Vulkan: Physical Devices

Application

Instance

Physical Device

Logical Device

Command Buffer

Querying the Number of Physical Devices

```c
uint32_t count;
result = vkEnumeratePhysicalDevices( Instance, OUT &count, OUT (VkPhysicalDevice *)nullptr );
VkPhysicalDevice * physicalDevices = new VkPhysicalDevice[ count ];
result = vkEnumeratePhysicalDevices( Instance, OUT &count, OUT physicalDevices );
```

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
" );
    return VK_SHOULD_EXIT;
}
fprintf(FpDebug, "
%d physical devices found.
", 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
", PhysicalDeviceCount );
    return VK_SHOULD_EXIT;
}
VkPhysicalDeviceProperties PhysicalDeviceFeatures;
vkGetPhysicalDeviceFeatures( IN physicalDevices[i], OUT &PhysicalDeviceFeatures );
if( result != VK_SUCCESS )
{
    fprintf( FpDebug, "Could not get the physical device properties of device %d
", i );
    return VK_SHOULD_EXIT;
}
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: 208
Device #0 selected ('RTX 2080 Ti')

  Physical Device Features:
    geometryShader = 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:
    geometryShader = 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 vpdm;</n>
VkGetPhysicalDeviceMemoryProperties(PhysicalDevice, OUT &vpdm);

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" );
}

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" );
  fprintf(FpDebug, "\n" );
}
```

Here’s What I Got

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

2 Memory Heaps:
  Heap 0: size = 0x0f000000 DeviceLocal
  Heap 1: size = 0x0fac00000
```
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, "\%d: queueCount = %d  ;   ", 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