МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №3**

**по курсу “Компьютерная графика”**

***Текстурирование***

Выполнил: Д.М. Ибрагимов

Группа: М8О-308Б-23

Преподаватель: В. Д. Бахарев

Москва, 2025

**Цель лабораторной работы**

В этой лабораторной работе вам предстоит загрузить изображение из файла, создать текстуру из изображения и, используя текстурные координаты, а также сэмплеры, наложить текстуру на объекты

**Требования**

Вы должны использовать Vulkan, GLFW и ImGUI. Стартовый код уже всё это содержит, но если хотите попробовать без него, то я похвалю за отвагу!  
P.S. Если хотите программировать на Direct3D под Windows, то используйте именно Direct3D 12, потому что он схож c Vulkan по принципам

**Базовое условие на 3**

- Загрузить изображение из файла (желательно, чтобы ширина и высота изображения имели значение степени двойки, сами знаете зачем)  
- Создать текстуру из изображения (использовать veekay::graphics::Texture)  
- Создать объект сэмплера и задать разумные параметры сэмплирования  
- Записать в набор дескрипторов новую привязку (дескриптор – изображение+сэмплер), чтобы его увидел фрагментный шейдер  
- Вершины объектов должны содержать текстурные координаты  
- Фрагментный шейдер должен сэмплировать текстуру, используя текстурные координаты, которые были переданы из вершинного шейдера

**Базовое условие на 4**

Условие то же, что и у базового условия на оценку 3, но расширяется условием реализовать использование разных текстур (VkImageView) и сэмплеров (VkSampler) моделями на сцене (с помощью использования разных наборов дескрипторов для каждой текстуры или набора текстур, то есть “материала”)

**Дополнительные задания**

1. Сделайте нетривиальное сэмплирование текстуры в шейдере (можно модулировать входящие текстурные координаты какой-либо сложной функцией в шейдере; также можно сэмплировать много раз и смешивать цвета множества текселей по нетривиальной схеме)
2. Реализуйте две (или больше) дополнительные текстуры, описывающие материал, например specular текстуру (для определенных участков, где будут видны блики) и emissive текстуру (для участков, которые будут игнорировать просчет освещения и затенения, они будут светиться в темноте)

**Метод решения**

### Общая идея текстурирования

Текстурирование в графике реального времени основано на сопоставлении точкам поверхности объекта координат на двумерном изображении. Каждая вершина модели, помимо позиции, содержит **текстурные координаты** , как правило в диапазоне . При растеризации треугольника значения и **интерполируются** между вершинами, и для каждого фрагмента формируется собственная пара координат, по которой выполняется выборка из текстуры.

Если три вершины треугольника имеют , то для произвольного фрагмента внутри треугольника итоговые координаты могут быть представлены через барицентрические коэффициенты:

Это обеспечивает непрерывное и корректное наложение изображения на поверхность.

### Сэмплирование и роль VkSampler

Выборка значения цвета из текстуры осуществляется функцией вида:

На практике значение зависит не только от , но и от настроек **сэмплера**: режима адресации (поведение при выходе за ), фильтрации и (при наличии) мип-уровней.

В реализации создан единый сэмплер для текстур сцены с параметрами:

* фильтрация **линейная** (для увеличения и уменьшения),
* адресация **REPEAT** по U/V/W (текстура повторяется при выходе координат за ).

Линейная фильтрация соответствует билинейной интерполяции: цвет вычисляется как взвешенная сумма соседних текселей, что уменьшает «лесенки» и мерцание на наклонных поверхностях.

### Загрузка изображения и создание текстуры на GPU

Изображение загружается из файла (в проекте используется загрузчик PNG), после чего на его основе создаётся объект veekay::graphics::Texture. Такой объект, как правило, инкапсулирует:

* VkImage (хранение данных текстуры на GPU),
* VkDeviceMemory (память),
* VkImageView (представление изображения для шейдера).

Для устойчивости приложения в коде предусмотрен **fallback**: если текстуру загрузить не удалось, создаётся процедурная «шахматная» текстура (вплоть до маленькой 2×2), которая позволяет корректно продолжать рендеринг и облегчает диагностику ошибок ресурсов.

### Передача текстур в шейдеры через дескрипторы

Доступ шейдера к ресурсам в Vulkan организуется через **наборы дескрипторов**. В реализации используется один VkDescriptorSet, включающий три привязки (bindings):

1. binding = 0 — uniform-буфер со сценовыми данными (матрицы и т.п.)
2. binding = 1 — **динамический** uniform-буфер на модель (параметры конкретного объекта; выбирается через dynamic offset)
3. binding = 2 — массив combined image sampler из нескольких текстур (в проекте предусмотрено до 8)

Таким образом, каждый объект сцены получает **индекс текстуры** texture\_index, и фрагментный шейдер выбирает нужный элемент массива textures[texture\_index]. Этот подход позволяет рисовать несколько объектов с разными текстурами в рамках одного пайплайна, не «перевязывая» дескриптор во время выполнения команд на GPU.

### Реализованный эффект: сдвиг RGB-каналов (анаглиф)

В данной работе реализовано нетривиальное сэмплирование текстуры, создающее эффект «анаглифа» (визуально напоминает 3D-картинку без очков). Суть эффекта состоит в том, что **красный, зелёный и синий каналы** берутся из **разных выборок** текстуры, выполненных с небольшими смещениями по координатам.

Пусть базовые координаты выборки — . Тогда формируются три смещения:

где — величина «диспаратности» (сдвига). В реализации зависит от положения пикселя на экране: используется -координата в NDC-пространстве

после чего диспаратность вычисляется как

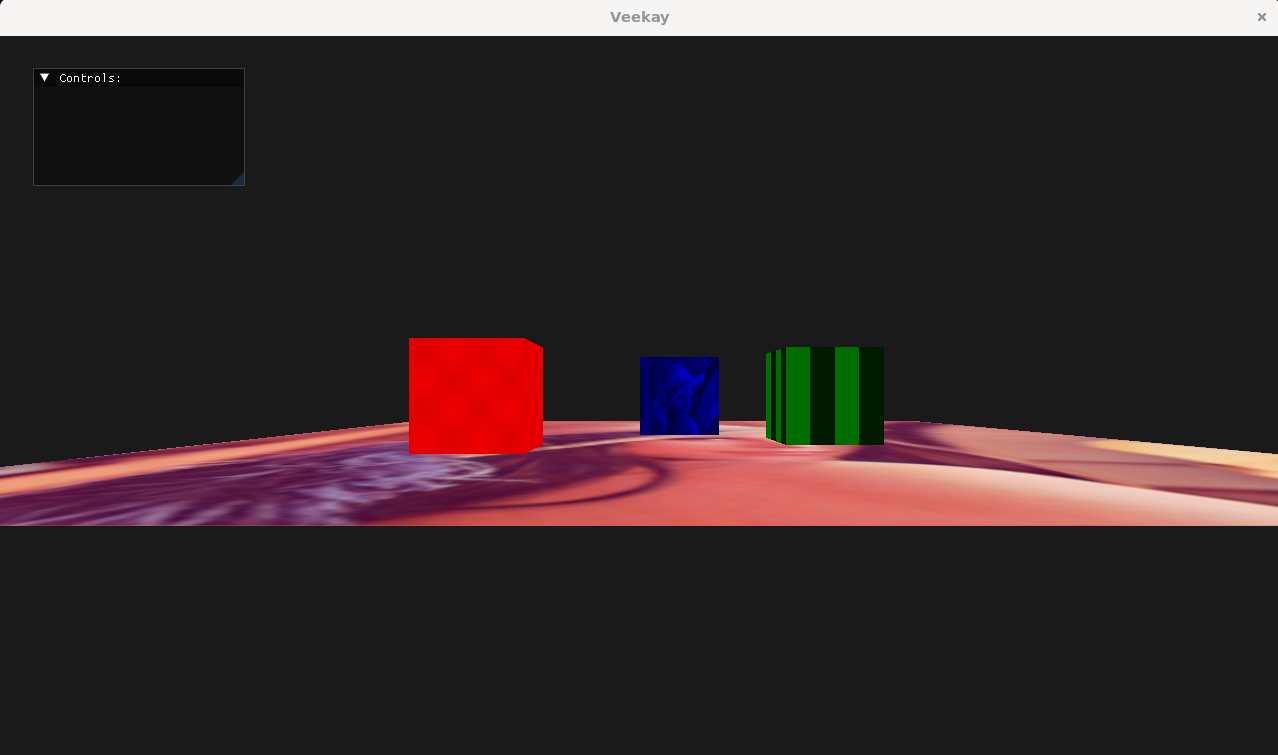
где — коэффициент силы эффекта.

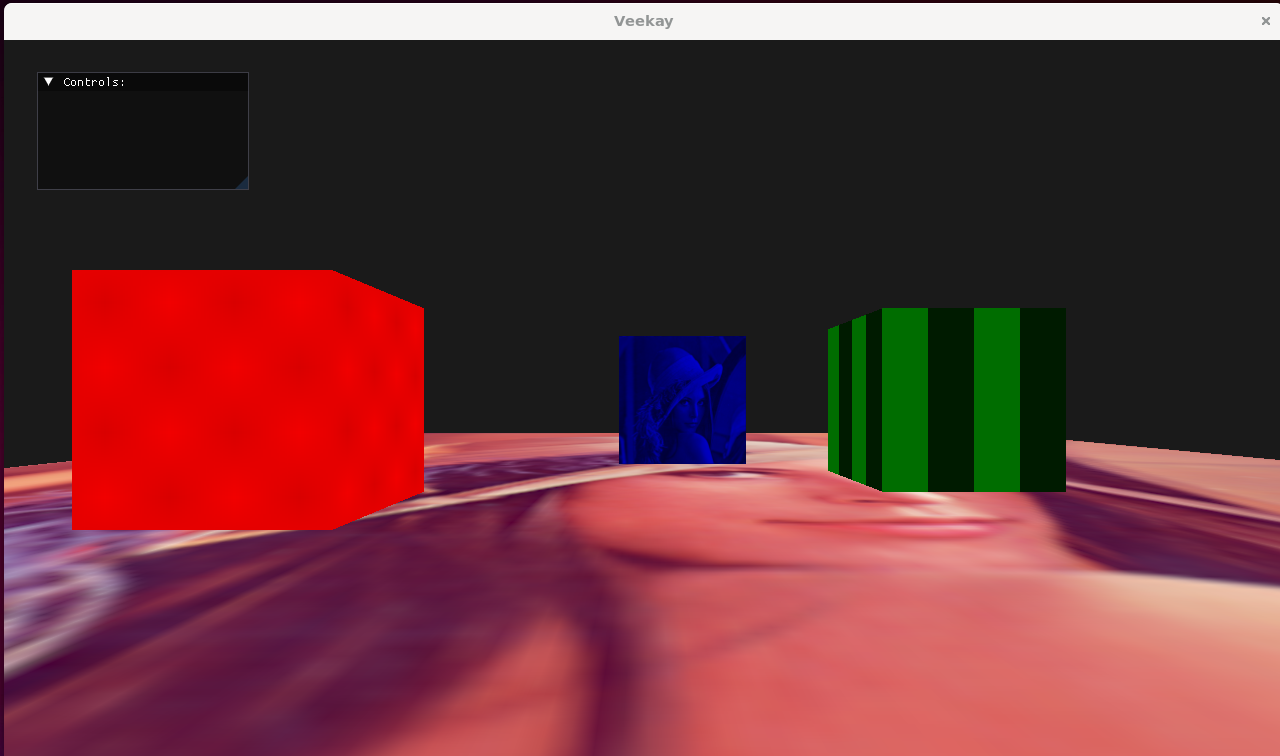
Далее выполняются три выборки:

Итоговый цвет собирается поканально:

Практически это даёт характерный «цветовой раздрай» по краям объектов, причём в центре экрана эффект слабее, а к краям — сильнее (из-за зависимости от ). Такой метод соответствует идее дополнительного задания про нетривиальное сэмплирование: текстура читается несколько раз, а результат смешивается нетривиальным образом.

**Результаты**





**Выводы**

В ходе выполнения лабораторной работы были изучены и реализованы базовые принципы текстурирования в графике реального времени: загрузка изображения из файла, создание текстурного ресурса на стороне GPU и передача его в шейдеры через механизм дескрипторов Vulkan. Были использованы текстурные координаты вершин и выполнено сэмплирование текстуры во фрагментном шейдере, что обеспечило корректное наложение изображения на объекты сцены.

Дополнительно был реализован нетривиальный шейдерный эффект, основанный на многократной выборке текстуры и раздельной сборке компонент RGB. Применение смещений при сэмплировании позволило получить визуальный эффект «анаглифа» (сдвиг цветовых каналов), зависящий от положения фрагмента на экране, что углубило понимание работы с UV-координатами и особенностей выборки из текстур.

В результате был получен практический опыт работы с текстурными ресурсами Vulkan, объектами VkSampler, VkImageView, а также настройкой дескрипторных привязок и взаимодействием CPU и GPU при обновлении данных сцены. Выполненная работа способствовала укреплению знаний о структуре графического конвейера и принципах применения текстурирования в современных API.