There are 3 types of Queries: Occlusion, Pipeline Statistics, and Timestamp

- Vulkan requires you to first setup “Query Pools”, one for each specific type
- This indicates that Vulkan thinks that Queries are time-consuming (relatively) to setup, and thus better to set them up in program-setup than in program-runtime

**Getting Information Back from the Graphics System**

**Setting up Query Pools**

- Vulkan requires you to first setup “Query Pools”, one for each specific type
- This indicates that Vulkan thinks that Queries are time-consuming (relatively) to setup, and thus better to set them up in program-setup than in program-runtime

```c
// VkQueryPoolCreateInfo
VkQueryPoolCreateInfo vqpci;

vqpci.sType = VK_STRUCTURE_TYPE_QUERY_POOL_CREATE_INFO;

vqpci.queryType = VK_QUERY_TYPE_OCCLUSION;
// VK_QUERY_TYPE_PIPELINE_STATISTICS
// VK_QUERY_TYPE_TIMESTAMP

vqpci.queryCount = 1;

vkCreateQueryPool(LogicalDevice, &vqpci, OUT &queryPool);
```

**Resetting, Filling, and Examining a Query Pool**

```c
vkCmdResetQueryPool(CommandBuffer, queryPool, 0, 1);

vkCmdBeginQuery(CommandBuffer, queryPool, 0, VK_QUERY_CONTROL_PRECISE_BIT);

uint32_t data[DATASIZE];

vkGetQueryPoolResults(LogicalDevice, queryPool, 0, 1, DATASIZE, data, stride, flags);
```
Occlusion Queries count the number of fragments drawn between the `vkCmdBeginQuery` and the `vkCmdEndQuery` that pass both the Depth and Stencil tests. This is commonly used to see what level-of-detail should be used when drawing a complicated object.

Some hints:
- Don't draw the whole scene – just draw the object(s) you are interested in
- Don't draw the whole object – just draw a simple bounding volume at least as big as the object(s)
- Don't draw the whole bounding volume – cut away the back faces (two reasons: time and correctness)
- Don't draw the colors – just draw the depths (especially if the fragment shader is time-consuming)

```c
uint32_t fragmentCount;
result = vkGetQueryPoolResults( LogicalDevice, occlusionQueryPool, 0, 1,
sizeof(uint32_t), &fragmentCount, 0, VK_QUERY_RESULT_WAIT_BIT );
```

Pipeline Statistics Queries count how many of various things get done between the `vkCmdBeginQuery` and the `vkCmdEndQuery`.

```c
uint32_t counts[ NUM_STATS ];
result = vkGetQueryPoolResults( LogicalDevice, statisticsQueryPool, 0, 1,
NUM_STATS*sizeof(uint32_t), counts, 0, VK_QUERY_RESULT_WAIT_BIT );
```

```c
// vqpci.pipelineStatistics = or'd bits of:
// VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_VERTICES_BIT
// VK_QUERY_PIPELINE_STATISTIC_INPUT_ASSEMBLY_PRIMITIVES_BIT
// ...
// VK_QUERY_PIPELINE_STATISTIC_COMPUTE_SHADER_INVOCATIONS_BIT
// VK_QUERY_PIPELINE_STATISTIC_GEOMETRY_SHADER_INVOCATIONS_BIT
// ...
// VK_QUERY_PIPELINE_STATISTIC_COMPUTE_SHADER_INVOCATIONS_BIT
// VK_QUERY_PIPELINE_STATISTIC_INPUT_ATTACHMENT_REFERENCES_BIT
// VK_QUERY_PIPELINE_STATISTIC_INPUT_ATTACHMENT_REFERENCES_BIT
// ...
```

Timestamp Queries count how many nanoseconds of time elapsed between the `vkCmdBeginQuery` and the `vkCmdEndQuery`.

```c
uint64_t nanosecondsCount;
result = vkGetQueryPoolResults( LogicalDevice, timestampQueryPool, 0, 1,
sizeof(uint64_t), &nanosecondsCount, 0, VK_QUERY_RESULT_64_BIT | VK_QUERY_RESULT_WAIT_BIT );
```

The `vkCmdWriteTimeStamp()` function produces the time between when this function is called and when the first thing reaches the specified pipeline stage. Even though the stages are “bits”, you are supposed to only specify one of them, not “or” multiple ones together.

```c
vkCmdWriteTimeStamp( CommandBuffer, pipelineStages, timestampQueryPool, 0 );
```

```c
// VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT
// VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
// VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
// VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
// VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
// ...
// VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
// VK_PIPELINE_STAGE_HOST_BIT
```