Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Новосибирский государственный технический университет»

Кафедра автоматики

**Отчет ПО ПРАКТИКЕ**

Производственная практика: практика по получению профессиональных умений и опыта

(наименование практики в соответствии с учебным планом)

профессиональной деятельности\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направление подготовки: 09.03.01 Информатика и вычислительная техника\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| Выполнил:  Студент Сизов Л.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Ф.И.О.)  Группа АВТ-318\_\_\_\_\_\_  Факультет АВТ\_\_\_\_\_\_\_\_\_\_\_\_.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись  «22» апреля 2017г. | Проверил:  Руководитель от НГТУ Гунько А.В.\_\_\_\_\_\_\_\_\_\_\_  (Ф.И.О.)  Балл: \_\_\_\_\_\_\_\_\_\_, ECTS\_\_\_\_\_\_\_\_\_\_\_\_\_,  Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «отлично», «хорошо», «удовлетворительно», «неуд.»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись  «22» апреля 2017г. |

Новосибирск 2017 г.

Оглавление

[Обзор библиотеки *Vulkan* и её функций для вычислений на графических процессорах 3](#_Toc480330199)

[1. Общие сведения 3](#_Toc480330200)

[1.1. Терминология 3](#_Toc480330201)

[1.2. Архитектурная модель 4](#_Toc480330202)

[1.3. Конвейеры *Vulkan* 6](#_Toc480330203)

[1.4. Модель выполнения 7](#_Toc480330204)

[1.5. Операции в очередях 8](#_Toc480330205)

[1.6. Объектная модель 8](#_Toc480330206)

[1.7. Механизмы синхронизации 8](#_Toc480330207)

[1.8. Многопоточное поведение 8](#_Toc480330208)

[2. Обзор функций, структур данных, утилит 8](#_Toc480330209)

# Обзор библиотеки *Vulkan* и её функций для вычислений на графических процессорах

# Общие сведения

## Терминология

В спецификации *Vulkan API* используются следующие понятия:

|  |  |  |
| --- | --- | --- |
| Понятие(англ.) | Понятие(рус.) | Значение и описание |
| *ABI* | Двоичный интерфейс приложения | Набор соглашений для доступа к операционной системе и другим низкоуровневым сервисам, спроектированный для переносимости исполняемого кода между машинами, имеющими совместимые ABI |
| *API* | Интерфейс программирования приложений | Набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) или операционной системой для использования во внешних программных продуктах |
| *Command* | Команда | Непосредственные команды для графического процессора |
| *Command Buffer* | Командный буфер | Структура данных, содержащая команды для графического процессора |
| *Compute Shader* | Вычислительный шейдер | Программа на специальном языке для графического процессора, предназначенная для вычислений |
| *CPU* | ЦП | Центральный процессор |
| *Device* | Устройство | Логическая сущность, представляющая собой физическое устройство. Содержит в себе информацию об устройстве |
| *Extension* | Расширение | Наборы расширения базового функционала Vulkan с помощью обратных вызовов |
| *Fragment Shader* | Фрагментный(пиксельный) шейдер | Один из этапов графического конвейера |
| *Geometry Shader* | Геометрический шейдер | Один из этапов графического конвейера |
| *GPU* | ГП | Графический процессор |
| *Host* | Хост | Программа, отправляющая команды на графический процессор |
| *Physical Device* | Физическое устройство | Графический процессор, выполняющий команды, отправленные с хоста |
| *Queue* | Очередь | Структура данных, в которую поступают команды для устройства |
| *Queue Family* | Семейство очередей | Набор очередей, предоставляющих определённые возможности |
| *Shader* | Шейдер | Программа на специальном языке, предназначенная для исполнения на ГП. В *Vulkan* представляет собой программируемую стадию графического конвейера |
| *Tessellation* | Тесселяция | автоматизированный процесс добавления новых выпуклых многоугольников в полигональную сетку с целью повышения детализации сетки. |
| *Tessellation Shader* | Шейдер тесселяции(замощения) | Один из этапов графического конвейера |
| *Validation Layer* | Слой валидации | Прослойка между драйвером и хостом для перехвата ошибок и обработок ошибок, записи логов и трассировки вызовов |
| *Vertex Shader* | Вершинный шейдер | Один из этапов графического конвейера |

## Архитектурная модель

Библиотека *Vulkan* спроектирована и написана для ЦП, ГП и других аппаратных ускорителей со следующими свойствами:

* Поддержка 8, 16, 32 и 64 битных знаковых и беззнаковых целых с обратным кодом для представления отрицательных чисел
* Поддержка 32 и 64 битных чисел с плавающей запятой с точностью в 6 знаков
* Порядок байтов в двоичном представлении чисел должен быть одинаковый для хоста и физического устройства

Библиотека *Vulkan* представляет собой интерфейс программирования приложений со свободными точками входа в функции. Большая часть параметров и возвращаемых типов представляет собой структуры языка *C,* обязательные для заполнения. Все указатели по умолчанию имеют размер 64 бита, если не указан явный размер в 32 бита или на платформе *x86.*

Двоичный интерфейс приложения предоставляется производителем драйвера для конкретной платформы. Например, поставщиком *ABI* для *Android* является компания *Google.*

В исходном виде, драйвер *Vulkan* не возвращает никаких ошибок, кроме критических и не проверяет поступающие данные. Для проверки данных используются слои, которые играют роль прослойки между хостом и драйвером. В этих слоях происходит проверка входящих данных, но никакой информации о ошибках или успехе проверок они не возвращают, а просто предотвращают критические сбои в работе программы.

|  |  |
| --- | --- |
| C:\Users\whyami\Desktop\normal.jpg | C:\Users\whyami\Desktop\fail.jpg |
| Рисунок 1а. | Рисунок 1б. |

Рисунки 1а и 1б демонстрируют механизм работы слоёв в случае успешного прохождения всех проверок (Рисунок 1а) и в случае провала одной из проверок (Рисунок 1б).

Базовый функционал *Vulkan* можно расширить с помощью расширений, основная часть которых поставляется вместе с *SDK*; другие расширения поддерживаются поставщиками драйверов и *ABI*. Примером расширения является базовое расширение *VK\_EXT\_debug\_report* позволяет получать детальную информацию о поведении программы и ошибках при её исполнении. Примером стороннего расширения является расширение *VK\_NV\_external\_memory\_capabilities* от *NVidia*, которое позволяет использовать память в приложении *Vulkan* память, выделенную средствами *Direct3D*.

При включенном расширении *VK\_EXT\_debug\_report* у слоя появляется возможность передавать информацию об ошибках на предыдущий слой. Тогда механизм работы слоёв меняется (Рисунок 2) и появляется возможность передавать отладочную информацию или информацию об ошибках при проверках в слоях назад.

## Конвейеры *Vulkan*

Библиотека *Vulkan* предоставляет два конвейера для использования: графический и вычислительный. Они изображены на рисунке 3. Структуру конвейеров и порядок стадий изменить нельзя (можно только пропустить некоторые стадии). Жёлтым цветов на рисунке 2 выделены программируемые стадии конвейера, называемые шейдерами.

В шейдер передаётся необходимая информация через *Buffer* или *Image*. Где первый может содержать любые данные, а второй только информацию о пикселях в определённом изображении.

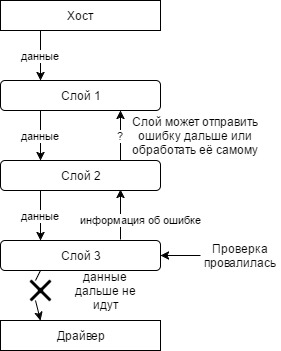


Рисунок 2. Механизм работы слоёв со включенным расширением обработки ошибок

Перед тем как запустить шейдер на выполнение, необходимо передать ему полученную из приложения информацию. Это происходит на этапе *Assembler,* который присутствует у каждой стадии конвейера. *Assembler* занимается тем, что преобразует читает и преобразует в нужный формат полученные данные и передают дальше в шейдер.

В графическом конвейере существуют следующие стадии:

* Вершинный шейдер – производит преобразования для переноса позиций вершин из пространства моделей в пространство экрана. Он так же передает данные по вершинам далее по конвейеру;
* Шейдер тесселяции (опциональный) – позволяет разбивать трёхмерную сетку модели на основе определённых правил для повышения качества сетки;
* Геометрический шейдер – позволяет добавлять, изменять и удалять трёхмерные примитивы
* Фрагментный(пиксельный) шейдер – производит выборку из текстур и/или математические операции над цветом и значением глубины пикселей для каждого отображаемого на экране пикселя



Рисунок 3. Графический и вычислительный конвейеры *Vulkan*

Вычислительный конвейер состоит из всего одной стадии – вычислительного шейдера. Он позволяет производить произвольные вычисления для, например, симуляции системы частиц или расчёта отражения лучей.

## Модель выполнения

Библиотека *Vulkan* даёт доступ к физическому устройству через одну или несколько логических сущностей Устройство. Оно в своё очередь предоставляет доступ к очередям выполнения команд. Наборы поддерживаемых ГП очередей разбиты на семейства. Каждое семейство поддерживает функциональность определённого типа и может содержать несколько очередей со схожими параметрами. Существует 4 типа очередей:

* Для работы с графикой
* Для работы с вычислениями
* Для перемещения данных
* Для работы с разреженной памятью

Память на физическом устройстве может быть выделена только явным образом с хоста. Каждое устройство может создать несколько блоков памяти типа куча, которые могут размещаться как на физическом устройстве, так и на хосте, причём вне зависимости от типа кучи, эта память всегда видна хосту.

Виды памяти типа куча:

* *DLHV heap* (*device local, host visible*) – память физически размещена на физическом устройстве, видна хосту
* *HLHV heap* (*host local, host visible*) – память размещена на хосте, видна и хосту и устройству

Приложение на *Vulkan* отправляет устройствам командный буфер, содержащий записанные команды для физического устройства. После того, как создан командный буфер, он может быть поставлен в очередь на исполнение неограниченное количество раз. Спецификация не определяет, что произойдёт, если отправить несколько командных буферов в несколько разных очередей – командные буферы могут исполниться параллельно или не в том порядке, в каком они были отправлены в очереди. Но в пределах одной очереди командные буферы выполняются в том порядке, в котором они были отправлены.

Выполнение командных буферов по отношению к приложению-хосту происходит асинхронно. О синхронизации между очередями, устройством и хостом должен заботиться хост.

Командные буферы могут находиться в нескольких состояниях:

* Первичное – в этом состоянии буфер находится, когда он только был создан
* Режим записи – буфер записывает все команды для исполнения устройством, которые были вызваны функциями вида *vkCmd\**
* Исполнение – буфер записал все нужные команды и готов быть переданным на исполнение в очередь
* В ожидании – буфер команд передан в очередь и исполняется устройством. В этом состоянии нельзя изменять буфер
* Ошибка – буфер в этом состоянии может быть только освобожден, сброшен на исходное состояние или включен в режим записи

## Операции в очередях

Очереди в *Vulkan* предоставляют возможность запуска команд на устройстве. Команды записываются в командный буфер вместе с командой *Queue submission* (англ. - отправка в очередь) и запускаются на исполнение специальными движками исполнения. После того, как команды были отправлены в очередь, хост не может никак повлиять на работу этих команд. Вместе с командным буфером можно передать набор семафоров для того, чтобы сигнализировать устройству о том, что можно начать выполнение, или для того, чтобы устройство сообщило об окончании определённой операции.

Спецификация *Vulkan* не определяет порядок выполнения команд в разных очередях – если передать два буфера команд в две разные очереди, то нельзя заранее сказать, какой из них выполнится первым.

Порядок выполнения командных буферов можно контролировать с помощью примитивов синхронизации, которые подробнее описаны в пункте 1.7. Не смотря на асинхронную природу выполнения команд устройством, гарантируется, что все задачи, стоящие до примитива синхронизации, будут выполнены к тому моменту, когда примитив синхронизации отправит сигнал о синхронизации устройству.

Команды в очередях могут выполнять действия (рисовать, копировать память, очищать, начинать/останавливать подпроходы отрисовки), изменять состояние (состояние подпрохода отрисовки, отправка констант, привязка буферов к очередям, конвейерам) или производить операции синхронизации.

Так как команды для устройства выполняются асинхронно, то управление может сразу же вернуться хосту после отправки командного буфера в очередь.

## Объектная модель

Устройства, очереди и сущности *Vulkan* представлены в виде объектов. Все объекты на уровне *API* управляются через их дескрипторы. Всего существуют дескрипторы двух видов: управляемые и неуправляемые. Управляемые дескрипторы — это указатели на непрозрачные типы данных (типы данных, не объявленные в интерфейсах, скрытые от глаз). Обычно такие дескрипторы используются для перехвата вызовов *API* драйвером и чаще всего передаются в функции первым параметром. Спецификация *Vulkan* определяет, что каждый управляемый объект должен иметь один уникальный управляемый дескриптор в течение всего времени жизни объекта.

Неуправляемые дескрипторы – целочисленные 64 битные типа данных, применение которых может меняться в зависимости от реализации. Неуправляемые объекты могут иметь несколько неуправляемых дескрипторов, причём гарантируется, что уничтожение одного дескриптора не уничтожит другие дескрипторы.

Все объекты, созданные с помощью сущности логического устройства *VkDevice* видны и могут быть использованы только этим логическим устройством.

Объекты создаются с помощью функций вида *vkCreate\**, другие объекты создаются выделением памяти функцией вида *vkAllocate\**. Как только объект был создан, его структура становится иммутабельной(неизменяемой), но данные в структуре изменять не запрещено.

Предполагается, что создание и уничтожение объектов происходит нечасто, в отличии от выделения и освобождения памяти под объекты. Поэтому объектов, требующих явного выделения памяти рекомендуется использоваться пулы и кучи.

Уничтожение объектов осуществляется функциями вида *vkDestroy\**, освобождение памяти – *vkFree\**.

Драйвер никак не контролирует время жизни объектов, перекладывая эту задачу на плечи хоста. Хост сам должен контролировать, чтобы объекты не уничтожались во время использования.

Существует также набор объектов, которые нельзя уничтожать при любых или определённых условиях.

* Никакой командой ни при каких условиях нельзя уничтожать объекты *VkShaderModule* и *VkPipelineCache*
* Нельзя уничтожать объект *VkPipelineLayout*, если какой-либо буфер команд находится в режиме записи
* Если командный буфер находится в режиме ожидания, то никакой из следующих объектов, используемых командным буфером, не должен быть уничтожен:
  + *VkEvent*
  + *VkQueryPool*
  + *VkBuffer*
  + *VkBufferView*
  + *VkImage*
  + *VkImageView*
  + *VkPipeline*
  + *VkSampler*
  + *VkDescriptorPool*
  + *VkFramebuffer*
  + *VkRenderPass*
  + *VkCommandBuffer*
  + *VkCommandPool*
  + *VkDeviceMemory*
  + *VkDescriptorSet*
* Если очередь выполняет команды, то следующие объекты, используемые в буферах команд, не должны быть уничтожены:
  + *VkFence*
  + *VkSemaphore*
  + *VkCommandBuffer*
  + *VkCommandPool*
* Объекты типа *VkQueue* не могут быть уничтожены явно – они уничтожаются автоматически вместе с объектом устройства *VkDevice*, с помощью которого они были созданы
* При уничтожении объекта пула, все объекты из этого пула уничтожаются неявным образом
* Объект логического устройства *VkDevice* может быть уничтожен только тогда, когда все связанные с ним очереди находятся в режиме ожидания, т.е. не имеют команд в очереди, и уничтожены следующие объекты, связанные с очередями: *VkFence, VkSemaphore, VkEvent, VkQueryPool, VkBuffer, VkBufferView, VkImage, VkImageView, VkShaderModule, VkPipelineCache, VkPipeline, VkPipelineLayout, VkSampler, VkDescriptorSetLayout, VkDescriptorPool, VkFramebuffer, VkRenderPass, VkCommandPool, VkCommandBuffer, VkDeviceMemory*
* Объект логического устройства *VkPhysicalDevice* не должен уничтожаться явным образом, он уничтожается вместе с объектом *VkInstance* в конце цикла жизни программы

## Механизмы синхронизации

Синхронизация доступа к ресурсам является первичной задачей приложения-хоста. Существует несколько неявных гарантий порядка запуска команд, однако в большинстве случаев это нужно делать самому, как и управление кэшем и другими оптимизациями.

*Vulkan* предоставляет 4 примитива синхронизации:

* Барьер/Забор (*Fence*) – используются для сообщения хосту о завершении команды
* Семафор – используется для контроля доступа к ресурсам из нескольких очередей
* События – используются для сообщения о наступлении какого-либо события из хоста или командного буфера.
* Конвейерный барьер (*Pipeline Barrier*) – используются для синхронизации внутри командных буферов, но только в одном месте за раз.

# Обзор функций, структур данных, утилит

# Список литературы