

# Компьютерная графика

## Лекция 8: Геометрические шейдеры, shadow volumes, shadow mapping и его разновидности

2021

# Геометрические шейдеры

- ▶ Тип шейдера, наравне в вершинным и фрагментным

# Геометрические шейдеры

- ▶ Тип шейдера, наравне в вершинным и фрагментным
- ▶ Создаётся как `glCreateShader(GL_GEOMETRY_SHADER)`

# Геометрические шейдеры

- ▶ Тип шейдера, наравне в вершинным и фрагментным
- ▶ Создаётся как `glCreateShader(GL_GEOMETRY_SHADER)`
- ▶ Встраивается после вершинного шейдера, до `perspective divide`

# Геометрические шейдеры

- ▶ Тип шейдера, наравне в вершинным и фрагментным
- ▶ Создаётся как `glCreateShader(GL_GEOMETRY_SHADER)`
- ▶ Встраивается после вершинного шейдера, до perspective divide
- ▶ Оперирует целыми примитивами (точками/линиями/треугольниками), т.е. наборами вершин

# Геометрические шейдеры

- ▶ Тип шейдера, наравне в вершинным и фрагментным
- ▶ Создаётся как `glCreateShader(GL_GEOMETRY_SHADER)`
- ▶ Встраивается после вершинного шейдера, до perspective divide
- ▶ Оперирует целыми примитивами (точками/линиями/треугольниками), т.е. наборами вершин
- ▶ Может менять тип примитива и количество вершин

# Геометрические шейдеры

- ▶ Тип шейдера, наравне в вершинным и фрагментным
- ▶ Создаётся как `glCreateShader(GL_GEOMETRY_SHADER)`
- ▶ Встраивается после вершинного шейдера, до perspective divide
- ▶ Оперирует целыми примитивами (точками/линиями/треугольниками), т.е. наборами вершин
- ▶ Может менять тип примитива и количество вершин
- ▶ Может варьировать количество вершин на выходе

# Геометрические шейдеры

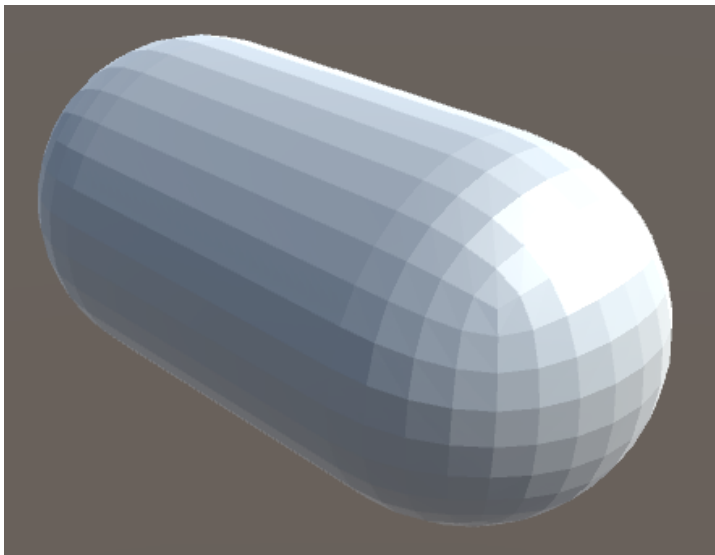
- ▶ Тип шейдера, наравне в вершинным и фрагментным
- ▶ Создаётся как `glCreateShader(GL_GEOMETRY_SHADER)`
- ▶ Встраивается после вершинного шейдера, до perspective divide
- ▶ Оперирует целыми примитивами (точками/линиями/треугольниками), т.е. наборами вершин
- ▶ Может менять тип примитива и количество вершин
- ▶ Может варьировать количество вершин на выходе
- ▶  $\Rightarrow$  может работать довольно медленно



# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a

## Flat shading



# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью

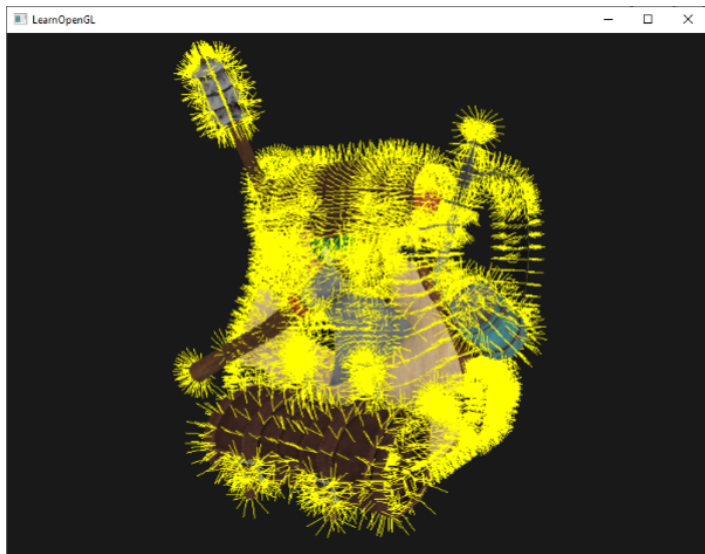
# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей

# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей
  - ▶ Вход: вершина с нормалью
  - ▶ Выход: линия из двух вершин – исходная вершина, исходная вершина + нормаль

# Визуализация нормалей



# Геометрические шейдеры: примеры использования

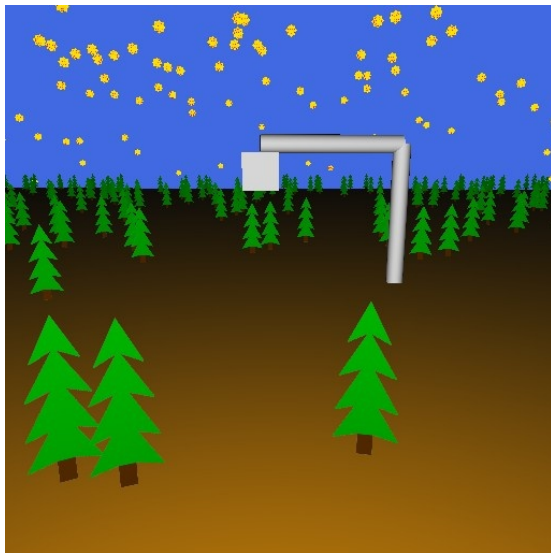
- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей
  - ▶ Вход: вершина с нормалью
  - ▶ Выход: линия из двух вершин – исходная вершина, исходная вершина + нормаль
- ▶ Shadow volumes – алгоритм рисования теней

# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей
  - ▶ Вход: вершина с нормалью
  - ▶ Выход: линия из двух вершин – исходная вершина, исходная вершина + нормаль
- ▶ Shadow volumes – алгоритм рисования теней
- ▶ Billboards – плоские фигуры, всегда смотрящие в сторону камеры



# Billboards



# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей
  - ▶ Вход: вершина с нормалью
  - ▶ Выход: линия из двух вершин – исходная вершина, исходная вершина + нормаль
- ▶ Shadow volumes – алгоритм рисования теней
- ▶ Billboards – плоские фигуры, всегда смотрящие в сторону камеры
  - ▶ Вход: одна вершина (точка)
  - ▶ Выход: набор треугольников

# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей
  - ▶ Вход: вершина с нормалью
  - ▶ Выход: линия из двух вершин – исходная вершина, исходная вершина + нормаль
- ▶ Shadow volumes – алгоритм рисования теней
- ▶ Billboards – плоские фигуры, всегда смотрящие в сторону камеры
  - ▶ Вход: одна вершина (точка)
  - ▶ Выход: набор треугольников
  - ▶ Системы частиц

## Billboards: система частиц (дым)



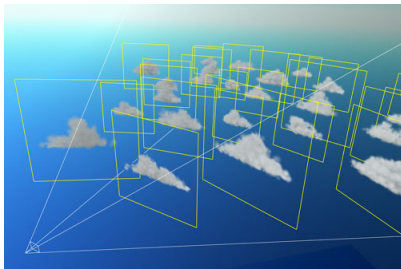
# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей
  - ▶ Вход: вершина с нормалью
  - ▶ Выход: линия из двух вершин – исходная вершина, исходная вершина + нормаль
- ▶ Shadow volumes – алгоритм рисования теней
- ▶ Billboards – плоские фигуры, всегда смотрящие в сторону камеры
  - ▶ Вход: одна вершина (точка)
  - ▶ Выход: набор треугольников
  - ▶ Системы частиц

# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей
  - ▶ Вход: вершина с нормалью
  - ▶ Выход: линия из двух вершин – исходная вершина, исходная вершина + нормаль
- ▶ Shadow volumes – алгоритм рисования теней
- ▶ Billboards – плоские фигуры, всегда смотрящие в сторону камеры
  - ▶ Вход: одна вершина (точка)
  - ▶ Выход: набор треугольников
  - ▶ Системы частиц
  - ▶ Облака

## Billboards: облака



# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей
  - ▶ Вход: вершина с нормалью
  - ▶ Выход: линия из двух вершин – исходная вершина, исходная вершина + нормаль
- ▶ Shadow volumes – алгоритм рисования теней
- ▶ Billboards – плоские фигуры, всегда смотрящие в сторону камеры
  - ▶ Вход: одна вершина (точка)
  - ▶ Выход: набор треугольников
  - ▶ Системы частиц
  - ▶ Облака



# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей
  - ▶ Вход: вершина с нормалью
  - ▶ Выход: линия из двух вершин – исходная вершина, исходная вершина + нормаль
- ▶ Shadow volumes – алгоритм рисования теней
- ▶ Billboards – плоские фигуры, всегда смотрящие в сторону камеры
  - ▶ Вход: одна вершина (точка)
  - ▶ Выход: набор треугольников
  - ▶ Системы частиц
  - ▶ Облака
  - ▶ Деревья

## Billboards: деревья



# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей
  - ▶ Вход: вершина с нормалью
  - ▶ Выход: линия из двух вершин – исходная вершина, исходная вершина + нормаль
- ▶ Shadow volumes – алгоритм рисования теней
- ▶ Billboards – плоские фигуры, всегда смотрящие в сторону камеры
  - ▶ Вход: одна вершина (точка)
  - ▶ Выход: набор треугольников
  - ▶ Системы частиц
  - ▶ Облака
  - ▶ Деревья

# Геометрические шейдеры: примеры использования

- ▶ Расчёт нормалей для flat shading'a
  - ▶ Вход: треугольник (тройка вершин)
  - ▶ Выход: те же вершины с посчитанной нормалью
- ▶ Визуализация нормалей
  - ▶ Вход: вершина с нормалью
  - ▶ Выход: линия из двух вершин – исходная вершина, исходная вершина + нормаль
- ▶ Shadow volumes – алгоритм рисования теней
- ▶ Billboards – плоские фигуры, всегда смотрящие в сторону камеры
  - ▶ Вход: одна вершина (точка)
  - ▶ Выход: набор треугольников
  - ▶ Системы частиц
  - ▶ Облака
  - ▶ Деревья
  - ▶ Трава

## Billboards: трава



## Геометрические шейдеры: пример

```
#version 330 core
uniform mat4 transform;
// Входные примитивы - точки
layout (points) in;
// Выходные примитивы - линии, в сумме не больше 2х вершин
layout (line_strip, max_vertices = 2) out;
// Данные из вершинного шейдера
in vec3 normal[];

void main() {
    gl_Position = transform * gl_in[0].gl_Position;
    EmitVertex();

    gl_Position = transform * (gl_in[0].gl_Position
        + vec4(normal[0], 0));
    EmitVertex();

    EndPrimitive();
}
```

## Геометрические шейдеры: пример

```
#version 330 core
// Входные примитивы - линии
layout (lines) in;
// Выходные примитивы - треугольники, в сумме не больше 4х вершин
layout (triangle_strip, max_vertices = 4) out;
// Данные для фрагментного шейдера
out vec4 color;

void main() {
    gl_Position = gl_in[0].gl_Position + vec4(-1.0, -1.0, 0.0, 0.0);
    color = vec4(1.0, 0.0, 0.0, 1.0);
    EmitVertex();

    gl_Position = gl_in[0].gl_Position + vec4( 1.0, -1.0, 0.0, 0.0);
    color = vec4(0.0, 1.0, 0.0, 1.0);
    EmitVertex();

    gl_Position = gl_in[1].gl_Position + vec4(-1.0,  1.0, 0.0, 0.0);
    color = vec4(0.0, 0.0, 1.0, 1.0);
    EmitVertex();

    gl_Position = gl_in[1].gl_Position + vec4( 1.0,  1.0, 0.0, 0.0);
    color = vec4(1.0, 1.0, 1.0, 1.0);
    EmitVertex();

    EndPrimitive();
}
```

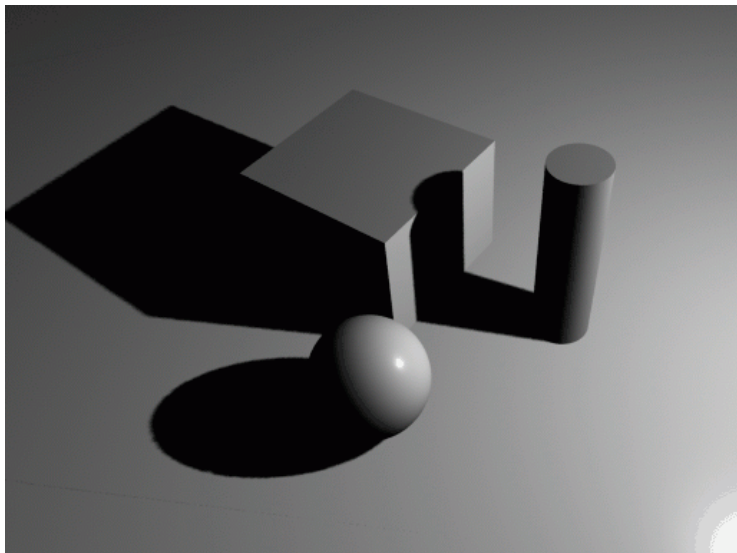
## Геометрические шейдеры: ссылки

- ▶ [khronos.org/opengl/wiki/Geometry\\_Shader](https://www.khronos.org/opengl/wiki/Geometry_Shader)
- ▶ [learnopengl.com/Advanced-OpenGL/Geometry-Shader](https://learnopengl.com/Advanced-OpenGL/Geometry-Shader)
- ▶ [open.gl/geometry](https://open.gl/geometry)
- ▶ [lighthouse3d.com/tutorials/glsl-tutorial/geometry-shader](https://lighthouse3d.com/tutorials/glsl-tutorial/geometry-shader)
- ▶ GPU Gems, Chapter 7. Rendering Countless Blades of Waving Grass



# Тени: теория

- ▶ Точка сцены, в которую не попадает (заблокирован чем-то) прямой свет из конкретного источника света
- ▶ Свойство точки по отношению к конкретному источнику света



# Тени: теория

- ▶ Если источник света точечный (или бесконечно удалённый), тень – бинарное свойство: луч из точки сцены в источник света или пересекает что-то (точка в тени), или нет (точка не в тени)  $\Rightarrow$  жёсткие тени (hard shadows)

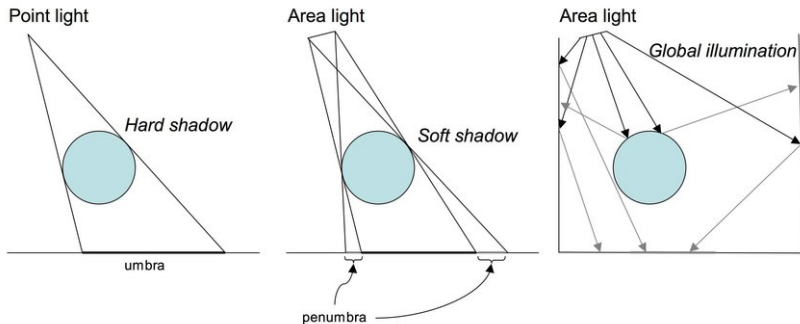
# Тени: теория

- ▶ Если источник света точечный (или бесконечно удалённый), тень – бинарное свойство: луч из точки сцены в источник света или пересекает что-то (точка в тени), или нет (точка не в тени)  $\Rightarrow$  жёсткие тени (hard shadows)
- ▶ Если источник света объёмный, точка сцены может находиться в тени относительно части источника света, и не находиться в тени относительно другой его части  $\Rightarrow$  мягкие тени (soft shadows)

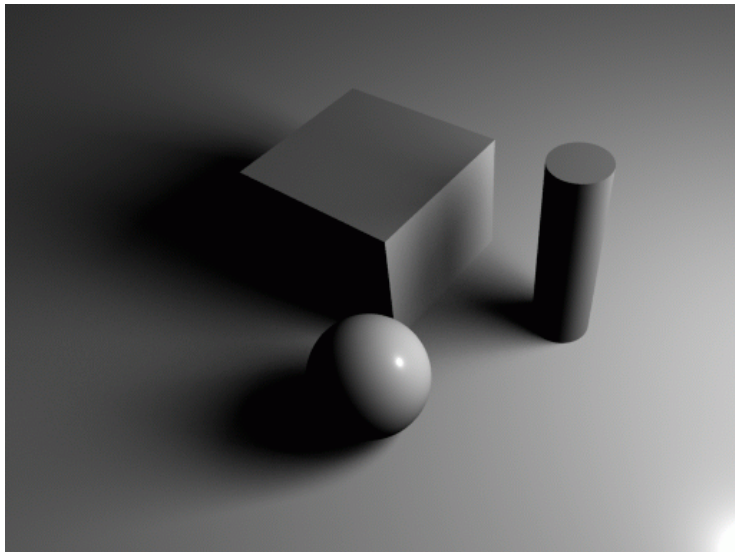
# Тени: теория

- ▶ Если источник света точечный (или бесконечно удалённый), тень – бинарное свойство: луч из точки сцены в источник света или пересекает что-то (точка в тени), или нет (точка не в тени)  $\Rightarrow$  жёсткие тени (hard shadows)
- ▶ Если источник света объёмный, точка сцены может находиться в тени относительно части источника света, и не находиться в тени относительно другой его части  $\Rightarrow$  мягкие тени (soft shadows)
  - ▶ Точки, полностью находящиеся в тени – *umbra*
  - ▶ Точки, частично находящиеся в тени – *penumbra*

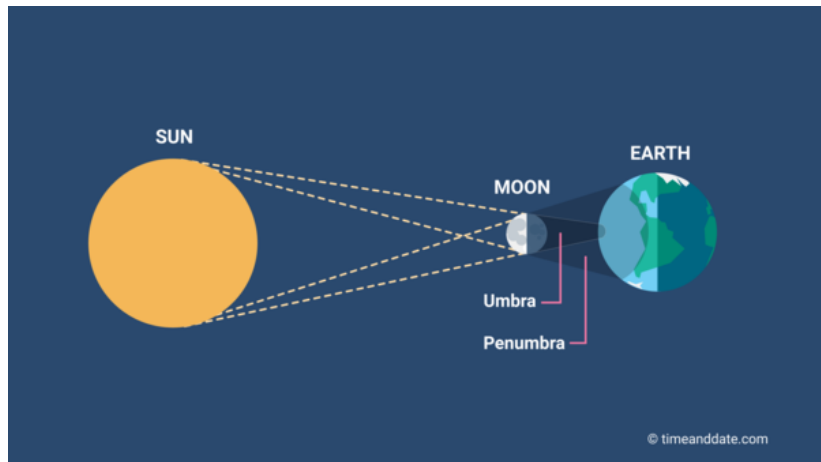
# Тени: мягкие vs жёсткие



# Мягкие тени



# Солнечное затмение





# Тени: теория

- ▶ Реальные источники света – объёмные

# Тени: теория

- ▶ Реальные источники света – объёмные
- ▶ Размер и форма penumbra зависит от размеров и формы источника света, а также от расстояния до него (чем дальше, тем меньше penumbra)

# Тени: теория

- ▶ Реальные источники света – объёмные
- ▶ Размер и форма penumbra зависит от размеров и формы источника света, а также от расстояния до него (чем дальше, тем меньше penumbra)
  - ▶ В пределе расстояния  $\rightarrow \infty$  мягкая тень вырождается в жёсткую

# Тени: теория

- ▶ Реальные источники света – объёмные
- ▶ Размер и форма penumbra зависит от размеров и формы источника света, а также от расстояния до него (чем дальше, тем меньше penumbra)
  - ▶ В пределе расстояния  $\rightarrow \infty$  мягкая тень вырождается в жёсткую
- ▶ В real-time графике получить правильные тени (как жёсткие, так и мягкие) довольно сложно

## Тени: raytracing

- ▶ При простом raytracing'е (без решения rendering equation) для получения жёсткой тени можно послать дополнительный луч в направлении света (shadow ray): если он что-то пересёк, точка находится в тени

# Тени: raytracing

- ▶ При простом raytracing'е (без решения rendering equation) для получения жёсткой тени можно послать дополнительный луч в направлении света (shadow ray): если он что-то пересёк, точка находится в тени
- ▶ При raytracing'е с монте-карло интегрированием для global illumination тени (как жёсткие, так и мягкие) получаются автоматически

# Тени: real-time алгоритмы

- ▶ Shadow volumes

# Тени: real-time алгоритмы

## ► Shadow volumes

- + Идеальные жёсткие тени
- — Алиасинг
- — Нужно на лету генерировать геометрию
- — Сложно сделать мягкие тени
- — Очень большой fill rate (количество обрабатываемых пикселей), плохо предсказуемая производительность



# Тени: real-time алгоритмы

- ▶ Shadow volumes
  - ▶ + Идеальные жёсткие тени
  - ▶ — Алиасинг
  - ▶ — Нужно на лету генерировать геометрию
  - ▶ — Сложно сделать мягкие тени
  - ▶ — Очень большой fill rate (количество обрабатываемых пикселей), плохо предсказуемая производительность
- ▶ Shadow mapping

# Тени: real-time алгоритмы

## ► Shadow volumes

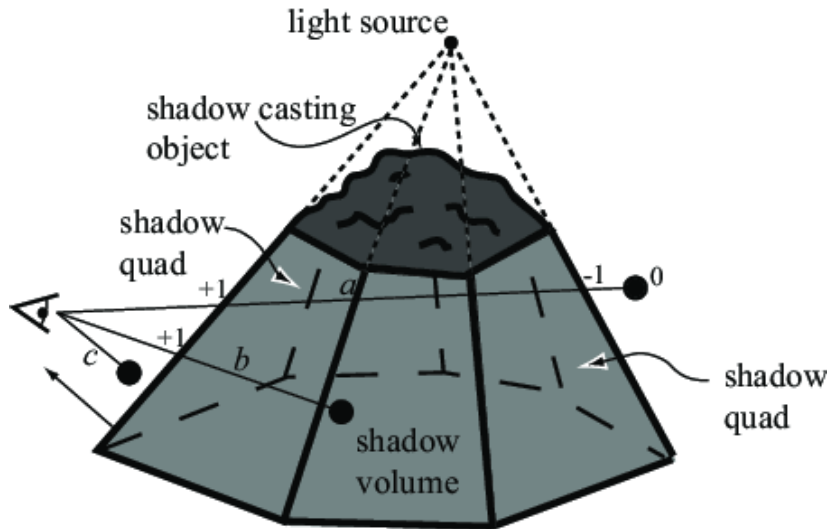
- + Идеальные жёсткие тени
- — Алиасинг
- — Нужно на лету генерировать геометрию
- — Сложно сделать мягкие тени
- — Очень большой fill rate (количество обрабатываемых пикселей), плохо предсказуемая производительность

## ► Shadow mapping

- + Производительность растёт +/- линейно с ростом сложности сцены
- — Крупный алиасинг ("пиксельные" тени)
- + Много вариаций, улучшающих качество и позволяющих делать как жёсткие, так и мягкие тени

## Shadow volumes (они же stencil shadows)

- ▶ Тень от конкретного объекта – некая (полубесконечная) трёхмерная область пространства (shadow volume)



## Shadow volumes (они же stencil shadows)

- ▶ Тень от конкретного объекта – некая (полубесконечная) трёхмерная область пространства (shadow volume)

## Shadow volumes (они же stencil shadows)

- ▶ Тень от конкретного объекта – некая (полубесконечная) трёхмерная область пространства (shadow volume)
- ▶ Нарисуем её как трёхмерный объект (построим и нарисуем её грани)

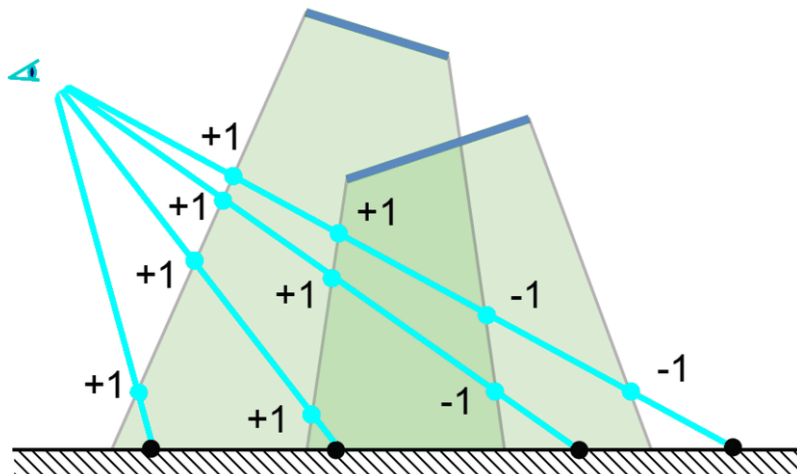
## Shadow volumes (они же stencil shadows)

- ▶ Тень от конкретного объекта – некая (полубесконечная) трёхмерная область пространства (shadow volume)
- ▶ Нарисуем её как трёхмерный объект (построим и нарисуем её грани)
- ▶ Если в конкретный пиксель попало одинаковое количество front-facing и back-facing граней, то он не в тени, иначе – в тени

## Shadow volumes (они же stencil shadows)

- ▶ Тень от конкретного объекта – некая (полубесконечная) трёхмерная область пространства (shadow volume)
- ▶ Нарисуем её как трёхмерный объект (построим и нарисуем её грани)
- ▶ Если в конкретный пиксель попало одинаковое количество front-facing и back-facing граней, то он не в тени, иначе – в тени
- ▶ Количество пересечений луча из камеры с гранями shadow volume вычисляем с помощью stencil буфера

## Shadow volumes (они же stencil shadows)





## Shadow volumes: алгоритм

- ▶ Выключаем рисование в цветовой буфер и буфер глубины (`glColorMask`, `glDepthMask`) (но не выключаем тест глубины!)

## Shadow volumes: алгоритм

- ▶ Выключаем рисование в цветовой буфер и буфер глубины (`glColorMask`, `glDepthMask`) (но не выключаем тест глубины!)
- ▶ Настраиваем stencil буфер: +1 для front-facing треугольников, -1 для back-facing

## Shadow volumes: алгоритм

- ▶ Выключаем рисование в цветовой буфер и буфер глубины (`glColorMask`, `glDepthMask`) (но не выключаем тест глубины!)
- ▶ Настраиваем stencil буфер: +1 для front-facing треугольников, -1 для back-facing
- ▶ Вычисляем и рисуем shadow volume (для каждого объекта)

## Shadow volumes: алгоритм

- ▶ Выключаем рисование в цветовой буфер и буфер глубины (`glColorMask`, `glDepthMask`) (но не выключаем тест глубины!)
- ▶ Настраиваем stencil буфер: +1 для front-facing треугольников, -1 для back-facing
- ▶ Вычисляем и рисуем shadow volume (для каждого объекта)
- ▶ Обратно включаем рисование цвета и глубины

## Shadow volumes: алгоритм

- ▶ Выключаем рисование в цветовой буфер и буфер глубины (`glColorMask`, `glDepthMask`) (но не выключаем тест глубины!)
- ▶ Настраиваем stencil буфер: +1 для front-facing треугольников, -1 для back-facing
- ▶ Вычисляем и рисуем shadow volume (для каждого объекта)
- ▶ Обратно включаем рисование цвета и глубины
- ▶ Настраиваем stencil-тест так, чтобы рисовать только пиксели, в которых `stencil=0`, и рисуем сцену шейдером, вычисляющим освещение

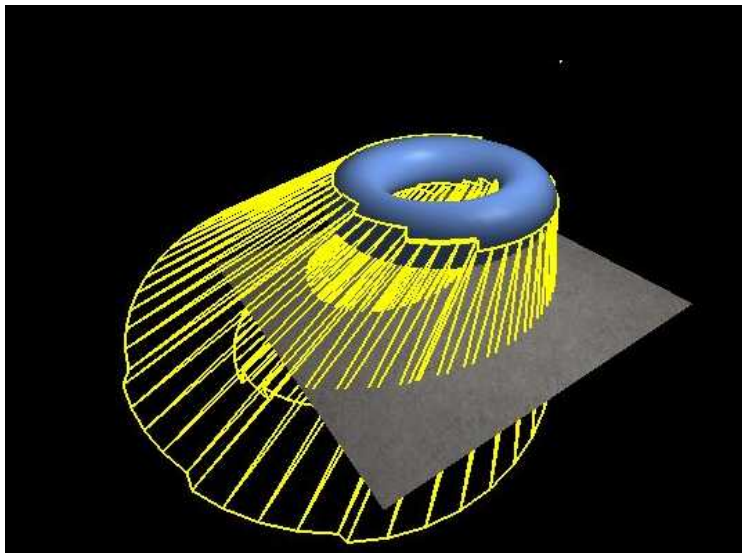
## Shadow volumes: алгоритм

- ▶ Выключаем рисование в цветовой буфер и буфер глубины (`glColorMask`, `glDepthMask`) (но не выключаем тест глубины!)
- ▶ Настраиваем stencil буфер: +1 для front-facing треугольников, -1 для back-facing
- ▶ Вычисляем и рисуем shadow volume (для каждого объекта)
- ▶ Обратно включаем рисование цвета и глубины
- ▶ Настраиваем stencil-тест так, чтобы рисовать только пиксели, в которых `stencil=0`, и рисуем сцену шейдером, вычисляющим освещение
- ▶ Настраиваем stencil-тест так, чтобы рисовать только пиксели, в которых `stencil!=0`, и рисуем сцену шейдером, игнорирующим освещение

## Shadow volumes: алгоритм

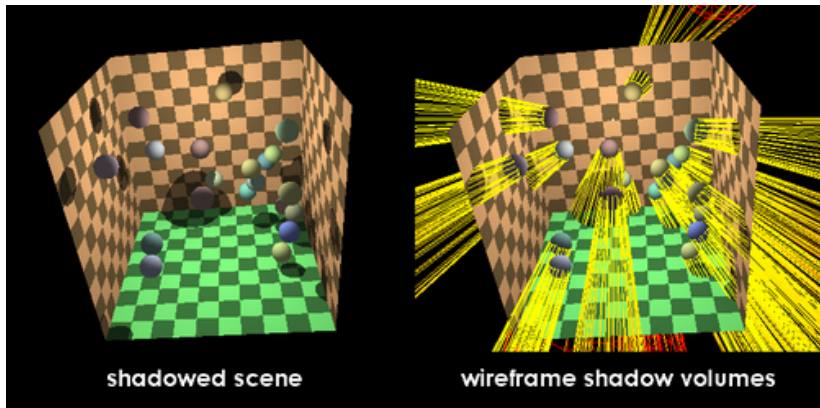
- ▶ Выключаем рисование в цветовой буфер и буфер глубины (`glColorMask`, `glDepthMask`) (но не выключаем тест глубины!)
- ▶ Настраиваем stencil буфер: +1 для front-facing треугольников, -1 для back-facing
- ▶ Вычисляем и рисуем shadow volume (для каждого объекта)
- ▶ Обратно включаем рисование цвета и глубины
- ▶ Настраиваем stencil-тест так, чтобы рисовать только пиксели, в которых `stencil=0`, и рисуем сцену шейдером, вычисляющим освещение
- ▶ Настраиваем stencil-тест так, чтобы рисовать только пиксели, в которых `stencil!=0`, и рисуем сцену шейдером, игнорирующим освещение
- ▶ Процедуру нужно повторить для каждого источника света

## Shadow volume





# Shadow volume



# Shadow volumes (Doom 3)



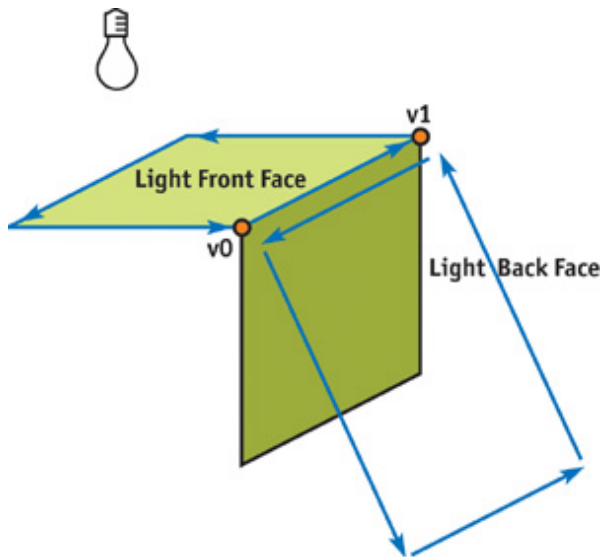
## Shadow volumes: как вычислять shadow volume?

- ▶ Шаг 1: определить shadow edges – рёбра модели, у которых один соседний треугольник обращён к источнику света ( $\vec{n} \cdot \vec{r} > 0$ ), а второй – против источника света ( $\vec{n} \cdot \vec{r} \leq 0$ )

## Shadow volumes: как вычислять shadow volume?

- ▶ Шаг 1: определить shadow edges – рёбра модели, у которых один соседний треугольник обращён к источнику света ( $\vec{n} \cdot \vec{r} > 0$ ), а второй – против источника света ( $\vec{n} \cdot \vec{r} \leq 0$ )
- ▶ Шаг 2: для каждого shadow edge построить четырёхугольную грань shadow volume – к точкам ребра  $p_1, p_2$  добавляются точки  $p_1 + D \cdot \frac{p_1 - o}{|p_1 - o|}$  и аналогично для  $p_2$ 
  - ▶  $o$  – координаты источника света
  - ▶  $D$  – расстояние, до которого отбрасывается тень (в идеале мы хотим  $D \rightarrow \infty$ , для этого можно воспользоваться однородными координатами)

## Shadow volumes (Doom 3)



## Shadow volumes: как вычислять shadow volume?

- ▶ Два варианта: на CPU или на GPU

# Shadow volumes: как вычислять shadow volume?

- ▶ Два варианта: на CPU или на GPU
- ▶ На CPU: очень дорого (положение модели и источника света могут меняться каждый кадр)

# Shadow volumes: как вычислять shadow volume?

- ▶ Два варианта: на CPU или на GPU
- ▶ На CPU: очень дорого (положение модели и источника света могут меняться каждый кадр)
- ▶ На GPU: геометрические шейдеры!
  - ▶ Нужен входной тип примитива *lines adjacency*, позволяющий передать 4 вершины как один примитив (две вершины ребра + две вершины соседних треугольников)



# Shadow volumes: как вычислять shadow volume?

- ▶ Два варианта: на CPU или на GPU
- ▶ На CPU: очень дорого (положение модели и источника света могут меняться каждый кадр)
- ▶ На GPU: геометрические шейдеры!
  - ▶ Нужен входной тип примитива *lines adjacency*, позволяющий передать 4 вершины как один примитив (две вершины ребра + две вершины соседних треугольников)
  - ▶  $\Rightarrow$  модель должна быть специально адаптирована для генерации shadow volume

# Shadow volumes: как вычислять shadow volume?

- ▶ Два варианта: на CPU или на GPU
- ▶ На CPU: очень дорого (положение модели и источника света могут меняться каждый кадр)
- ▶ На GPU: геометрические шейдеры!
  - ▶ Нужен входной тип примитива *lines adjacency*, позволяющий передать 4 вершины как один примитив (две вершины ребра + две вершины соседних треугольников)
  - ▶  $\Rightarrow$  модель должна быть специально адаптирована для генерации shadow volume
  - ▶ Шейдер проверяет, является ли ребро shadow edge, и генерирует грань shadow volume

# Shadow volumes

- ▶ + Pixel-perfect жёсткие тени
- ▶ — Алиасинг
- ▶ — Сложно получить мягкие тени
- ▶ — Нужно на лету генерировать геометрию
- ▶ — Требуется правильной геометрии (без дырок и дублирования)
- ▶ — Не работает, если камера находится внутри тени (исправляется с помощью т.н. Carmack's Reverse)
- ▶ — Ресурсозатратен: даже для маленького объекта его shadow volume может занимать значительную часть сцены  
⇒ слишком много растеризации, чтения/записи памяти (stencil buffer)

## Shadow volumes с плохой геометрией



# Shadow volumes

- ▶ Thief 3
- ▶ Doom 3
- ▶ Quake 4
- ▶ Prey
- ▶ Far Cry
- ▶ F.E.A.R 1, 2, 3
- ▶ S.T.A.L.K.E.R.
- ▶ Timeshift
- ▶ и др.

## Shadow volumes: ссылки

- ▶ [en.wikipedia.org/wiki/Shadow\\_volume](https://en.wikipedia.org/wiki/Shadow_volume)
- ▶ [ogldev.org/www/tutorial40/tutorial40.html](http://ogldev.org/www/tutorial40/tutorial40.html)
- ▶ GPU Gems, Chapter 9. Efficient Shadow Volume Rendering

# Shadow mapping: идея

- ▶ Самое популярное семейство алгоритмов рисования теней

# Shadow mapping: идея

- ▶ Самое популярное семейство алгоритмов рисования теней
- ▶ Идея: представим, что источник света – это камера (наблюдатель)
- ▶ Тень – это то, что **не видит** источник света



# Shadow mapping: идея

- ▶ Самое популярное семейство алгоритмов рисования теней
- ▶ Идея: представим, что источник света – это камера (наблюдатель)
- ▶ Тень – это то, что **не видит** источник света
- ▶ Как понять, что видит источник света?

# Shadow mapping: идея

- ▶ Самое популярное семейство алгоритмов рисования теней
- ▶ Идея: представим, что источник света – это камера (наблюдатель)
- ▶ Тень – это то, что **не видит** источник света
- ▶ Как понять, что видит источник света? Нарисовать!

# Shadow mapping: алгоритм

- ▶ Рисуем сцену в текстуру (shadow map, карта теней) с точки зрения источника света (цветовой буфер не нужен, достаточно буфера глубины)

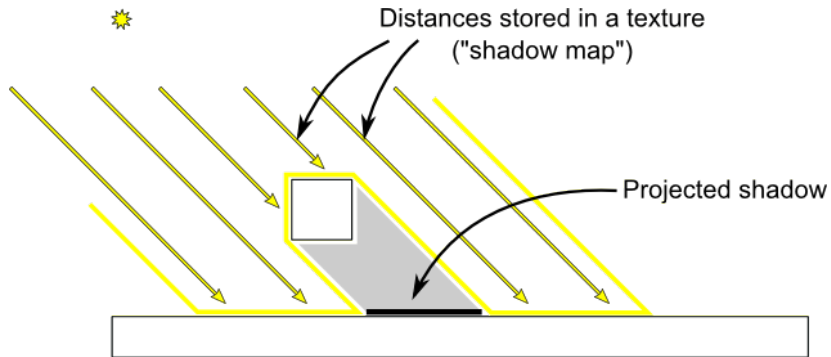
# Shadow mapping: алгоритм

- ▶ Рисуем сцену в текстуру (shadow map, карта теней) с точки зрения источника света (цветовой буфер не нужен, достаточно буфера глубины)
- ▶ Рисуем сцену на экран, фрагментный шейдер читает *буфер глубины* (shadow map), нарисованный с точки зрения источника света – там содержатся расстояния от источника до ближайшей (к источнику) поверхности
- ▶ Если расстояние от источника до рисуемого пикселя больше, чем значение из буфера глубины, – текущий пиксель находится в тени

## Shadow mapping: алгоритм

- ▶ Рисуем сцену в текстуру (shadow map, карта теней) с точки зрения источника света (цветовой буфер не нужен, достаточно буфера глубины)
- ▶ Рисуем сцену на экран, фрагментный шейдер читает *буфер глубины* (shadow map), нарисованный с точки зрения источника света – там содержатся расстояния от источника до ближайшей (к источнику) поверхности
- ▶ Если расстояние от источника до рисуемого пикселя больше, чем значение из буфера глубины, – текущий пиксель находится в тени
- ▶ N.B. Линейная фильтрация для буфера глубины не имеет смысла – будут артефакты на границах объектов

# Shadow mapping: алгоритм



## Shadow mapping: фрагментный шейдер

```
#version 330 core
// Матрица с которой рисовался shadow map
uniform mat4 shadow_projection;
// Буфер глубины (shadow map)
uniform sampler2D shadow_depth;

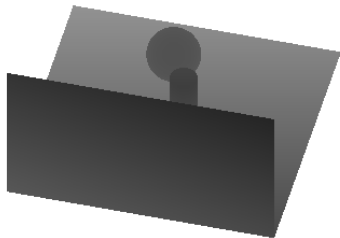
in vec3 position;

void main(){
    vec4 shadow_coord = shadow_projection * vec4(position, 1.0);
    shadow_coord /= shadow_coord.w; // perspective divide
    if (abs(shadow_coord.x) < 1.0 && abs(shadow_coord.y) < 1.0){
        shadow_coord = shadow_coord * 0.5 + vec2(0.5);

        bool in_shadow = texture(shadow_depth,
            shadow_coord.xy).r < shadow_coord.z;

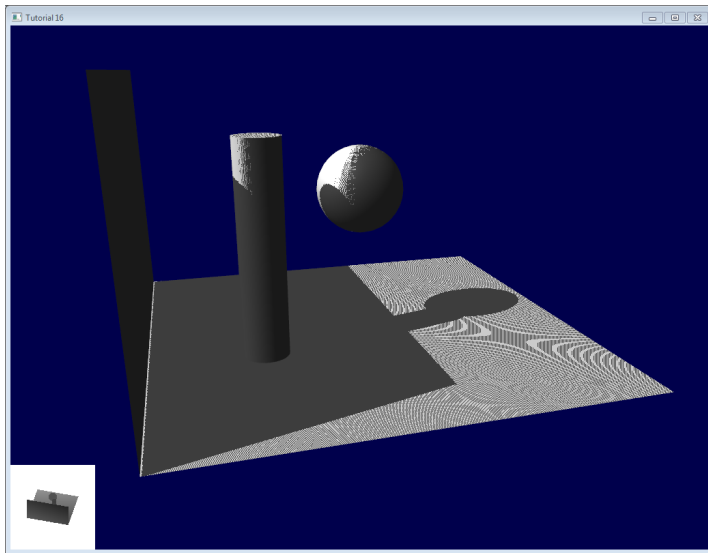
        ...
    }
}
```

## Shadow mapping: буфер глубины





# Shadow mapping: результат



# Shadow mapping: проекция

- ▶ Как выбрать проекцию камеры (`shadow_transform`) для shadow map?

# Shadow mapping: проекция

- ▶ Как выбрать проекцию камеры (`shadow_transform`) для shadow map?
- ▶ Удалённый источник освещает всю сцену с одного направления  $\Rightarrow$  удобно использовать ортографическую проекцию

# Shadow mapping: проекция

- ▶ Как выбрать проекцию камеры (`shadow_transform`) для shadow map?
- ▶ Удалённый источник освещает всю сцену с одного направления  $\Rightarrow$  удобно использовать ортографическую проекцию
- ▶ Проекцию надо подобрать так, чтобы в видимую область попала вся сцена

# Shadow mapping: проекция

- ▶ Как выбрать проекцию камеры (`shadow_transform`) для shadow map?
- ▶ Удалённый источник освещает всю сцену с одного направления  $\Rightarrow$  удобно использовать ортографическую проекцию
- ▶ Проекцию надо подобрать так, чтобы в видимую область попала вся сцена
- ▶ Можно спроецировать все объекты на плоскость, перпендикулярную направлению на свет (которое можно считать Z-осью этой камеры), взять любые два ортогональных вектора X и Y в этой плоскости, и найти bounding box сцены в координатах XYZ – это даст размеры и ориентацию области shadow map

# Shadow mapping: проекция

- ▶ Точечный источник светит во все стороны  $\Rightarrow$  удобно использовать перспективную проекцию и subemap-текстуру

# Shadow mapping: проекция

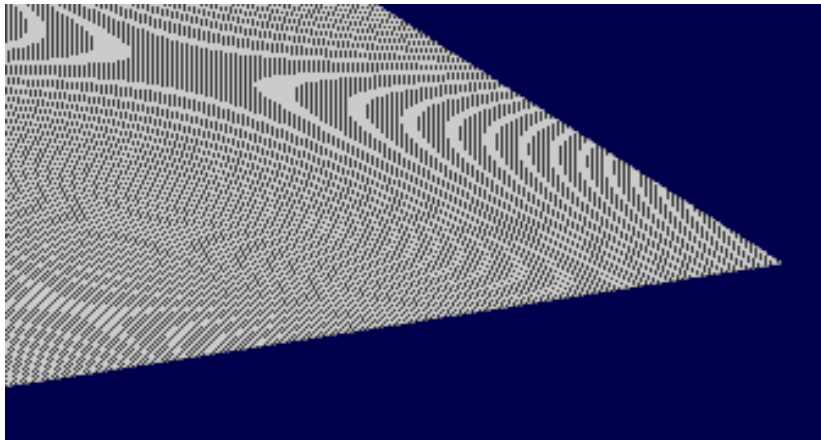
- ▶ Точечный источник светит во все стороны  $\Rightarrow$  удобно использовать перспективную проекцию и subemap-текстуру
- ▶ 6 shadow map (по одной на каждую грань) с соответствующей перспективной проекцией

# Shadow mapping: проекция

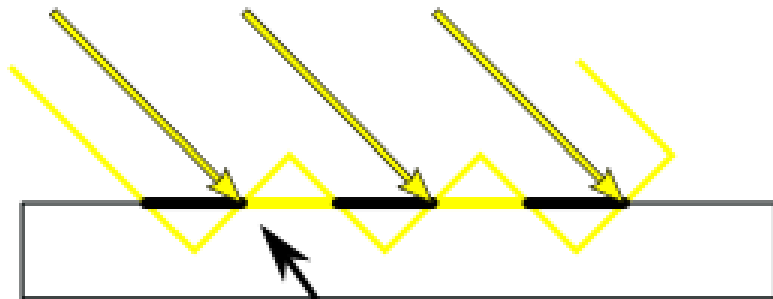
- ▶ Точечный источник светит во все стороны  $\Rightarrow$  удобно использовать перспективную проекцию и subemap-текстуру
- ▶ 6 shadow map (по одной на каждую грань) с соответствующей перспективной проекцией
- ▶ Положение камеры совпадает с положением источника света, виртуальный 'экран' совпадает с отрисовываемой гранью shadow map



## Shadow acne



## Shadow acne



Lightmap pixels

## Shadow acne

- ▶ Артефакт, возникающий из-за дискретности shadow map: если луч света не перпендикулярен плоскости объекта, рядом с центром пикселя shadow map будут точки с меньшей или большей глубиной

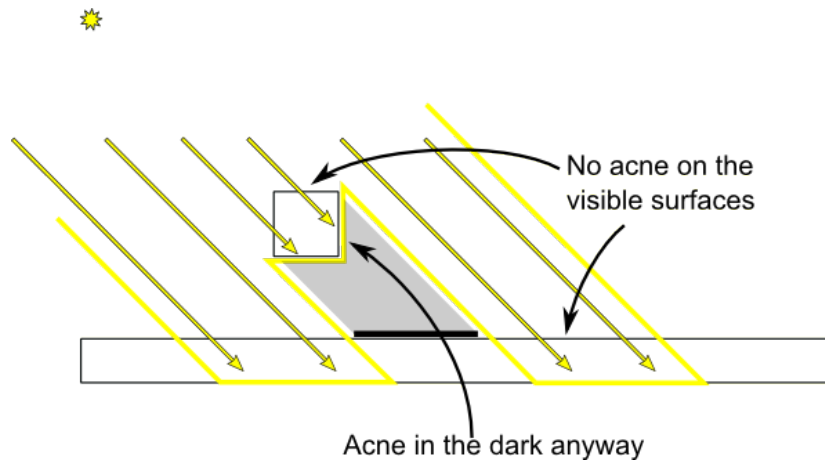
# Shadow acne

- ▶ Артефакт, возникающий из-за дискретности shadow map: если луч света не перпендикулярен плоскости объекта, рядом с центром пикселя shadow map будут точки с меньшей или большей глубиной
- ▶ Первый способ исправить: добавить к значению, прочитанному из shadow map, некую константу (shadow bias)
  - ▶ Часто эту константу умножают на тангенс угла падения света

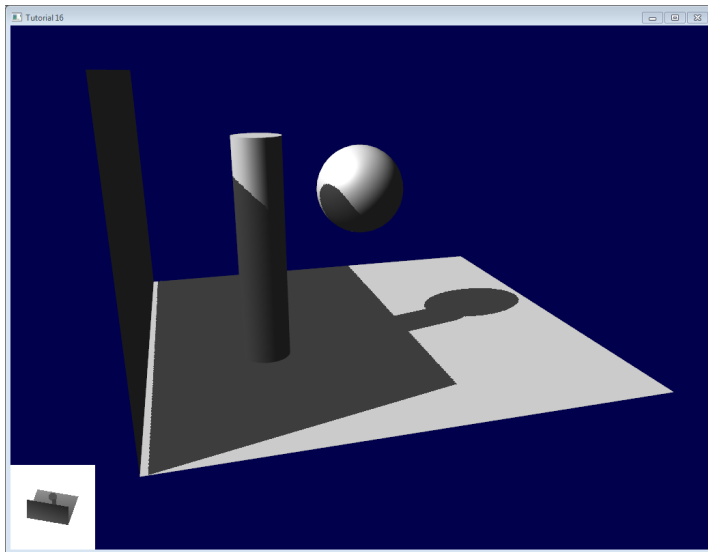
# Shadow acne

- ▶ Артефакт, возникающий из-за дискретности shadow map: если луч света не перпендикулярен плоскости объекта, рядом с центром пикселя shadow map будут точки с меньшей или большей глубиной
- ▶ Первый способ исправить: добавить к значению, прочитанному из shadow map, некую константу (shadow bias)
  - ▶ Часто эту константу умножают на тангенс угла падения света
- ▶ Второй способ исправить: рисовать только back-facing грани (т.е. обращённые против света) в shadow map
  - ▶ Теперь shadow acne возникнет на back-facing гранях, но мы и без shadow map знаем, что они в тени!
  - ▶ Работает только с правильной геометрией (двусторонней, без дырок, и т.п.)

## Back-facing shadows



# Peter panning



# Peter panning

- ▶ Объекты кажутся приподнятыми над поверхностью из-за того, что их тень начинается не сразу под объектом



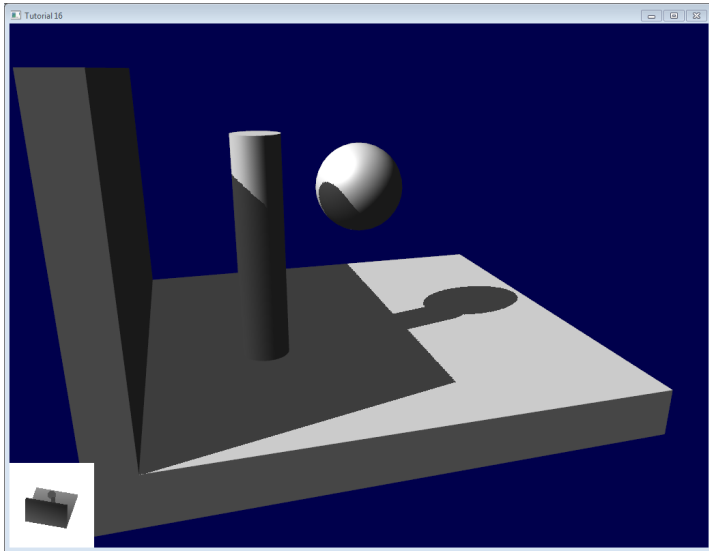
# Peter panning

- ▶ Объекты кажутся приподнятыми над поверхностью из-за того, что их тень начинается не сразу под объектом
- ▶ Усиливается при использовании shadow bias

# Peter panning

- ▶ Объекты кажутся приподнятыми над поверхностью из-за того, что их тень начинается не сразу под объектом
- ▶ Усиливается при использовании shadow bias
- ▶ Лучший способ исправить – не использовать слишком тонкую геометрию

# Shadow mapping



## Разрешение shadow map



# Shadow mapping

- ▶ При низком разрешении shadow map хорошо видны 'пиксели' теней

# Shadow mapping

- ▶ При низком разрешении shadow map хорошо видны 'пиксели' теней
- ▶ Чем выше разрешение, тем лучше жёсткая тень

# Shadow mapping

- ▶ При низком разрешении shadow map хорошо видны 'пиксели' теней
- ▶ Чем выше разрешение, тем лучше жёсткая тень
  - ▶ Алгоритм silhouette mapping позволяет получить жёсткую тень без слишком большого разрешения

# Shadow mapping

- ▶ При низком разрешении shadow map хорошо видны 'пиксели' теней
- ▶ Чем выше разрешение, тем лучше жёсткая тень
  - ▶ Алгоритм silhouette mapping позволяет получить жёсткую тень без слишком большого разрешения
- ▶ Можно слегка размыть тень, избавившись от 'пикселей' и получив аппроксимацию мягких теней



# Shadow mapping

- ▶ При низком разрешении shadow map хорошо видны 'пиксели' теней
- ▶ Чем выше разрешение, тем лучше жёсткая тень
  - ▶ Алгоритм silhouette mapping позволяет получить жёсткую тень без слишком большого разрешения
- ▶ Можно слегка размыть тень, избавившись от 'пикселей' и получив аппроксимацию мягких теней
  - ▶ Размывать нужно результат сравнения глубины со значением в shadow map, а **не** саму shadow map!

# Shadow mapping

- ▶ При низком разрешении shadow map хорошо видны 'пиксели' теней
- ▶ Чем выше разрешение, тем лучше жёсткая тень
  - ▶ Алгоритм silhouette mapping позволяет получить жёсткую тень без слишком большого разрешения
- ▶ Можно слегка размыть тень, избавившись от 'пикселей' и получив аппроксимацию мягких теней
  - ▶ Размывать нужно результат сравнения глубины со значением в shadow map, а **не** саму shadow map!
  - ▶ Percentage-closer filtering (PCF)

# Shadow mapping

- ▶ При низком разрешении shadow map хорошо видны 'пиксели' теней
- ▶ Чем выше разрешение, тем лучше жёсткая тень
  - ▶ Алгоритм silhouette mapping позволяет получить жёсткую тень без слишком большого разрешения
- ▶ Можно слегка размыть тень, избавившись от 'пикселей' и получив аппроксимацию мягких теней
  - ▶ Размывать нужно результат сравнения глубины со значением в shadow map, а **не** саму shadow map!
  - ▶ Percentage-closer filtering (PCF)
  - ▶ В OpenGL есть встроенная реализация: shadow samplers

# Shadow mapping

- ▶ При низком разрешении shadow map хорошо видны ‘пиксели’ теней
- ▶ Чем выше разрешение, тем лучше жёсткая тень
  - ▶ Алгоритм silhouette mapping позволяет получить жёсткую тень без слишком большого разрешения
- ▶ Можно слегка размыть тень, избавившись от ‘пикселей’ и получив аппроксимацию мягких теней
  - ▶ Размывать нужно результат сравнения глубины со значением в shadow map, а **не** саму shadow map!
  - ▶ Percentage-closer filtering (PCF)
  - ▶ В OpenGL есть встроенная реализация: shadow samplers
- ▶ Более современные вариации shadow maps пишут в текстуру не глубину, а какие-нибудь значения, которые имеет смысл складывать и усреднять

# Shadow mapping

- ▶ При низком разрешении shadow map хорошо видны ‘пиксели’ теней
- ▶ Чем выше разрешение, тем лучше жёсткая тень
  - ▶ Алгоритм silhouette mapping позволяет получить жёсткую тень без слишком большого разрешения
- ▶ Можно слегка размыть тень, избавившись от ‘пикселей’ и получив аппроксимацию мягких теней
  - ▶ Размывать нужно результат сравнения глубины со значением в shadow map, а **не** саму shadow map!
  - ▶ Percentage-closer filtering (PCF)
  - ▶ В OpenGL есть встроенная реализация: shadow samplers
- ▶ Более современные вариации shadow maps пишут в текстуру не глубину, а какие-нибудь значения, которые имеет смысл складывать и усреднять
  - ▶ Позволяет использовать фильтрацию текстур, mipmap, размытие shadow map

# Shadow mapping

- ▶ При низком разрешении shadow map хорошо видны 'пиксели' теней
- ▶ Чем выше разрешение, тем лучше жёсткая тень
  - ▶ Алгоритм silhouette mapping позволяет получить жёсткую тень без слишком большого разрешения
- ▶ Можно слегка размыть тень, избавившись от 'пикселей' и получив аппроксимацию мягких теней
  - ▶ Размывать нужно результат сравнения глубины со значением в shadow map, а **не** саму shadow map!
  - ▶ Percentage-closer filtering (PCF)
  - ▶ В OpenGL есть встроенная реализация: shadow samplers
- ▶ Более современные вариации shadow maps пишут в текстуру не глубину, а какие-нибудь значения, которые имеет смысл складывать и усреднять
  - ▶ Позволяет использовать фильтрацию текстур, mipmap, размытие shadow map
  - ▶ Exponential shadow maps, variance shadow maps (следующая лекция)

## Shadow sampler

- ▶ В OpenGL есть ограниченная поддержка PCF в виде shadow sampler'ов

# Shadow sampler

- ▶ В OpenGL есть ограниченная поддержка PCF в виде shadow sampler'ов
- ▶ Для текстуры нужно настроить параметры:
  - ▶ `GL_TEXTURE_COMPARE_MODE` в значение `GL_COMPARE_REF_TO_TEXTURE` (по умолчанию – `GL_NONE`)
  - ▶ `GL_TEXTURE_COMPARE_FUNC` в значение `GL_LEQUAL` (теоретически могут быть `GL_ALWAYS` и т.п.)



# Shadow sampler

- ▶ В OpenGL есть ограниченная поддержка PCF в виде shadow sampler'ов
- ▶ Для текстуры нужно настроить параметры:
  - ▶ `GL_TEXTURE_COMPARE_MODE` в значение `GL_COMPARE_REF_TO_TEXTURE` (по умолчанию – `GL_NONE`)
  - ▶ `GL_TEXTURE_COMPARE_FUNC` в значение `GL_LEQUAL` (теоретически могут быть `GL_ALWAYS` и т.п.)
- ▶ Текстуру с `GL_COMPARE_REF_TO_TEXTURE` **нельзя** использовать как `sampler2D`, для неё есть отдельный тип sampler'a: `sampler2DShadow`

# Shadow sampler

- ▶ В OpenGL есть ограниченная поддержка PCF в виде shadow sampler'ов
- ▶ Для текстуры нужно настроить параметры:
  - ▶ `GL_TEXTURE_COMPARE_MODE` в значение `GL_COMPARE_REF_TO_TEXTURE` (по умолчанию – `GL_NONE`)
  - ▶ `GL_TEXTURE_COMPARE_FUNC` в значение `GL_LEQUAL` (теоретически могут быть `GL_ALWAYS` и т.п.)
- ▶ Текстуру с `GL_COMPARE_REF_TO_TEXTURE` **нельзя** использовать как `sampler2D`, для неё есть отдельный тип sampler'a: `sampler2DShadow`
- ▶ Функция `texture(shadow_map, texcoord)` принимает трёхмерный вектор текстурных координат:
  - ▶ Первые две – обычные X,Y текстурные координаты
  - ▶ Третья – используется для сравнения со значением в shadow map

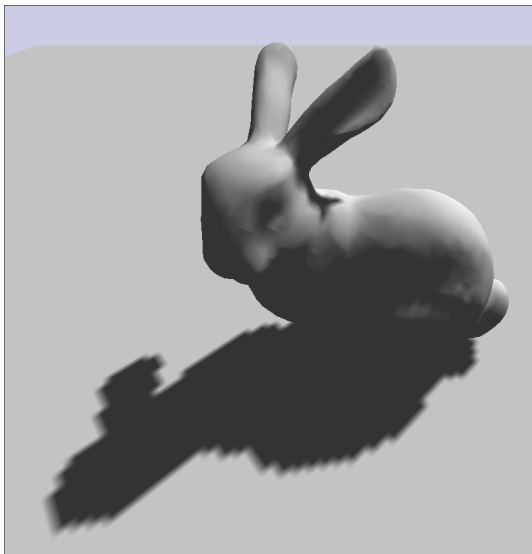
# Shadow sampler

- ▶ В OpenGL есть ограниченная поддержка PCF в виде shadow sampler'ов
- ▶ Для текстуры нужно настроить параметры:
  - ▶ `GL_TEXTURE_COMPARE_MODE` в значение `GL_COMPARE_REF_TO_TEXTURE` (по умолчанию – `GL_NONE`)
  - ▶ `GL_TEXTURE_COMPARE_FUNC` в значение `GL_LEQUAL` (теоретически могут быть `GL_ALWAYS` и т.п.)
- ▶ Текстуру с `GL_COMPARE_REF_TO_TEXTURE` **нельзя** использовать как `sampler2D`, для неё есть отдельный тип sampler'a: `sampler2DShadow`
- ▶ Функция `texture(shadow_map, texcoord)` принимает трёхмерный вектор текстурных координат:
  - ▶ Первые две – обычные X,Y текстурные координаты
  - ▶ Третья – используется для сравнения со значением в shadow map
- ▶ Если сравнение успешно, возвращается 1, иначе 0 (функция сравнения настраивается `GL_TEXTURE_COMPARE_FUNC`)

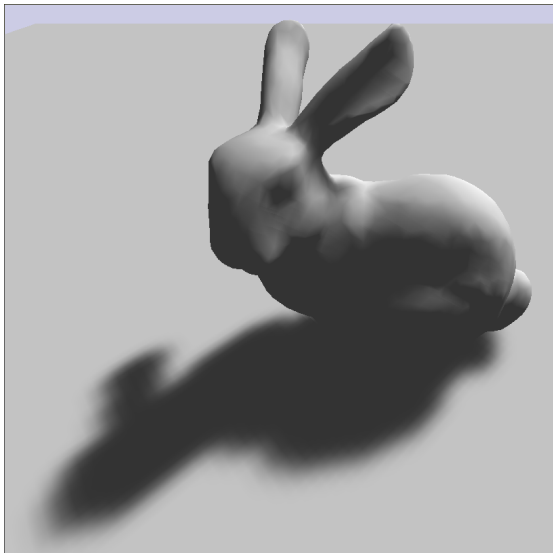
# Shadow sampler

- ▶ В OpenGL есть ограниченная поддержка PCF в виде shadow sampler'ов
- ▶ Для текстуры нужно настроить параметры:
  - ▶ `GL_TEXTURE_COMPARE_MODE` в значение `GL_COMPARE_REF_TO_TEXTURE` (по умолчанию – `GL_NONE`)
  - ▶ `GL_TEXTURE_COMPARE_FUNC` в значение `GL_LEQUAL` (теоретически могут быть `GL_ALWAYS` и т.п.)
- ▶ Текстуру с `GL_COMPARE_REF_TO_TEXTURE` **нельзя** использовать как `sampler2D`, для неё есть отдельный тип sampler'a: `sampler2DShadow`
- ▶ Функция `texture(shadow_map, texcoord)` принимает трёхмерный вектор текстурных координат:
  - ▶ Первые две – обычные X,Y текстурные координаты
  - ▶ Третья – используется для сравнения со значением в shadow map
- ▶ Если сравнение успешно, возвращается 1, иначе 0 (функция сравнения настраивается `GL_TEXTURE_COMPARE_FUNC`)
- ▶ Для такой текстуры можно включить линейную фильтрацию – интерполироваться будут не значения глубины, а *результаты сравнения*

## Shadow sampler с линейной фильтрацией



## Shadow sampler с линейной фильтрацией и $7 \times 7$ размытием по Гауссу



## Shadow mapping: ссылки

- ▶ [learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping](http://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping)
- ▶ [opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping](http://opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping)
- ▶ [ogldev.org/www/tutorial23/tutorial23.html](http://ogldev.org/www/tutorial23/tutorial23.html)
- ▶ GPU Gems 2, Chapter 17. Efficient Soft-Edged Shadows Using Pixel Shader Branching