Physical Devices

Vulkan: Overall Block Diagram

Application

Instance

Physical Device

Logical Device

Queue

Command Buffer

Logical Device

Queue

Command Buffer

Logical Device

Queue

Command Buffer
Vulkan: a More Typical (and Simplified) Block Diagram

Application

Instance

Physical Device

Logical Device

Queue

Command Buffer

Command Buffer

Command Buffer

uint32_t count;
result = vkEnumeratePhysicalDevices( Instance, OUT &count, OUT (VkPhysicalDevice *)&nullptr );

VkPhysicalDevice * physicalDevices = new VkPhysicalDevice[count];
result = vkEnumeratePhysicalDevices( Instance, OUT &count, OUT physicalDevices );

Querying the Number of Physical Devices

This way of querying information is a recurring OpenCL and Vulkan pattern (get used to it):

result = vkEnumeratePhysicalDevices( Instance, &count, nullptr );
result = vkEnumeratePhysicalDevices( Instance, &count, physicalDevices );
VkResult result = VK_SUCCESS;
result = vkEnumeratePhysicalDevices( Instance, OUT &PhysicalDeviceCount, (VkPhysicalDevice *)nullptr );
if( result != VK_SUCCESS || PhysicalDeviceCount <= 0 )
{
    fprintf( FpDebug, "Could not count the physical devices\n" );
    return VK_SHOULD_EXIT;
}

fprintf(FpDebug, "\n%d physical devices found.\n", PhysicalDeviceCount);
VkPhysicalDevice * physicalDevices = new VkPhysicalDevice[ PhysicalDeviceCount ];
result = vkEnumeratePhysicalDevices( Instance, OUT &PhysicalDeviceCount, OUT physicalDevices );
if( result != VK_SUCCESS )
{
    fprintf( FpDebug, "Could not enumerate the %d physical devices\n", PhysicalDeviceCount );
    return VK_SHOULD_EXIT;
}

Vulkan: Identifying the Physical Devices

int discreteSelect = -1;
int integratedSelect = -1;
for( unsigned int i = 0; i < PhysicalDeviceCount; i++ )
{
    VkPhysicalDeviceProperties vpdp;
    vkGetPhysicalDeviceProperties( IN physicalDevices[i], OUT &vpdp );
    if( result != VK_SUCCESS )
    {
        fprintf( FpDebug, "Could not get the physical device properties of device %d,\n", i );
        return VK_SHOULD_EXIT;
    }
    fprintf( FpDebug, "\n
Device %2d:\n", i );
    fprintf( FpDebug, "API version: %d\n", vpdp.apiVersion );
    fprintf( FpDebug, "Driver version: %d\n", vpdp.apiVersion );
    fprintf( FpDebug, "Vendor ID: 0x%04x\n", vpdp.vendorID );
    fprintf( FpDebug, "Device ID: 0x%04x\n", vpdp.deviceID );
    fprintf( FpDebug, "Physical Device Type: %d =\n", vpdp.deviceType );
    if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_DISCRETE_GPU )    fprintf( FpDebug, " (Discrete GPU)\n" );
    if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU ) fprintf( FpDebug, " (Integrated GPU)\n" );
    if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_VIRTUAL_GPU )    fprintf( FpDebug, " (Virtual GPU)\n" );
    if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_CPU )            fprintf( FpDebug, " (CPU)\n" );
    fprintf( FpDebug, "Device Name: %s\n", vpdp.deviceName );
    fprintf( FpDebug, "Pipeline Cache Size: %d\n", vpdp.pipelineCacheSize );
}

Which Physical Device to Use, I
// need some logical here to decide which physical device to select:

if (vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_DISCRETE_GPU)
    discreteSelect = i;
else if (vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU)
    integratedSelect = i;

int which = -1;
if (discreteSelect >= 0)
    which = discreteSelect;
else if (integratedSelect >= 0)
    which = integratedSelect;
else
    fprintf(FpDebug, "Could not select a Physical Device\n");
    return VK_SHOULD_EXIT;

VkPhysicalDeviceProperties PhysicalDeviceFeatures;
vkGetPhysicalDeviceProperties( IN PhysicalDevice, OUT &PhysicalDeviceFeatures );

fprintf(FpDebug, "nPhysical Device Features:\n");
fprintf(FpDebug, "geometryShader = %2d\n", PhysicalDeviceFeatures.geometryShader);
fprintf(FpDebug, "tessellationShader = %2d\n", PhysicalDeviceFeatures.tessellationShader);
fprintf(FpDebug, "multiDrawIndirect = %2d\n", PhysicalDeviceFeatures.multiDrawIndirect);
fprintf(FpDebug, "wideLines = %2d\n", PhysicalDeviceFeatures.wideLines);
fprintf(FpDebug, "largePoints = %2d\n", PhysicalDeviceFeatures.largePoints);
fprintf(FpDebug, "multiViewport = %2d\n", PhysicalDeviceFeatures.multiViewport);
fprintf(FpDebug, "occlusionQueryPrecise = %2d\n", PhysicalDeviceFeatures.occlusionQueryPrecise);
fprintf(FpDebug, "pipelineStatisticsQuery = %2d\n", PhysicalDeviceFeatures.pipelineStatisticsQuery);
fprintf(FpDebug, "shaderFloat64 = %2d\n", PhysicalDeviceFeatures.shaderFloat64);
fprintf(FpDebug, "shaderInt64 = %2d\n", PhysicalDeviceFeatures.shaderInt64);
fprintf(FpDebug, "shaderInt16 = %2d\n", PhysicalDeviceFeatures.shaderInt16);
Here's What the NVIDIA RTX 2080 Ti Produced

vkEnumeratePhysicalDevices:

Device 0:
API version: 4198499
Driver version: 4198499
Vendor ID: 0x10de
Device ID: 0x1e04
Physical Device Type: 2 = (Discrete GPU)
Device Name: RTX 2080 Ti
Pipeline Cache Size: 206

Device #0 selected ('RTX 2080 Ti')

Physical Device Features:
galleryShader = 1
tessellationShader = 1
multiDrawIndirect = 1
wideLines = 1
largePoints = 1
multiViewport = 1
occlusionQueryPrecise = 1
pipelineStatisticsQuery = 1
shaderFloat64 = 1
shaderInt64 = 1
shaderInt16 = 1

Here's What the Intel HD Graphics 520 Produced

vkEnumeratePhysicalDevices:

Device 0:
API version: 4194360
Driver version: 4194360
Vendor ID: 0x8086
Device ID: 0x1916
Physical Device Type: 1 = (Integrated GPU)
Device Name: Intel(R) HD Graphics 520
Pipeline Cache Size: 213

Device #0 selected ('Intel(R) HD Graphics 520')

Physical Device Features:
galleryShader = 1
tessellationShader = 1
multiDrawIndirect = 1
wideLines = 1
largePoints = 1
multiViewport = 1
occlusionQueryPrecise = 1
pipelineStatisticsQuery = 1
shaderFloat64 = 1
shaderInt64 = 1
shaderInt16 = 1
Asking About the Physical Device’s Different Memories

```
VkPhysicalDeviceMemoryProperties vpdmp;
vkGetPhysicalDeviceMemoryProperties( PhysicalDevice, OUT &vpdmp );

fprintf( FpDebug, "%d Memory Types:
", vpdmp.memoryTypeCount );
for( unsigned int i = 0; i < vpdmp.memoryTypeCount; i++ )
{
    VkMemoryType vmt = vpdmp.memoryTypes[i];
    fprintf( FpDebug, "Memory %2d: ", i );
    if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT       ) != 0 )    fprintf( FpDebug, " DeviceLocal" );
    if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT      ) != 0 )    fprintf( FpDebug, " HostVisible" );
    if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_HOST_COHERENT_BIT      ) != 0 )    fprintf( FpDebug, " HostCoherent" );
    if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_HOST_CACHED_BIT        ) != 0 )    fprintf( FpDebug, " HostCached" );
    if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT   ) != 0 )    fprintf( FpDebug, " LazilyAllocated" );
    fprintf(FpDebug, "\n\n" );
}

fprintf( FpDebug, "%d Memory Heaps:
", vpdmp.memoryHeapCount );
for( unsigned int i = 0; i < vpdmp.memoryHeapCount; i++ )
{
    VkMemoryHeap vmh = vpdmp.memoryHeaps[i];
    fprintf(FpDebug, "Heap %d: ", i);
    if( ( vmh.flags & VK_MEMORY_HEAP_DEVICE_LOCAL_BIT ) != 0 ) fprintf(FpDebug, " DeviceLocal" );
    if( ( vmh.flags & VK_MEMORY_HEAP_HOST_VISIBLE_BIT ) != 0 ) fprintf(FpDebug, " HostVisible" );
    if( ( vmh.flags & VK_MEMORY_HEAP_HOST_COHERENT_BIT ) != 0 ) fprintf(FpDebug, " HostCoherent" );
    if( ( vmh.flags & VK_MEMORY_HEAP_HOST_CACHED_BIT ) != 0 ) fprintf(FpDebug, " HostCached" );
    fprintf(FpDebug, "size = 0x%08lx", (unsigned long int)vmh.size);
    if( ( vmh.flags & VK_MEMORY_HEAP_DEVICE_LOCAL_BIT ) != 0 ) fprintf(FpDebug, " DeviceLocal" );
    fprintf(FpDebug, "\n\n" );
}
```

Here’s What I Got

```
11 Memory Types:
Memory 0:  DeviceLocal
Memory 1:  DeviceLocal
Memory 2:  HostVisible HostCoherent
Memory 3:  HostVisible HostCoherent HostCached
Memory 4:  
Memory 5:  
Memory 6:  
Memory 7:  
Memory 8:  
Memory 9:  
Memory 10:  

2 Memory Heaps:
Heap 0:  size = 0xb7c000000 DeviceLocal
Heap 1:  size = 0x09000000
```
uint32_t count = -1;
vkGetPhysicalDeviceQueueFamilyProperties(IN PhysicalDevice, &count, OUT (VkQueueFamilyProperties *)nullptr);
fprintf(FpDebug, "Found %d Queue Families:\n", count);
VkQueueFamilyProperties *vqfp = new VkQueueFamilyProperties[count];
vkGetPhysicalDeviceQueueFamilyProperties(IN PhysicalDevice, &count, OUT vqfp);
for(unsigned int i = 0; i < count; i++) {
fprintf(FpDebug, "\t%d: queueCount = %2d  ;   ", i, vqfp[i].queueCount);
if((vqfp[i].queueFlags & VK_QUEUE_GRAPHICS_BIT) != 0) fprintf(FpDebug, " Graphics”);
if((vqfp[i].queueFlags & VK_QUEUE_COMPUTE_BIT) != 0) fprintf(FpDebug, " Compute ”);
if((vqfp[i].queueFlags & VK_QUEUE_TRANSFER_BIT) != 0) fprintf(FpDebug, " Transfer”);
fprintf(FpDebug, "\n”);
}

Here’s What I Got

Found 3 Queue Families:
0: queueCount = 16 ;  Graphics Compute Transfer
1: queueCount = 2 ;  Transfer
2: queueCount = 8 ;  Compute