Physical Devices

Vulkan: Overall Block Diagram

Application

Instance

Physical Device

Logical Device

Queue

Instance

Physical Device

Logical Device

Queue

Physical Device

Logical Device

Queue

Logical Device

Queue

Logical Device

Queue

Logical Device

Queue

Command Buffer

Command Buffer

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

Application

Instance

Physical Device

Logical Device

Command Buffer

Command Buffer

Command Buffer

Queue

Querying the Number of Physical Devices

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\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;
}

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, "\nDevice %2d:\n", i );
fprintf( FpDebug, "\tAPI version: %d\n", vpdp.apiVersion );
fprintf( FpDebug, "\tDriver version: %d\n", vpdp.driverVersion );
fprintf( FpDebug, "\tVendor ID: 0x%04x\n", vpdp.vendorID );
fprintf( FpDebug, "\tDevice ID: 0x%04x\n", vpdp.deviceID );
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, "\tDevice Name: %s\n", vpdp.deviceName );
if( vpdp.pipelineCacheSize != 0 )
    fprintf( FpDebug, "\tPipeline Cache Size: %d\n", vpdp.pipelineCacheSize );
Which Physical Device to Use, I

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, "\nDevice %2d:\n", i );
    fprintf( FpDebug, "\tAPI version: %d\n", vpdp.apiVersion );
    fprintf( FpDebug, "\tDriver version: %d\n", vpdp.driverVersion );
    fprintf( FpDebug, "\tVendor ID: 0x%04x\n", vpdp.vendorID );
    fprintf( FpDebug, "\tDevice ID: 0x%04x\n", vpdp.deviceID );
    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, "\tDevice Name: %s\n", vpdp.deviceName );
    if( vpdp.pipelineCacheSize != 0 )
        fprintf( FpDebug, "\tPipeline Cache Size: %d\n", vpdp.pipelineCacheSize );
}
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 Features

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

fprintf(FpDebug, "nPhysical Device Features:\n");
fprintf(FpDebug, "geometryShader = %d\n", PhysicalDeviceFeatures.geometryShader);
fprintf(FpDebug, "tessellationShader = %d\n", PhysicalDeviceFeatures.tessellationShader );
fprintf(FpDebug, "multiDrawIndirect = %d\n", PhysicalDeviceFeatures.multiDrawIndirect );
fprintf(FpDebug, "largePoints = %d\n", PhysicalDeviceFeatures.largePoints );
fprintf(FpDebug, "multiViewport = %d\n", PhysicalDeviceFeatures.multiViewport );
fprintf(FpDebug, "occlusionQueryPrecise = %d\n", PhysicalDeviceFeatures.occlusionQueryPrecise );
fprintf(FpDebug, "pipelineStatisticsQuery = %d\n", PhysicalDeviceFeatures.pipelineStatisticsQuery );
fprintf(FpDebug, "shaderFloat64 = %d\n", PhysicalDeviceFeatures.shaderFloat64 );
fprintf(FpDebug, "shaderInt16 = %d\n", PhysicalDeviceFeatures.shaderInt16 );
Here’s What the NVIDIA A6000 Produced

Init03PhysicalDeviceAndGetQueueFamilyProperties

Device 0:
  API version: 4206797
  Driver version: 4206797
  Vendor ID: 0x10de
  Device ID: 0x2230
  Physical Device Type: 2 = (Discrete GPU)
  Device Name: NVIDIA RTX A6000
  Pipeline Cache Size: 72
  Device #0 selected (‘NVIDIA RTX A6000’)

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

Init03PhysicalDeviceAndGetQueueFamilyProperties

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

```c
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 %d: ", 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++ ) {
    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\n");
```

---

Here's What I Got on the A6000's

<table>
<thead>
<tr>
<th>Memory Types:</th>
<th>Memory Heaps:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory 0:</td>
<td>Heap 0: size = 0xdbb00000 DeviceLocal</td>
</tr>
<tr>
<td>Memory 1: DeviceLocal</td>
<td>Heap 1: size = 0xd504000</td>
</tr>
<tr>
<td>Memory 2: HostVisible HostCoherent</td>
<td>Heap 2: size = 0xd600000 DeviceLocal</td>
</tr>
<tr>
<td>Memory 3: HostVisible HostCoherent HostCached</td>
<td>Heap 3: size = 0x2000000 DeviceLocal</td>
</tr>
<tr>
<td>Memory 4: DeviceLocal HostVisible HostCoherent</td>
<td></td>
</tr>
<tr>
<td>Memory 5: DeviceLocal</td>
<td></td>
</tr>
</tbody>
</table>

---

Oregon State University
Computer Graphics

mjb – December 21, 2022
Asking About the Physical Device’s Queue Families

```c
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 = %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 on the A6000’s

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