Vulkan C++
Markus Tavenrath, Senior DevTech Software Engineer
Professional Visualization
Who am I?
Markus Tavenrath

• Senior Dev Tech Software Engineer - Professional Visualization
• Joined NVIDIA 8 years ago to work on middleware
  • Goal: Make graphics programming easy and efficient
• Working with CAD ISVs to optimizing their graphics pipelines
• Working with driver team on Vulkan and OpenGL performance
What is Vulkan?

What developers have been asking for

Reduce CPU overhead
Scale well with multiple threads
Precompiled shaders
Predictable - no hitching
Clean, modern and consistent API - no cruft
Native tiling and mobile GPU support
What is Vulkan?

Downsides

• Vulkan moves responsibility from driver to application
  • Application has to write ‘driver‘-code
  • Simple HelloWorld is ~750 lines of code
• So much to do, hard to start
• It‘s easy make errors
• It‘s hard to find those errors
• Is there a way to simplify Vulkan usage, especially for beginners?
Goals

Preview

Native (~750loc)

VkCPP (~200loc)
Goals

C++11 API on top of Vulkan

- Simplify Vulkan usage by
  - reducing risk of errors, i.e. type safety, automatic initialization of sType, ...
  - Reduce #lines of written code, i.e. constructors, initializer lists for arrays, ...
  - Add utility functions for common tasks (suballocators, resource tracking, ...)

- Make it easy to use Vulkan!
Implementation

- Two C++ based layers
  - Autogenerated 'low-level' layer using vulkan.xml
    - Type safety
    - Syntactic sugar
  - Hand-coded 'high level' layer
    - Reduce code complexity
    - Exception safety, RAII, resource lifetime tracking, ...
- Layers are work in progress, showing current state
Low Level Layer

vk namespace

- All Vulkan symbols replicated in vk namespace to avoid conflicts
  - Structs
    - VkRect2D -> vk::Rect2D
  - Functions
    - VkCmdDraw(...) -> vk::cmdDraw(...)
  - Enums
    - VkImageViewType -> vk::ImageViewType
  - Flags
    - VkQueueFlags -> vk::QueueFlags
  - ...

Low Level Layer

Enums

• Enforce type safety for enums by using scoped enums
  • `VK_IMAGE_VIEW_TYPE_CUBE` -> `vk::ImageViewType::CUBE`
  • Special case:
    • Symbols starting with a number are prefixed with a ‘_’
    • `VK_IMAGE_VIEW_TYPE_1D` -> `vk::ImageViewType::1D`
Low Level Layer

Flags

- Enforce type safety for flags with `vk::Flags` template class
  - `typedef vk::Flags<QueueFlagBits> QueueFlags;
  - `QueueFlags flags(vk::QueueFlagBits::GRAPHICS)`
  - Most bit-operators supported
    - `flags = vk::QueueFlagBits::GRAPHICS | vk::QueueFlagBits::COMPUTE;`
    - `flags |= vk::QueueFlagBits::COMPUTE;`
    - `flags &= vk::QueueFlagBits::COMPUTE;`
    - ...
Low Level Layer

Basic struct initialization

- Implement class vk::* for struct Vk*

  - class Extent2D {
    public:
      Rect2D(int width, int height) // add constructors
      operator vkRect2D const &() { return m_rect2D; } // nop in release
    private:
      vkRect2D m_rect2D;
  };

  - C++ style `vk::Rect2D r(vk::Offset2D(0,0), vk::Extent2D(1920, 1080));`

  - C++11 style `vk::Rect2D r {{0,0}, {1920, 1080}}`
Low Level Layer

C99 designated initializers style

• Named parameter idiom closest thing one can get in C++

• `class Extent2D {
   public:
   ...
   Extent2D& width(int width_) // setter
   int const &width(); // getter
};`

• `Vk::Extent2D e = vk::Extent2D().width(1920).height(1080);`

• Just set what you need - in any order
Low Level Layer

*CreateInfo initialization

- *CreateInfo structs initialization simplified
  - `sType` and `pNext` set automatically in constructor
  - Native:
    ```
    VkMemoryAllocateInfo mai = {VK_MEMORY_ALLOCATE_INFO};
    mai.allocationSize = 1024;
    mai.memoryTypeIndex = 0;
    ```
  - vkcpp:
    ```
    vk::MemoryAllocateInfo mai(1024, 0);
    or
    vk::MemoryAllocateInfo mai = {1024, 0};
    ```
Low Level Layer

Unions

• Add one constructor for each enum type

• ```
struct ClearColorValue {
    ClearColorValue(std::array<float,4> float32
        = { 0.0f, 0.0f, 0.0f, 0.0f } );
    ClearColorValue(std::array<int32_t,4> int32);    
    ClearColorValue(std::array<uint32_t,4> uint32);  

    ...
};
```

• ```
ClearColorValue c = {1, 0, 0, 0}; uses expected constructor (int32_t);
but
VkClearColorValue c = {1, 0, 0, 0}; interprets values as floats (on msvc)
```
Low Level Layer

Default Parameters?

- Constructors would allow to define default parameters
  - Useful in some cases
- Unfortunately no default parameters available in vulkan.xml
  - What are good defaults?
- Enhance vulkan.xml or add separate xml file?
- Could be quite useful in combination with named parameter idiom
Low Level layer

Conclusion

• Autogenerated C++ wrapper removes some potential errors
  • Wrong enums for member variables or bitfields
  • Passing Enums for non-enums fields
  • Missing/Incorrect pType field in createInfos
  • Uninitialized fields when using constructors
• Constructors make code horizontal, less lines of code
Low Level Layer

Conclusion

• Low level layer good for 'ninjas' which prefer C++-style over C-style
  • Still hard to use for people who 'just want to render' a large number of objects

• Introduce higher level interface for people who 'just want to use Vulkan'
  • 'Close' to native Vulkan
  • Common tasks handled by utility functions or interfaces
  • Object lifetime, resource allocation, resource tracking, ...
High Level Layer

Initialization

• RAII with shared_ptrs and factories

```cpp
std::shared_ptr<nvk::Instance> instance;
instance = nvk::Instance::create("GLUTHelloVulkan", 0, ...);
```

• Parameterized interface, keep code horizontal
Initialization

Device creation & hierarchy

- PhysicalDevices are permanent -> getPhysicalDevice

```cpp
// get first physical device
std::shared_ptr<nvk::PhysicalDevice> physicalDevice;
physicalDevice = instance->getPhysicalDevice(0);

// create device from PhysicalDevice
std::shared_ptr<nvk::Device> device;
device = physicalDevice->createDevice();
```
Instance must not be deleted before all devices had been destroyed

Use shared_ptr to establish object hierarchy lifetime upwards

Use weak_ptr for 'permanent' object like PhysicalDevice and Queue
High Level Layer

Hierarchy Lifetime

- Instance
- PhysicalDevice
- Device

- CommandPool
- CommandBuffer
- DescriptorPool
- DescriptorSet
- QueryPool
- Buffer
- BufferView
- ShaderModule
- Pipeline
- Framebuffer
- RenderPass
- DeviceMemory
- Image
- ImageView
- Semaphore
- Event
- Fence
- Sampler
- PipelineCache
Framebuffer creation

Image creation

- Create Image for color buffer

```cpp
colorImage = device->createImage(  
    vk::Format::R8G8B8A8_UNORM,  
    windowSize, 1, 1,  
    vk::SampleFormat::_1,  
    vk::ImageTiling::OPTIMAL,  
    vk::ImageUsage::COLOR_ATTACHMENT,  
    vk::ImageLayout::COLOR_ATTACHMENT_OPTIMAL, 0, heap);
```

Scoped Enums

Allocator Interface
Low-level memory control

Heap interface

• Introduce new Heap interface
  
  ```cpp
  class Heap {
    virtual std::shared_ptr<nvk::Allocation> allocate(Requirements) = 0;
  };
  ```

• Allocation keeps reference to Heap and
• ~Allocation responsible for free
• Device provides default heap

Interface class by Heap
  std::shared_ptr<vk::DeviceMemory>, offset
Framebuffer creation

ImageView creation

• It’s not uncommon that imageView has colorFormat and imageType as Image
• Query information from vk::Image, avoid 'conflicting' types
• Creating ImageViews with different types also possible

```cpp
colorImageView = colorImage->createImageView();
```
Framebuffer creation

Framebuffer

- Framebuffer gets
  - `windowSize` (redundant, query from `imageView`?)
  - `std::vector<nvk::ImageView>` for attachments
  - `RenderPass`

```cpp
Framebuffer framebuffer = device->createFramebuffer(
    windowHeight, { colorImageView }, renderPass);
```

- `std::vector of std::shared_ptr<vk::ImageView>`
GraphicsPipeline creation

**VertexInputState**

- `nvk::GraphicsPipelineCreateInfo` is class for full GraphicsPipeline creation state
- Sets ‘good’ defaults upon initialization, i.e. 1 Viewport, triangles, ...

```cpp
nvk::GraphicsPipelineCreateInfo gpci;
gpci.setVertexInputState(
    {{0, sizeof(Vertex)}},
    {{0, 0, vk::Format::R32G32_SFLOAT, offsetof(Vertex, position)},
     {1, 0, vk::Format::R8G8B8A8_UNORM, offsetof(Vertex, color) }});
```

std::vector<VertexInputBindingDescription>

std::vector<VertexInputAttributeDescription>
GraphicsPipeline creation

Shader

```
shaderVertex = device->createShaderModule(srcVertex);
shaderFragment = device->createShaderModule(srcFragment);
gpci.setShaderStages(
    { { vk::ShaderStage::VERTEX, shaderVertex, "main" },
      { vk::ShaderStage::FRAGMENT, shaderFragment, "main" } } )
);

// create actual pipeline
pipeline = device->createGraphicsPipeline(gpci);
```
Resource Tracking

Lifetime

- IndexBuffers
- VertexBuffers
- Images

Bind Resources

- Set of Resources
- specifying
- referencing

Keep Resources alive as long as GPU is referencing them!

GPU processing

CommandBuffer

Submit

Queue

Fence signal
ResourceTracking

Automation

- Introduce ResourceTracker, used by CommandBuffer

```cpp
class ResourceTracker {
    // track resources
    void track(std::shared_ptr<vk::Buffer> const & buffer);
    void track(std::shared_ptr<vk::Event> const & event);

    // track usage on GPU
    void addFence(std::shared_ptr<vk::Fence> const & fence);

    // check if used on GPU, also removes finished entries
    bool isUsed();
};
```
Resource Tracking

Automation

- CommandBuffer holds ResourceTracker implementation
- Calls resourceTracker->add() for used resources, build up resource set
- On queue->submit() add entry to a map<sp<Fence>, sp<ResourceTracker>>
- Queue keeps ResourceTracker alive as long as Fence hasn‘t been reached
  -> ResourceTracker keeps references to resources
Resource Tracking

Potential problems
- Performance, tracking all resources each frame might be slow
  - ResourceTracker is interface, write your own tracker which keeps track of frameset
  - In debug mode you could validate if all resources are referenced

How to trigger resource cleanup
- Async: Wait for fence in a separate thread
- Sync: Provide function Queue::releaseResources() which frees resources for all reached fences
Questions?

Markus Tavenrath, Senior DevTech Software Engineer
Professional Visualization