AGENDA

- Heaps & Types
- Tips & Tricks
- The VMA Library
- Conclusion
EFFORT VS. CONTROL

- Achieve smaller footprint
- Better optimize for specific platforms
- Aliasing!

Legacy APIs
Heaps & Types
YOUR MISSION, JIM...

Create a resource → Resource → Figure out what type, alignment and size → Allocate memory → Memory → Bind
**ALLOCATING SOME MEMORY**

```c
VkResult vkAllocateMemory(
    VkDevice device,
    const VkMemoryAllocateInfo* pAllocateInfo,
    const VkAllocationCallbacks* pAllocator,
    VkDeviceMemory* pMemory);
```

```c
typedef struct VkMemoryAllocateInfo {
    VkStructureType sType;
    const void* pNext;
    VkDeviceSize allocationSize;
    uint32_t memoryTypeIndex;
} VkMemoryAllocateInfo;
```
VkResult vkAllocateMemory(
    VkDevice device,
    const VkMemoryAllocateInfo* pAllocateInfo,
    const VkAllocationCallbacks* pAllocator,
    VkDeviceMemory* pMemory);
vkGetPhysicalDeviceMemoryProperties(
    VkPhysicalDevice physicalDevice,
    VkPhysicalDeviceMemoryProperties* pMemoryProperties);

typedef struct VkPhysicalDeviceMemoryProperties {
    uint32_t memoryTypeCount;
    VkMemoryType memoryTypes[VK_MAX_MEMORY_TYPES];
    uint32_t memoryHeapCount;
    VkMemoryHeap memoryHeaps[VK_MAX_MEMORY_HEAPS];
} VkPhysicalDeviceMemoryProperties;
MEMORY TYPES VS. HEAPS
(AMD RX VEGA 64)

Heap 0 -> Type 0

Heap 1

Heap 2 -> Type 2

Type 0

Type 1

Type 3
MEMORY TYPES CHEAT SHEET
(AMD RX VEGA 64)

Maps to
VK_MEMORYPROPERTYDEVICELOCAL_BIT in VkMemoryPropertyFlagBits.

- Storage  - Visible  - Cached

- Fast  - Slow

Most of VRAM
Fixed 256MiB

Size
## MEMORY TYPES CHEAT SHEET
**(AMD RX VEGA 64)**

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Storage</th>
<th>Visible</th>
<th>Cached</th>
<th>Size</th>
<th>Most of VRAM</th>
<th>Fixed 256MiB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>1</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>2</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>3</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Maps to `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT` and `VK_MEMORY_PROPERTY_CACHED_BIT` respectively.
**MEMORY TYPES CHEAT SHEET**
*(AMD RX VEGA 64)*

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Storage</th>
<th>Visible</th>
<th>Cached</th>
<th>Read</th>
<th>Write</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>Most of VRAM</td>
</tr>
<tr>
<td>1</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>Fixed 256MiB</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

- **Storage**
- **Visible**
- **Cached**
- **Fast**
- **Slow**
Okay, not *that* bad since we benefit from GPU caches, but certainly worse than just reading from `DEVICE_LOCAL`. 
### MEMORY TYPES CHEAT SHEET

**AMD RX VEGA 64**

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Storage</th>
<th>Visible</th>
<th>Cached</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>✅️</td>
<td>✅️</td>
<td>✗️</td>
</tr>
<tr>
<td>1</td>
<td>✗️</td>
<td>✅️</td>
<td>✅️</td>
</tr>
<tr>
<td>2</td>
<td>✅️</td>
<td>✅️</td>
<td>✅️</td>
</tr>
<tr>
<td>3</td>
<td>✗️</td>
<td>✅️</td>
<td>✅️</td>
</tr>
</tbody>
</table>

- **Storage**: Most of VRAM
- **Visible**: Fixed 256MiB

On current GPUs & drivers, PC Windows® everything that is HOST_VISIBILE is also marked COHERENT.

On other architectures you may need: vkInvalidateMappedMemoryRanges before reads and vkFlushMappedMemoryRanges after writes.

BEWARE: Unmapping won’t do this for you!
## MEMORY TYPES VS. HEAPS
**(AMD RX VEGA 64)**

<table>
<thead>
<tr>
<th>Heap 0</th>
<th>Heap 1</th>
<th>Heap 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type 0</strong></td>
<td><strong>Type 1</strong></td>
<td><strong>Type 2</strong></td>
</tr>
<tr>
<td><strong>Type 3</strong></td>
<td><strong>Usage Notes</strong></td>
<td><strong>Usage Notes</strong></td>
</tr>
<tr>
<td>Use for “upload-once” resources, render targets, depth buffers, etc.</td>
<td>Staging for uploads to VRAM.</td>
<td>Written by GPU, read by CPU. E.g.: Downsampling luminance.</td>
</tr>
<tr>
<td>Fixed size of 256MiB. Use for dynamic, write once, read once resources.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
WARNING!

- Not a good idea to hardcode the memory type indices.
  - Driver may change in future
  - May be different on other/newer hardware
  - Magic numbers
- Query the info using vkGetPhysicalDeviceMemoryProperties.
- Map to engine-specific enums.
Would you make a unique allocation from the heap for each structure in a C/C++ program?
Would you make a unique allocation from the heap for each structure in a C/C++ program?
ALLOCATE STRATEGY

Same idea on GPU for similar reasons:
- Fragmentation
- Performance
- Data locality
- Personal sanity

Allocate reasonably large chunks of memory (256MiB).
- Just 16 allocations fills 4GiB of VRAM.
- Good balance between flexibility and performance.
  - On Windows®7 Vulkan memory allocations have larger overhead.

Sub-allocate the memory for resources from these blocks.
ALLOCATION STRATEGY

- Linear allocator
- Stack allocator
- Double stack allocator
- Block allocator
- Ring buffer

- Used memory
- Free memory
What happens when you run out of DEVICE_LOCAL memory (VRAM) and try vkAllocateMemory?
What happens when you run out of DEVICE_LOCAL memory (VRAM) and try vkAllocateMemory?
OVER-SUBSCRIPTION
VK_ERROR_OUT_OF_DEVICE_MEMORY

- Allocation fails.
- Application must handle out-of-memory conditions.
- Out-of-memory potentially changes per driver/hardware.
Allocation succeeds.

Some blocks are silently migrated to system memory.

Why would you want this?

– Useful for development purposes - Artists don’t always stick to budgets.
– Some of your blocks might get paged anyway (you’re not alone on the machine).
– Application doesn’t have to handle out-of-memory.

Accessing blocks migrated to system memory can degrade GPU performance.
No way is exposed to control residency manually.

No way is exposed to query the used/free memory.

To make things worse, there are other implicit resources which need memory too:
- Swap chains
- Command buffers
- Descriptors
- Shaders / PSOs
- Query results

Use `VkMemoryHeap::size` then apply some “informed adjustments”:

<table>
<thead>
<tr>
<th>Flags</th>
<th>Hack</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVICE_LOCAL</td>
<td><code>VkMemoryHeap::size * 0.8f</code></td>
</tr>
<tr>
<td>DEVICE_LOCAL</td>
<td>HOST_VISIBLE</td>
</tr>
</tbody>
</table>
As resolutions get larger, render targets follow suit.

As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.
As resolutions get larger, render targets follow suit.

As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.

Once lighting is done, we don’t need the g-buffer anymore. Let’s use it!
As resolutions get larger, render targets follow suit.

As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.

---

Post processing often does a lot of ping-ponging of RTs. Why not use again here?
As resolutions get larger, render targets follow suit.

As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.
As resolutions get larger, render targets follow suit.

As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.

For second, third, etc. use of aliased resource best to assume it contains garbage.
As resolutions get larger, render targets follow suit.

As many resources are transient, aliasing can be a solution to keep render target/UAV memory in check.

For second, third, etc. use of aliased resource best to assume it contains garbage.

>50% memory saved in some titles [ODonnell17].
Which queue should you use to copy a resource from host memory to DEVICE_LOCAL memory?

- VK_QUEUE_TRANSFER_BIT
- VK_QUEUE_COMPUTE_BIT
Which queue should you use to copy a resource from host memory to DEVICE_LOCAL memory?

- VK_QUEUE_TRANSFER_BIT
- VK_QUEUE_COMPUTE_BIT
The transfer queue is great.
- DMA hardware that you drive asynchronously – doesn’t touch other queues.
- The fastest way to copy across PCIe bus.

Kick DMAs off as early as you can, waaaaaay before you need them on graphics/compute queue.

Some hardware even has 2 transfer queues.
- E.g. RX 580
What about DEVICE_LOCAL to DEVICE_LOCAL copies?
- Choice not as clear cut.
- Peak transfer rates of Graphics/Compute are probably faster, but it “clogs up” GPU.
- If you can pipeline the copies, transfer queue can still be a win.

Use the queue to defrag your allocations.
- Copy to a new address.
- Next frame, update descriptor.

General rules of thumb:
- Need it now? Graphics/Compute queue.
- It can wait? Transfer queue.
- Respect granularity of the queue. Full sub-resource is fine.
- Measure, measure, measure. Queue semaphores can cost you.
Having your entire memory block persistently mapped is generally OK.
- No longer any need to unmap before using stuff on GPU!

Exceptions:
- AMD, Windows® version < 10. Blocks of DEVICE_LOCAL that are also HOST_VISIBLE (type 2) that are still mapped at the time of Submit or Present will be migrated to system memory.
- Keeping many large memory blocks mapped may impact stability/performance of debugging tools.
Avoid the following Vulkan ‘lazy-mode’ features:

- **VK_IMAGE_LAYOUT_GENERAL**
  - Prefer to transition to appropriate \texttt{VK_IMAGE_LAYOUT_*OPTIMAL} state.

- **VK_SHARING_MODE_CONCURRENT** on render targets or depth buffers
  - It nobs DCC compression.
  - Go for \texttt{VK_SHARING_MODE_EXCLUSIVE} and do explicit queue family ownership barriers.

- **VK_IMAGE_TILING_LINEAR**
  - \texttt{VK_IMAGE_TILING_OPTIMAL} is more... well... optimal.
  - You can always copy to a buffer to de-tile things.

- Setting too many usage bits on your stuff.
  - Great way to confuse drivers into flushing more caches, and draining the GPU.
Querying the size required for two identical resources always returns the exact same size?
Querying the size required for two identical resources always returns the exact same size?
Memory requirements - e.g. size - can vary for different, similar resources.
- Same format, width, height and mip-levels.

Not just the preserve of ‘spec wonks’. This *really* happens - don’t cache the results when querying sizes.

Make sure you query *each* resource for it’s specific requirements.
Mixing large and small allocations carelessly can be painful.

Consider routing allocations to different blocks of memory based on their sizes.

Pool larger allocations together in one block, and smaller allocations together in another one.
The Vulkan Memory Allocator
VULKAN MEMORY ALLOCATOR (VMA)

  - https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator
- Simple, C99 interface. Same style as Vulkan™.
- Battle tested, already getting some love in the community.
VULKAN MEMORY ALLOCATOR (VMA)

- Function that help to choose the correct and optimal memory type based on intended usage.
- Functions that allocate memory blocks, reserve and return parts of them to the user.
- Allocation tracker, look at used/unused, and fragmentation.
- Respects alignment and buffer/image granularity.

```cpp
VkBufferCreateInfo bufferInfo = { VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO };
bufferInfo.size = 65536;
bufferInfo.usage = VK_BUFFER_USAGE_VERTEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT;

VmaAllocationCreateInfo allocInfo = {};
allocInfo.usage = VMA_MEMORY_USAGE_GPU_ONLY;

VkBuffer buffer;
VmaAllocation allocation;
vmaCreateBuffer(allocator, &bufferInfo, &allocInfo, &buffer, &allocation, NULL);
```
VULKAN MEMORY ALLOCATOR (VMA)

- Even has some tooling!
- VMA can dump allocator state to JSON.
- Python script generates PNG file which shows the allocator contents.
CONCLUSION

- Vulkan is lower-level and requires explicit memory management.
  - Creating resources is a multi-stage process.
  - Former driver magic is now under your control.

- You need to deal with differences between GPUs.

- By following good practices you can achieve optimal performance on any GPU.

- Vulkan Memory Allocator (VMA) is battle-tested and can really help a lot.
SPECIAL THANKS

Adam Sawicki
Dominik Baumeister
Lou Kramer
Matthäus G. Chajdas
Rys Sommefeldt
Timothy Lottes
Nicolas Thibiero

Alon Or-Bach
[O'Donnell17]
Yuriy O’Donnell – FrameGraph: Extensible Rendering Architecture in Frostbite
https://www.gdcvault.com/play/1024612/FrameGraph-Extensible-Rendering-Architecture-in
DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2018 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.