Компьютерная графика

Практика 8: Shadow mapping

2021

Задание 1

Создаём и настраиваем shadow map и framebuffer

- Выбираем размер shadow map: например, `shadow_map_res = 1024`

Задание 1

Создаём и настраиваем shadow map и framebuffer

- Выбираем размер shadow map: например, `shadow_map_res = 1024`
- Создаём текстуру для shadow map: min/mag фильтры - `GL_NEAREST`, размеры - `shadow_map_res` на `shadow_map_res`, 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`

Задание 1

Создаём и настраиваем shadow map и framebuffer

- Выбираем размер shadow map: например, `shadow_map_res = 1024`
- Создаём текстуру для shadow map: min/mag фильтры - `GL_NEAREST`, размеры - `shadow_map_res` на `shadow_map_res`, 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

Задание 1

Создаём и настраиваем shadow map и framebuffer

- Выбираем размер shadow map: например, `shadow_map_res = 1024`
- Создаём текстуру для shadow map: min/mag фильтры - `GL_NEAREST`, размеры - `shadow_map_res` на `shadow_map_res`, 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 (`GL_DEPTH_ATTACHMENT`)

Задание 1

Создаём и настраиваем shadow map и framebuffer

- Выбираем размер shadow map: например, `shadow_map_res = 1024`
- Создаём текстуру для shadow map: min/mag фильтры - `GL_NEAREST`, размеры - `shadow_map_res` на `shadow_map_res`, 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 (`GL_DEPTH_ATTACHMENT`)
- Проверяем, что фреймбуффер настроен правильно (`glCheckFramebufferStatus`)

Задание 2

Добавляем дебажную шейдерную программу, чтобы видеть содержимое нашей shadow map

- Вершинный шейдер: выдаёт захардкоженные координаты вершин, используя `gl_VertexID`, и передаёт во фрагментный шейдер текстурные координаты
  - Должно быть 6 вершин - два треугольника, образующих прямоугольник
  - Координаты вершин должны быть где-то в нижнем левом углу экрана (например, [-1.0 .. -0.75] по обеим осям)
  - Текстурные координаты должны быть [0.0 .. 1.0]

Задание 2

Добавляем дебажную шейдерную программу, чтобы видеть содержимое нашей shadow map

- Вершинный шейдер: выдаёт захардкоженные координаты вершин, используя `gl_VertexID`, и передаёт во фрагментный шейдер текстурные координаты
  - Должно быть 6 вершин - два треугольника, образующих прямоугольник
  - Координаты вершин должны быть где-то в нижнем левом углу экрана (например, [-1.0 .. -0.75] по обеим осям)
  - Текстурные координаты должны быть [0.0 .. 1.0]
- Фрагментный шейдер: читает цвет из переданной текстуры и выводит в `out_color`

Задание 2

Добавляем дебажную шейдерную программу, чтобы видеть содержимое нашей shadow map

- Вершинный шейдер: выдаёт захардкоженные координаты вершин, используя `gl_VertexID`, и передаёт во фрагментный шейдер текстурные координаты
  - Должно быть 6 вершин - два треугольника, образующих прямоугольник
  - Координаты вершин должны быть где-то в нижнем левом углу экрана (например, [-1.0 .. -0.75] по обеим осям)
  - Текстурные координаты должны быть [0.0 .. 1.0]
- Фрагментный шейдер: читает цвет из переданной текстуры и выводит в `out_color`
- Создаём фиктивный VAO (без настройки атрибутов вершин)

Задание 2

Добавляем дебажную шейдерную программу, чтобы видеть содержимое нашей shadow map

- Вершинный шейдер: выдаёт захардкоженные координаты вершин, используя `gl_VertexID`, и передаёт во фрагментный шейдер текстурные координаты
  - Должно быть 6 вершин - два треугольника, образующих прямоугольник
  - Координаты вершин должны быть где-то в нижнем левом углу экрана (например, [-1.0 .. -0.75] по обеим осям)
  - Текстурные координаты должны быть [0.0 .. 1.0]
- Фрагментный шейдер: читает цвет из переданной текстуры и выводит в `out_color`
- Создаём фиктивный VAO (без настройки атрибутов вершин)
- Рисуем с помощью `glDrawArrays(GL_TRIANGLES, 0, 6)` (не забываем сделать текущими VAO, программу и текстуру shadow map)

Задание 3

Генерируем shadow map

- Выбираем проекцию для shadow map: для начала сгодится проекция "снизу-вверх"(как будто камера смотрит сверху)

Задание 3

Генерируем shadow map

- Выбираем проекцию для shadow map: для начала сгодится проекция "снизу-вверх"(как будто камера смотрит сверху)
  - `light_Z = glm::vec3(0, -1, 0)`
  - `light_X = glm::vec3(1,  0, 0)`
  - `light_Y = glm::cross(light_X, light_Z)`

Задание 3

Генерируем shadow map

- Выбираем проекцию для shadow map: для начала сгодится проекция "снизу-вверх"(как будто камера смотрит сверху)
  - `light_Z = glm::vec3(0, -1, 0)`
  - `light_X = glm::vec3(1,  0, 0)`
  - `light_Y = glm::cross(light_X, light_Z)`
  - `light_X, light_Y, light_Z` - строки матрицы проекции (N.B. в GLM `m[i][j]` это `i`-ый **столбец** и `j`-ая **строка**)

Задание 3

Генерируем shadow map

- Выбираем проекцию для shadow map: для начала сгодится проекция "снизу-вверх"(как будто камера смотрит сверху)
  - `light_Z = glm::vec3(0, -1, 0)`
  - `light_X = glm::vec3(1,  0, 0)`
  - `light_Y = glm::cross(light_X, light_Z)`
  - `light_X, light_Y, light_Z` - строки матрицы проекции (N.B. в GLM `m[i][j]` это `i`-ый **столбец** и `j`-ая **строка**)
- Пишем шейдерную программу:

Задание 3

Генерируем shadow map

- Выбираем проекцию для shadow map: для начала сгодится проекция "снизу-вверх"(как будто камера смотрит сверху)
  - `light_Z = glm::vec3(0, -1, 0)`
  - `light_X = glm::vec3(1,  0, 0)`
  - `light_Y = glm::cross(light_X, light_Z)`
  - `light_X, light_Y, light_Z` - строки матрицы проекции (N.B. в GLM `m[i][j]` это `i`-ый **столбец** и `j`-ая **строка**)
- Пишем шейдерную программу:
  - Вершинный шейдер преобразует вершины (`gl_Position = shadow_transform * model * ...`)
  - Фрагментный шейдер ничего не делает (пустая функция `main`)

Задание 3

Генерируем shadow map

- Выбираем проекцию для shadow map: для начала сгодится проекция "снизу-вверх"(как будто камера смотрит сверху)
  - `light_Z = glm::vec3(0, -1, 0)`
  - `light_X = glm::vec3(1,  0, 0)`
  - `light_Y = glm::cross(light_X, light_Z)`
  - `light_X, light_Y, light_Z` - строки матрицы проекции (N.B. в GLM `m[i][j]` это `i`-ый **столбец** и `j`-ая **строка**)
- Пишем шейдерную программу:
  - Вершинный шейдер преобразует вершины (`gl_Position = shadow_transform * model * ...`)
  - Фрагментный шейдер ничего не делает (пустая функция `main`)
- Перед рисованием основного кадра: используем shadow framebuffer для рисования, настраиваем `glViewport`, очищаем буфер глубины, включаем front-face culling, рисуем нашу модель созданной шейдерной программой

Задание 3

Генерируем shadow map

- Выбираем проекцию для shadow map: для начала сгодится проекция "снизу-вверх"(как будто камера смотрит сверху)
  - `light_Z = glm::vec3(0, -1, 0)`
  - `light_X = glm::vec3(1,  0, 0)`
  - `light_Y = glm::cross(light_X, light_Z)`
  - `light_X, light_Y, light_Z` - строки матрицы проекции (N.B. в GLM `m[i][j]` это `i`-ый **столбец** и `j`-ая **строка**)
- Пишем шейдерную программу:
  - Вершинный шейдер преобразует вершины (`gl_Position = shadow_transform * model * ...`)
  - Фрагментный шейдер ничего не делает (пустая функция `main`)
- Перед рисованием основного кадра: используем shadow framebuffer для рисования, настраиваем `glViewport`, очищаем буфер глубины, включаем front-face culling, рисуем нашу модель созданной шейдерной программой
- Не забываем вернуть дефолтный framebuffer для рисования и вернуть `glViewport`

Задание 3

Генерируем shadow map

- Выбираем проекцию для shadow map: для начала сгодится проекция "снизу-вверх"(как будто камера смотрит сверху)
  - `light_Z = glm::vec3(0, -1, 0)`
  - `light_X = glm::vec3(1,  0, 0)`
  - `light_Y = glm::cross(light_X, light_Z)`
  - `light_X, light_Y, light_Z` - строки матрицы проекции (N.B. в GLM `m[i][j]` это `i`-ый **столбец** и `j`-ая **строка**)
- Пишем шейдерную программу:
  - Вершинный шейдер преобразует вершины (`gl_Position = shadow_transform * model * ...`)
  - Фрагментный шейдер ничего не делает (пустая функция `main`)
- Перед рисованием основного кадра: используем shadow framebuffer для рисования, настраиваем `glViewport`, очищаем буфер глубины, включаем front-face culling, рисуем нашу модель созданной шейдерной программой
- Не забываем вернуть дефолтный framebuffer для рисования и вернуть `glViewport`
- Модель должна появиться в нашем дебажном прямоугольнике

Задание 4

Используем shadow map

- Передаём текстуру shadow map и проекцию для неё в основную шейдерную программу

Задание 4

Используем shadow map

- Передаём текстуру shadow map и проекцию для неё в основную шейдерную программу
- Во фрагментном шейдере:
  - Проверяем текущий пиксель на попадание в видимую область shadow map (все координаты `shadow_pos` после всех преобразований должны быть в диапазоне [0..1])
  - Сравниваем значение из shadow map с Z-координатой `shadow_pos`
  - Если пиксель внутри видимой области и его Z-координата больше считанной из shadow map, он в тени (к нему не нужно применять прямое освещение, но ambient остаётся)

Задание 5

Вычисляем настоящую проекцию

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