Память в Vulkan API

Быковских Дмитрий Александрович

05.10.2024

Память в Vulkan API
Быковских Дмитрий Александрович
05.10.2024

Введение

- Рассмотреть объекты Vulkan API
- Рассмотреть модель графического конвейера Vulkan API
- Далее по слайдам....

Введение
- Рассмотреть объекты Vulkan API
- Рассмотреть модель графического конвейера Vulkan API
- Далее по слайдам....

Для правильного выбора и настройки ресурсов, используемых в шейдерах, следует понимать, какие типы данных существуют, где их следует хранить, как использовать/связывать и т.д.
Например, вершинные атрибуты (позиция, цвет, нормали и т.д.), буферы, используемые для хранения общей информации о моделях, изображения (сэмплеры) с целью быстрого достоптупа к изображению,

и т.д.

Иерархия памяти современной графической карты

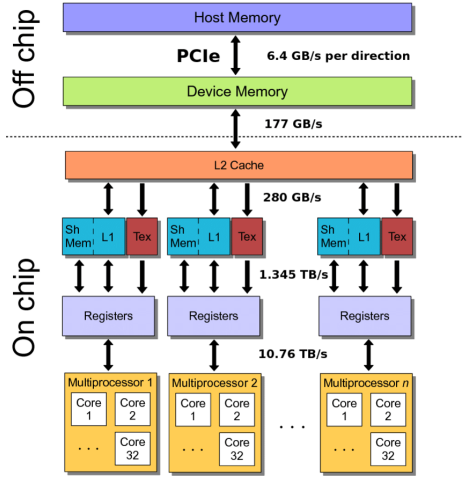
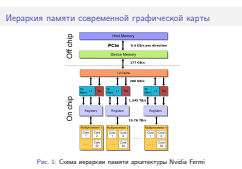


Рис. 1: Схема иерархии памяти архитектуры Nvidia Fermi

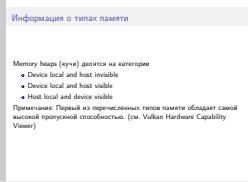


Информация о типах памяти

Memory heaps (кучи) делятся на категории

- Device local and host invisible
- Device local and host visible
- Host local and device visible

Примечание: Первый из перечисленных типов памяти обладает самой высокой пропускной способностью. (см. Vulkan Hardware Capability Viewer)



При разработке программ следует понимать следующие типы памяти

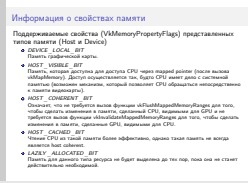
- Стек (stack) — это структура данных, организованная по принципу LIFO (Last In, First Out — последним пришёл, первым ушёл). Особенность его использования связана с вызовом функций. В первую очередь стек хранит адрес возврата и локальные переменные функций. Каждая новая вызванная функция добавляется в стек, а после завершения работы функции стек «сворачивается» обратно.
- Куча (heap) — это область динамической памяти, управляемая программой вручную. Она используется для хранения объектов, которые создаются в процессе выполнения программы, когда невозможно заранее предугадать размер памяти, который потребуется. Доступ к элементам в куче происходит напрямую через указатели или ссылки. Управление памятью в куче выполняется самостоятельно (т.е. нет сборщика мусора, может приводить к утечке).

Замечания:
1. Операции со стеком, как правило, быстрее, поскольку работают с последним добавленным элементом, тогда как в куче требуется больше времени для поиска и управления памятью.
2. Стек имеет ограничение на количество выделяемой памяти, в то время, как куча ограничена размерами физической памяти.

Информация о свойствах памяти

Поддерживаемые свойства (VkMemoryPropertyFlags) представленных типов памяти (Host и Device)

- *DEVICE_LOCAL_BIT*
  Память графической карты.
- *HOST_VISIBLE_BIT*
  Память, которая доступна для доступа CPU через mapped pointer (после вызова vkMapMemory). Доступ осуществляется так, будто CPU имеет дело с системной памятью (возможен механизм, который позволяет CPU обращаться непосредственно к памяти видеокарты).
- *HOST_COHERENT_BIT*
  Означает, что не требуется вызов функции vkFlushMappedMemoryRanges для того, чтобы сделать изменения в памяти, сделанный CPU, видимыми для GPU и не требуется вызов функции vkInvalidateMappedMemoryRanges для того, чтобы сделать изменения в памяти, сделанные GPU, видимыми для CPU.
- *HOST_CACHED_BIT*
  Чтение CPU из такой памяти более эффективно, однако такая память не всегда является host coherent.
- *LAZILY_ALLOCATED_BIT*
  Память для данного типа ресурса не будет выделена до тех пор, пока она не станет действительно необходимой.


Информация о свойствах памяти

Поддерживаемые свойства (VkMemoryPropertyFlags) представленных типов памяти (Host и Device)

- *DEVICE_LOCAL_BIT*
  Память графической карты.
- *HOST_VISIBLE_BIT*
  Память, которая доступна для доступа CPU через mapped pointer (после вызова vkMapMemory). Доступ осуществляется так, будто CPU имеет дело с системной памятью (возможен механизм, который позволяет CPU обращаться непосредственно к памяти видеокарты).
- *HOST_COHERENT_BIT*
  Означает, что не требуется вызов функции vkFlushMappedMemoryRanges для того, чтобы сделать изменения в памяти, сделанный CPU, видимыми для GPU и не требуется вызов функции vkInvalidateMappedMemoryRanges для того, чтобы сделать изменения в памяти, сделанные GPU, видимыми для CPU.
- *HOST_CACHED_BIT*
  Чтение CPU из такой памяти более эффективно, однако такая память не всегда является host coherent.
- *LAZILY_ALLOCATED_BIT*
  Память для данного типа ресурса не будет выделена до тех пор, пока она не станет действительно необходимой.

Некоторые примеры:
1. Область памяти с только одним флагом *DEVICE_LOCAL_BIT* используется для данных, которые будут использоваться исключительно GPU (вершинные буферы, текстуры и т.п.). Такой способ обеспечивает высокую производительность для GPU-доступа. Если необходимо, то для передачи данных используются промежуточные буферы с флагами *HOST_VISIBLE_BIT* и *HOST_COHERENT_BIT*.
2. Область памяти с флагами *HOST_VISIBLE_BIT* и *HOST_COHERENT_BIT* подходит для данных, которые будут часто обновляться CPU и передаваться GPU (uniform-буферы, динамические буферы), при этом имеет легкий доступ CPU без необходимости явной синхронизации.

Процесс управления ресурсами

Этапы работы

- Создание объекта с данными (например, загрузка моделей)
- Запрос подходящего типа памяти и создание типа объекта (например, буфер или изображение)
- Установка требований к выделению памяти
- Выделение области памяти и сохранение данных в него (возможно, потребуется промежуточный буфер)
- Связывание область памяти с данными с типом созданного объекта (выполняется через дескрипторы)

Процесс управления ресурсами

Этапы работы
- Создание объекта с данными (например, загрузка моделей)
- Запрос подходящего типа памяти и создание типа объекта (например, буфер или изображение)
- Установка требований к выделению памяти
- Выделение области памяти и сохранение данных в него (возможно, потребуется промежуточный буфер)
- Связывание область памяти с данными с типом созданного объекта (выполняется через дескрипторы)

Набор дескрипторов — интерфейс, связывающий ресурсы (буферы, изображения и т.д.) и шейдеры.

Пул дескрипторов, позволяет сохранить такие наборы, которые можно использовать в дальнейшем.