**Міністерство освіти і науки України Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» Факультет інформатики та обчислювальної техніки Кафедра обчислювальної техніки**

# Лабораторна робота №5

з дисципліни

«Комп’ютерна графіка та мультимедіа»

Виконав:

студент групи ІП-05

Гапій Денис Едуардович

номер залікової : 0504

Перевірив:

Родіонов П. Ю.

Київ 2022

# Тема: «Робота з Uniform Buffers у середовищі API Vulkan»

**Мета**: навчитися створювати та редагувати Uniform Buffers у середовищі API Vulkan.

**Контрольні запитання:**

1. Поняття макету дескриптору та буферу.

Якщо власними словами, то дескриптор - це зручне рішення для отримання доступу до вершиного шейдеру за допомогою матрице-подібної структури даних з інформацією про буфер чи зображення. Використання дескрипторів складається з трьох частин:

- Укажіть макет дескриптора під час створення конвеєра
- Виділити набір дескрипторів із пулу дескрипторів
- Прив’яжіть набір дескрипторів під час візуалізації

2. Характеристика та призначення набору дескрипторів.

Макет дескриптора вказує типи ресурсів, до яких конвеєр матиме доступ, подібно до того, як перехід візуалізації вказує типи вкладень, до яких буде доступ. Набір дескрипторів визначає фактичний буфер або ресурси зображення, які будуть прив’язані до дескрипторів, подібно до того, як буфер кадрів визначає фактичні перегляди зображень, які потрібно прив’язати до вкладень пропуску рендерингу. Тоді набір дескрипторів прив’язується до команд малювання так само, як буфери вершин і буфер кадрів.

3. Поняття уніформного буферу.

Буферний об’єкт , який використовується для зберігання уніфікованих даних (тобто таких які не змінюються від одного виклику шейдера до наступного в рамках певного виклику візуалізації, тому їх значення однакове для всіх викликів) для шейдерної програми , називається уніфікованим буферним об’єктом. Їх можна використовувати для спільного використання уніформ між різними програмами, а також для швидкого перемикання між наборами

уніформ для одного програмного об’єкта. Перемикання між уніфікованими прив’язками буферів зазвичай відбувається швидше, ніж перемикання десятків уніформ у програмі. Таким чином, уніфіковані буфери можна використовувати для швидкого переходу між різними наборами уніфікованих даних для різних об’єктів, які спільно використовують ту саму програму.
Крім того, уніфіковані буферні об’єкти зазвичай можуть зберігати більше даних, ніж небуферизовані уніформи. Тому їх можна використовувати для зберігання та доступу до більших блоків даних, ніж небуферизовані уніфіковані значення. Нарешті, їх можна використовувати для обміну інформацією між різними програмами. Таким чином, зміна одного буфера може ефективно дозволити оновлювати уніформи в кількох програмах.

4. Оновлення уніформних даних.
Спочатку виконуються операції, описані в *pDescriptorWrites*, а потім операції, описані в *pDescriptorCopies*. У кожному масиві операції виконуються в тому порядку, в якому вони з’являються в масиві. Кожен елемент у масиві *pDescriptorWrites* описує операцію оновлення набору дескрипторів за допомогою дескрипторів для ресурсів, указаних у структурі.
Кожен елемент у масиві *pDescriptorCopies* є структурою *VkCopyDescriptorSet*, що описує операцію копіювання дескрипторів між наборами. Якщо член *dstSet* будь-якого елемента *pDescriptorWrites* або *pDescriptorCopies* прив’язаний, доступний або змінений будь-якою командою, яка була записана до буфера команд, який наразі перебуває у стані запису або виконуваного стану , і будь-які зв’язки дескрипторів, які оновлюються, не були створені за допомогою *VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT* або *VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT* біт встановлено, буфер команд стає недійсним .
У нашому випадку є функція updateUniformBuffer, що генеруватиме нове перетворення кожного кадру, щоб геометрія оберталася. Тепер ми визначимо трансформації моделі, вигляду та проекції в єдиному буферному об’єкті.

Обертання моделі буде простим обертанням навколо осі Z з використанням змінної часу.

5. Вимоги до вирівнювання графічних об’єктів.

Вкладена структура має бути вирівняна за базовим вирівнюванням її елементів, округлене до числа, кратного 16. Матриця mat4 повинна мати таке ж вирівнювання, що й матриця vec4. Можна знайти повний перелік вимог до вирівнювання в специфікації. Оригінальний шейдер лише з трьома полями mat4 уже відповідає вимогам вирівнювання. Оскільки кожен mat4 має розмір 4 x 4 x 4 = 64 байти, модель має зміщення 0, view має зміщення 64, а proj має зміщення 128. Усе це кратно 16, тому це працювало ефективно.

Всі наявні вимоги до вирівнювання:

- Існують різні вимоги до узгодження залежно від конкретних ресурсів і функцій, увімкнених на пристрої.
- Типи матриць визначаються в термінах масивів наступним чином:
- Матриця з головним стовпцем із C стовпцями та R рядками еквівалентна C елементному масиву векторів із R компонентами.
- Матриця з основними рядками з C стовпців і R рядків еквівалентна масиву R елементів векторів із C компонентами.
- Скалярне вирівнювання типу OpTypeStructчлена визначається рекурсивно наступним чином:
- Скаляр розміру N має скалярне вирівнювання N .
- Векторний тип має скалярне вирівнювання, рівне вирівнюванню його компонентного типу.
- Тип масиву має скалярне вирівнювання, рівне вирівнюванню його типу елемента.
- Структура має скалярне вирівнювання, яке дорівнює найбільшому скалярному вирівнюванню будь-якого з її членів.

- Тип матриці успадковує скалярне вирівнювання з оголошення еквівалентного масиву.
- Базове вирівнювання типу OpTypeStructчлена визначається рекурсивно наступним чином:
- Скаляр має базове вирівнювання, яке дорівнює його скалярному вирівнюванню.
- Базове вирівнювання двокомпонентного вектора дорівнює його подвоєному скалярному вирівнюванню.
- Три- або чотирикомпонентний вектор має базове вирівнювання, яке вчетверо помножене на його скалярне вирівнювання.
- Базове вирівнювання масиву дорівнює базовому вирівнюванню його типу елемента.
- Структура має базове вирівнювання, яке дорівнює найбільшому базовому вирівнюванню будь-якого з її членів. Порожня структура має базове вирівнювання, що дорівнює розміру найменшого скалярного типу, дозволеного можливостями, заявленими в модулі SPIR-V. (наприклад, для 1-байтової вирівняної порожньої структури в StorageBuffer класі зберігання StorageBuffer8BitAccessабо UniformAndStorageBuffer8BitAccess має бути оголошено в модулі SPIR-V.)
- Матричний тип успадковує базове вирівнювання від еквівалентного оголошення масиву.
- Розширене вирівнювання типу OpTypeStructчлена визначається так само:
- Скалярний або векторний тип має розширене вирівнювання, яке дорівнює базовому вирівнюванню.
- Тип масиву або структури має розширене вирівнювання, яке дорівнює найбільшому розширеному вирівнюванню будь-якого з його елементів, округленому до числа, кратного 16.

- Матричний тип успадковує розширене вирівнювання від еквівалентного оголошення масиву.
- Член вважається таким, що неналежним чином працює, якщо виконується одне з наведеного нижче:
- Це вектор із загальним розміром, меншим або дорівнює 16 байтам, і має Offsetдекорації, які розміщують його перший байт у F і останній байт у L , де floor(F / 16) != floor(L / 16) .
- Це вектор із загальним розміром понад 16 байтів і має свої Offsetприкраси, які розміщують його перший байт у нецілому кратному 16.

**Приклади коду:**

```
#define GLM_FORCE_RADIANS // переконатися, що такі функції, як glm::rotate аргументи, використовують радіани

#include <glm/glm.hpp>

#include <glm/gtc/matrix_transform.hpp> //розкриває функції, які можна використовувати для генерації перетворень моделі, таких як glm::rotate, перетворень перегляду, таких як glm::lookAt і проекційних перетворень, таких як glm::perspective

#include <chrono> // надає функції для точного відліку часу.


struct UniformBufferObject { // структура, що буде використовуватися дескриптором як уніфікований об’єкт

    alignas(16) glm::mat4 model; // структура типу mat4 з вирівнюванням 16

    alignas(16) glm::mat4 view;

    alignas(16) glm::mat4 proj; // perspective projection

};
```

```
    void createUniformBuffers() {
        VkDeviceSize bufferSize = sizeof(UniformBufferObject);
        uniformBuffers.resize(MAX_FRAMES_IN_FLIGHT);
        uniformBuffersMemory.resize(MAX_FRAMES_IN_FLIGHT);
        uniformBuffersMapped.resize(MAX_FRAMES_IN_FLIGHT);
        for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
            createBuffer(bufferSize,
VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT,
VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT |
VK_MEMORY_PROPERTY_HOST_COHERENT_BIT, uniformBuffers[i],
uniformBuffersMemory[i]);
            vkMapMemory(device, uniformBuffersMemory[i], 0,
bufferSize, 0, &uniformBuffersMapped[i]); // отримання покажчика, до
якого ми можемо записати дані пізніше
        }
    }


    void createDescriptorPool() {
        VkDescriptorPoolSize poolSize{}; // опис, які типи
дескрипторів будуть містити наші набори дескрипторів і скільки їх
        poolSize.type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
        poolSize.descriptorCount =
static_cast<uint32_t>(MAX_FRAMES_IN_FLIGHT);
        VkDescriptorPoolCreateInfo poolInfo{}; // виділимо один із цих
дескрипторів для кожного кадру
        poolInfo.sType =
VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
```

```
        poolInfo.poolSizeCount = 1;
        poolInfo.pPoolSizes = &poolSize;
        poolInfo.maxSets =
static_cast<uint32_t>(MAX_FRAMES_IN_FLIGHT); // максимальна кількість
наборів дескрипторів, які можуть бути виділені:
        if (vkCreateDescriptorPool(device, &poolInfo, nullptr,
&descriptorPool) != VK_SUCCESS) {
            throw std::runtime_error("failed to create descriptor
pool!");
        }
    }

    void createDescriptorSets() {
        std::vector<VkDescriptorSetLayout>
layouts(MAX_FRAMES_IN_FLIGHT, descriptorSetLayout);
        VkDescriptorSetAllocateInfo allocInfo{}; // пул дескрипторів
для виділення, кількість наборів дескрипторів для виділення та макет
дескрипторів, на основі яких вони будуть засновані:
        allocInfo.sType =
VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO;
        allocInfo.descriptorPool = descriptorPool;
        allocInfo.descriptorSetCount =
static_cast<uint32_t>(MAX_FRAMES_IN_FLIGHT);
        allocInfo.pSetLayouts = layouts.data();
        descriptorSets.resize(MAX_FRAMES_IN_FLIGHT);
        if (vkAllocateDescriptorSets(device, &allocInfo,
descriptorSets.data()) != VK_SUCCESS) { // утримувати маркери набору
дескрипторів і розподілити їх
```

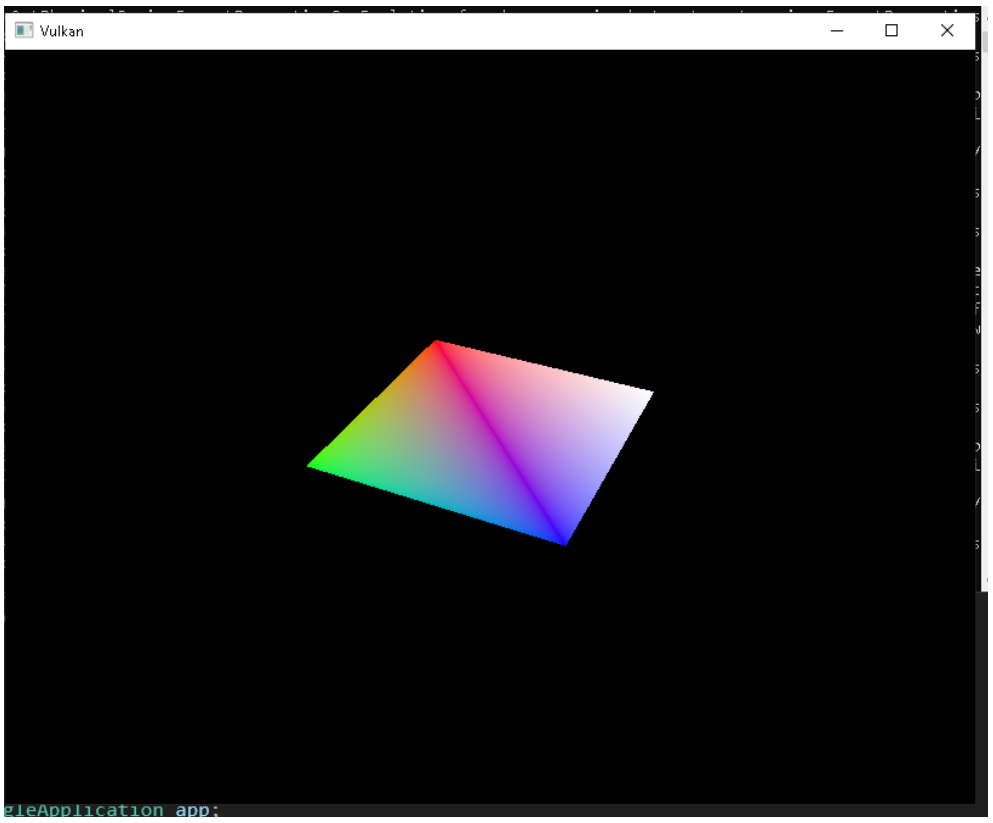
```
            throw std::runtime_error("failed to allocate descriptor
sets!");
        }
        for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) { //  цикл
для заповнення кожного дескриптора:
            VkDescriptorBufferInfo bufferInfo{};
            bufferInfo.buffer = uniformBuffers[i];
            bufferInfo.offset = 0;
            bufferInfo.range = sizeof(UniformBufferObject);
            VkWriteDescriptorSet descriptorWrite{};
            descriptorWrite.sType =
VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET; //  набір дескрипторів для
оновлення
            descriptorWrite.dstSet = descriptorSets[i]; // прив’язка
            descriptorWrite.dstBinding = 0; // індекс зв’язування
буфера
            descriptorWrite.dstArrayElement = 0; // не використовуємо
масив
            descriptorWrite.descriptorType =
VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
            descriptorWrite.descriptorCount = 1; // скільки елементів
масиву потрібно оновити.
            descriptorWrite.pBufferInfo = &bufferInfo;

            vkUpdateDescriptorSets(device, 1, &descriptorWrite, 0,
nullptr); // Оновлення застосовуються
        }
    }
```

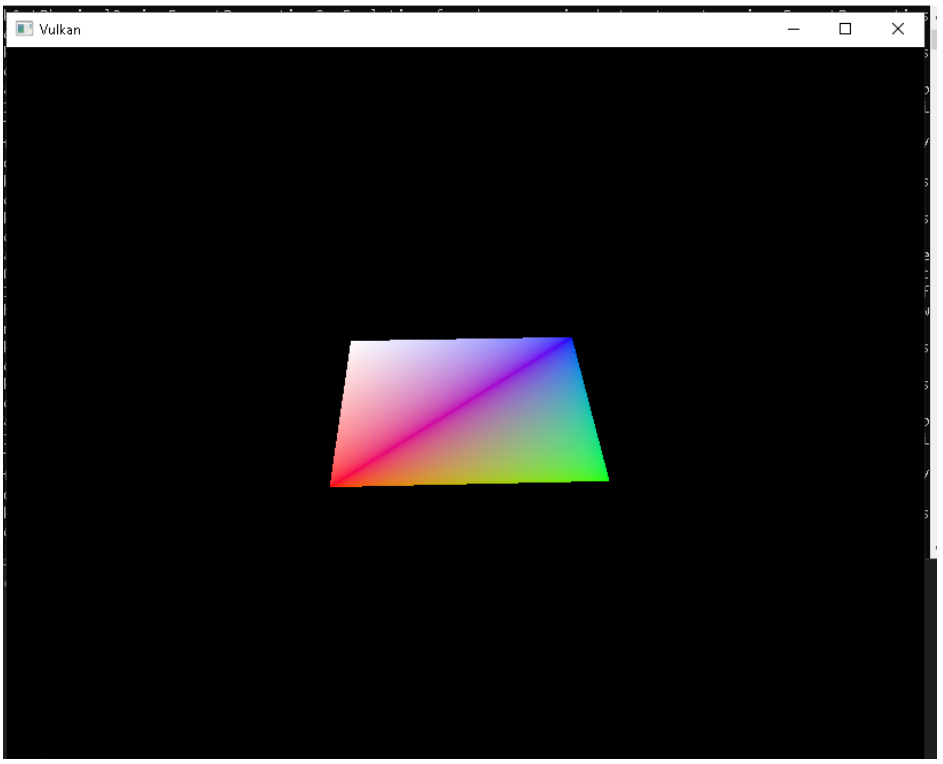
```
    void updateUniformBuffer(uint32_t currentImage) { // еометрія повертається на 90 градусів за секунду незалежно від частоти кадрів.
        static auto startTime = std::chrono::high_resolution_clock::now();
        auto currentTime = std::chrono::high_resolution_clock::now();
        float time = std::chrono::duration<float, std::chrono::seconds::period>(currentTime - startTime).count();
        UniformBufferObject ubo{};
        ubo.model = glm::rotate(glm::mat4(1.0f), time * glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0f)); // приймає наявне перетворення, кут повороту та вісь обертання як параметри
        ubo.view = glm::lookAt(glm::vec3(2.0f, 2.0f, 2.0f), glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 0.0f, 1.0f)); // подивитися на геометрію зверху під кутом 45 градусів. lookAt приймає положення очей, центральне положення та вісь вгору як параметри.
        ubo.proj = glm::perspective(glm::radians(45.0f), swapChainExtent.width / (float) swapChainExtent.height, 0.1f, 10.0f); // кут зору проекції, співвідношення сторін, ближня та далека площини огляду
        ubo.proj[1][1] *= -1;
        memcpy(uniformBuffersMapped[currentImage], &ubo, sizeof(ubo)); //  скопіювати дані з об’єкта уніфікованого буфера в поточний уніфікований буфер
    }
```

**Приклад нового вершинного шейдеру:**

```
#version 450

layout(binding = 0) uniform UniformBufferObject {

    mat4 model;

    mat4 view;

    mat4 proj;

} ubo;

layout(location = 0) in vec2 inPosition;

layout(location = 1) in vec3 inColor;

layout(location = 0) out vec3 fragColor;

void main() {

    gl_Position = ubo.proj * ubo.view * ubo.model * vec4(inPosition,
0.0, 1.0);

    fragColor = inColor;

}
```

**Результат запуску програми:**




Вікно створене за допомогою GLFW відображає 3-д об’єкт, що являє собою площину (plane) складену з двох примітивних фігур, а саме трикутників. Розфарбоване градієнтом відносно кольорів, що прикріплені до вершин у шейдері. З

ефектом трансформації - обертання об’єкту навколо осі Z (вертикальної), відносно змінної часу.

**Висновок:**

Я успішно навчився створювати та редагувати Uniform Buffers у середовищі API Vulkan. Це дало мені ширше уявлення про можливості параметрів при створення шейдерів. Тепер опановано не лише атрибутні, а й уніфіковані дані.

Додатково опрацьовані види трансформацій (такі як обертання, змінення розміру і тд) та інші концепції представлення об’єктів, по типу проекція, точка / кут зору, модель та інше.

Використавши уніфікований (уніформний) буфер додали можливість відображати нашу фігуру у 3-ьох вимірному представлені, що є зручнішим та ‘іноваційнішим’ ніж відображення примітивів на площині.