Push Constants

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License
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:

```
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
Push Constants

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

- Describe the pipeline creation process using `VkPipelineLayoutCreateInfo`
- Use `vkCreatePipelineLayout()` to create a pipeline layout
- Use `VkPipelineShaderStageCreateInfo` to define shader stages
- Use `VkPipelineVertexInputStateCreateInfo` to describe vertex input state
- Use `VkVertexInputBindingDescription` for binding descriptions
- Use `VkViewportStateCreateInfo` for viewport state
- Use `VkPipelineRasterizationStateCreateInfo` for rasterization state
- Use `VkPipelineDepthStencilStateCreateInfo` for depth stencil state
- Use `VkPipelineColorBlendStateCreateInfo` for color blend state
- Use `VkPipelineDynamicStateCreateInfo` for dynamic state
- Use `VkGraphicsPipelineCreateInfo` to create a graphics pipeline
- Use `vkCreateGraphicsPipeline()` to create a graphics pipeline
- Use `vkCreatePipelineLayout()` to create a pipeline layout
- Use `VkPipelineLayoutCreateInfo` to create a pipeline layout

Diagram:
- VkPipelineLayoutCreateInfo
- VkPipelineShaderStageCreateInfo
- VkPipelineVertexInputStateCreateInfo
- VkVertexInputBindingDescription
- VkViewportStateCreateInfo
- VkPipelineRasterizationStateCreateInfo
- VkPipelineDepthStencilStateCreateInfo
- VkPipelineColorBlendStateCreateInfo
- VkPipelineDynamicStateCreateInfo
- VkGraphicsPipelineCreateInfo
- VkPipelineLayoutCreateInfo
- VkDescriptorSet
- Push Constants
- VkSpecializationInfo
- VkShaderModule
- VkPipelineShadersStageCreateInfo
- VkPipelineInputAssemblyStateCreateInfo
- VkPipelineInputAssemblyStateCreateInfo
- VkPipelineViewportStateCreateInfo
- VkPipelineRasterizationStateCreateInfo
- VkPipelineDepthStencilStateCreateInfo
- VkPipelineColorBlendStateCreateInfo
- VkPipelineDynamicStateCreateInfo
- VkGraphicsPipelineCreateInfo
- VkCreateGraphicsPipeline()