The Vulkan Memory Allocator (VMA)

Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu

Oregon State University
Computer Graphics

The Vulkan Memory Allocator (VMA)

The **Vulkan Memory Allocator**, developed by AMD, is a set of functions to simplify your view of allocating buffer memory. It is all included in our class VMA sample code, but if you want to go get it for yourself, the github link is:

https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator

This repository includes a smattering of documentation and sample programs.

Vulkan Memory Allocator Setup

Do this in just one .cpp file:

```
#define VMA_IMPLEMENTATION
#define VMA_VULKAN_VERSION  1001000          // if vulkan version 1.1
#include "vk_mem_alloc.h"
```

Do this in all the other .cpp files:

```
#define VMA_VULKAN_VERSION  1001000          // if vulkan version 1.1
#include "vk_mem_alloc.h"
```

Do the usual Vulkan setup for:

```
VkPhysicalDevice        PhysicalDevice;
VkLogicalDevice         LogicalDevice;
VkInstance              Instance;
```

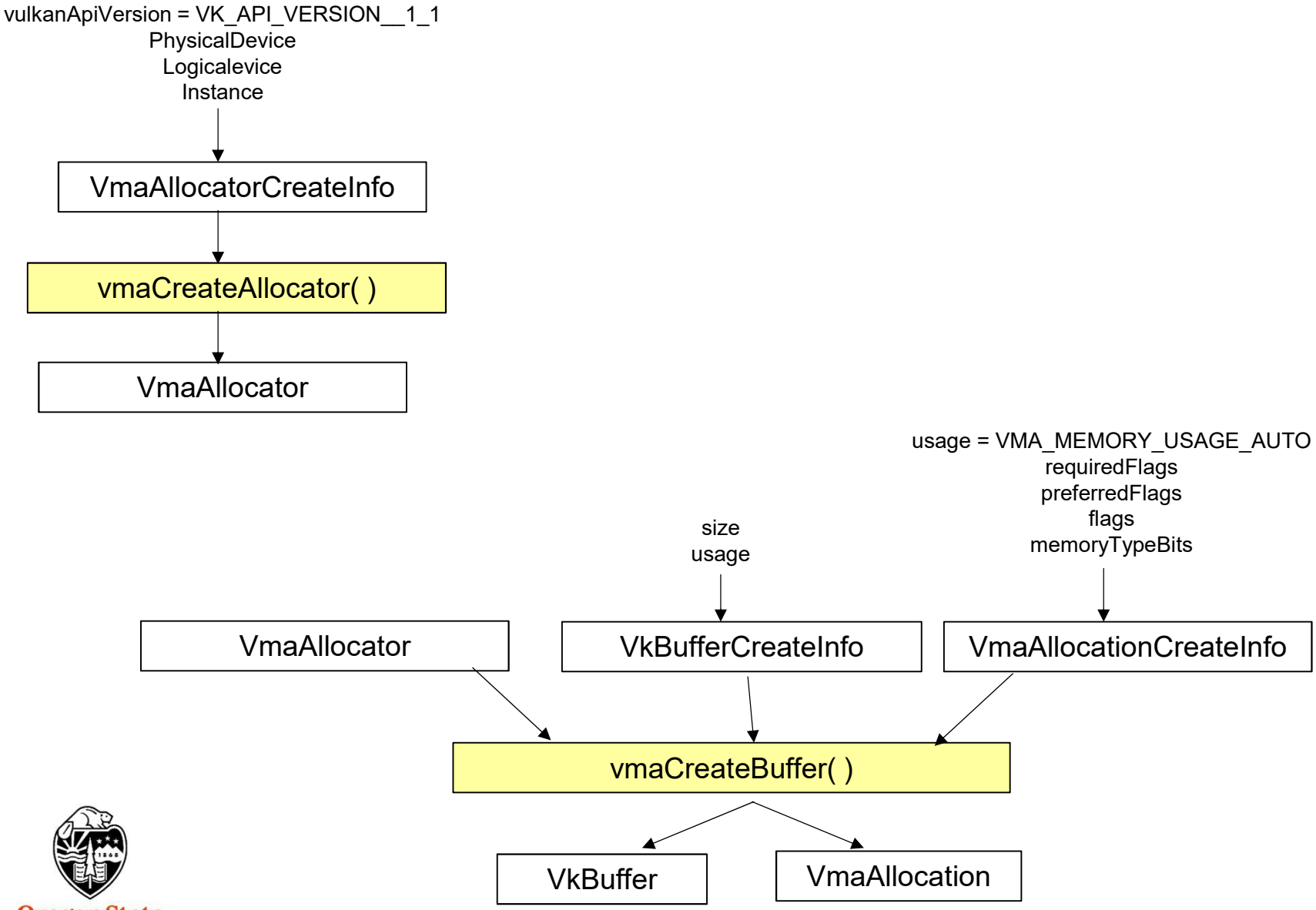
Add one new global variable for VMA:

```
VmaAllocator            Allocator;
```

Add one new global variable for each VMA allocation you do:

```
VmaAllocation           Allocation;
```

Vulkan Memory Allocator (VMA)



Create the VMA Allocator


```
VmaAllocator          Allocator;        // global

. . .

VmaAllocatorCreateInfo                  vrci;
        vrci.vulkanApiVersion = VK_API_VERSION_1_1;
        vrci.flags                    = 0;                    // VmaAllocatorCreateFlagBits enum
        vrci.physicalDevice     = PhysicalDevice;       // from usual vulkan setup
        vrci.device                   = LogicalDevice;        // from usual vulkan setup
        vrci.instance                 = Instance;             // from usual vulkan setup
        vrci.pVulkanFunctions = nullptr;


vmaCreateAllocator( IN &vrci, OUT &Allocator );
```

The Allocator acts as the keeper of the system knowledge for VMA

Oregon State University
Computer Graphics

VmaAllocatorCreateInfo .flags Bits

VMA_ALLOCATOR_CREATE_EXTERNALLY_SYNCHRONIZED_BIT
VMA_ALLOCATOR_CREATE_KHR_DEDICATED_ALLOCATION_BIT
VMA_ALLOCATOR_CREATE_KHR_BIND_MEMORY2_BIT
VMA_ALLOCATOR_CREATE_EXT_MEMORY_BUDGET_BIT
VMA_ALLOCATOR_CREATE_AMD_DEVICE_COHERENT_MEMORY_BIT
VMA_ALLOCATOR_CREATE_BUFFER_DEVICE_ADDRESS_BIT
VMA_ALLOCATOR_CREATE_EXT_MEMORY_PRIORITY_BIT

Create the Information for a Vulkan Data Buffer


```
VkBuffer  Buffer;              // or "VkDataBuffer   Buffer"

VkBufferCreateInfo  vbci;
        vbci.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
        vbci.pNext = nullptr;
        vbci.flags = 0;
        vbci.size = << buffer size in bytes >>
        vbci.usage = <<or'ed bits of: >>
                VK_USAGE_TRANSFER_SRC_BIT
                VK_USAGE_TRANSFER_DST_BIT
                VK_USAGE_UNIFORM_TEXEL_BUFFER_BIT
                VK_USAGE_STORAGE_TEXEL_BUFFER_BIT
                VK_USAGE_UNIFORM_BUFFER_BIT
                VK_USAGE_STORAGE_BUFFER_BIT
                VK_USAGE_INDEX_BUFFER_BIT
                VK_USAGE_VERTEX_BUFFER_BIT
                VK_USAGE_INDIRECT_BUFFER_BIT
        vbci.sharingMode = << one of: >>
                VK_SHARING_MODE_EXCLUSIVE
                VK_SHARING_MODE_CONCURRENT
        vbci.queueFamilyIndexCount = 0;
        vbci.pQueueFamilyIndices = (const iont32_t) nullptr;

// DO NOT CREATE THE BUFFER – LET VMA DO IT!
// result = vkCreateBuffer( LogicalDevice, IN &vbci, PALLOCATOR,  OUT &Buffer );
```

"or" these bits together to specify how this buffer will be used

Creating the Buffer


```
#include “vk_mem_alloc.h”
. . .
VkBuffer                        Buffer;         // global
. . .

VmaAllocationCreateInfo         vaci;
        vaci.usage          = VMA_MEMORY_USAGE_AUTO;    // select what it thinks is the best type (recommended)
        vaci.requiredFlags  = VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT;
        vaci.preferredFlags = VK_MEMORY_PROPERTY_HOST_COHERENT_BIT | VK_MEMORY_PROPERTY_HOST_CACHED_BIT;
        vaci.flags          = VMA_ALLOCATION_CREATE_HOST_ACCESS_SEQUENTIAL_WRITE_BIT;

vmaCreateBuffer( IN Allocator, IN &vbci, IN &vaci, OUT &Buffer. OUT &Allocation, nullptr );
```

Both allocates and binds in one call

The Allocation acts as the keeper of this specific buffer knowledge for VMA

```
vmaDestroyBuffer( Allocator, Buffer. Allocation );

vmaDestroyAllocator( Allocator );
```

VmaAllocationCreateInfo Values for .usage

VMA_MEMORY_USAGE_UNKNOWN
VMA_MEMORY_USAGE_GPU_ONLY
VMA_MEMORY_USAGE_CPU_ONLY
VMA_MEMORY_USAGE_CPU_TO_GPU
VMA_MEMORY_USAGE_GPU_TO_CPU
VMA_MEMORY_USAGE_CPU_COPY
VMA_MEMORY_USAGE_GPU_LAZILY_ALLOCATED
VMA_MEMORY_USAGE_AUTO
VMA_MEMORY_USAGE_AUTO_PREFER_DEVICE
VMA_MEMORY_USAGE_AUTO_PREFER_HOST

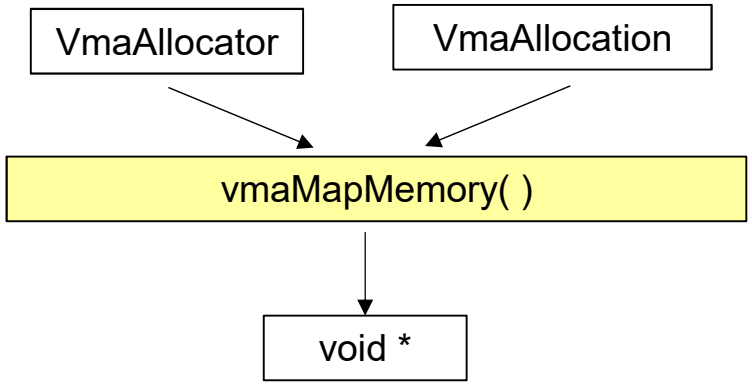
VmaAllocationCreateInfo Bit Values for .flags

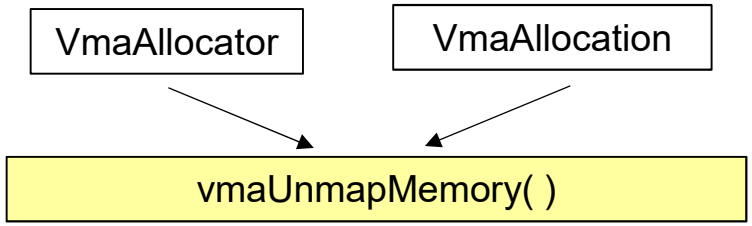
VMA_ALLOCATION_CREATE_DEDICATED_MEMORY_BIT
VMA_ALLOCATION_CREATE_NEVER_ALLOCATE_BIT
VMA_ALLOCATION_CREATE_MAPPED_BIT
VMA_ALLOCATION_CREATE_USER_DATA_COPY_STRING_BIT
VMA_ALLOCATION_CREATE_UPPER_ADDRESS_BIT
VMA_ALLOCATION_CREATE_DONT_BIND_BIT
VMA_ALLOCATION_CREATE_WITHIN_BUDGET_BIT
VMA_ALLOCATION_CREATE_CAN_ALIAS_BIT
VMA_ALLOCATION_CREATE_HOST_ACCESS_SEQUENTIAL_WRITE_BIT
VMA_ALLOCATION_CREATE_HOST_ACCESS_RANDOM_BIT
VMA_ALLOCATION_CREATE_HOST_ACCESS_ALLOW_TRANSFER_INSTEAD_BIT
VMA_ALLOCATION_CREATE_STRATEGY_MIN_MEMORY_BIT
VMA_ALLOCATION_CREATE_STRATEGY_MIN_TIME_BIT
VMA_ALLOCATION_CREATE_STRATEGY_MIN_OFFSET_BIT

Memory Property Flag Bits for .requiredFlags and .preferredFlags

VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT
VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT
VK_MEMORY_PROPERTY_HOST_COHERENT_BIT
VK_MEMORY_PROPERTY_HOST_CACHED_BIT
VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT
VK_MEMORY_PROPERTY_PROTECTED_BIT
VK_MEMORY_PROPERTY_DEVICE_COHERENT_BIT_AMD
VK_MEMORY_PROPERTY_DEVICE_UNCACHED_BIT_AMD
VK_MEMORY_PROPERTY_RDMA_CAPABLE_BIT_NV

VMA Memory-Mapped I/O





VMA Memory-Mapped I/O, Example I


```
void *mappedDataAddr;

vmaMapMemory( Allocator, Allocation, OUT &mappedDataAddr );

        memcpy( mappedDataAddr, &VertexData, sizeof(VertexData) );

vmaUnmapMemory( Allocator, Allocation );
```

Mamory-Mapped I/O, Example II


```
struct vertex *vp;

vmaMapMemory( Allocator, Allocation, OUT (void *)&vp );

for( int i = 0; i < numTrianglesInObjFile; i++ )        // number of triangles
{
        for( int j = 0; j < 3; j++ )                    // 3 vertices per triangle
        {
                vp->position = glm::vec3( . . . );
                vp->normal = glm::vec3( . . . );
                vp->color = glm::vec3( . . . );
                vp->texCoord = glm::vec2( . . . );
                vp++;
        }
}

vmaUnmapMemory( Allocator, Allocation );
```