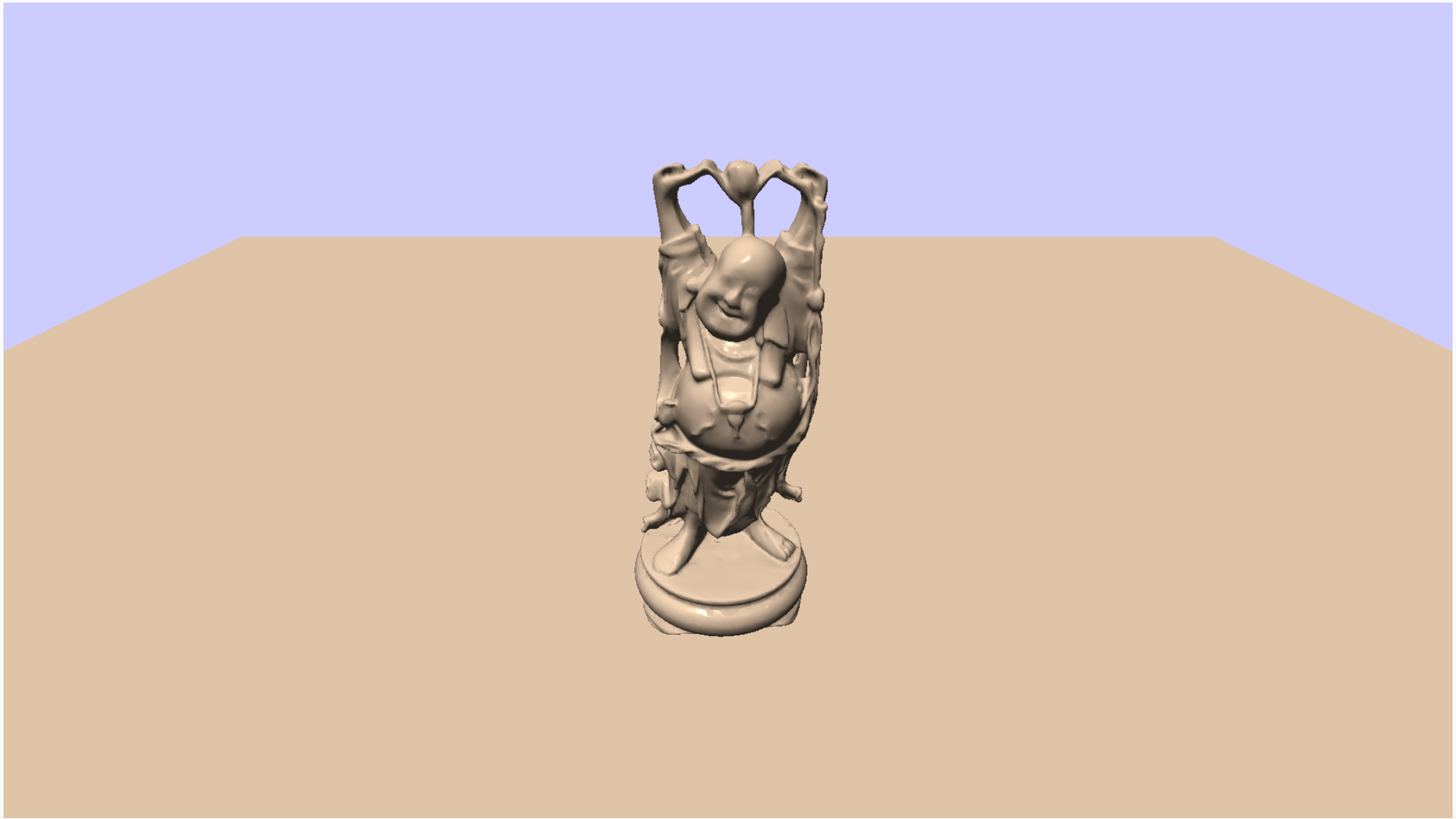
Компьютерная графика

Практика 8: Shadow mapping

2023

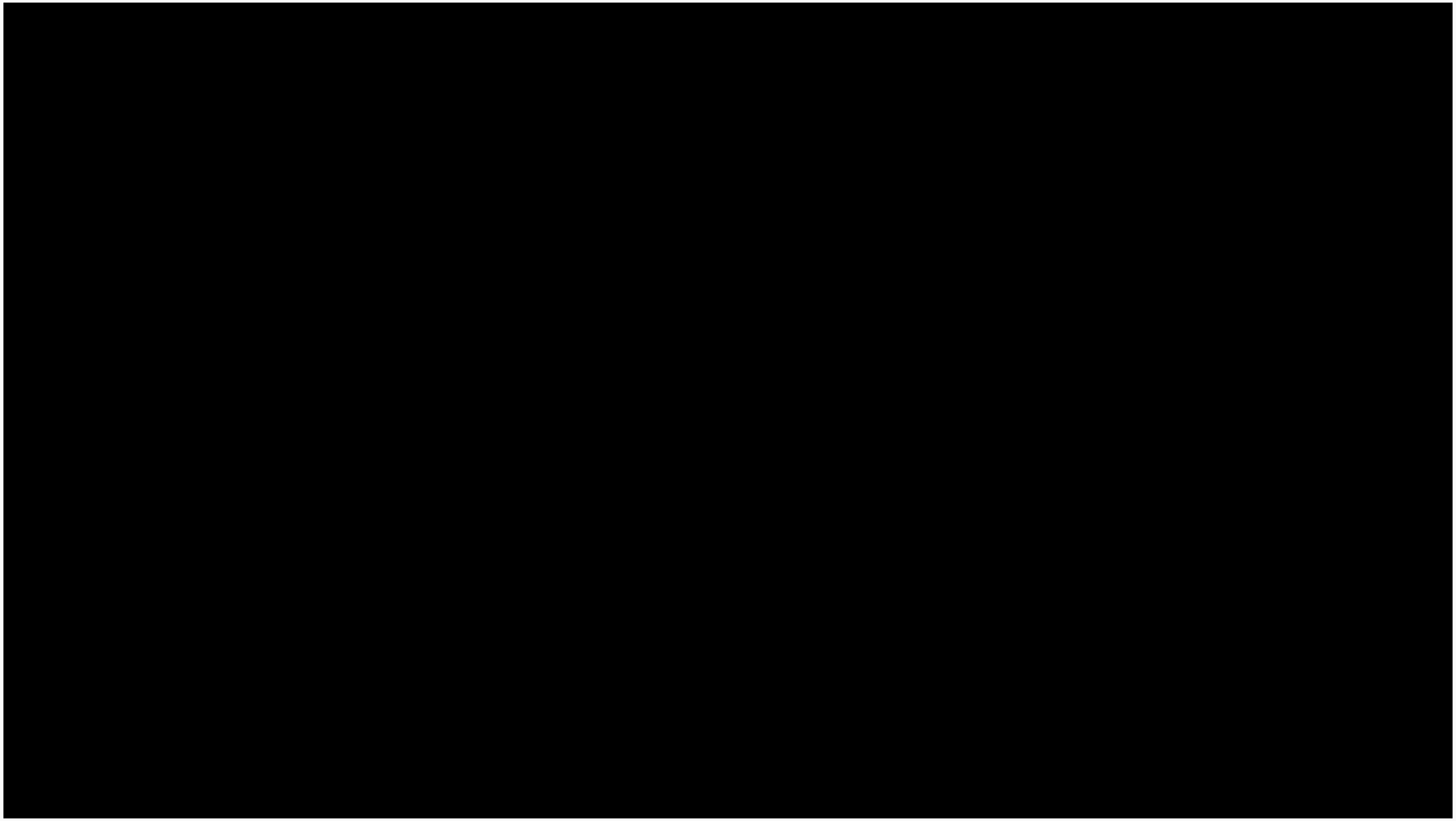


Задание 1

При инициализации 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`, сразу после обработки событий) делаем текущим дефолтный (ID = 0) фреймбуфер, чтобы снова увидеть сцену
- Создаём новый вершинный шейдер: выдаёт (в `gl_Position` захардкоженные координаты вершин, используя `gl_VertexID` (как в первой практике), и передаёт (через `out vec2 texcoord`) 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] по обеим осям, чтобы они покрыли всю текстуру, т.е. (0, 0) у левого нижнего угла, (1, 0) у правого нижнего, и т.д.

Задание 2

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

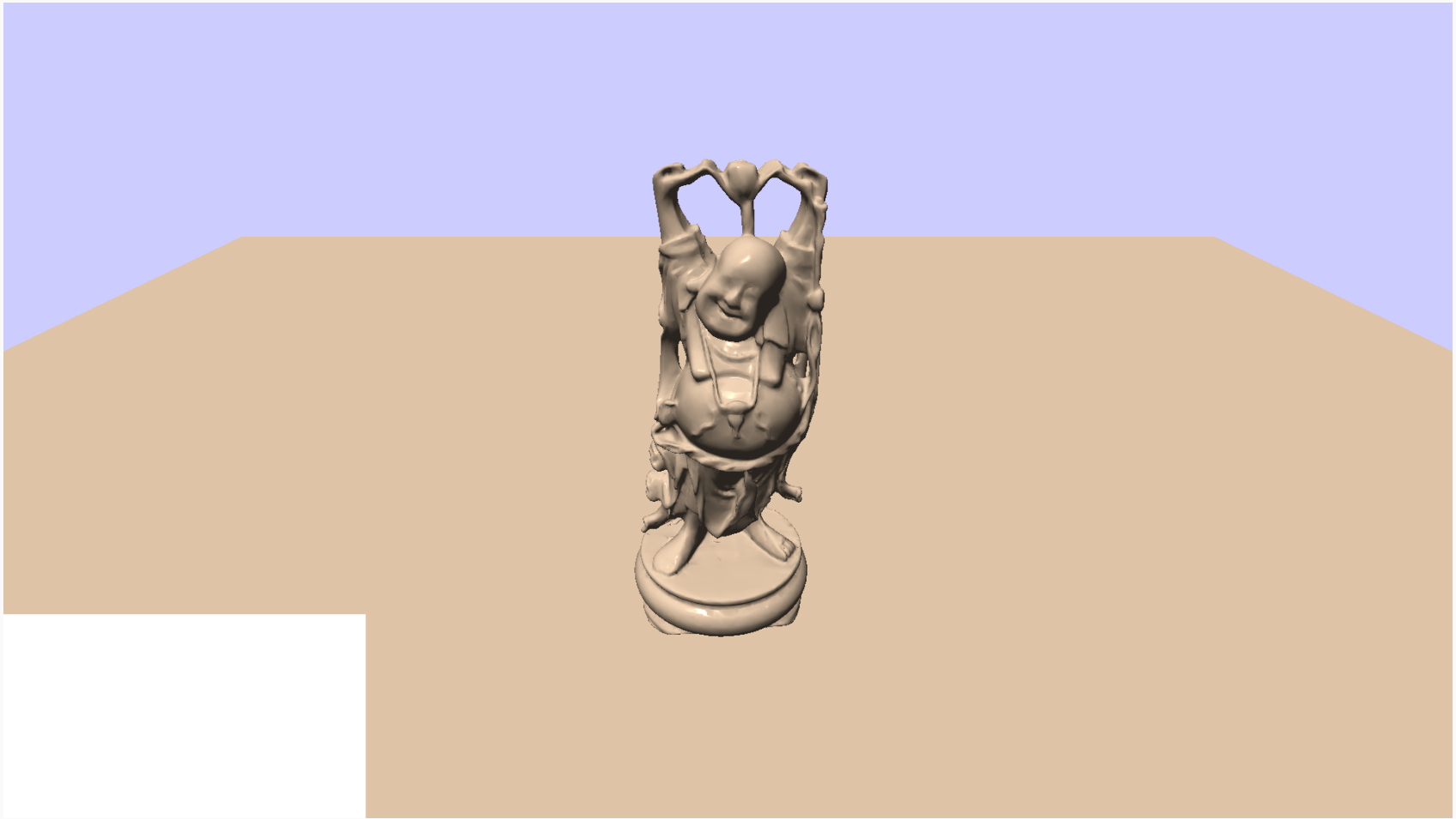
- Фрагментный шейдер: читает цвет из переданной текстуры (`uniform sampler2D`) и выводит в `out_color`, можно только красный канал: `vec4(texture(...).r)` (другие каналы содержат нули – так себя ведёт тип пикселя `GL_DEPTH_COMPONENT`)
- При инициализации создаём фиктивный VAO (без настройки атрибутов вершин)
- После рисования основной модели, перед `SDL_GL_SwapBuffers` рисуем прямоугольник с помощью `glDrawArrays(GL_TRIANGLES, 0, 6)` (не забываем сделать текущими созданный VAO, новую шейдерную программу и текстуру shadow map, а также выключить тест глубины, чтобы прямоугольник не оказался ‘за’ основной сценой)
- По-хорошему для связи `sampler2D` и текстуры нужен texture unit; для простоты можем воспользоваться тем, что по умолчанию активный texture unit – нулевой, и значение uniform-переменных по умолчанию – тоже ноль
- N.B. прямоугольник будет белым (или чёрным, зависит от драйвера), так как shadow map пока пустой

Задание 2

Теперь весь код рисования кадра должен выглядеть как-то так:

- Делаем текущим дефолтный (ID = 0) фреймбуфер, настраиваем viewport, очищаем color и depth буферы, настраиваем depth test и culling
- Включаем основную шейдерную программу, рисуем сцену
- Включаем новую шейдерную программу, рисуем прямоугольник

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)` – направление, противоположное вгзляду камеры
- `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)))` (пользуемся тем, что матрица из этих трёх векторов – ортогональная; в общем случае `transpose` надо заменить на `inverse`: см. лекцию про камеры и проекции)

Задание 3

Генерируем shadow map

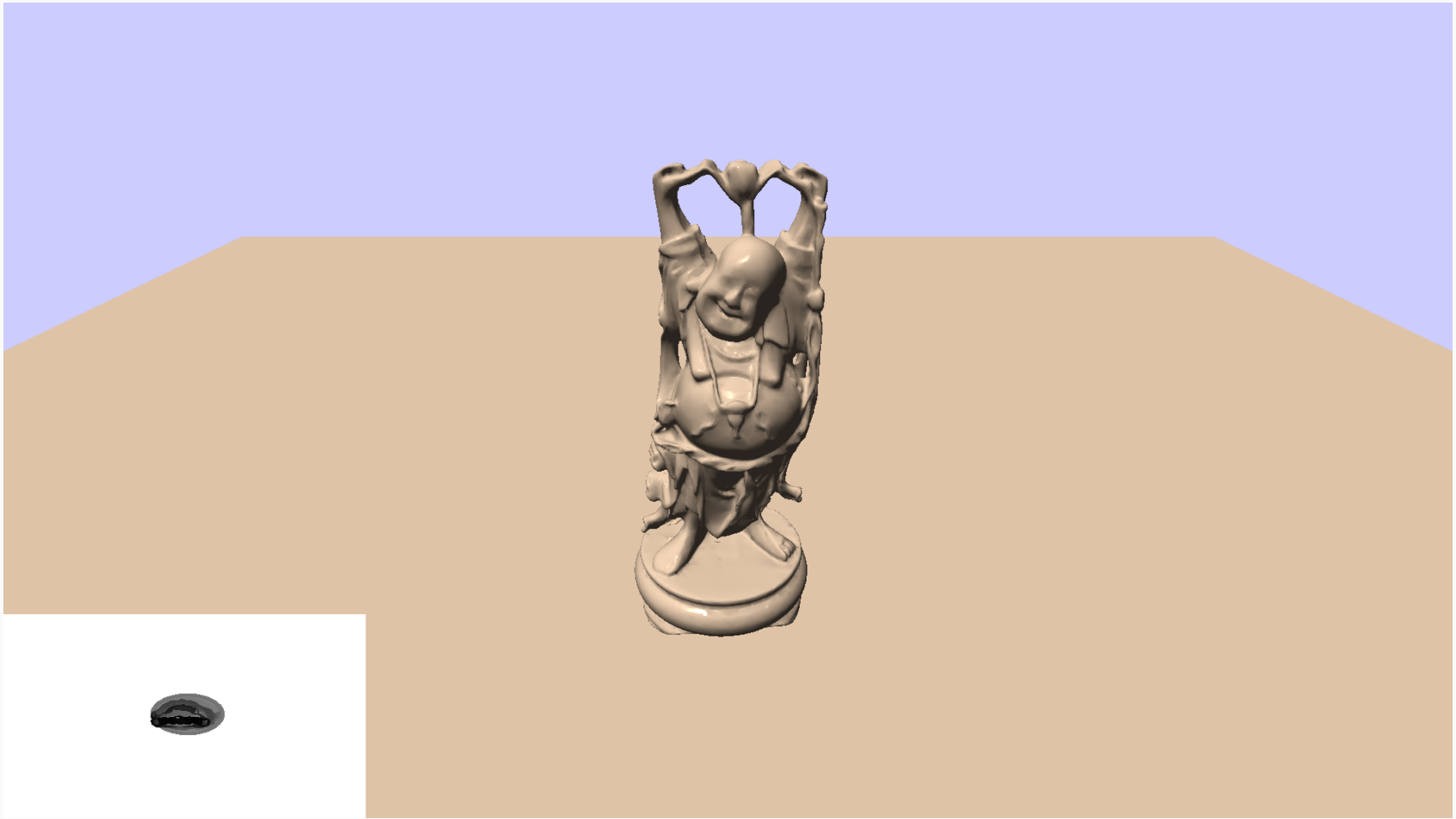
- Пишем новую шейдерную программу для вычисления shadow map:
- Вершинный шейдер преобразует вершины `gl_Position = shadow_projection * model * vec4(in_position, 1.0)` (матрица view здесь не нужна – мы не настраиваем реальную камеру, а просто вычисляем тени)
- Фрагментный шейдер ничего не делает (пустая функция `main`; глубина пикселя, которая нам и нужна, пишется сама, автоматически)
- **Перед** рисованием основной сцены и прямоугольника: используем созданный ранее фреймбуфер для рисования, настраиваем viewport (размер – `shadow_map_size x shadow_map_size`), очищаем буфер глубины, включаем front-face culling (чтобы избавиться от shadow acne), 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

Теперь весь код рисования кадра должен выглядеть как-то так:

- Делаем текущим созданный в задании 1 фреймбуфер, настраиваем viewport, очищаем depth буфер, настраиваем depth test и front-face culling
- Включаем шейдерную программу для рисования shadow map, рисуем сцену
- Делаем текущим дефолтный (ID = 0) фреймбуфер, настраиваем viewport, очищаем color и depth буферы, настраиваем depth test и back-face culling
- Включаем основную шейдерную программу, рисуем сцену
- Включаем шейдерную программу для прямоугольника, рисуем прямоугольник

Задание 3

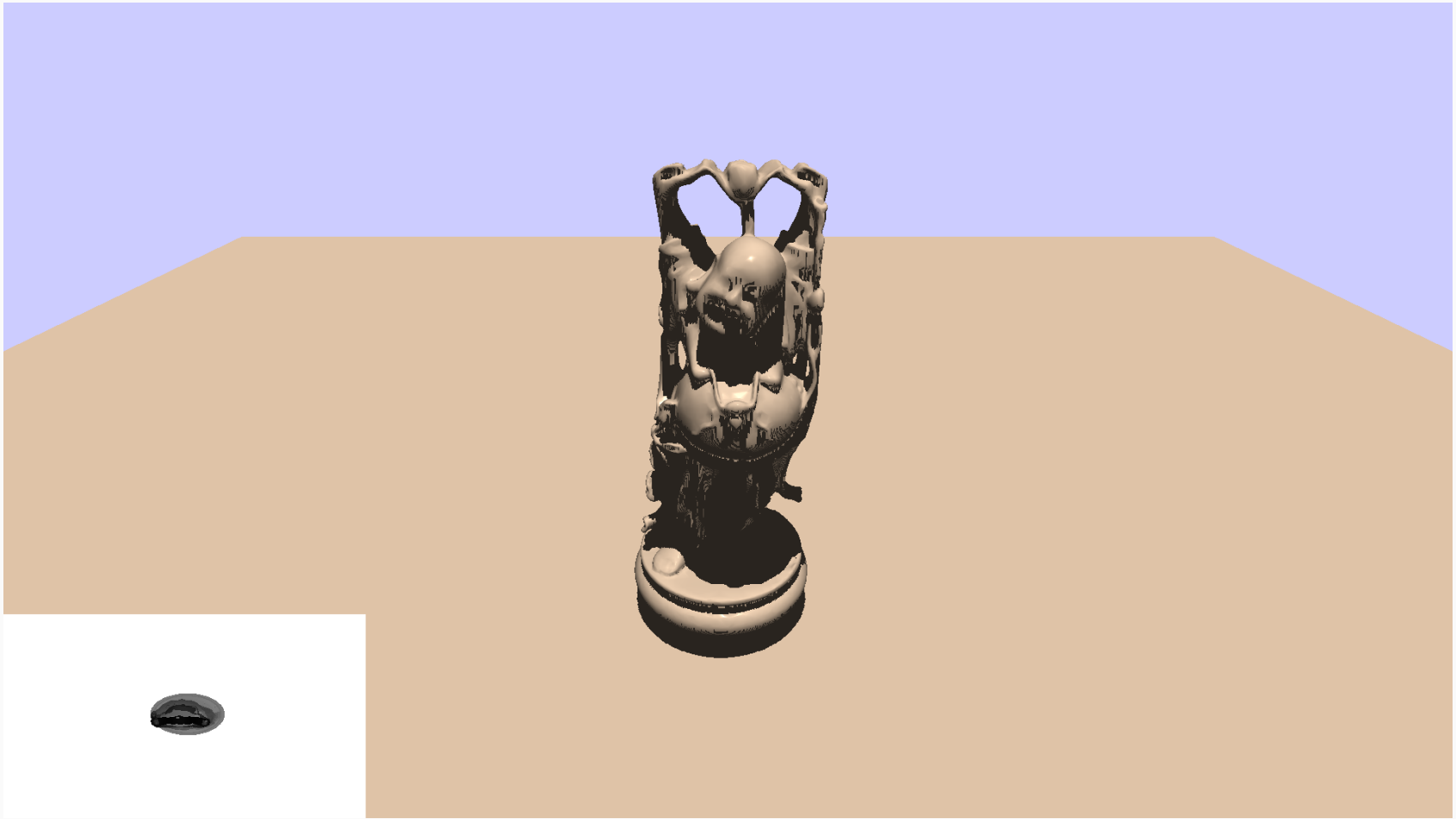


Задание 4

Используем shadow map

- Передаём текстуру shadow map (`uniform sampler2D`) и проекцию для неё (`uniform mat4 projection`) 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 `shadow_texcoord = ndc.xy * 0.5 + 0.5` и глубину `shadow_depth = ndc.z * 0.5 + 0.5`
  - Если значение в shadow map `texture(shadow_map, shadow_texcoord)` 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

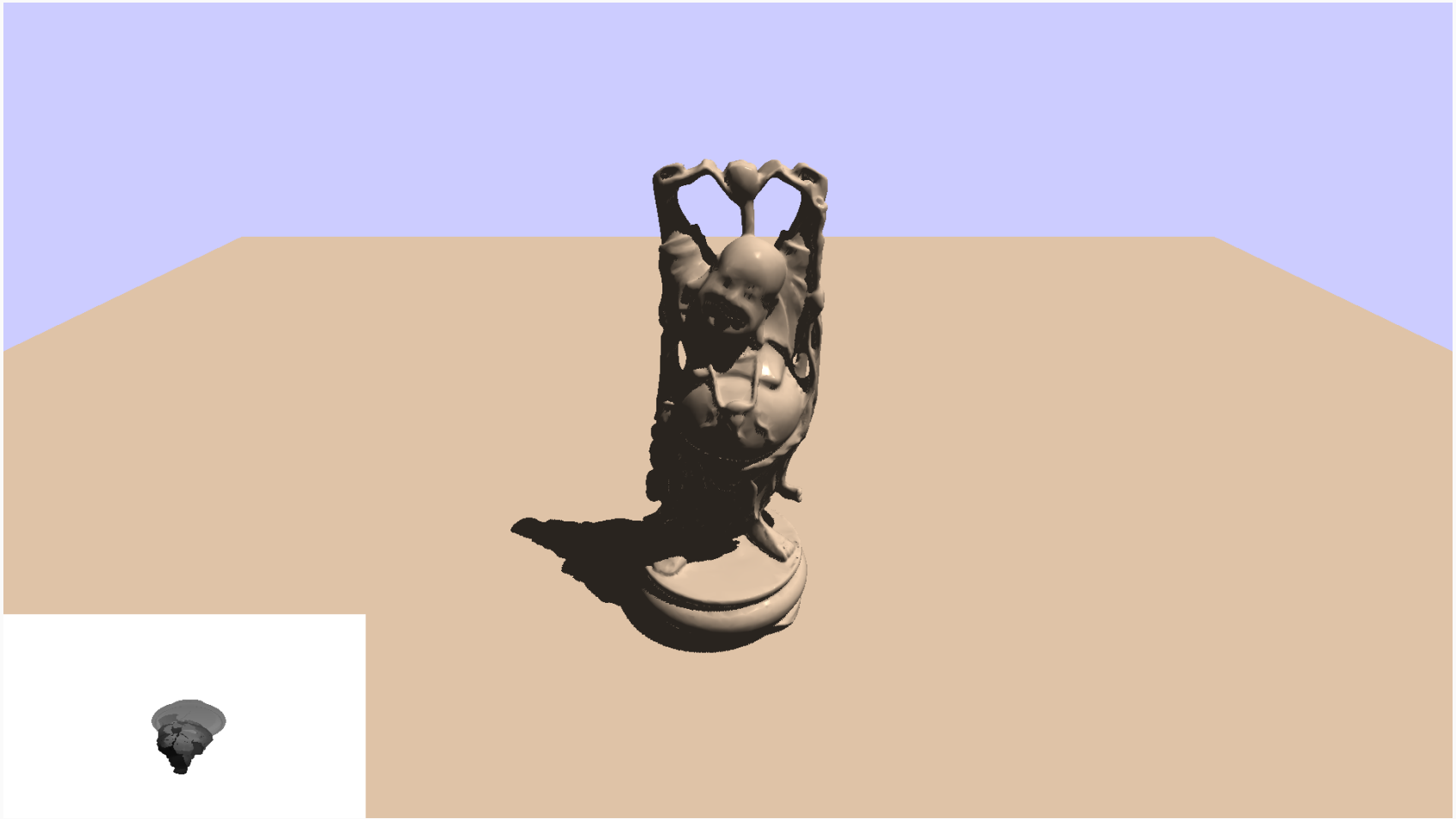


Задание 5

Вычисляем настоящую проекцию

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

Задание 5

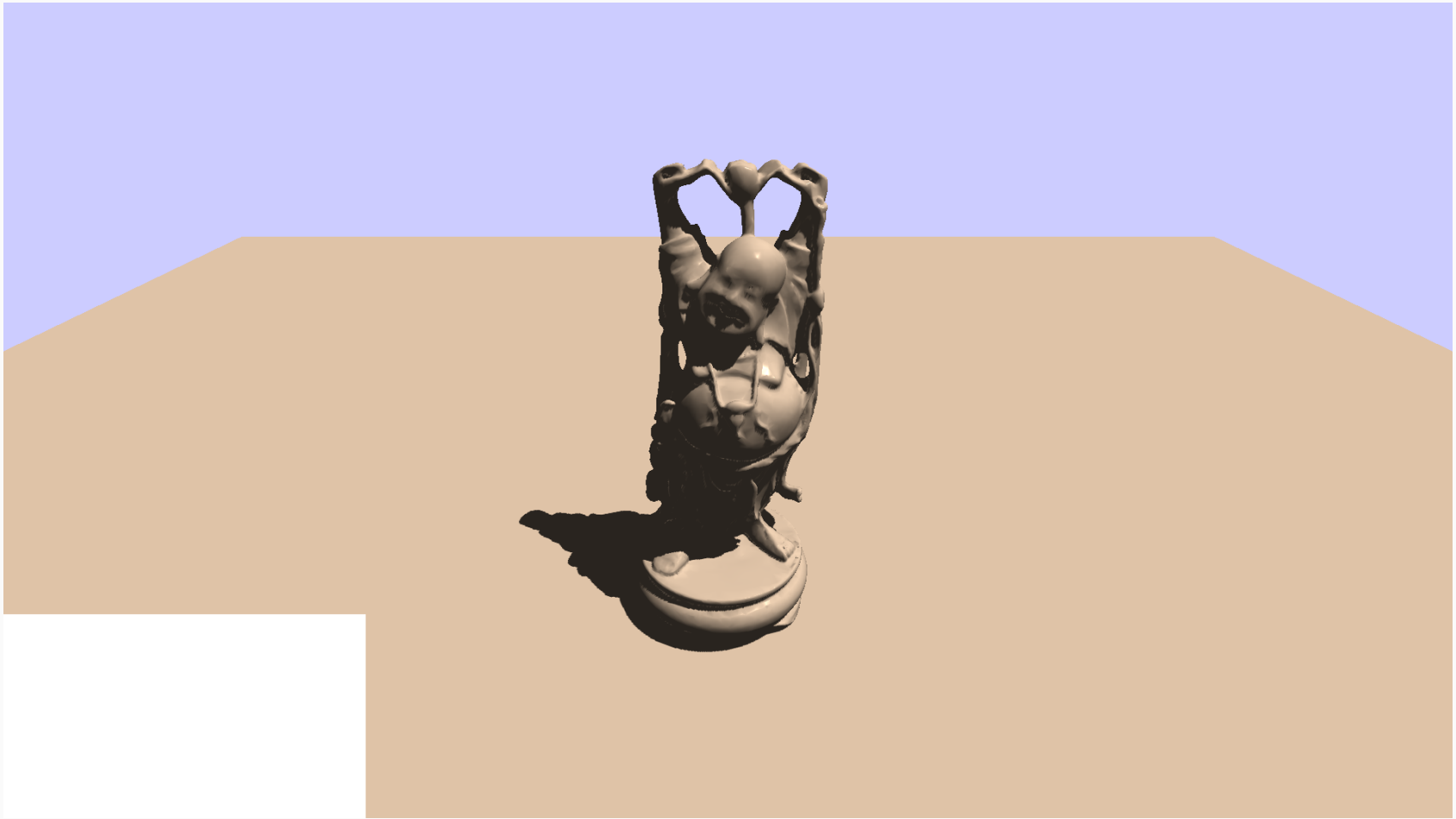


Задание 6

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` (тоже через `glTexParameteri`)
- Заменяем в основном фрагментном шейдере `sampler2D shadow_map` на `sampler2DShadow`
- Сравнение `texture(shadow_map, shadow_texcoord) < shadow_depth` заменяется на один вызов `texture(shadow_map, shadow_texcoord)` – вернёт значение от 0 до 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, shadow_texcoord)` читаем значения из соседних пикселей (надо будет что-то прибавить к `shadow_texcoord`) и усредняем по Гауссу
- Тени должны получиться более размытыми

Задание 7*