What's your Vulkan Memory Type?

By Mathias Schott, posted Aug 29 2016 at 10:52AM
Tags:
Vulkan (/taxonomy/term/586), GameWorks (/taxonomy/term/315),
GameWorks Expert Developer (/category/tags/gameworks-expert-developer), DesignWorks (/taxonomy/term/701)

Authors: Mathias Schott & Jeff Bolz

This blog post discusses recent improvements and additions to the memory management interface of our Vulkan implementation, based on collaborating with game developers porting their games to Vulkan.

- Host-local memory types (in "GPU accessible system memory") to allow applications to put buffers, textures and render targets into host-local memory to gracefully handle device-local memory ("video memory") oversubscription.
- Introduction of the VK_NV_dedicated_allocation extension that allows applications to provide additional explicit information in order to enable additional optimizations. This functionality has been standardized as the VK_KHR_dedicated_allocation extension since the original publication of this article. See NVIDIA Vulkan Developer Driver for Khronos Vulkan Spec update 1.0.54 (/nvidia-vulkan-developer-driver-khronos-vulkan-spec-update-1054) for details.

We also provide the DedicatedAllocationVk sample which illustrates basic use cases for the extension.

Background

The Vulkan memory allocation interface is designed to enable applications to intelligently manage their memory usage, including the ability to share and reuse memory between resources without freeing and reallocating memory, and to sub-allocate resources within a small number of relatively large memory allocations. The API includes the flexibility to accommodate different memory architectures, such as a discrete GPU with a local framebuffer or a mobile SoC with unified memory, by enumerating an abstract set of memory heaps and memory types.

We initially tried to expose a minimal set of memory types, but subtle interactions between hardware, driver, and operating system issues have prompted us to expand our advertised set of memory types, as discussed by the following sections.

Memory Types for Resources in System Memory

In OpenGL or DirectX 11, the driver traditionally has been supporting the application's resource allocation by moving resources between device local and system memory in case of oversubscription of device memory, which can happen if the user might select image quality settings that exceed the amount of available device local memory. This naturally has the potential for reduced performance, especially if done excessively, but can work pretty well in practice when used in moderation.

However Vulkan, as an explicit API, does not provide for this at all and instead expects the application to handle those cases by itself. A simple, but ungraceful approach is to simply exit the application if a device memory allocation fails (e.g. somewhere deep within a level loading operation) which might be the only feasible way to handle this since restarting the content loading with reduced sized textures or graphics quality settings might not be possible due to overall engine structure.

For those cases, reduced application performance is likely a better trade-off for the user than application termination. This can be implemented by the application placing those resources into GPU accessible system memory when vkAllocateMemory (https://www.khronos.org/registry/vulkan/specs/1.0/man/html/vkAllocateMemory.html) returns

VK_ERROR_OUT_OF_DEVICE_MEMORY for a device-local memory type.

To enable this, we are exposing additional host-local memory types:

- A memory type for buffers
- A memory type for color images of any format
- Separate memory types for depth/stencil images for each depth/stencil format in system memory

A memory allocator that follows the rules and guidance of the Vulkan specification (https://www.khronos.org/registry/vulkan/specs/1.0/xhtml/vkspec.html#memory-device) should be able to handle all these memory types gracefully by properly interpreting the VkMemoryRequirements::memoryTypeBits (https://www.khronos.org/registry/vulkan/specs/1.0/xhtml/vkspec.html#VkMemoryRequirements) member when selecting an allocation for a specific resource.

Dedicated Allocation Extension

Vulkan’s memory allocation API provides a flexible toolbox to handle resource & memory management. But complete flexibility and freedom do not necessarily provide full performance. Advanced memory features - such as memory aliasing or sparse binding - could interfere with optimizations like framebuffer compression or efficient page table usage. These optimizations are important for performance of render targets and very large resources, which typically do not use memory aliasing or sparse binding.

An application can explicitly express the intent to never alias and never sparsely bind a specific resource via the VK_NV_dedicated_allocation extension (https://github.com/KhronosGroup/Vulkan-Docs/blob/1.0/doc/specs/vulkan/appendices/VK_NV_dedicated_allocation.txt), which adds a few structures to the resource creation and memory allocation functions. Those together allow device memory to be allocated for a particular buffer or image resource. Astute readers are likely going to notice the similarities to DirectX 12’s ID3D12Device::CreateCommittedResource (https://msdn.microsoft.com/en-us/library/windows/desktop/dn899178(v=vs.85).aspx).

On some devices this can significantly improve the performance of that resource. Where the application was more memory bandwidth limited, we’ve seen performance improvements of ~15%. Additionally, dedicated allocations improve the opportunities for the OS/driver to handle global video memory oversubscription by paging allocations between video memory to system memory, e.g. when another resource demanding application is running concurrently and thus competing for device local memory.

Usage Guidelines

Do's

- Use Dedicated Allocations for
  - Render targets for improved GPU performance
  - Very large buffers/images (dozens of MB)
- Use regular allocations and sub-allocations specifically for everything else
- Make sure that the total number of memory allocations is below VkPhysicalDeviceLimits::maxMemoryAllocationCount
  - Ideally stay < 1024 allocations to reduce CPU overhead on Windows 7
- Base your memory allocation strategy on correct interpretation of the results of the various related Vulkan API calls to safe-guard against implementation differences:
  - vkGetBufferMemoryRequirements (https://www.khronos.org/registry/vulkan/specs/1.0/xhtml/vkspec.html#vkGetBufferMemoryRequirements)
  - vkGetImageMemoryRequirements (https://www.khronos.org/registry/vulkan/specs/1.0/xhtml/vkspec.html#vkGetImageMemoryRequirements)
  - vkGetPhysicalDeviceMemoryProperties (https://www.khronos.org/registry/vulkan/specs/1.0/xhtml/vkspec.html#vkGetPhysicalDeviceMemoryProperties)

- Explicitly look for the VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT when picking a memory type for resources that are intended to be device local.

Don'ts

- Don’t put every resource into a Dedicated Allocation!
- Don’t pick the “first” memory type for memory allocations that are intended to be in device local memory
- Don’t put an excessive amount of image resources into GPU accessible system memory since it can reduce performance severely

How to get started?

Specification And Documentation

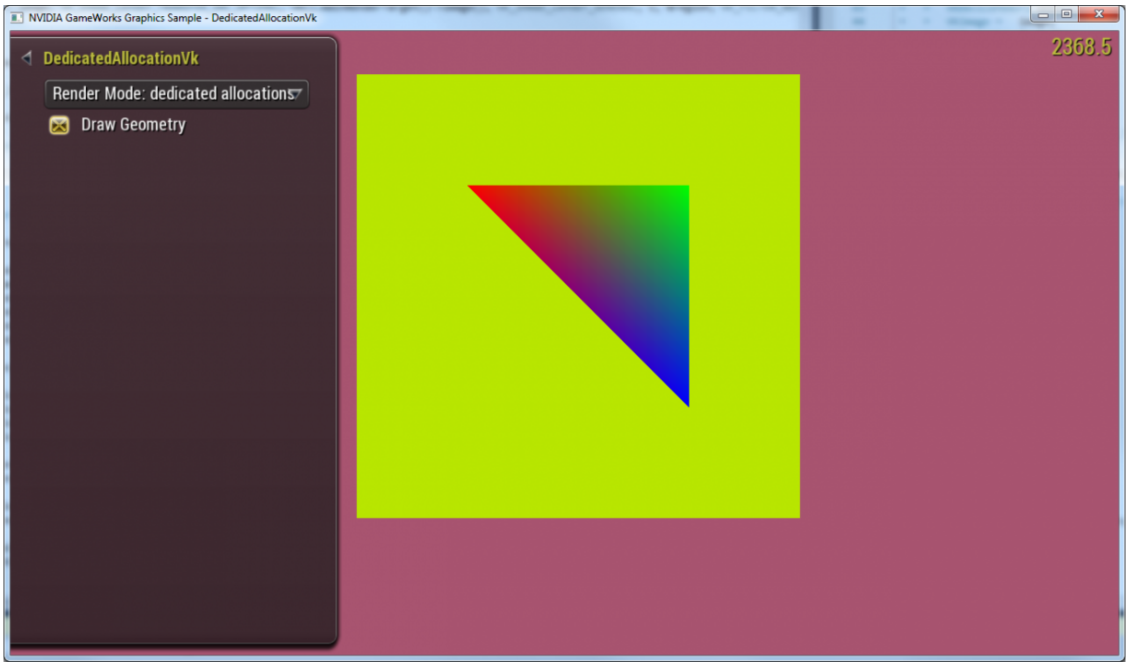
The specification (https://github.com/KhronosGroup/Vulkan-Docs/blob/1.0/doc/specs/vulkan/appendices/VK_NV_dedicated_allocation.txt) and headers (https://github.com/KhronosGroup/Vulkan-Docs/blob/1.0/src/vulkan/vulkan.h) can be found at the Khronos Vulkan API Documentation Github repository. Note:The Vulkan API documentation recently moved to a model where all extensions are documented in a single branch. Please refer to the “1.0” branch since the per extension branches are deprecated and won’t be updated in the future.

Drivers

We started exposing the dedicated allocation extension with version 368.69 (http://www.nvidia.com/download/driverResults.aspx/104726/en-us) drivers for Windows and with version 367.27 (http://www.nvidia.com/download/driverResults.aspx/104284/en-us) drivers for Linux. Android support in our SHIELD family of devices is expected to be provided by a future OTA.

Sample Code

The DedicatedAllocationVk sample application is available on Github (source (https://github.com/NVIDIAGameWorks/GraphicsSamples/tree/master/samples/vk10-kepler/DedicatedAllocationVk), documentation (http://nvidiagameworks.github.io/GraphicsSamples/DedicatedAllocationVulkanSample.htm)) and shows how to use the extension to bind an image to a dedicated allocation and use it as a render target which then gets blitted to the screen.


NVIDIA GameWorks Graphics Sample - DedicatedAllocationVk
DedicatedAllocationVk
Render Mode: dedicated allocations
Draw Geometry
2368.5