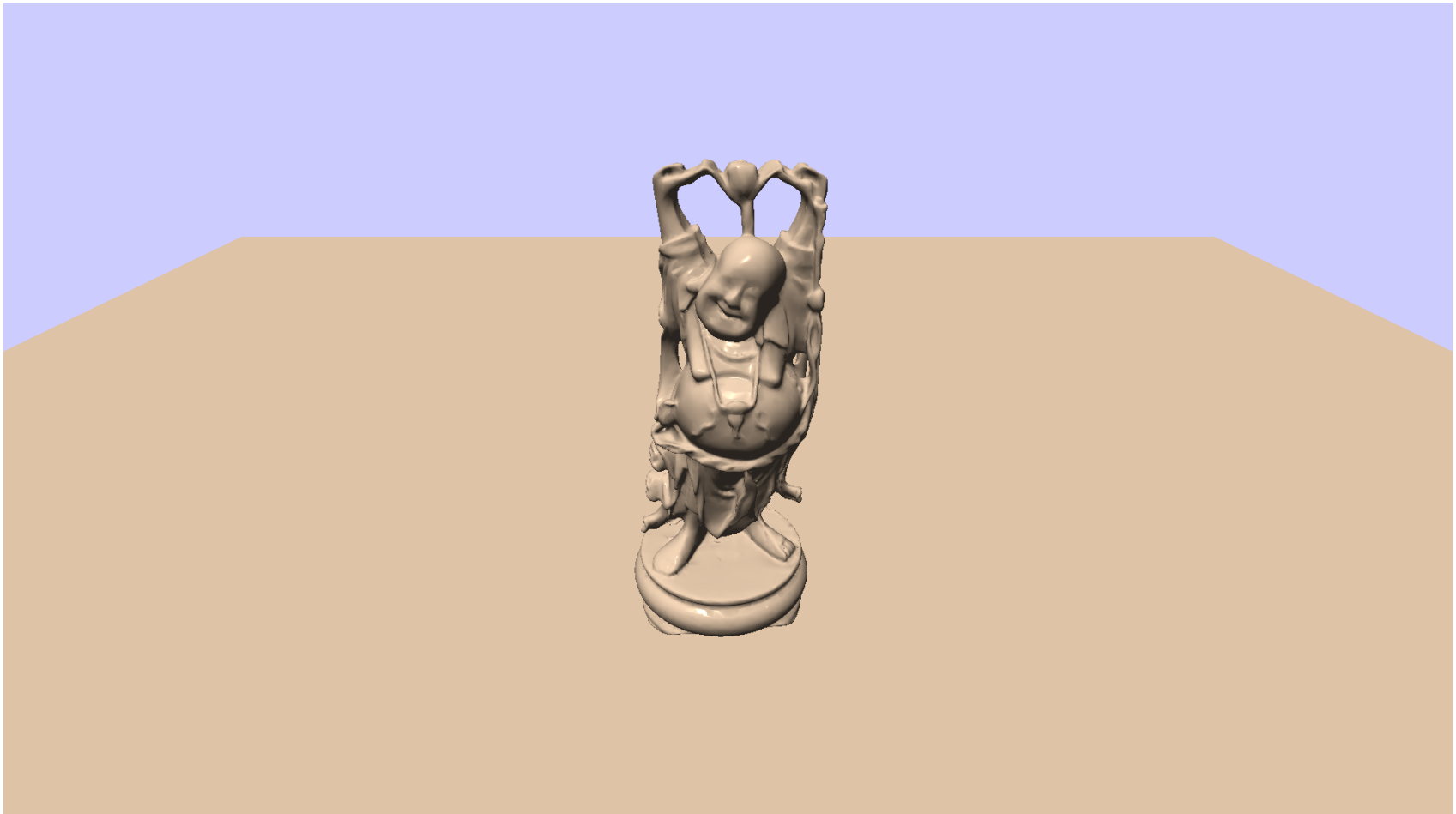
Компьютерная графика

Практика 8: Shadow mapping

2021



Задание 1

Создаём и настраиваем shadow map и framebuffer

- Выбираем размер shadow map: например, `shadow_map_size = 1024`
- Создаём текстуру для shadow map: min/mag фильтры – `GL_NEAREST`, размеры – `shadow_map_size x shadow_map_size`, internal format – `GL_DEPTH_COMPONENT24`, format – `GL_DEPTH_COMPONENT`, type – `GL_FLOAT`, в данных – `nullptr`
- Настраиваем ей параметры `GL_TEXTURE_WRAP_S` и `GL_TEXTURE_WRAP_T` в значение `GL_CLAMP_TO_EDGE`
- Создаём framebuffer, присоединяем к нему нашу текстуру в качестве глубины (`glFramebufferTexture`, `GL_DEPTH_ATTACHMENT`), `target` лучше использовать `GL_DRAW_FRAMEBUFFER`
- Проверяем, что фреймбуффер настроен правильно (`glCheckFramebufferStatus`)
- N.B. Экран будет чёрный, так как мы не сделали дефолтный фреймбуфер текущим :)

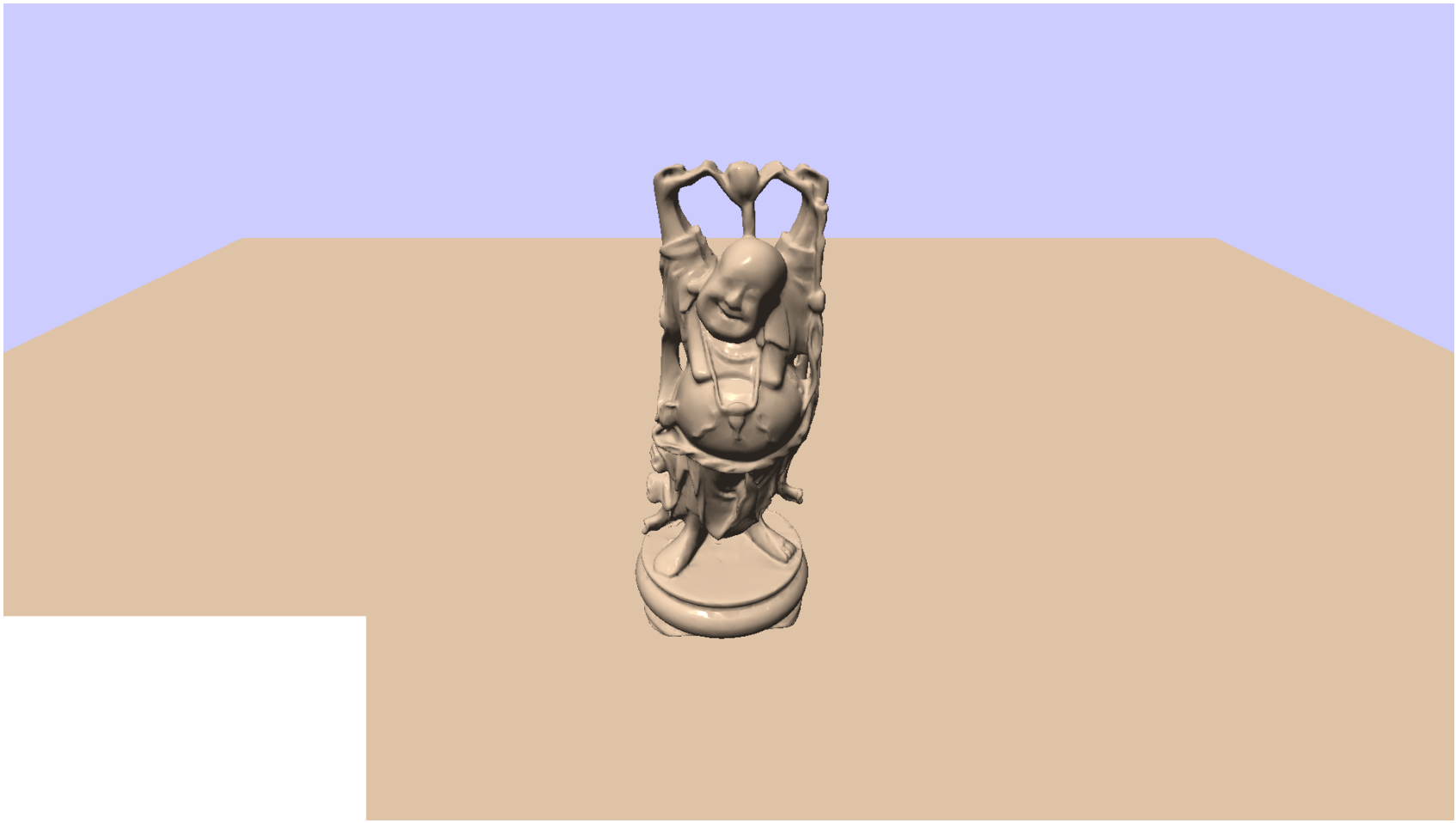
Задание 1

Задание 2

Добавляем дебажный прямоугольник с собственной шейдерной программой, чтобы видеть содержимое нашей shadow map

- В начале рендеринга (перед `glClear`) делаем текущим дефолтный фреймбуфер, чтобы снова увидеть сцену
- Создаём вершинный шейдер: выдаёт захардкоженные координаты вершин, используя `gl_VertexID` (как в первой практике), и передаёт во фрагментный шейдер текстурные координаты (без каких-либо матриц)
  - Должно быть 6 вершин – два треугольника, образующих прямоугольник
  - Координаты вершин должны быть где-то в нижнем левом углу экрана (например, [-1.0 .. -0.5] по обеим осям)
  - Текстурные координаты должны быть [0.0 .. 1.0]
- Фрагментный шейдер: читает цвет из переданной текстуры и выводит в `out_color`, можно только красный канал: `vec4(texture(...).r)`
- Создаём фиктивный VAO (без настройки атрибутов вершин)
- После рисования основной модели рисуем прямоугольник с помощью `glDrawArrays(GL_TRIANGLES, 0, 6)` (не забываем сделать текущими VAO, программу и текстуру shadow map, а также выключить тест глубины)
- N.B. прямоугольник будет белым, так как shadow map пока пустой

Задание 2



Задание 3

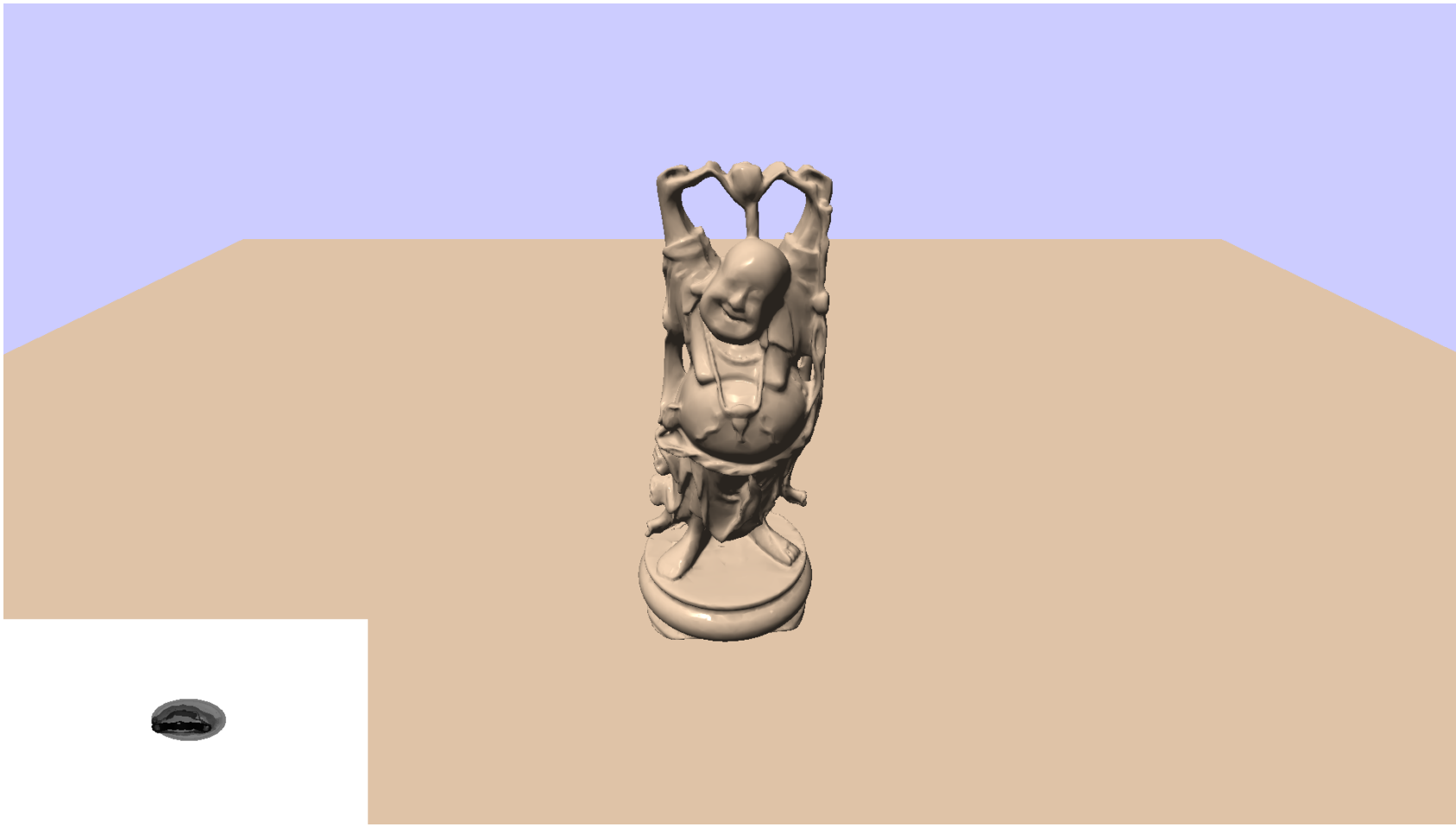
Генерируем shadow map

- Выбираем проекцию для shadow map: для начала сгодится проекция ‘снизу-вверх’ (как будто камера смотрит сверху)
- При рисовании кадра вычисляем оси проекции:
  - `light_Z = glm::vec3(0, -1, 0)`
  - `light_X = glm::vec3(1,  0, 0)`
  - `light_Y = glm::cross(light_X, light_Z)`
- Матрица проекции:

```
glm::mat4(glm::transpose(
    glm::mat3(light_X, light_Y, light_Z)))
```

- Пишем шейдерную программу для вычисления shadow map:
  - Вершинный шейдер преобразует вершины (`gl_Position = shadow_projection * model * ...`)
  - Фрагментный шейдер ничего не делает (пустая функция `main`)
- Перед рисованием основного кадра: используем созданный ранее shadow framebuffer для рисования, настраиваем `glViewport`, очищаем буфер глубины, включаем front-face culling, включаем depth test, рисуем нашу модель созданной шейдерной программой
- После этого не забываем вернуть back-face culling
- Модель должна появиться в нашем дебажном прямоугольнике

Задание 3



Задание 4

Используем shadow map

- Передаём текстуру shadow map и проекцию для неё в основную шейдерную программу
- Во фрагментном шейдере:
  - Вычисляем ndc-координаты текущей точки после применения проекции:
    `vec4 ndc = shadow_projection * vec4(position, 1.0)`
  - Проверяем точку на попадание в видимую область shadow map (XY-координаты `ndc` должны быть в диапазоне [-1..1])
  - Если точка попала в shadow map, вычисляем её текстурные координаты для shadow map `ndc.xy * 0.5 + 0.5` и глубину `ndc.z * 0.5 + 0.5`
  - Если значение в shadow map меньше глубины нашей точки, она в тени (к ней не нужно применять прямое освещение, но ambient остаётся)
- Тень будет выглядеть так, будто свет падает сверху

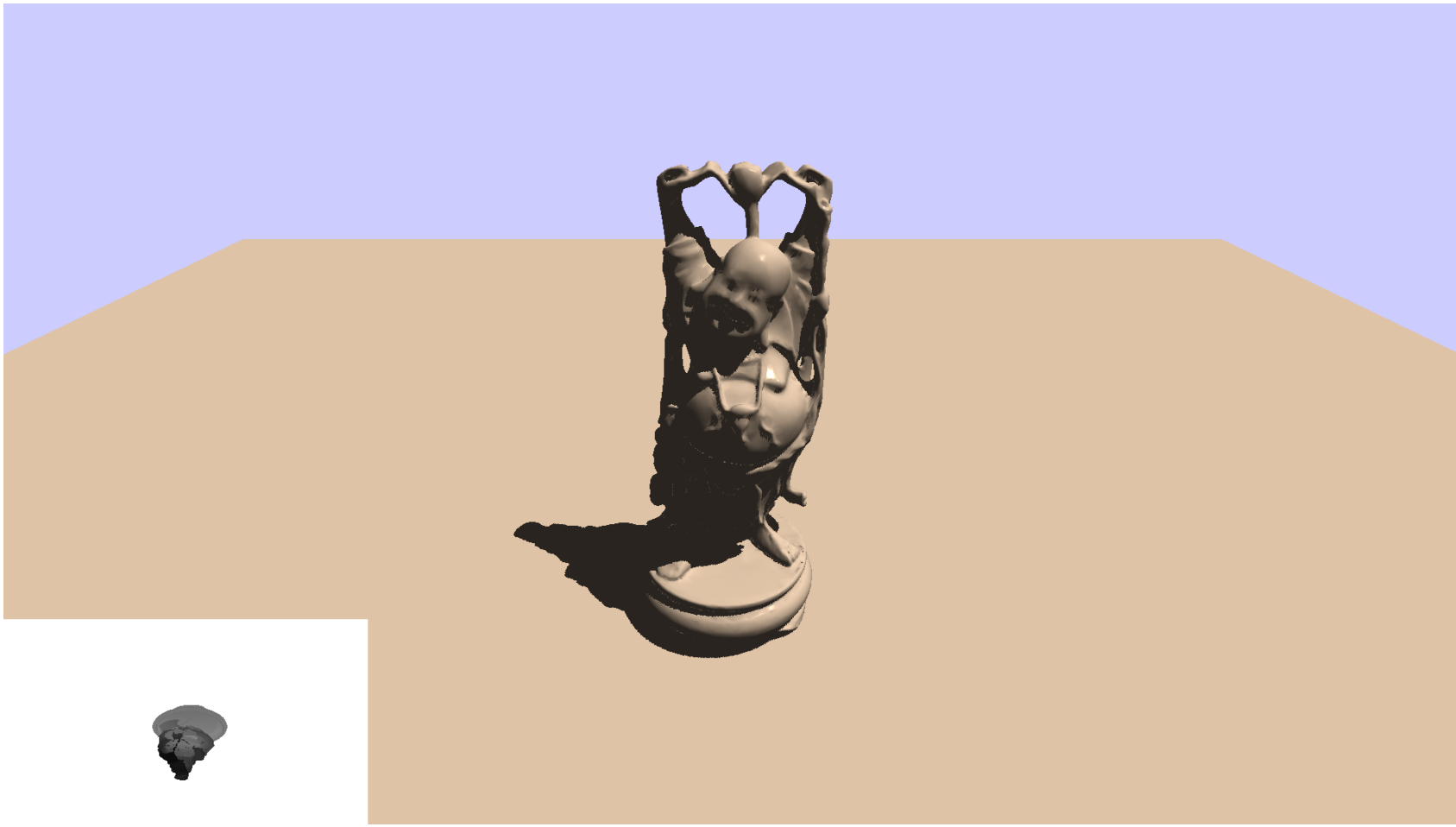
Задание 4

Задание 5

Вычисляем настоящую проекцию

- `light_Z = -light_direction`
- `light_X` – любой вектор, ортогональный `light_Z`
- `light_Y = glm::cross(light_X, light_Z)`

Задание 5

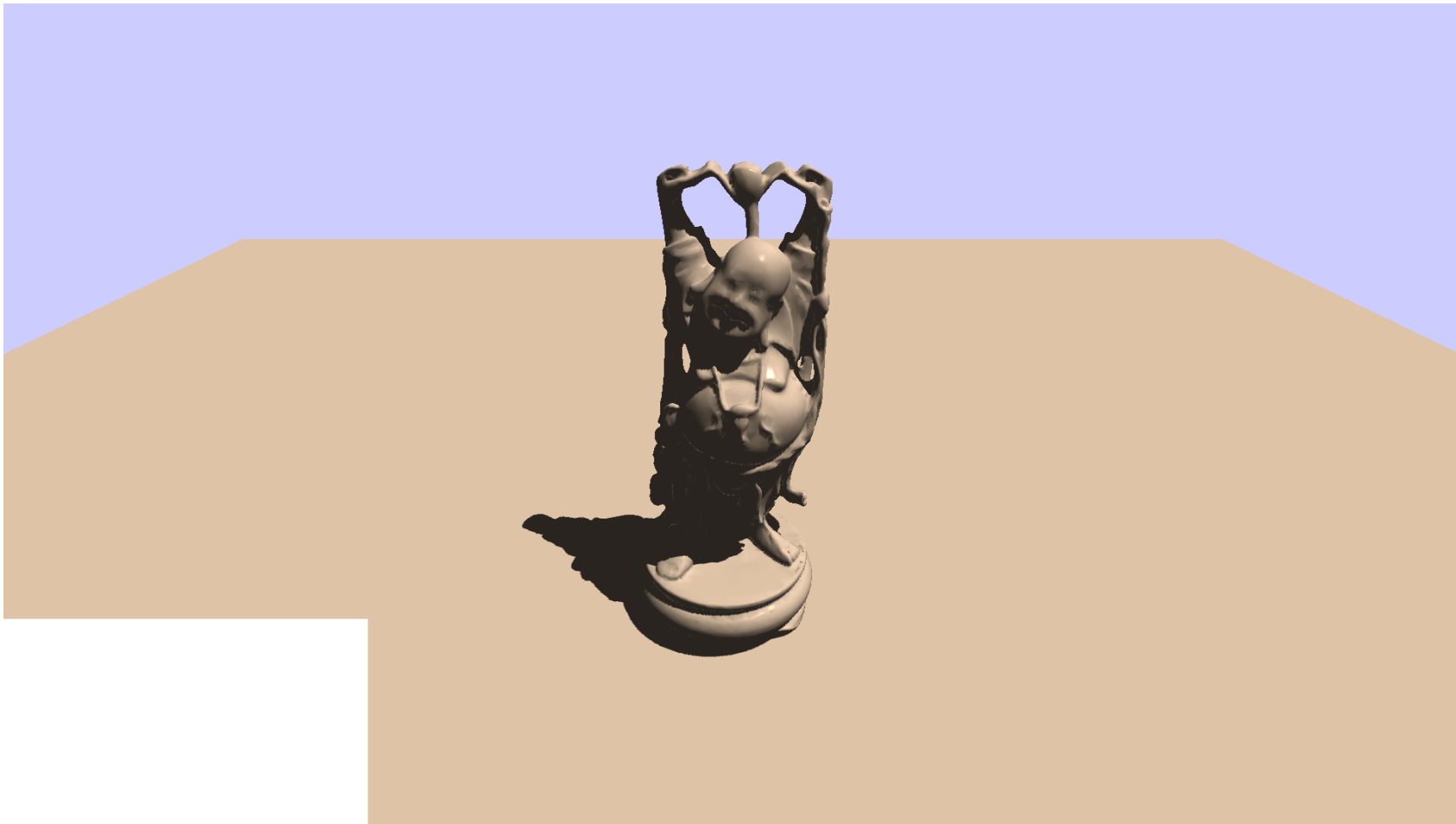


Задание 6

Включаем PCF

- Меняем min/mag фильтры shadow map на `GL_LINEAR`
- Настраиваем текстуре shadow map опции `GL_TEXTURE_COMPARE_MODE = GL_COMPARE_REF_TO_TEXTURE` и `GL_TEXTURE_COMPARE_FUNC = GL_LEQUAL`
- Заменяем в основном фрагментном шейдере `sampler2D shadow_map` на `sampler2DShadow`
- Сравнение `texture(shadow_map, texcoord) < depth` заменяется на один вызов `texture(shadow_map, ndc * 0.5 + 0.5)` – вернёт 0 или 1 (если в тени или не в тени, соответственно)
- N.B. дебажный прямоугольник перестанет работать :(

Задание 6



Задание 7*

Добавляем размытие к PCF

- Во фрагментном шейдере, вместо однократного чтения shadow map `texture(shadow_map, ndc * 0.5 + 0.5)` читаем значения из соседних пикселей (надо будет что-то прибавить к `ndc`) и усредняем по Гауссу
- Тени должны получиться более размытыми

Задание 7