

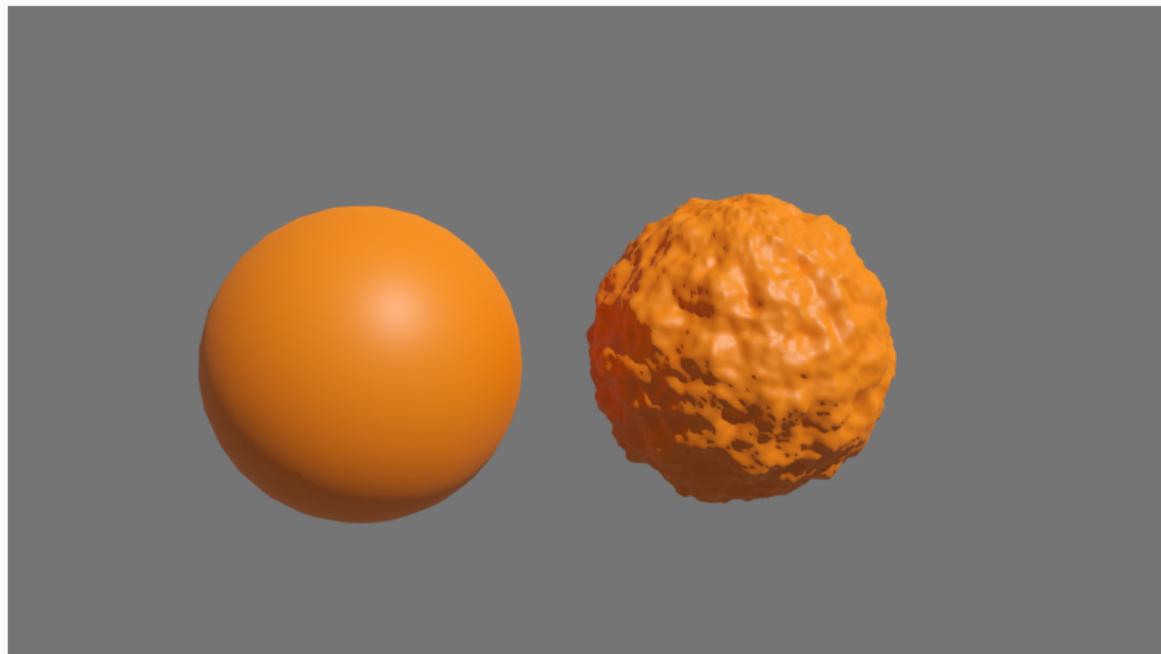
Компьютерная графика

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

2023

Normal mapping

- Хотим высокую детализацию поверхности объекта

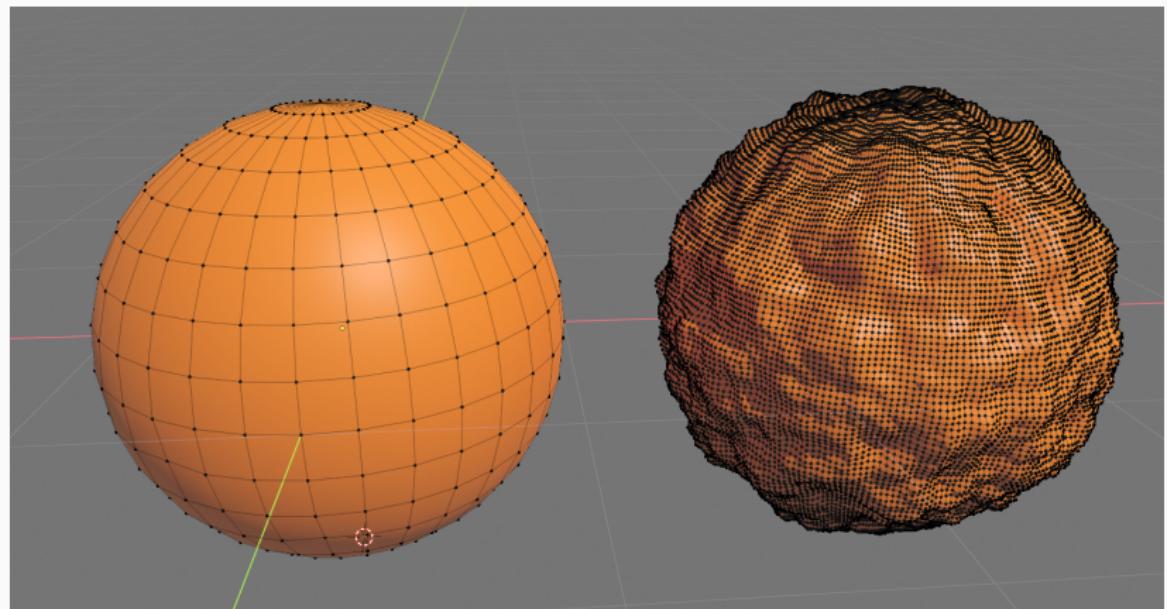


Normal mapping

- Можно взять очень много вершин

Normal mapping

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



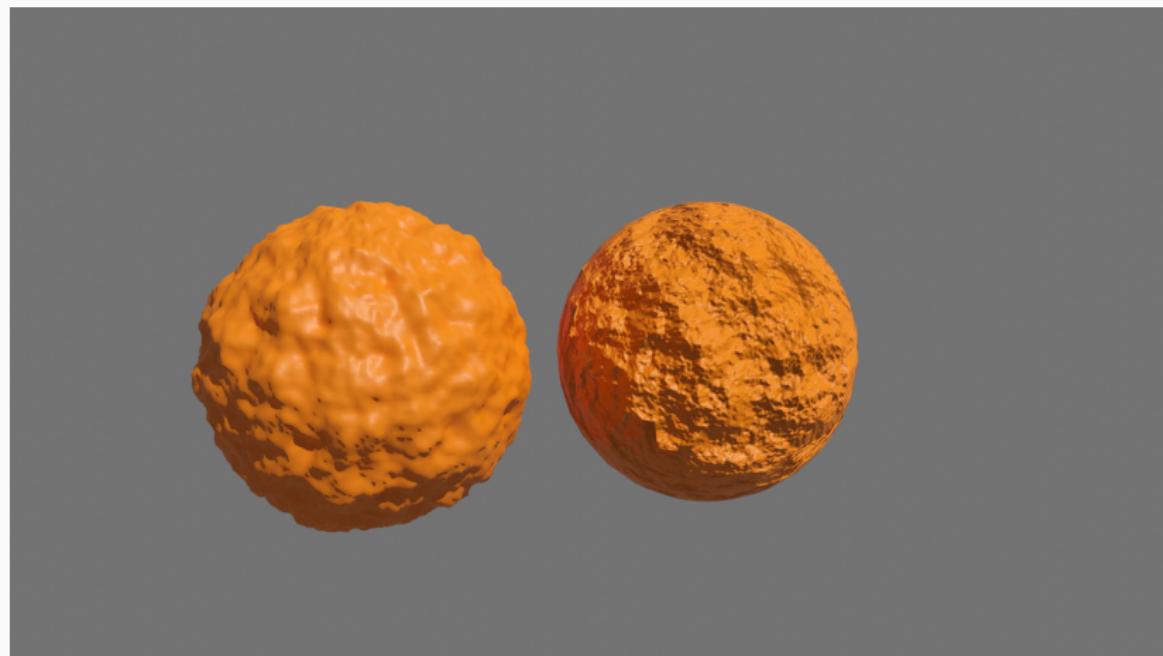
Normal mapping

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

Normal mapping

- Наблюдение: высокая детализация даёт 2 вещи
 - Детализацию геометрии – слабо заметна
 - Детализацию нормалей – 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

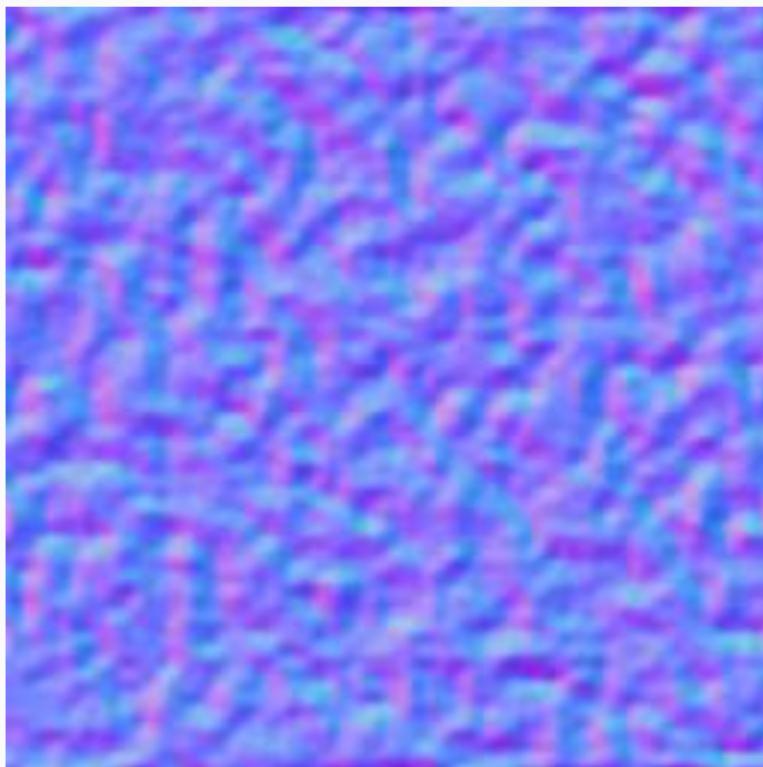
Normal mapping

- *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

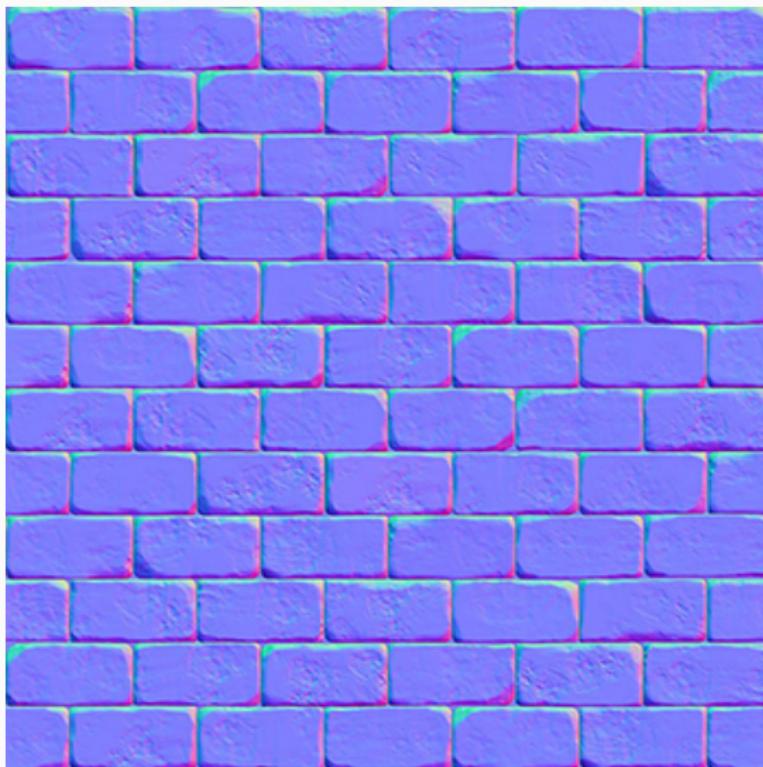
Normal mapping



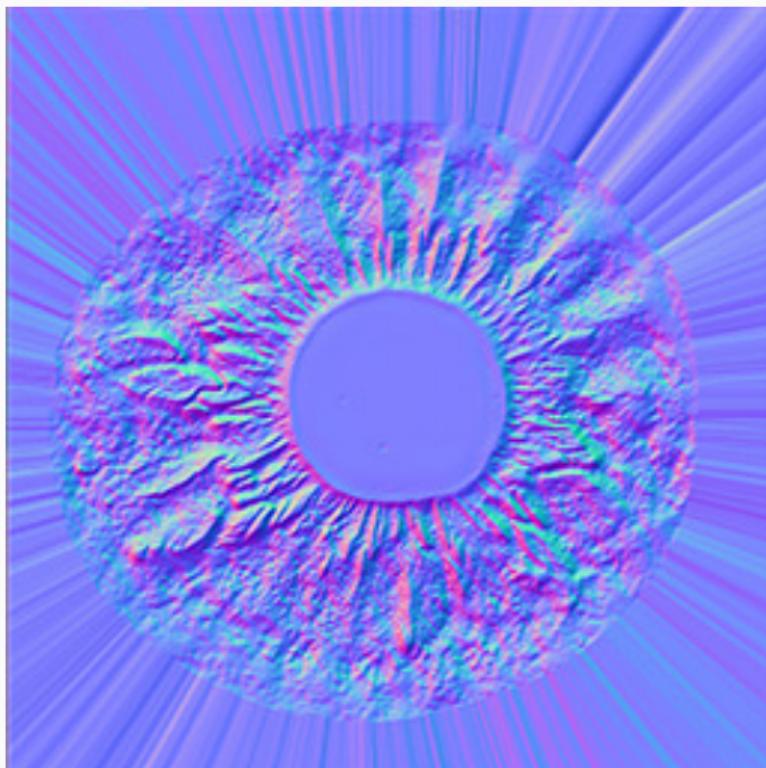
Normal mapping



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) влечёт пересчёт текстуры 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) влечёт пересчёт текстуры нормалей
- В локальной системе координат пикселя, учитывая геометрию объекта
 - — Нужно восстанавливать локальную систему координат точки в шейдере
 - + Можно сэкономить и хранить только две координаты нормали
 - + Такая текстура 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: TBN и shader derivatives

- Есть способ восстановить TBN используя $dFdx$ & $dFdy$:

Normal mapping: TBN и shader derivatives

- Есть способ восстановить TBN используя $dFdx$ & $dFdy$:
- Нас интересует производная первой текстурной координаты по мировым координатам точки (т.е. в каком направлении растёт эта координата)

Normal mapping: TBN и shader derivatives

- Есть способ восстановить TBN используя $dFdx$ & $dFdy$:
- Нас интересует производная первой текстурной координаты по мировым координатам точки (т.е. в каком направлении растёт эта координата)
- Через $dFdy$ & $dFdy$ мы можем получить производную текстурной координаты по пикселям экрана и производную мировых координат по пикселям экрана

Normal mapping: TBN и shader derivatives

- Есть способ восстановить TBN используя $dFdx$ & $dFdy$:
- Нас интересует производная первой текстурной координаты по мировым координатам точки (т.е. в каком направлении растёт эта координата)
- Через $dFdy$ & $dFdy$ мы можем получить производную текстурной координаты по пикселям экрана и производную мировых координат по пикселям экрана
- Мы знаем $\frac{\partial A}{\partial C}$ и $\frac{\partial B}{\partial C}$, можем вычислить $\frac{\partial A}{\partial B} = \frac{\partial A}{\partial C} \cdot \left(\frac{\partial B}{\partial C}\right)^{-1}$

Normal mapping: TBN и shader derivatives

- Есть способ восстановить TBN используя $dFdx$ & $dFdy$:
- Нас интересует производная первой текстурной координаты по мировым координатам точки (т.е. в каком направлении растёт эта координата)
- Через $dFdy$ & $dFdy$ мы можем получить производную текстурной координаты по пикселям экрана и производную мировых координат по пикселям экрана
- Мы знаем $\frac{\partial A}{\partial C}$ и $\frac{\partial B}{\partial C}$, можем вычислить $\frac{\partial A}{\partial B} = \frac{\partial A}{\partial C} \cdot \left(\frac{\partial B}{\partial C}\right)^{-1}$
- Сводится к операциям над матрицами 2×2 в шейдере

Normal mapping: TBN и shader derivatives

- Есть способ восстановить TBN используя $dFdx$ & $dFdy$:
- Нас интересует производная первой текстурной координаты по мировым координатам точки (т.е. в каком направлении растёт эта координата)
- Через $dFdy$ & $dFdy$ мы можем получить производную текстурной координаты по пикселям экрана и производную мировых координат по пикселям экрана
- Мы знаем $\frac{\partial A}{\partial C}$ и $\frac{\partial B}{\partial C}$, можем вычислить $\frac{\partial A}{\partial B} = \frac{\partial A}{\partial C} \cdot \left(\frac{\partial B}{\partial C}\right)^{-1}$
- Сводится к операциям над матрицами 2×2 в шейдере
- Метод менее точен, но позволяет избежать дополнительных данных в вершинах

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 на сложных поверхностях (brdf с усреднённой нормалью – не то же самое, что усреднённые значения brdf)

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 на сложных поверхностях (brdf с усреднённой нормалью – не то же самое, что усреднённые значения brdf)
- 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

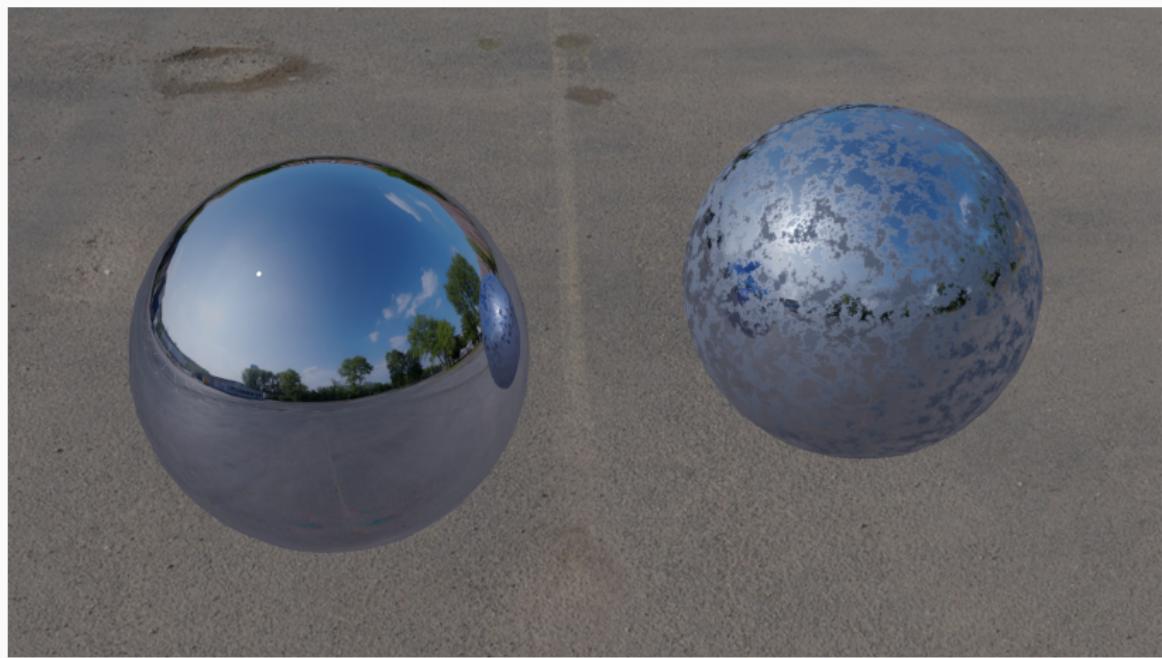
Material mapping

- 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)
- *Material mapping*: варьировать их, читая из текстуры, так же как альбено или нормали

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
- *Environment mapping*: рисовать статическую (обычно), заранее подготовленную картинку ‘environment map’ (облака, горы, etc.)

Environment mapping: как хранить?

- Environment map задаёт цвет 'фона' в каждом возможном направлении (формально – количество света, приходящего в сцену из этого направления)

Environment mapping: как хранить?

- Environment map задаёт цвет 'фона' в каждом возможном направлении (формально – количество света, приходящего в сцену из этого направления)
- **N.B.:** лучше, если ambient освещение совпадает с усреднённым значением environment map, иначе картинка будет выглядеть неестественно

Environment mapping: как хранить?

- Environment map задаёт цвет 'фона' в каждом возможном направлении (формально – количество света, приходящего в сцену из этого направления)
- **N.B.:** лучше, если ambient освещение совпадает с усреднённым значением environment map, иначе картинка будет выглядеть неестественно
- Варианты хранения:
 - Cubemap-текстура

Environment mapping: как хранить?

- Environment map задаёт цвет 'фона' в каждом возможном направлении (формально – количество света, приходящего в сцену из этого направления)
- **N.B.:** лучше, если ambient освещение совпадает с усреднённым значением environment map, иначе картинка будет выглядеть неестественно
- Варианты хранения:
 - 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 (формально – количество света, приходящего в сцену из этого направления)
- **N.B.:** лучше, если ambient освещение совпадает с усреднённым значением environment map, иначе картинка будет выглядеть неестественно
- Варианты хранения:
 - Cubemap-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, используя нормаль к поверхности (возможно, с normal mapping'ом)

Reflection mapping: реализация

- Во фрагментном шейдере вычисляем направление луча из камеры в точку
- Вычисляем направление отражённого луча, используя нормаль к поверхности (возможно, с normal mapping'ом)
- Используем вычисленное направление для чтения из environment map

Reflection mapping: реализация

- Во фрагментном шейдере вычисляем направление луча из камеры в точку
- Вычисляем направление отражённого луча, используя нормаль к поверхности (возможно, с normal mapping'ом)
- Используем вычисленное направление для чтения из environment map
- Можно явно использовать тиртарты 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'е, но вместо заранее подготовленного environment тар нарисуем сцену в текстуру

Отражения

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

Сиветар отражения

- Удобнее всего использовать сиветар-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 тіртар'ы или размыть сиветар чтобы имитировать нечёткие отражения

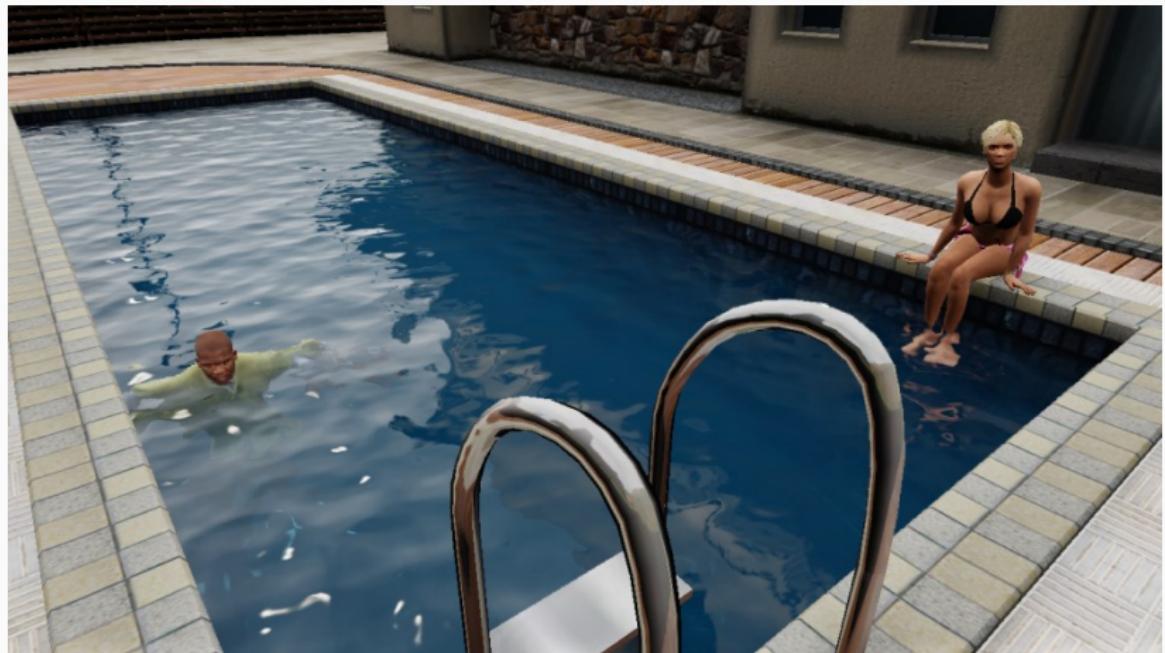
Сиветар отражения

- 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

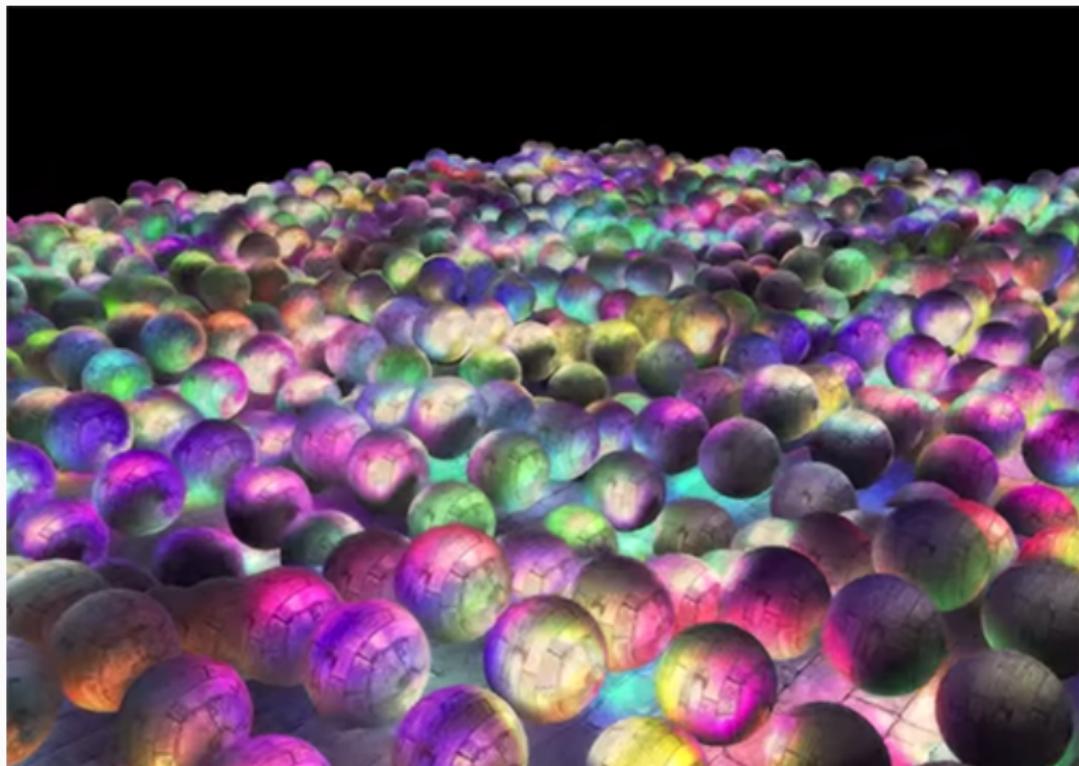
Сиветар отражения

- 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
- Можно сэкономить, расставляя зеркала только внутри зданий/комнат, где заранее известен набор отражаемых объектов

Субетап 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]`
 - Альтернативно, можно загрузить весь массив сразу функциями `glUniform*v`

Как обрабатывать много источников света?

- Сколько источников света можно так обработать?

Как обрабатывать много 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.)

Как обрабатывать много 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

Как обрабатывать много 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?

- Вместо массивов uniform-переменных можно использовать:
 - Uniform buffer objects (UBO) – обычно 16К
 - Buffer textures – от 64К
 - Shader storage buffer objects (SSBO, OpenGL 4.3) – от 128М

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 \Rightarrow больше изменений uniform-переменных при рендеринге, меньше возможность для batching'a (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 \Rightarrow больше изменений uniform-переменных при рендеринге, меньше возможность для batching'a (объединения объектов в группы или в общие буферы)
- Более современные решения: *deferred shading*, *tiled/clustering 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.
- Шейдер читает из G-buffer'а параметры материала в этом пикселе и вычисляет вклад освещения для всех источников света

Deferred shading

- Можно применять освещение одним шейдером на все источники, а можно аддитивным blending'ом по одному

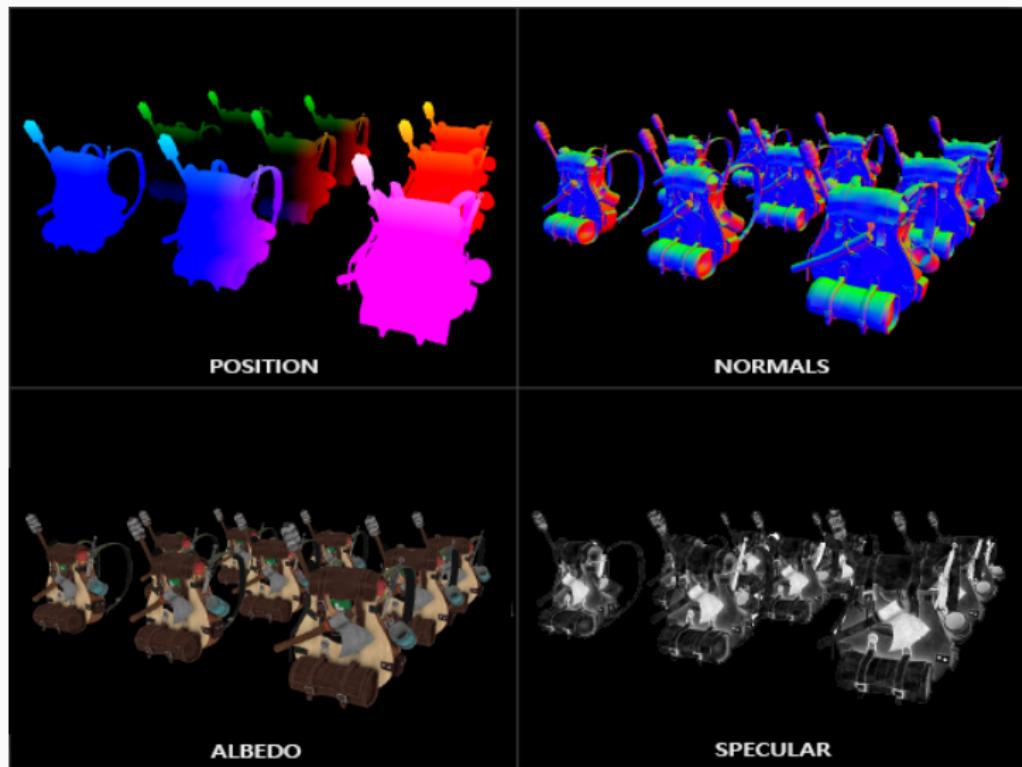
Deferred shading

- Можно применять освещение одним шейдером на все источники, а можно аддитивным blending'ом по одному
- В любом случае нужно рисовать полноэкранный прямоугольник

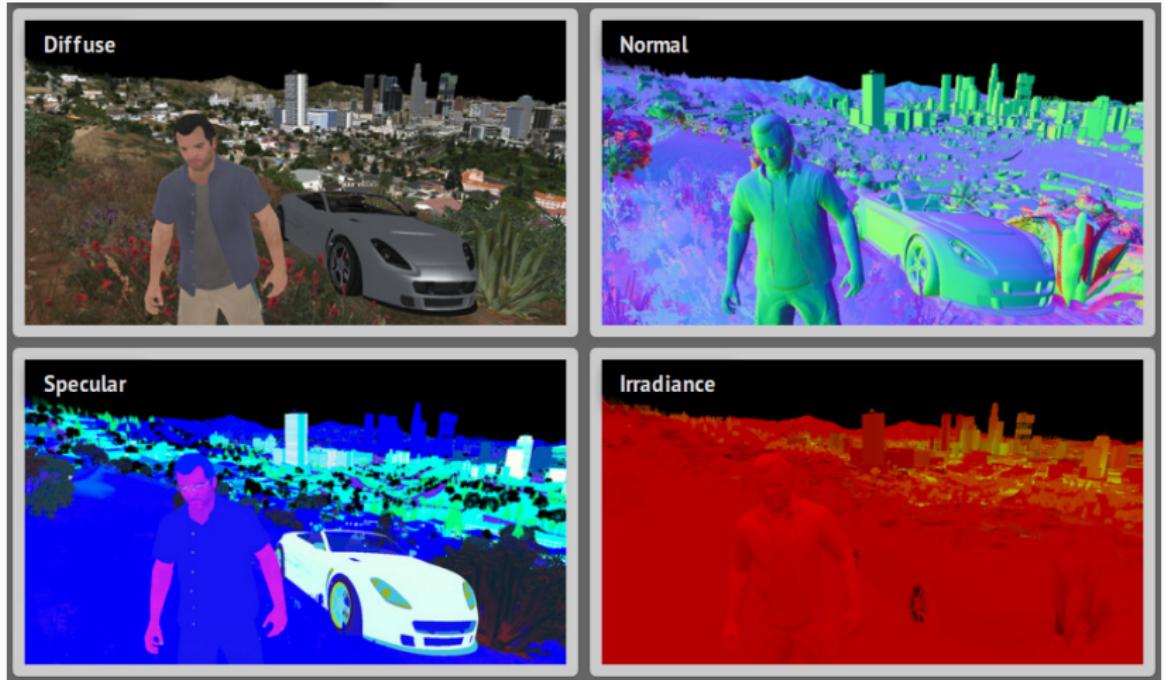
Deferred shading

- Можно применять освещение одним шейдером на все источники, а можно аддитивным blending'ом по одному
- В любом случае нужно рисовать полноэкранный прямоугольник
- В варианте с blending'ом можно вычислять освещение только для пикселей, расположенных рядом с источником света, вместо полноэкранного прямоугольника рисуя некую геометрию, ограничивающую область, освещаемую этим источником света (*proxy geometry*)

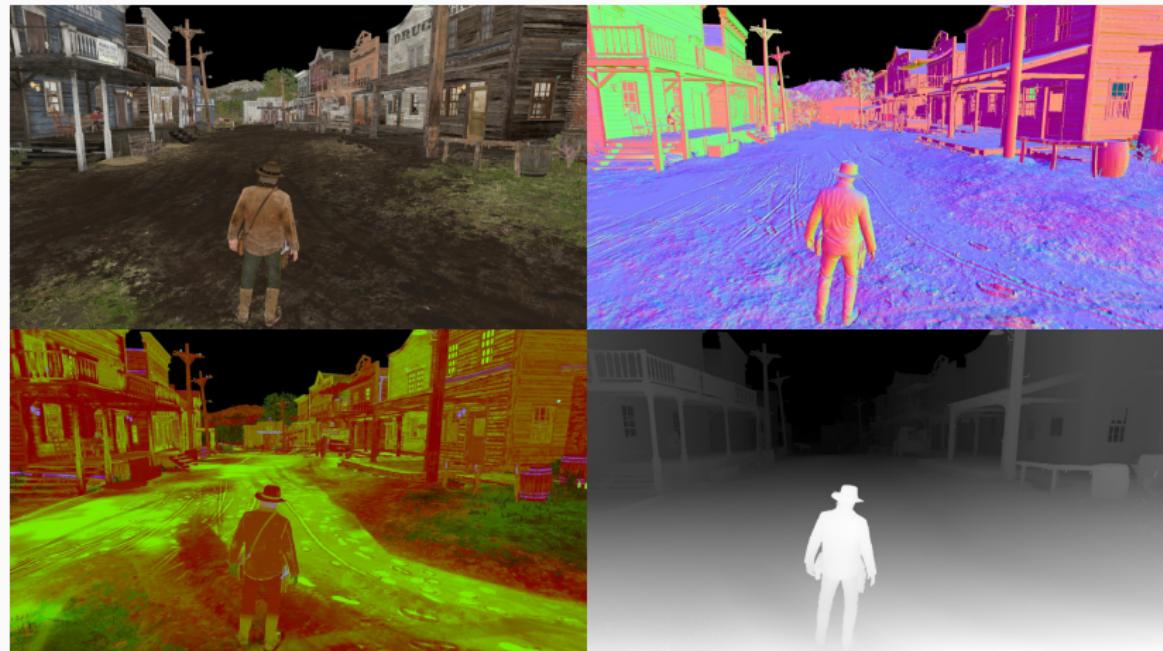
Deferred shading: G-buffer



Deferred shading: G-buffer (GTA V)



Deferred shading: G-buffer (RDR 2)



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'a (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 и чтения памяти (G-buffer) на каждый кадр
- Всё ещё один из самых популярных алгоритмов

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

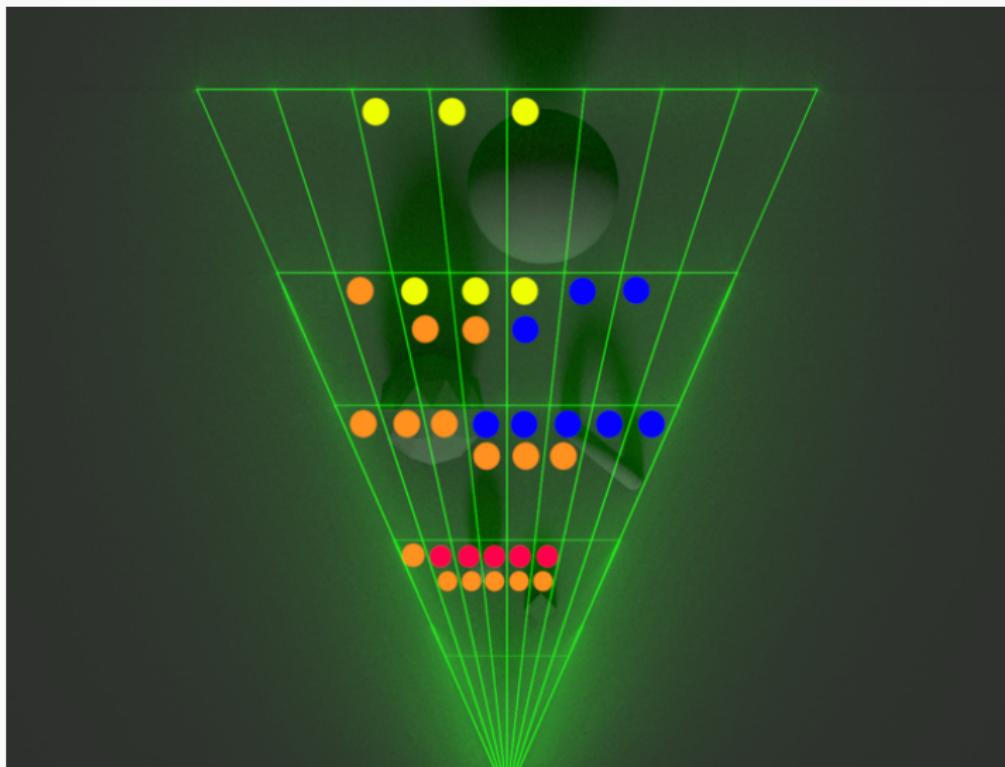
Tiled/clustered shading

- 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)

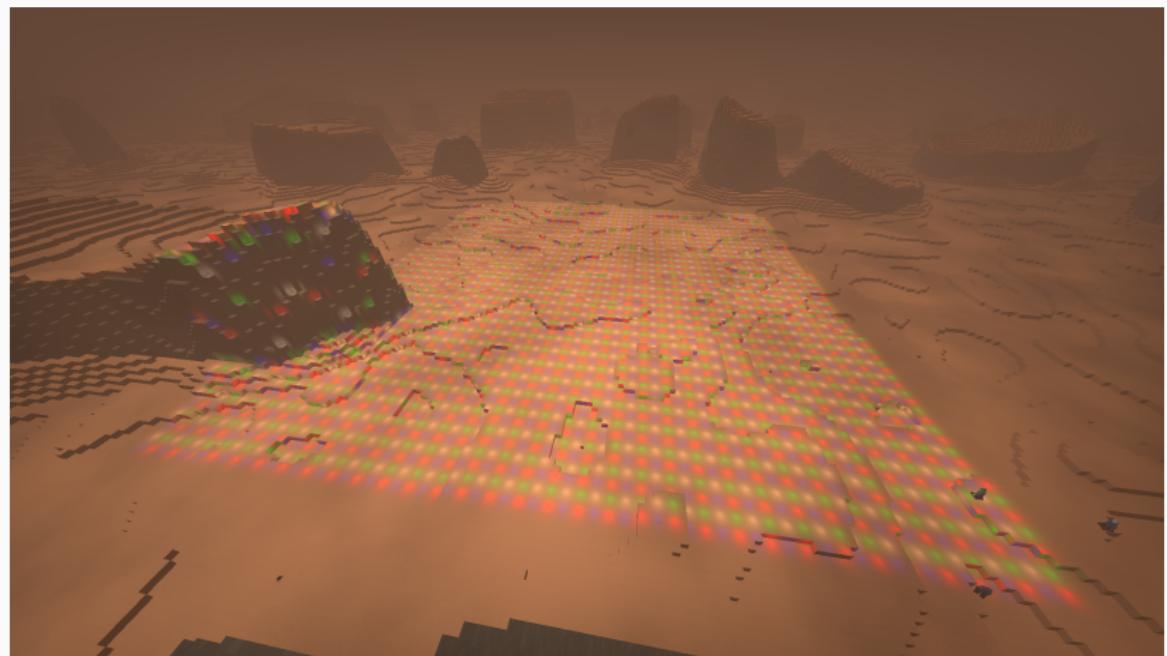
Tiled/clustered shading

- 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
- Из-за меньшего количества операций с памятью обычно работает быстрее, чем deferred shading

Тени от многих источников света

- Для каждого источника придётся рисовать shadow map

Тени от многих источников света

- Для каждого источника придётся рисовать shadow map
- Как хранить много shadow map?

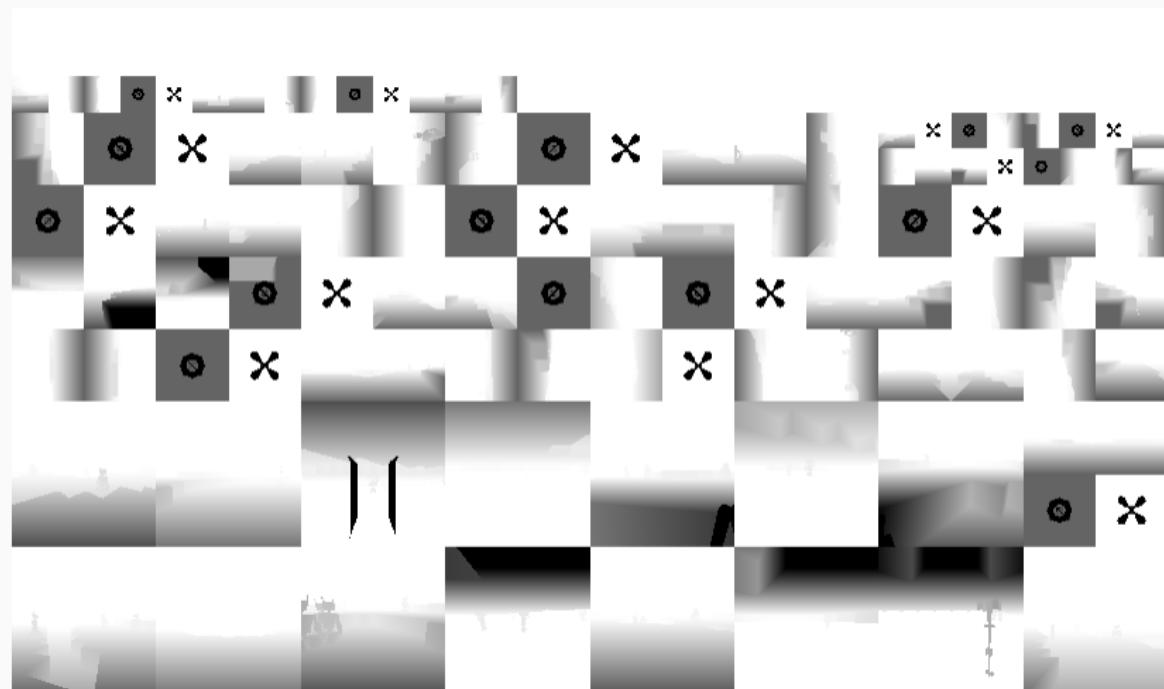
Тени от многих источников света

- Для каждого источника придётся рисовать shadow map
- Как хранить много shadow map?
- Array texture
 - + Удобный доступ (слой = номер источника)
 - - Все shadow map будут одинакового размера ⇒ нельзя сэкономить на тенях далёких источников

Тени от многих источников света

- Для каждого источника придётся рисовать shadow map
- Как хранить много shadow map?
- Array texture
 - + Удобный доступ (слой = номер источника)
 - - Все shadow map будут одинакового размера ⇒ нельзя сэкономить на тенях далёких источников
- *Shadow atlas*: использовать одну большую текстуру, в кусках которой хранятся shadow maps всех источников
 - + Можно варьировать размер shadow map для каждого источника
 - - Сложнее доступ к shadow map
 - - Нужно аккуратно работать с текстурными координатами, чтобы не залезть в чужой shadow map
 - - Нужен алгоритм упаковки многих текстур в одну
 - Самый распространённый способ

Shadow atlas



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