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

```c
result = vkEnumeratePhysicalDevices( Instance, &count, nullptr );
result = vkEnumeratePhysicalDevices( Instance, &count, physicalDevices );
```
Vulkan: Identifying the Physical Devices

```c
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;
}
```
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, "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, "Device Name: %s\n", vpdp.deviceName );
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, "Pipeline Cache Size: %d\n", vpdp.pipelineCacheUUID[0] );
Asking About the Physical Device’s Features

VkPhysicalDeviceProperties PhysicalDeviceFeatures;
vkGetPhysicalDeviceFeatures(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 1080ti Produced

vkEnumeratePhysicalDevices:

Device 0:
  API version: 4194360  
  Driver version: 4194360  
  Vendor ID: 0x10de  
  Device ID: 0x1b06  
  Physical Device Type: 2 = (Discrete GPU)  
  Device Name: GeForce GTX 1080 Ti  
  Pipeline Cache Size: 13

Device #0 selected ('GeForce GTX 1080 Ti')

Physical Device Features:
  geometryShader = 1  
  tessellationShader = 1  
  multiDrawIndirect = 1  
  wideLines = 1  
  largePoints = 1  
  multiViewport = 1  
  occlusionQueryPrecise = 1  
  pipelineStatisticsQuery = 1  
  shaderFloat64 = 1  
  shaderInt64 = 0  
  shaderInt16 = 0
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
Which Physical Device to Use, II

// need some logical here to decide which physical device to select:

if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_DISCRETE_GPU )
    discreteSelect = i;

if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU )
    integratedSelect = i;
}

int which = -1;
if( discreteSelect >= 0 )
{
    which = discreteSelect;
    PhysicalDevice = physicalDevices[which];
}
else if( integratedSelect >= 0 )
{
    which = integratedSelect;
    PhysicalDevice = physicalDevices[which];
}
else
{
    fprintf( FpDebug, "Could not select a Physical Device\n" );
    return VK_SHOULD_EXIT;
}
Asking About the Physical Device’s Different Memories

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

fprintf( FpDebug, "\n%d Memory Types:\n", 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, "\n%d Memory Heaps:\n", vpdmp.memoryHeapCount );
for( unsigned int i = 0; i < vpdmp.memoryHeapCount; i++ )
{
    fprintf(FpDebug, "Heap %d: ", i);
    VkMemoryHeap vmh = vpdmp.memoryHeaps[i];
    fprintf( FpDebug, " size = 0x%08lx", (unsigned long int)vmh.size );
    if( ( vmh.flags & VK_MEMORY_HEAP_DEVICE_LOCAL_BIT ) != 0 )    fprintf( FpDebug, " DeviceLocal" );  // only one in use
    fprintf(FpDebug, "\n");
}
```

Oregon State University
Computer Graphics
## Here’s What I Got

<table>
<thead>
<tr>
<th>Memory Types</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory 0:</td>
<td>DeviceLocal</td>
</tr>
<tr>
<td>Memory 1:</td>
<td>DeviceLocal</td>
</tr>
<tr>
<td>Memory 2:</td>
<td>HostVisible</td>
</tr>
<tr>
<td>Memory 3:</td>
<td>HostCoherent</td>
</tr>
<tr>
<td>Memory 4:</td>
<td>HostCached</td>
</tr>
<tr>
<td>Memory 5:</td>
<td></td>
</tr>
<tr>
<td>Memory 6:</td>
<td></td>
</tr>
<tr>
<td>Memory 7:</td>
<td></td>
</tr>
<tr>
<td>Memory 8:</td>
<td></td>
</tr>
<tr>
<td>Memory 9:</td>
<td></td>
</tr>
<tr>
<td>Memory 10:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory Heaps</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap 0: size = 0xb7c00000 DeviceLocal</td>
<td></td>
</tr>
<tr>
<td>Heap 1: size = 0xfac00000</td>
<td></td>
</tr>
</tbody>
</table>
uint32_t count = -1;
vkGetPhysicalDeviceQueueFamilyProperties( IN PhysicalDevice, &count, OUT (VkQueueFamilyProperties *)nullptr );
fprintf( FpDebug, "\nFound %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

<table>
<thead>
<tr>
<th>Queue Family</th>
<th>queueCount</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
<td>Graphics Compute Transfer</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Transfer</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>Compute</td>
</tr>
</tbody>
</table>