МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №4 по курсу
“Компьютерная графика”

Основы 3D графики

Выполнил: К.А. Ефременко
Группа: М8О-312Б-23
Преподаватель: В. Д. Бахарев

Москва, 2025

Условие

Цель работы

Реализовать технику наложения теней (Shadow Mapping). Освоить работу с рендерингом вне кадра "от лица" направленного источника света в текстуру глубины с использованием расширения Vulkan 1.2 Dynamic Rendering. Научиться использовать данные о глубине сцены для создания эффекта тени на поверхностях моделей.

Базовое условие

Выполнены все пункты базового условия:

- Подготовлена матрица проекции для направленного источника света
- Создана текстура глубины (shadow map) для рендеринга сцены "от лица" источника освещения
- Использовано расширение Vulkan 1.2 Dynamic Rendering (`vkCmdBeginRendering`, `vkCmdEndRendering`) для рисования в текстуру глубины
- Создан объект сэмплера с поддержкой сравнения значений глубины
- Реализовано использование shadow map в fragment shader для затенения пикселей
- Интегрированы все изученные техники: освещение по модели Блинна-Фонга, текстурирование и тени **Дополнительное задание**

Выполнено дополнительное задание:

Тени от точечных источников света: Реализованы тени от двух точечных источников света с использованием кубических shadow maps (omnidirectional shadow mapping). Стратегия выбора источников для отбрасывания теней основана на дистанции до камеры и радиусе свечения.

Метод решения

Алгоритм Shadow Mapping

Shadow Mapping — это техника рендеринга теней, которая работает в два прохода:

1. **Shadow Pass (проход теней):** Рендеринг сцены с точки зрения источника света в текстуру глубины
2. **Lighting Pass (проход освещения):** Обычный рендеринг сцены с использованием shadow map для определения затенённых участков

Принцип работы

Основная идея: если фрагмент дальше от источника света, чем ближайший объект (записанный в shadow map), то он находится в тени.

// Псевдокод проверки тени float shadow_depth = texture(shadow_map, shadow_coords.xy).r; float current_depth = shadow_coords.z; if (current_depth >

shadow_depth + bias) { // Фрагмент в тени shadow_factor = 0.0; } else { // Фрагмент освещён shadow_factor = 1.0; }

Подготовка матрицы проекции для источника света

Для направленного света

Направленный свет (например, солнце) использует ортографическую проекцию:

// Расчёт матрицы вида для направленного света Vector light_position = -light.direction * distance; Vector light_target = {0, 0, 0}; Vector up = {0, 1, 0}; Matrix light_view = lookAt(light_position, light_target, up); // Ортографическая проекция float ortho_size = 10.0f; Matrix light_projection = orthographic(-ortho_size, ortho_size, // left, right -ortho_size, ortho_size, // bottom, top 0.1f, 50.0f // near, far); Matrix light_view_projection = light_projection * light_view;

Ортографическая проекция используется потому, что лучи направленного света параллельны.

Для точечного света

Точечный свет излучает во все стороны, поэтому требуется 6 проекций (кубическая карта):

// Перспективная проекция для каждой грани куба Matrix light_projection = perspective(90.0f, 1.0f, near, far); // 6 направлений: +X, -X, +Y, -Y, +Z, -Z Vector directions[6] = { {1, 0, 0}, {-1, 0, 0}, {0, 1, 0}, {0, -1, 0}, {0, 0, 1}, {0, 0, -1} }; Vector ups[6] = { {0, -1, 0}, {0, -1, 0}, {0, 0, 1}, {0, 0, -1}, {0, -1, 0}, {0, -1, 0} }; for (int i = 0; i < 6; i++) { Matrix view = lookAt(light_position, light_position + directions[i], ups[i]); light_view_projections[i] = light_projection * view; }

Создание текстуры глубины (Shadow Map)

Shadow map — это специальная текстура, которая хранит значения глубины сцены с точки зрения источника света.

Параметры создания

VkImageCreateInfo shadow_map_info{ .sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO, .imageType = VK_IMAGE_TYPE_2D, .format = VK_FORMAT_D32_SFLOAT, // 32-битная глубина .extent = {2048, 2048, 1}, // Высокое разрешение для качества .mipLevels = 1, .arrayLayers = 1, .samples = VK_SAMPLE_COUNT_1_BIT, .tiling = VK_IMAGE_TILING_OPTIMAL, .usage = VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT | VK_IMAGE_USAGE_SAMPLED_BIT, // Для чтения в шейдере .initialLayout = VK_IMAGE_LAYOUT_UNDEFINED }; **Ключевые параметры:**

- `VK_FORMAT_D32_SFLOAT` — 32-битная точность глубины для минимизации артефактов
- Разрешение 2048×2048 — компромисс между качеством и производительностью
- `SAMPLED_BIT` — возможность чтения текстуры в fragment shader

Для кубической карты (точечный свет)

VkImageCreateInfo cube_shadow_map_info{ // ... аналогично выше .flags = VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT, .arrayLayers = 6, // 6 граней куба // ... };

Vulkan 1.2 Dynamic Rendering

Dynamic Rendering — это расширение Vulkan 1.2, которое упрощает процесс рендеринга, устраняя необходимость в `VkRenderPass` и `VkFramebuffer`.

Преимущества Dynamic Rendering

- Меньше boilerplate кода
- Динамическая настройка attachments
- Более гибкая архитектура
- Проще работать с offscreen рендерингом

Shadow Pass с Dynamic Rendering

// Описание depth attachment для shadow map VkRenderingAttachmentInfo depth_attachment{ .sType = VK_STRUCTURE_TYPE_RENDERING_ATTACHMENT_INFO, .imageView = shadow_map_view, .imageLayout = VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, .loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR, .storeOp = VK_ATTACHMENT_STORE_OP_STORE, .clearValue = {.depthStencil = {1.0f, 0}} }; // Начало рендеринга VkRenderingInfo rendering_info{ .sType = VK_STRUCTURE_TYPE_RENDERING_INFO, .renderArea = {{0, 0}, {2048, 2048}}, .layerCount = 1, .pDepthAttachment = &depth_attachment, .pStencilAttachment = nullptr }; // Переход layout перед рендерингом transitionImageLayout(shadow_map, VK_IMAGE_LAYOUT_UNDEFINED, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL); vkCmdBeginRendering(command_buffer, &rendering_info); // Рендеринг сцены с точки зрения света vkCmdBindPipeline(cmd, VK_PIPELINE_BIND_POINT_GRAPHICS, shadow_pipeline); vkCmdBindDescriptorSets(cmd, ...); // ... draw calls для всех объектов vkCmdEndRendering(command_buffer); // Переход layout для чтения в shader transitionImageLayout(shadow_map, VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_OPTIMAL, VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL);

Важно: после shadow pass необходимо перевести shadow map в `SHADER_READ_ONLY_OPTIMAL` для использования в lighting pass.

Сэмплер с поддержкой сравнения глубины

Для автоматического сравнения глубины используется специальный сэмплер:

VkSamplerCreateInfo shadow_sampler_info{ .sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO, .magFilter = VK_FILTER_LINEAR, .minFilter = VK_FILTER_LINEAR, .mipmapMode = VK_SAMPLER_MIPMAP_MODE_NEAREST, .addressModeU = VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE, .addressModeV = VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE, .addressModeW = VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE, .compareEnable = VK_TRUE, // Включение сравнения .compareOp = VK_COMPARE_OP_LESS_OR_EQUAL, // Оператор сравнения

.borderColor = VK_BORDER_COLOR_FLOAT_OPAQUE_WHITE }; vkCreateSampler(device, &shadow_sampler_info, nullptr, &shadow_sampler); **Ключевые параметры:**

- `compareEnable = VK_TRUE` — включает аппаратное сравнение
- `compareOp = LESS_OR_EQUAL` — фрагмент освещён, если его глубина ≤ значению в shadow map
- `CLAMP_TO_EDGE` — предотвращает артефакты на краях
- `LINEAR` фильтрация — сглаживает края теней (аппаратный PCF)

Использование Shadow Map в Fragment Shader

Vertex Shader layout (location = 0) in vec3 v_position; layout (location = 1) in vec3 v_normal; layout (location = 2) in vec2 v_uv; layout (location = 0) out vec3 f_position; layout (location = 1) out vec3 f_normal; layout (location = 2) out vec2 f_uv; layout (location = 3) out vec4 f_shadow_coord; // Координаты в shadow map layout (binding = 0) uniform SceneUniforms { mat4 view_projection; vec3 camera_position; mat4 light_view_projection; // Матрица света }; layout (binding = 1) uniform ModelUniforms { mat4 model; // ... }; void main() { vec4 world_pos = model * vec4(v_position, 1.0); gl_Position = view_projection * world_pos; f_position = world_pos.xyz; f_normal = normalize((model * vec4(v_normal, 0.0)).xyz); f_uv = v_uv; // Преобразование в координаты света f_shadow_coord = light_view_projection * world_pos; }

Fragment Shader layout (binding = 6) uniform sampler2DShadow shadow_map; // Shadow sampler layout
(location = 3) in vec4 f_shadow_coord; float calculateShadow(vec4 shadow_coord) { // Perspective divide vec3 proj_coords = shadow_coord.xyz / shadow_coord.w; // Преобразование из NDC [-1,1] в UV [0,1] proj_coords.xy = proj_coords.xy * 0.5 + 0.5; // Проверка выхода за границы shadow map if (proj_coords.x < 0.0 || proj_coords.x > 1.0 || proj_coords.y < 0.0 || proj_coords.y > 1.0 || proj_coords.z > 1.0) { return 1.0; // Вне shadow map — освещён } // Shadow bias для предотвращения shadow acne float bias = 0.005; proj_coords.z -= bias; // Аппаратное сравнение через sampler2DShadow // Возвращает 1.0 если освещён, 0.0 если в тени float shadow = texture(shadow_map, proj_coords); return shadow; } // Интеграция с освещением void main() { // ... расчёт освещения по модели Блинна-Фонга vec3 lighting = calculateLighting(...); // Применение тени float shadow_factor = calculateShadow(f_shadow_coord); vec3 final_color = ambient + shadow_factor * lighting; out_color = vec4(final_color, 1.0); }

PCF (Percentage Closer Filtering)

PCF — это техника смягчения краёв теней путём усреднения нескольких сэмплов из shadow map.

Аппаратный PCF

При использовании `sampler2DShadow` с линейной фильтрацией, GPU автоматически выполняет 2×2 PCF (4 сэмпла).

Ручная реализация PCF 3×3

float calculateShadowPCF(vec4 shadow_coord) { vec3 proj_coords = shadow_coord.xyz / shadow_coord.w; proj_coords.xy = proj_coords.xy * 0.5 + 0.5; if (proj_coords.x < 0.0 || proj_coords.x > 1.0 || proj_coords.y < 0.0 || proj_coords.y > 1.0) { return 1.0; } float bias = 0.005; proj_coords.z -= bias; // Размер текселя vec2 texel_size = 1.0 / textureSize(shadow_map, 0); float shadow = 0.0; // 3×3 = 9 сэмплов for (int x = -1; x <= 1; x++) { for (int y = -1; y <= 1; y++) { vec2 offset = vec2(x, y) * texel_size; shadow += texture(shadow_map, proj_coords.xy + offset, proj_coords.z); } } return shadow / 9.0; // Усреднение }

PCF создаёт мягкие края теней, устраняя ступенчатость.

Борьба с артефактами

1. Shadow Acne (Угревая сыпь теней)

Проблема: самозатенение поверхности из-за недостаточной точности глубины.

Решение: Добавление bias (смещения) к глубине фрагмента:

float bias = max(0.05 * (1.0 - dot(normal, light_dir)), 0.005); proj_coords.z -= bias;

Bias зависит от угла между нормалью и направлением света — больше для скользящих углов.

2. Peter Panning (Эффект Питера Пэна)

Проблема: при слишком большом bias тени "отрываются" от объектов.

Решение: Балансировка bias — достаточный для устранения acne, но минимальный.

3. Shadow Map Resolution

Проблема: низкое разрешение создаёт блочные тени.

Решение: Использование высокого разрешения (2048×2048 или 4096×4096) и PCF фильтрации.

Интеграция всех техник

Финальный fragment shader объединяет все изученные техники:

void main() { // 1. Текстурирование vec3 albedo = texture(albedo_texture, f_uv).rgb * albedo_color; vec3 specular_map = texture(specular_texture, f_uv).rgb; vec3 emissive_map = texture(emissive_texture, f_uv).rgb; // 2. Расчёт теней float shadow = calculateShadowPCF(f_shadow_coord); // 3. Освещение по модели Блинна-Фонга vec3 result = vec3(0.0); // Ambient (всегда присутствует) result += ambient_light.color * ambient_light.intensity * albedo; // Directional light (с тенями) vec3 N = normalize(f_normal); vec3 L = normalize(-directional_light.direction); vec3 V = normalize(camera_position - f_position); vec3 H = normalize(L + V); float diff = max(dot(N, L), 0.0); float spec = pow(max(dot(N, H), 0.0), shininess); vec3 diffuse = directional_light.color * diff * albedo; vec3 specular = directional_light.color * spec * specular_map; // Применение тени только к

directional light result += shadow * directional_light.intensity * (diffuse + specular); // Point lights (без теней или с отдельными shadow maps) for (int i = 0; i < point_light_count; i++) { // ... расчёт освещения } // 4. Emissive (светящиеся участки) result += emissive_map * 2.0; out_color = vec4(result, 1.0); }

Результаты

Демонстрация работы программы

Скриншот 1

Рис. 1. Демонстрация теней от направленного источника света. На сцене видны три куба (красный, синий и фиолетовый), отбрасывающие чёткие тени на текстурированную плоскость с изображением аниме-персонажа. Панель управления показывает настройки Directional Light: направление (-0.200, 0.363, -0.300), белый цвет и интенсивность 0.800. Тени имеют мягкие края благодаря PCF фильтрации. Видна корректная работа shadow mapping: тени соответствуют позициям объектов и направлению света. Эмиссивная сетка на полу (голубые светящиеся линии) остаётся видимой даже в затенённых областях, демонстрируя правильную интеграцию всех техник (освещение, текстурирование, тени).

Скриншот 2

Рис. 2. Вид сцены с другого ракурса, демонстрирующий динамическую природу теней. Тени объектов адаптируются к новому положению камеры и продолжают корректно отображаться на поверхности. Красный куб на переднем плане отбрасывает чёткую тень, которая частично закрывает синий куб. Видно, как тени взаимодействуют с текстурой пола — затемнение применяется поверх текстурных деталей, сохраняя их видимость. Освещение сцены изменилось из-за ракурса, но тени остаются физически корректными. Хорошо видна работа shadow bias — отсутствуют артефакты самозатенения (shadow acne), и тени не "отрываются" от объектов (peter panning). Эмиссивные линии сетки продолжают светиться независимо от теней, добавляя глубину сцене.

Функциональные возможности

Реализованная программа предоставляет следующие возможности:

Система теней

- **Shadow Mapping для направленного света:** Реалистичные тени с учётом направления освещения
- **PCF фильтрация:** Мягкие края теней без ступенчатости (9 сэмплов)
- **Динамический shadow bias:** Автоматическая коррекция в зависимости от угла поверхности
- **Высокое разрешение shadow map:** 2048×2048 для детализированных теней
- **Offscreen рендеринг:** Эффективная генерация shadow map через Dynamic Rendering

Интеграция техник

- **Освещение Блинна-Фонга:** Ambient, diffuse и specular компоненты с тенями
- **Текстурирование:** Albedo, specular и emissive карты работают с тенями
- **Множественные источники света:** Directional, point и spot lights (point lights с опциональными тенями)
- **Материалы:** Разные наборы текстур для разных объектов

Vulkan 1.2 Dynamic Rendering

- **Упрощённая архитектура:** Без явных VkRenderPass и VkFramebuffer
- **Гибкость:** Динамическая настройка attachments на лету
- **Offscreen рендеринг:** Простое создание shadow pass
- **Image layout transitions:** Автоматическое управление состояниями текстур

Технические характеристики

Параметр	Значение
Vulkan версия	1.2+ (Dynamic Rendering)
Разрешение shadow map	2048 × 2048 пикселей
Формат shadow map	VK_FORMAT_D32_SFLOAT (32-бит)
PCF ядро	3×3 (9 сэмплов)
Shadow bias	Динамический, 0.005 - 0.05
Сэмплер	sampler2DShadow с compareEnable
Оператор сравнения	VK_COMPARE_OP_LESS_OR_EQUAL
Режим адресации	VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
Количество проходов	2 (shadow pass + lighting pass)
Image layouts	DEPTH_ATTACHMENT_OPTIMAL → SHADER_READ_ONLY_OPTIMAL

Архитектура рендеринга

Этап	Описание
1. Shadow Pass	Рендеринг сцены с точки зрения света в shadow map (только depth)
2. Layout Transition	Переход shadow map: DEPTH_ATTACHMENT → SHADER_READ_ONLY
3. Lighting Pass	Обычный рендеринг с чтением shadow map для расчёта теней
4. Shadow Calculation	PCF фильтрация + bias + сравнение глубины
5. Light Integration	Применение shadow factor к диффузному и зеркальному освещению

Выводы

В ходе выполнения лабораторной работы была успешно реализована техника Shadow Mapping с использованием современных возможностей Vulkan 1.2. Выполнены все пункты базового условия и дополнительное задание.

Приобретённые знания и навыки:

- **Shadow Mapping:** Освоил классическую технику рендеринга теней:
  - Понимание двухпроходной архитектуры (shadow pass + lighting pass)
  - Создание и настройка depth-only текстур для хранения карты глубины
  - Расчёт матриц проекции для источников света (ортографическая и перспективная)
  - Преобразование координат фрагмента в пространство света
- **Vulkan 1.2 Dynamic Rendering:** Изучил современный подход к рендерингу:
  - Использование `vkCmdBeginRendering` / `vkCmdEndRendering`
  - Динамическая настройка attachments без предварительного создания renderpass
  - Упрощение архитектуры offscreen рендеринга
  - Гибкость в настройке load/store операций
- **Depth-only рендеринг:** Понял специфику рендеринга только глубины:
  - Создание pipeline без color attachments
  - Оптимизация шейдеров — минимальный vertex shader, отсутствие fragment shader
  - Настройка depth test и depth write
- **Сэмплеры с сравнением:** Освоил работу с `sampler2DShadow`:
  - Настройка `compareEnable` и `compareOp`
  - Понимание аппаратного сравнения глубины
  - Автоматический PCF через линейную фильтрацию
  - Работа с результатом сравнения (0.0 или 1.0)
- **PCF (Percentage Closer Filtering):** Реализовал смягчение краёв теней:
  - Аппаратный PCF 2×2 через linear sampler
  - Ручная реализация PCF 3×3 для лучшего качества
  - Расчёт размера текселя и смещений
  - Усреднение результатов множественных сэмплов
- **Борьба с артефактами теней:** Научился решать распространённые проблемы: – **Shadow Acne:** динамический bias в зависимости от угла поверхности
  - **Peter Panning:** балансировка bias для предотвращения отрыва теней
  - **Aliasing:** высокое разрешение shadow map и PCF фильтрация
  - Обработка граничных случаев (выход за пределы shadow map)

- **Image Layout Management:** Понял важность управления состояниями текстур: – UNDEFINED → DEPTH_ATTACHMENT_OPTIMAL для записи
  - DEPTH_ATTACHMENT_OPTIMAL → SHADER_READ_ONLY_OPTIMAL для чтения
  - Pipeline barriers для синхронизации
  - Оптимальные layouts для разных операций
- **Интеграция с освещением:** Научился корректно применять тени к освещению:
  - Ambient освещение игнорирует тени (всегда присутствует)
  - Diffuse и specular компоненты умножаются на shadow factor
  - Emissive участки не затеняются
  - Раздельная обработка разных типов источников света • **Omnidirectional Shadow Mapping (дополнительное задание):** – Кубические shadow maps для точечных источников света
  - 6 проекций для покрытия всех направлений
  - Стратегия выбора источников по дистанции и радиусу
  - Оптимизация — рендеринг только ближайших источников **Технические достижения:**

- Построена полнофункциональная система shadow mapping с высоким качеством теней
- Реализован эффективный двухпроходный рендеринг с минимальными overhead
- Достигнуты мягкие края теней через PCF без существенного падения производительности
- Устранены все основные артефакты (acne, peter panning, aliasing)
- Успешная интеграция с предыдущими техниками (освещение, текстурирование)
- Использованы современные возможности Vulkan 1.2 для упрощения кода

Понимание концепций:

Работа дала глубокое понимание того, как рендерятся тени в реальном времени. Ключевой инсайт — тени это не отдельный эффект, а результат проверки видимости с точки зрения источника света. Shadow map — это "взгляд света на сцену", и мы просто сравниваем, что видно свету с тем, что видим мы.

Особенно ценным оказалось понимание компромиссов в shadow mapping: разрешение shadow map напрямую влияет на качество, но требует памяти и производительности. PCF улучшает визуал, но увеличивает количество texture fetches. Bias решает одну проблему, но может создать другую. Эти знания критичны для принятия архитектурных решений в реальных проектах.

Dynamic Rendering показал себя как значительное улучшение API Vulkan. Код стал проще и понятнее, особенно для offscreen рендеринга. Отсутствие необходимости предварительно создавать renderpass и framebuffer существенно упрощает архитектуру.

Практическое применение:

Shadow Mapping — это индустриальный стандарт для теней в реальном времени. Эта техника используется в играх, симуляторах, системах визуализации архитектуры. Понимание shadow mapping открывает путь к более продвинутым техникам: Cascaded Shadow Maps (для больших открытых пространств), Variance Shadow Maps (для мягких теней), Ray-Traced Shadows (для фотореализма).

Опыт работы с Dynamic Rendering будет полезен в будущем, так как это направление развития Vulkan API. Многие новые расширения (например, VK_KHR_dynamic_rendering_local_read) строятся поверх этой концепции.