil буфер, чтобы перевёрнутая сцена нарисовалась только там, где находится зеркало
 - ▶ – Рисовать заново всю сцену – дорого
 - ▶ – Не работает для неплоских зеркал
- ▶ Можно нарисовать сцену целиком, вычисляя проекцию объекта на зеркало в вершинном шейдере
 - ▶ – Дорого
 - ▶ – Будут артефакты с растягиванием вершин (зависит от формы зеркала)

Отражения

- ▶ Апроксимация: использовать ту же идею, что и в reflection mapping'e, но вместо заранее подготовленного environment map нарисуем сцену в текстуру

Отражения

- ▶ Аппроксимация: использовать ту же идею, что и в reflection mapping'e, но вместо заранее подготовленного environment map нарисуем сцену в текстуру
 - ▶ - Дорого (рисуем всю сцену ещё раз)
 - ▶ - Отражения не зависят от точки, в которой происходит отражение
 - ▶ + Обычно артефакты не очень заметны

Cubemap отражения

- ▶ Удобнее всего использовать cubemap-текстуру

Cubemap отражения

- ▶ Удобнее всего использовать cubemap-текстуру
- ▶ Алгоритм:
 - ▶ Выбираем точку, из которой будет строиться текстура для отражений (обычно – позиция камеры или игрока)

Сиветар отражения

- ▶ Удобнее всего использовать сиветар-текстуру
- ▶ Алгоритм:
 - ▶ Выбираем точку, из которой будет строиться текстура для отражений (обычно – позиция камеры или игрока)
 - ▶ Рисуем сцену в сиветар-текстуру камерой, находящейся в этой точке (как в shadow mapping от точечного источника света)

Сиветар отражения

- ▶ Удобнее всего использовать сиветар-текстуру
- ▶ Алгоритм:
 - ▶ Выбираем точку, из которой будет строиться текстура для отражений (обычно – позиция камеры или игрока)
 - ▶ Рисуем сцену в сиветар-текстуру камерой, находящейся в этой точке (как в shadow mapping от точечного источника света)
 - ▶ При рисовании сцены на экран вычисляем отражённый луч из камеры (как в reflection mapping)

Сиветар отражения

- ▶ Удобнее всего использовать сиветар-текстуру
- ▶ Алгоритм:
 - ▶ Выбираем точку, из которой будет строиться текстура для отражений (обычно – позиция камеры или игрока)
 - ▶ Рисуем сцену в сиветар-текстуру камерой, находящейся в этой точке (как в shadow mapping от точечного источника света)
 - ▶ При рисовании сцены на экран вычисляем отражённый луч из камеры (как в reflection mapping)
 - ▶ Читаем значение из сиветар-текстуры по этому направлению – это отражённый цвет

Сиветар отражения

- ▶ Удобнее всего использовать сиветар-текстуру
- ▶ Алгоритм:
 - ▶ Выбираем точку, из которой будет строиться текстура для отражений (обычно – позиция камеры или игрока)
 - ▶ Рисуем сцену в сиветар-текстуру камерой, находящейся в этой точке (как в shadow mapping от точечного источника света)
 - ▶ При рисовании сцены на экран вычисляем отражённый луч из камеры (как в reflection mapping)
 - ▶ Читаем значение из сиветар-текстуры по этому направлению – это отражённый цвет
- ▶ Как в reflection mapping, можно использовать тиртарт'ы или размыть сиветар чтобы имитировать нечёткие отражения

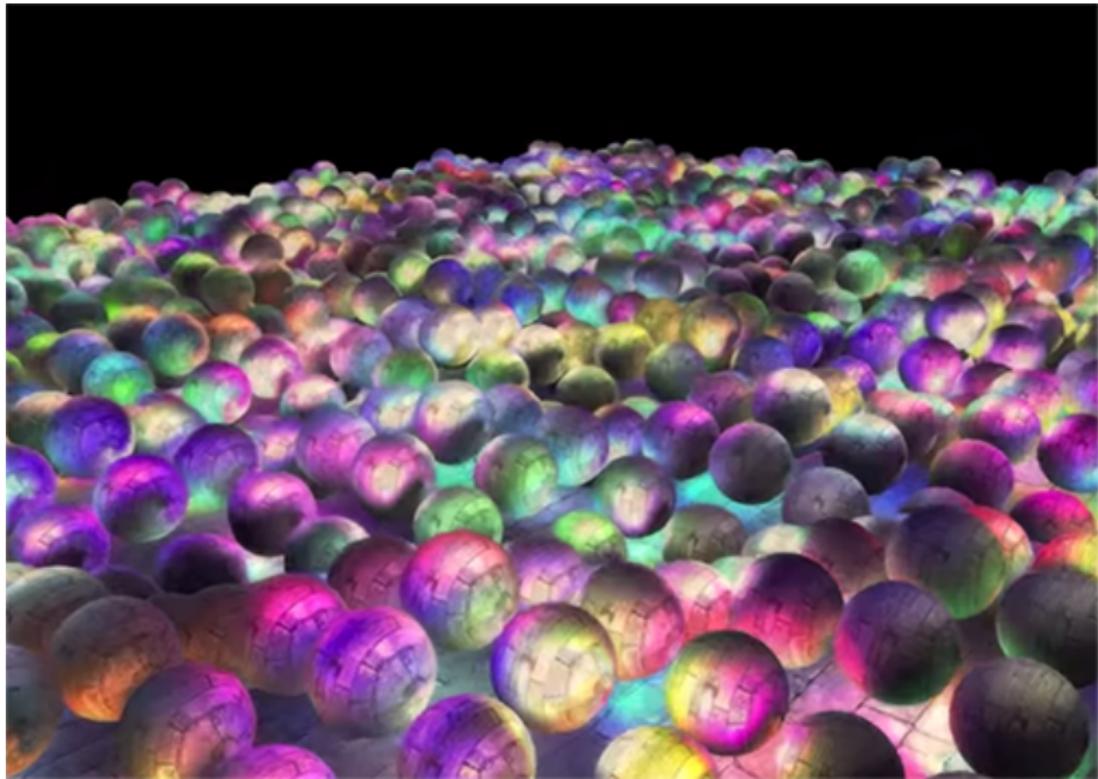
Сиветар отражения

- ▶ Удобнее всего использовать сиветар-текстуру
- ▶ Алгоритм:
 - ▶ Выбираем точку, из которой будет строиться текстура для отражений (обычно – позиция камеры или игрока)
 - ▶ Рисуем сцену в сиветар-текстуру камерой, находящейся в этой точке (как в shadow mapping от точечного источника света)
 - ▶ При рисовании сцены на экран вычисляем отражённый луч из камеры (как в reflection mapping)
 - ▶ Читаем значение из сиветар-текстуры по этому направлению – это отражённый цвет
- ▶ Как в reflection mapping, можно использовать тіртар'ы или размыть сиветар чтобы имитировать нечёткие отражения
- ▶ Можно сэкономить, взяв сиветар низкого качества, и рисуя в него только крупные объекты

Субмар отражения (GTA 5)



Много источников света



Как обрабатывать много источников света?

- ▶ Простейший способ – использовать по отдельному набору uniform-переменных для каждого источника света, и обрабатывать их все сразу в шейдере

Как обрабатывать много источников света?

- ▶ Простейший способ – использовать по отдельному набору uniform-переменных для каждого источника света, и обрабатывать их все сразу в шейдере
- ▶ Можно использовать массивы uniform-переменных

Как обрабатывать много источников света?

- ▶ Простейший способ – использовать по отдельному набору uniform-переменных для каждого источника света, и обрабатывать их все сразу в шейдере
- ▶ Можно использовать массивы uniform-переменных
 - ▶ Задаются как `uniform vec3 light_position[N]` (N – константа)
 - ▶ Каждый элемент массива – отдельная uniform-переменная, для неё нужно отдельно вызвать `glGetUniformLocation`
 - ▶ Имя этой переменной – имя массива + индекс, например `light_position[0]`

Несколько источников света

- ▶ Сколько источников света можно так обработать?

Несколько источников света

- ▶ Сколько источников света можно так обработать?
- ▶ OpenGL гарантирует, что под uniform-переменные есть как минимум 1024 выделенных компоненты (vec3 занимает 3 компоненты, и т.п.)

Несколько источников света

- ▶ Сколько источников света можно так обработать?
- ▶ OpenGL гарантирует, что под uniform-переменные есть как минимум 1024 выделенных компоненты (vec3 занимает 3 компоненты, и т.п.)
- ▶ В варианте 3 vec3 на источник (position, color, attenuation)
 - около 113 источников света

Несколько источников света

- ▶ Сколько источников света можно так обработать?
- ▶ OpenGL гарантирует, что под uniform-переменные есть как минимум 1024 выделенных компоненты (vec3 занимает 3 компоненты, и т.п.)
- ▶ В варианте 3 vec3 на источник (position, color, attenuation) – около 113 источников света
- ▶ Чем больше источников, тем больше вычислений в шейдере, тем больше нагрузка на GPU

Как обрабатывать много источников света?

- ▶ Другой способ: рисовать сцену по одному разу на каждый источник света с аддитивным blending'ом, 'добавляя' вклад каждого источника света

Как обрабатывать много источников света?

- ▶ Другой способ: рисовать сцену по одному разу на каждый источник света с аддитивным blending'ом, 'добавляя' вклад каждого источника света
- ▶ Очень дорого:
 - ▶ Многократно повторяются одни и те же вычисления в вершинных шейдерах
 - ▶ Многократно читаются одни и те же данные атрибутов вершин, текстур, и т.д.

Несколько источников света

- ▶ Оба подхода выполняют много лишних операций:
 - ▶ Вычисляется полное освещение для объекта, который может быть скрыт другим объектом

Несколько источников света

- ▶ Оба подхода выполняют много лишних операций:
 - ▶ Вычисляется полное освещение для объекта, который может быть скрыт другим объектом
 - ▶ Вычисляется вклад в освещение для объекта, находящегося слишком далеко от источника света

Несколько источников света

- ▶ Оба подхода выполняют много лишних операций:
 - ▶ Вычисляется полное освещение для объекта, который может быть скрыт другим объектом
 - ▶ Вычисляется вклад в освещение для объекта, находящегося слишком далеко от источника света
- ▶ Можно частично решить, вычисляя для каждого объекта список источников, которыми он освещён ⇒ больше изменений uniform-переменных при рендеринге, меньше возможность для batching'а (объединения объектов в группы или в общие буферы)

Несколько источников света

- ▶ Оба подхода выполняют много лишних операций:
 - ▶ Вычисляется полное освещение для объекта, который может быть скрыт другим объектом
 - ▶ Вычисляется вклад в освещение для объекта, находящегося слишком далеко от источника света
- ▶ Можно частично решить, вычисляя для каждого объекта список источников, которыми он освещён ⇒ больше изменений uniform-переменных при рендеринге, меньше возможность для batching'а (объединения объектов в группы или в общие буферы)
- ▶ Более современные решения: deferred shading, tiled/clustered shading

Deferred shading

- ▶ Идея: вместо рисования сцены на экран, сначала нарисуем её (с точки зрения камеры) в набор буферов, вместе называющийся G-buffer, содержащих
 - ▶ Цвет пикселей (альбедо)
 - ▶ Нормали пикселей
 - ▶ Материалы пикселей
 - ▶ Позиции пикселей в мировой системе координат
 - ▶ И т.п.

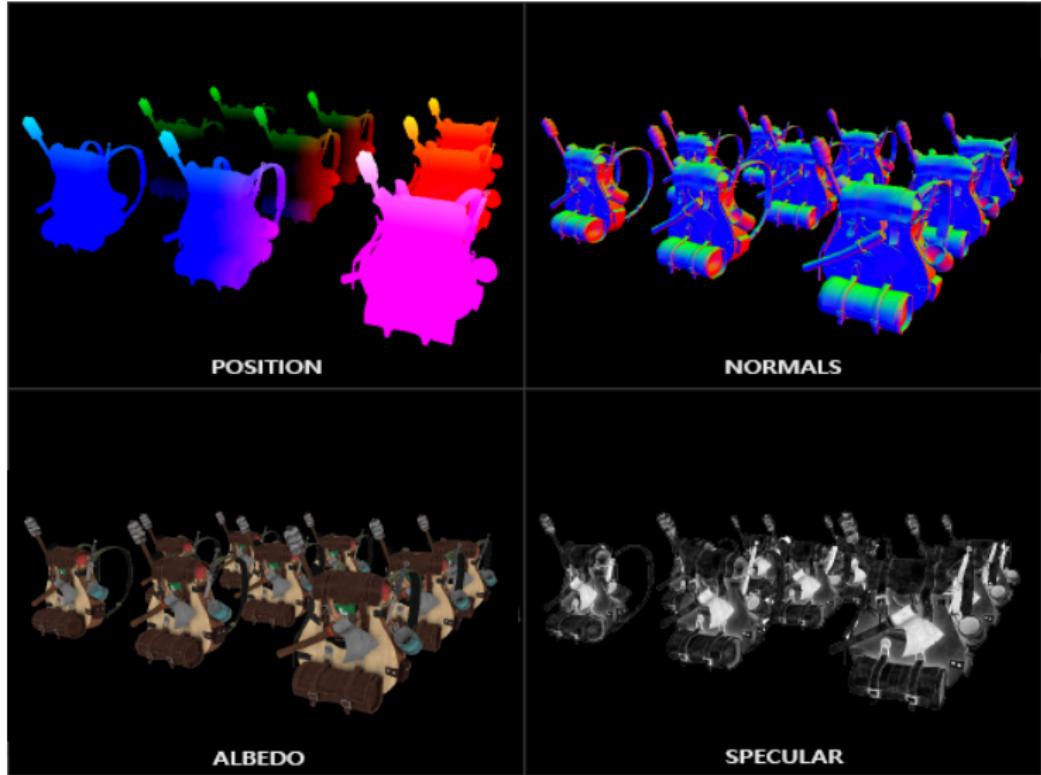
Deferred shading

- ▶ Идея: вместо рисования сцены на экран, сначала нарисуем её (с точки зрения камеры) в набор буферов, вместе называющийся G-buffer, содержащих
 - ▶ Цвет пикселей (альбедо)
 - ▶ Нормали пикселей
 - ▶ Материалы пикселей
 - ▶ Позиции пикселей в мировой системе координат
 - ▶ И т.п.
- ▶ После этого вычисляем освещение любым способом: одним большим шейдером на все источники, или аддитивным blending'ом по одному

Deferred shading

- ▶ Идея: вместо рисования сцены на экран, сначала нарисуем её (с точки зрения камеры) в набор буферов, вместе называющийся G-buffer, содержащих
 - ▶ Цвет пикселей (альбедо)
 - ▶ Нормали пикселей
 - ▶ Материалы пикселей
 - ▶ Позиции пикселей в мировой системе координат
 - ▶ И т.п.
- ▶ После этого вычисляем освещение любым способом: одним большим шейдером на все источники, или аддитивным blending'ом по одному
- ▶ В варианте с blending'ом можно вычислять освещение только для пикселей, расположенных рядом с источником света

Deferred shading: G-buffer



Deferred shading

- ▶ + Позволяет учесть тысячи источников света
- ▶ + После применения освещения остаётся много информации (G-buffer), которую можно использовать для других эффектов (SSAO, screen-space reflections, ...)
- ▶ + Не вычисляет освещение для объектов, скрытых другими объектами
- ▶ + Есть много вариантов оптимизации хранения G-buffer'а (сжимать нормали, восстанавливать позицию по глубине)
- ▶ - Не позволяет учесть освещённые полупрозрачные объекты (на каждый слой прозрачных объектов нужен свой G-buffer)
- ▶ - Очень много операций записи памяти (в G-buffer) на каждый кадр

Tiled/clustered shading

- ▶ Идея: разобьём видимую область на кусочки
 - ▶ Tiled: экран разбивается на маленькие 'тайлы', например 8x8 пикселей
 - ▶ Clustered: трёхмерная видимая область (view frustum) разбивается на 'кластеры', например 16x8x24 (всего 3072 кластера) по X, Y и Z соответственно

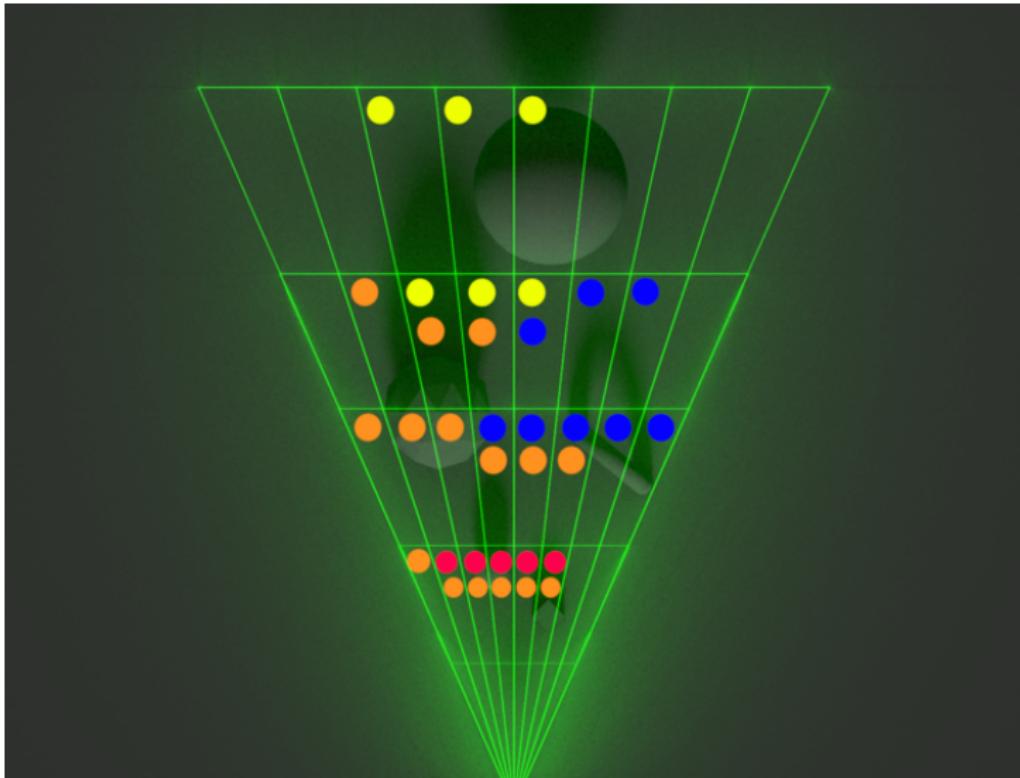
Tiled/clustered shading

- ▶ Идея: разобьём видимую область на кусочки
 - ▶ Tiled: экран разбивается на маленькие 'тайлы', например 8x8 пикселей
 - ▶ Clustered: трёхмерная видимая область (view frustum) разбивается на 'кластеры', например 16x8x24 (всего 3072 кластера) по X, Y и Z соответственно
- ▶ Для каждого тайла/кластера вычисляем список источников света, влияющих на объекты в этом тайле/кластере

Tiled/clustered shading

- ▶ Идея: разобьём видимую область на кусочки
 - ▶ Tiled: экран разбивается на маленькие 'тайлы', например 8x8 пикселей
 - ▶ Clustered: трёхмерная видимая область (view frustum) разбивается на 'кластеры', например 16x8x24 (всего 3072 кластера) по X, Y и Z соответственно
- ▶ Для каждого тайла/кластера вычисляем список источников света, влияющих на объекты в этом тайле/кластере
- ▶ Рисуем сцену как обычно; при расчёте освещения вычисляем, в каком мы находимся тайле/кластере, и читаем оттуда список источников света

Clustered shading



Tiled/clustered shading

- ▶ Хранить список источников можно в buffer textures или в shader storage buffer objects (OpenGL 4.3)

Tiled/clustered shading

- ▶ Хранить список источников можно в buffer textures или в shader storage buffer objects (OpenGL 4.3)
- ▶ Вычислять список источников для каждого тайла/кластера можно на CPU или на GPU (compute shaders)

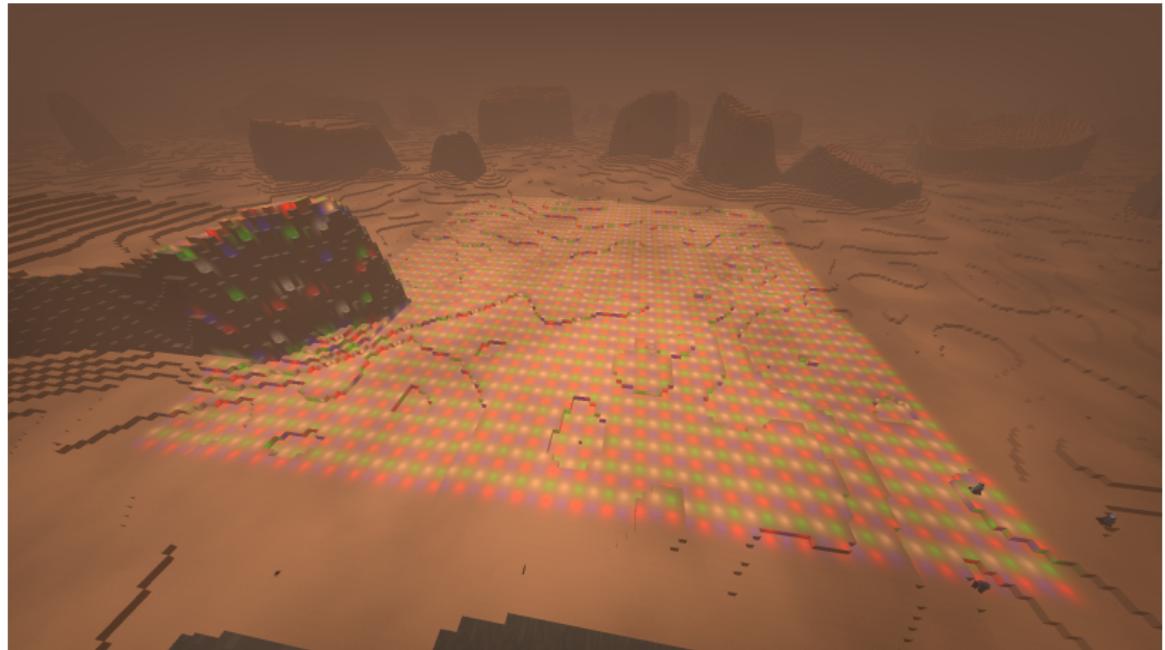
Tiled/clustered shading

- ▶ Хранить список источников можно в buffer textures или в shader storage buffer objects (OpenGL 4.3)
- ▶ Вычислять список источников для каждого тайла/кластера можно на CPU или на GPU (compute shaders)
- ▶ На GPU быстрее, но сложнее реализация и формат хранения

Tiled/clustered shading

- ▶ Хранить список источников можно в buffer textures или в shader storage buffer objects (OpenGL 4.3)
- ▶ Вычислять список источников для каждого тайла/кластера можно на CPU или на GPU (compute shaders)
- ▶ На GPU быстрее, но сложнее реализация и формат хранения
- ▶ Даже на CPU позволяет обрабатывать тысячи источников света

Clustered shading



Tiled/clustered shading

- ▶ + Позволяет учесть тысячи источников света
- ▶ + Позволяет рисовать прозрачные освещённые объекты
- ▶ - Вычисляет освещение и для объектов, скрытых другими объектами

Несколько источников света

- ▶ learnopengl.com/Lighting/Multiple-lights
- ▶ en.wikibooks.org/wiki/GLSL_Programming/GLUT/Multiple_Lights
- ▶ learnopengl.com/Advanced-Lighting/Deferred-Shading
- ▶ www.aortiz.me/2018/12/21/CG.html