Push Constants

In an effort to expand flexibility and retain efficiency, Vulkan provides something called Push Constants. Like the name implies, these let you push constant values out to the shaders. These are typically used for small, frequently-updated data values. There is only one push constant setup allowed per pipeline.

By “small”, Vulkan specifies that these must be at least 128 bytes in size, although they can be larger. (You can query this limit by looking at the maxPushConstantSize parameter in the VkPhysicalDeviceLimits structure. The maximum size is 256 bytes on the NVIDIA 1080ti.) Unlike uniform buffers and vertex buffers, these are not backed by memory. They are actually part of the Vulkan pipeline.

Push Constants

On the shader side, the use of push constants looks like this:

```cpp
layout( push_constant ) uniform matrix
{
  mat4 modelMatrix;
} Matrix;
```

where this is a 430-type layout.

On the application side, push constants are pushed at the shaders by binding them to the Command Buffer:

```cpp
vkCmdPushConstants( CommandBuffer, PipelineLayout, stageFlags, offset, size, pValues );
```

where:
- stage flags are or'ed bits of VK_PIPELINE_STAGE_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT, etc.
- Size is in bytes
- pValues is a void * pointer to the data

Prior to that, however, the pipeline layout needs to be told about the push constants:

```cpp
VkPushConstantRange vpcr;
  vpcr.stageFlags = VK_PIPELINE_STAGE_VERTEX_SHADER_BIT | VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
  vpcr.offset = 0;
  vpcr.size = sizeof( glm::mat4 );

VkPipelineLayoutCreateInfo vplci;
  ...
  vplci.ConstantRangeCount = 1;
  vplci.pPushConstantRanges = &vpcr;
```

Creating a Pipeline

The process of creating a pipeline in Vulkan includes setting up various states and structures. Here’s a simplified overview:

1. **Prerequisites**:
   - A Vulkan instance
   - A logical device
   - A physical device

2. **Creating a Pipeline Layout**:
   - Use `vkCreatePipelineLayout` to create a pipeline layout
   - Specify the states you want to use (e.g., vertex input, rasterization, color blend)

3. **Creating the Pipeline**:
   - Use `vkCreateGraphicsPipeline` to create the graphics pipeline
   - Specify the shader modules, layout, and other parameters
   - A pipeline handle is returned

4. **Using the Pipeline**:
   - Bind the pipeline to the graphics commands
   - Use `vkCmdBindPipeline` to set the active pipeline
   - Draw or present frames as needed

Throughout the process, errors can be handled using `vkGetPipelineCreationInfoKHR` to get creation info and `vkGetPipelineInfoKHR` to get pipeline IDs.

For a complete list of Vulkan functions related to pipelines, see the Vulkan specification documentation.