

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

**Лабораторная работа №X по
курсу “Компьютерная графика”**

Основы 3D графики

Выполнил: К.А. Ефременко

Группа: М8О-312Б-23

Преподаватель: В. Д. Бахарев

Москва, 2025

Условие

Цель работы

Познакомиться с основами 3D-графики: построением простых 3D-объектов, проекцией на 2D-плоскость, а также научиться работать с матрицами перспективы, ортографической проекции и аффинными преобразованиями с использованием Vulkan API.

Вариант задания №4: Цилиндр с ортографической проекцией

Требуется сгенерировать цилиндр (~50 вершин, без крышек) и отобразить его с использованием ортографической проекции. Цилиндр должен перемещаться по круговой траектории. Необходимо реализовать следующие функции:

- Генерация геометрии цилиндра программным способом
- Задание цвета через uniform/push constant
- Использование ортографической проекции
- Анимация перемещения цилиндра по круговой траектории
- UI элементы для изменения радиуса траектории

Дополнительные задания

Для повышения оценки были выполнены два дополнительных задания:

1. **Интерполяция цвета между вершинами:** Модифицирован вершинный буфер с добавлением атрибута цвета (RGB) для каждой вершины. Цвет передается из vertex-шейдера во fragment-шейдер для интерполяции, что создает градиентный эффект на поверхности цилиндра.
2. **Анимация с паузой и реверсом:** Добавлено управление анимацией через UI элементы для паузы/возобновления и реверса направления движения. Реализовано через переменную времени и модификатор направления в модельной матрице.

Метод решения

Математические основы

1. Генерация геометрии цилиндра

Цилиндр генерируется с использованием параметрических уравнений окружности. Для создания цилиндра без крышек используется 16 сегментов (32 вершины: 16 на верхнем кольце и 16 на нижнем):

$$x = r \times \cos(\theta) \quad z = r \times \sin(\theta) \quad y = \pm h/2 \quad \text{где } \theta = 2\pi \times i / n, i = 0..n-1, n = 16 \quad (\text{количество сегментов}) \\ r = 1.0 \quad (\text{радиус цилиндра}) \quad h = 2.0 \quad (\text{высота цилиндра})$$

Каждый сегмент цилиндра формируется двумя треугольниками, соединяющими соответствующие вершины верхнего и нижнего колец. Общее количество индексов: 16

$\times 6 = 96$.

2. Ортографическая проекция

Для отображения 3D-объектов используется ортографическая проекция, которая сохраняет параллельность линий и не искажает размеры объектов в зависимости от расстояния. Матрица ортографической проекции имеет вид:

$P =$

$$\begin{bmatrix} 2/(r-l) & 0 & 0 & -(r+l)/(r-l) \\ 0 & 2/(t-b) & 0 & -(t+b)/(t-b) \\ 0 & 0 & 1/(f-n) & -n/(f-n) \end{bmatrix}$$

$[0 \ 0 \ 0 \ 1]$ где l, r, b, t, n, f — границы viewing frustum (left, right, bottom, top, near, far).

3. Аффинные преобразования

Для позиционирования и анимации объекта используются следующие трансформации:

- **Перенос (translation):** Матрица переноса на вектор (x, y, z)
- **Вращение (rotation):** Матрица вращения вокруг произвольной оси на угол α , использующая формулу Родрига
- **Композиция трансформаций:** Комбинирование матриц путем их последовательного перемножения

4. Круговая траектория движения

Позиция цилиндра на круговой траектории вычисляется по формулам:

$x(t) = R \times \cos(t \times v \times d)$ $y(t) = 0$ $z(t) = R \times \sin(t \times v \times d) + \text{offset}$ где R — радиус траектории v — скорость анимации d — направление (± 1) t — время offset — смещение по оси Z

Используемые объекты Vulkan

Объект Vulkan	Назначение
VkBuffer	Хранение вершинных данных (vertex buffer) и индексов (index buffer)
VkDeviceMemory	Выделение памяти на GPU для буферов
VkShaderModule	Загрузка скомпилированных SPIR-V шейдеров (vertex и fragment)
VkPipeline	Графический конвейер, определяющий состояние рендеринга
VkPipelineLayout	Описание ресурсов, доступных шейдерам (push constants)
VkCommandBuffer	Запись команд рендеринга (draw calls)
VkRenderPass	Определение последовательности рендеринга и attachments
VkFramebuffer	Целевой буфер для рендеринга (связан со swapchain)

Архитектура решения

Структура данных вершины

Каждая вершина содержит два атрибута:

```
struct Vertex { Vector position; // Позиция в 3D пространстве (x, y, z) Vector color; // Цвет вершины (r, g, b) };
```

Push Constants

Для передачи данных в шейдеры используется механизм push constants, который позволяет эффективно обновлять небольшие объемы данных каждый кадр:

```
struct ShaderConstants { Matrix projection; // Матрица проекции (ортографическая) Matrix transform; // Модельная матрица (комбинация трансформаций) Vector base_color; // Базовый цвет (не используется при интерполяции) };
```

Конвейер рендеринга

- Инициализация:** Создание буферов, загрузка шейдеров, настройка pipeline
- Обновление (update):** Вычисление позиции объекта, обработка UI элементов
- Рендеринг (render):** Обновление констант, запись команд отрисовки

Реализация интерполяции цвета

Для создания градиентного эффекта вершинам цилиндра присваиваются разные цвета. Нижние вершины окрашены в синий цвет, верхние — в красный, с плавным переходом между ними:

```
// Вычисление цвета для каждой вершины float colorFactor = static_cast<float>(i) / segments; bottomVertex.color = mix(blueColor, redColor, colorFactor); topVertex.color = mix(redColor, blueColor, colorFactor);
```

В vertex shader цвет передается во fragment shader, где происходит автоматическая интерполяция между вершинами треугольника:

```
// Vertex Shader layout(location = 1) in vec3 in_color; layout(location = 0) out vec3 frag_color; void main() { gl_Position = constants.projection * constants.transform * vec4(in_position, 1.0); frag_color = in_color; }
```

```
// Fragment Shader layout(location = 0) in vec3 frag_color; layout(location = 0) out vec4 out_color; void main() { out_color = vec4(frag_color, 1.0); }
```

Результаты

Демонстрация работы программы

Скриншот 1

Рис. 1. Цилиндр с градиентной раскраской в начальной позиции траектории. Видна панель управления с элементами UI для настройки параметров анимации, включая

скорость вращения, радиус траектории и угол наклона цилиндра. Состояние анимации: PLAYING (FORWARD), время 45.36 секунд.

Скриншот 2

Рис. 2. Цилиндр в процессе движения по круговой траектории (время 14.75 секунд).

Демонстрируется работа интерполяции цвета между вершинами — плавный градиент от красного к фиолетовому и синему. Увеличен радиус траектории до 3.0, что видно по измененной позиции объекта (-1.72, 0.00, 7.45).

Функциональные возможности

Реализованная программа предоставляет следующие возможности управления:

- **Translation:** Ручное позиционирование цилиндра в 3D пространстве по осям X, Y, Z
- **Rotation:** Управление углом вращения цилиндра вокруг оси Y (от 0 до 2π)
- **Spin:** Включение/выключение автоматического вращения
- **Color:** Выбор базового цвета (не влияет на градиент при интерполяции)
- **Animate Motion:** Включение/выключение движения по траектории
- **Animation Speed:** Регулировка скорости анимации (0.1 - 5.0)
- **Trajectory Radius:** Изменение радиуса круговой траектории (1.0 - 10.0)
- **Pause/Resume:** Приостановка и возобновление анимации
- **Forward/Reverse:** Изменение направления движения
- **Cylinder Tilt:** Наклон цилиндра относительно оси X (-90° до 90°)

Технические характеристики реализации

Параметр	Значение
Количество сегментов цилиндра	16
Количество вершин	32 (16×2 кольца)
Количество индексов	96 (16 сегментов \times 6 индексов)
Радиус цилиндра	1.0
Высота цилиндра	2.0
Тип проекции	Ортографическая
Размер viewing frustum	$5.0 \times \text{aspect_ratio}$
Near/Far planes	0.01 / 100.0

Выводы

В ходе выполнения лабораторной работы были успешно освоены основы 3D-графики с использованием современного графического API Vulkan. Реализован вариант задания №4 с цилиндром и ортографической проекцией, а также выполнены два дополнительных задания.

Приобретенные знания и навыки:

- Научился генерировать 3D геометрию программно с использованием параметрических уравнений
- Освоил работу с ортографической проекцией и понял отличия от перспективной
- Изучил применение аффинных преобразований (перенос, вращение) и композицию матриц
- Получил практический опыт работы с Vulkan API: создание буферов, шейдеров, pipeline
- Реализовал vertex и fragment шейдеры на языке GLSL с поддержкой интерполяции
- Научился работать с вершинными атрибутами для передачи цветовой информации
- Освоил механизм push constants для эффективной передачи данных в шейдеры
- Реализовал интерактивное управление через ImGui для параметров анимации
- Разработал систему управления анимацией с поддержкой паузы и реверса

Технические достижения:

- Эффективная организация кода с разделением на функции инициализации, обновления и рендера
- Корректная работа с памятью GPU: выделение, связывание, маппинг и освобождение
- Оптимальное использование индексных буферов для минимизации дублирования вершин
- Плавная анимация с возможностью динамического изменения параметров в реальном времени
- Визуально привлекательный градиентный эффект благодаря интерполяции цвета

Полученные знания являются фундаментальными для дальнейшего изучения компьютерной графики и могут быть применены для создания более сложных 3D приложений, игр и систем визуализации. Опыт работы с низкоуровневым API Vulkan дал глубокое понимание работы графического конвейера и принципов оптимизации рендера.