МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Лабораторная работа №1
по курсу “Компьютерная графика”

Основы 3D графики

Выполнил: В. Н. Шиширин
Группа: М8О-310Б-23
Преподаватель: В. Д. Бахарев

Москва, 2025

Условие

В этой лабораторной работе вы познакомитесь с основами 3D-графики: построением простых 3D-объектов, проекцией на 2D-плоскость, а также научитесь работать с матрицами перспективы, ортографической проекции и аффинными преобразованиями.

Вариант задания: Сфера с пульсирующим масштабом. Сгенерировать сферу (~100 вершин, параметрические уравнения). Цвет фиксированный. Перспективная проекция. Анимация: сфера масштабируется по синусоиде (sin(time)). Добавить элементы UI для поворота камеры. Интерполяция цвета между вершинами. Модифицировать вершинный буфер, добавив к каждой вершине атрибут цвета (RGB). В vertex-шейдере передать цвет во fragment-шейдер для интерполяции.

Например, для куба можно задать разные цвета углам, чтобы получить градиентный эффект.

Метод решения

Краткое описание задачи

Вариант: сфера с пульсирующим масштабом. Требуется сгенерировать сетку сферы (параметрическая формула), назначить каждой вершине цвет (пер-вершинный RGB), реализовать перспективную проекцию, анимацию масштаба по синусоиде и дать управление положением/ориентацией камеры через UI.

1. Построение сетки сферы:
   Сфера строится параметрически по двум параметрам: меридианам (stacks) и параллелям (slices). Для каждой пары параметров вычисляются координаты вершины по формулам:
   $x = r * \sin(\text{phi}) * \cos(\text{theta})$
   $y = r * \cos(\text{phi})$
   $z = r * \sin(\text{phi}) * \sin(\text{theta})$
   где $\text{phi} \in [0, \pi]$ (stacks), $\text{theta} \in [0, 2\pi]$ (slices), r — радиус. В реализации (файл main.cpp) используется вложенный цикл по стэкам и слайсам: каждая ячейка сетки разбивается на два треугольника (6 индексов), поэтому применяется индексный буфер для экономии памяти и повторного использования вершин.
2. Интерполяция цвета:
   Каждой вершине присваивается атрибут цвета color (RGB). Вершинный буфер содержит структуру Vertex { position, color }. Цвет вычисляется

как простая функция от координат (например, r = (x + 1) * 0.5, g = (y + 1) * 0.5, b = (z + 1) * 0.5). Вершинный шейдер передаёт цвет во фрагментный шейдер; аппаратная интерполяция выдаёт плавный градиент по поверхности.

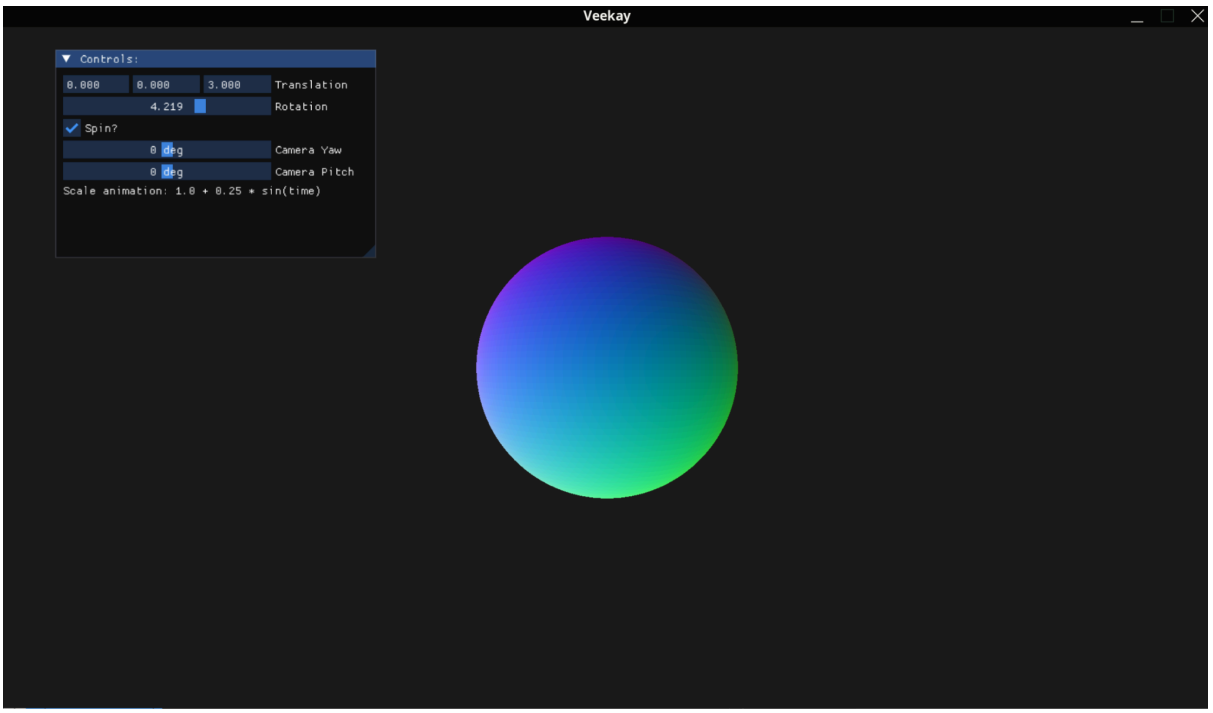
3. Модель — Вид — Проекция (Model / View / Projection):
   - Модельная матрица M_model строится как последовательность аффинных преобразований: масштабирование S(s), поворот R(axis, angle) и трансляция T(t). В коде матрицы реализованы явными функциями scaleMatrix(s), rotation(axis, angle) и translation(t), а итог вычисляется как:
     modelMat = R * (S * T) (в коде — композиция матриц через функцию multiply).
   - Анимация масштаба: масштаб анимируется функцией времени:
     s(t) = 1.0 + 0.25 * sin(t)
     Значение s(t) подставляется в модельную матрицу перед отрисовкой.
   - Видовая матрица view реализована как два поворота по осям (yaw и pitch):
     viewMat = R_pitch * R_yaw
     Параметры yaw и pitch меняются через элементы UI (ImGui).
   - Проекционная матрица projection(fov, aspect, near, far) реализована в коде функцией projection() и передаётся в шейдер вместе с transform = viewMat * modelMat.
4. Передача данных в шейдер:
   Для передачи малых по размеру данных (матрицы проекции, итоговый трансформ, цвет) используется механизм push constants. Структура ShaderConstants содержит поля projection (матрица), transform (матрица) и color (вектор). В коде данные отправляются вызовом:
   vkCmdPushConstants(cmd, pipeline_layout, stages, 0, sizeof(ShaderConstants), &constants)
5. Взаимодействие с Vulkan (используемые объекты)
   Перечень ключевых Vulkan-объектов, используемых в реализации (main.cpp):
   - VkBuffer + VkDeviceMemory: vertex buffer (VK_BUFFER_USAGE_VERTEX_BUFFER_BIT) и index buffer (VK_BUFFER_USAGE_INDEX_BUFFER_BIT). Буферы создаются и заполняются через функцию createBuffer(...), память выделяется с

флагами HOST_VISIBLE | HOST_COHERENT для простой загрузки с CPU.
- VkShaderModule: загружаются shader.vert.spv и shader.frag.spv (функция loadShaderModule()).
- VkPipelineLayout: содержит диапазон push constants (VkPushConstantRange) для передачи ShaderConstants.
- VkPipeline (Graphics pipeline): настраивается с vertex input (stride = sizeof(Vertex), атрибуты position и color), input assembly (triangle list), rasterization (cull back), depth/stencil (depth test & write enabled), viewport/scissor и связью с render pass (veekay::app.vk_render_pass).
- Командные буферы и команды отрисовки: vkCmdBindPipeline, vkCmdBindVertexBuffers, vkCmdBindIndexBuffer, vkCmdPushConstants, vkCmdDrawIndexed.
- Render pass и framebuffer: используются из фреймворка veekay (контекст рендеринга и управление swapchain берёт на себя обвязка приложения).

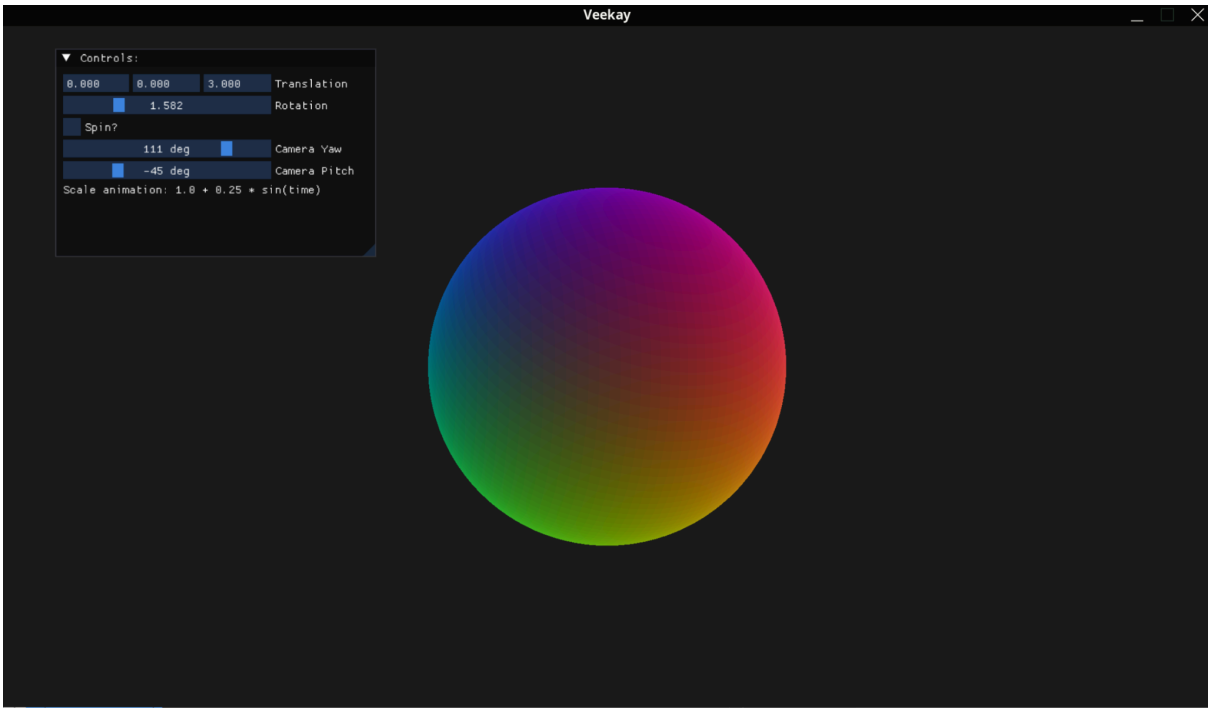
6. Алгоритм выполнения кадра:
   - При инициализации генерируется сетка вершин и индексов сферы, создаются vertex/index buffers.
   - Каждого кадра: обновляются параметры анимации (model_scale = 1.0 + 0.25 * sin(time); model_rotation при включённой анимации), формируются матрицы model, view и projection.
   - Формируется структура ShaderConstants и отправляется в шейдер через push constants.
   - Выполняется привязка pipeline, буферов и вызывается vkCmdDrawIndexed для отрисовки всей сетки.

Результаты

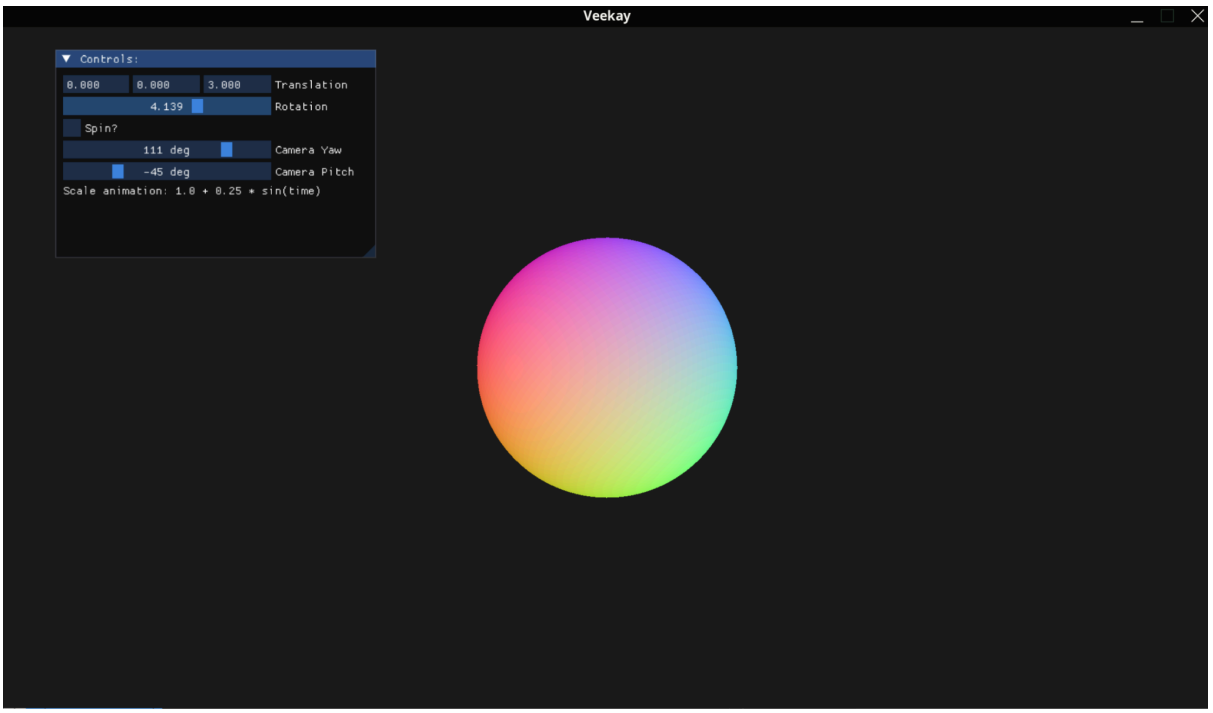
Сцена в исходной конфигурации: сгенерированная параметрически сфера (поля stacks/slices) с пер-вершинной окраской, слева видна панель Control для управления трансформациями и камерой. Масштаб примерно 1.0, видны плавные градиенты цвета по поверхности.



Сфера в пиковой фазе пульсации (максимальный масштаб).



Сцена с повернутой камерой — проверка view-матрицы и глубинного теста.



Выводы

Я научился генерировать параметрическую сетку сферы и строить для неё индексный буфер, задавать пер‑вершинные атрибуты цвета и получать их аппаратную интерполяцию в шейдерах; формировать и комбинировать модельную, видовую и перспективную матрицы, а также реализовывать анимированную шкалу модели $s(t)=1.0+0.25\cdot\sin(t)$. Практически освоил создание и загрузку vertex/index буферов в Vulkan, настройку графического пайплайна и передачу небольших структур в шейдеры через push‑constants; научился управлять камерой через ImGui и проверять корректность глубинного теста и видовых преобразований. В результате получено работающее приложение, демонстрирующее базовые принципы 3D‑рендеринга и готовое для включения в отчёт.