

Vulkan.

Specialization Constants



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



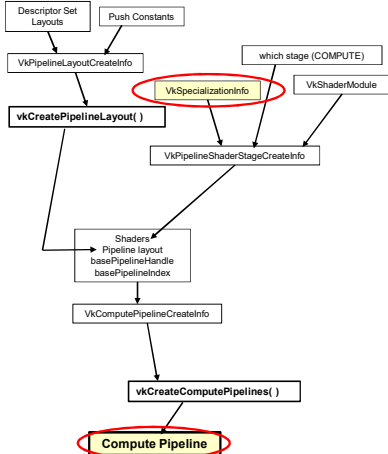
This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/)



SpecializationConstants.pptx


mjb - December 29, 2022

Remember the Compute Pipeline?



```

    graph TD
      DSL[Descriptor Set Layouts] --> VKPL[VkPipelineLayoutCreateInfo]
      PC[Push Constants] --> VKPL
      VKPL --> VKCPL[vkCreatePipelineLayout()]
      VKCPL --> SH[Shaders]
      SH --> PLS[Pipeline layout basePipelineHandle basePipelineIndex]
      PLS --> VKCPC[VkComputePipelineCreateInfo]
      VKCPC --> VKCCP[vkCreateComputePipelines()]
      VKCCP --> CP[Compute Pipeline]
      
      VKCPL --> VKSI[VkSpecializationInfo]
      VKSI --> VKCPC
      
      VKSI --> WSC[which stage COMPUTE]
      WSC --> VKSS[VkPipelineShaderStageCreateInfo]
      VKSS --> VKCPC
      
      VKSI --> VKSM[VkShaderModule]
      VKSM --> VKSS
  
```



mjb - December 29, 2022

What Are Specialization Constants?

In Vulkan, all shaders get half-way-compiled into SPIR-V and then the rest-of-the-way compiled by the Vulkan driver.

Normally, the half-way compile finalizes all constant values and compiles the code that uses them.

But, it would be nice every so often to have your Vulkan program sneak into the half-way-compiled binary and manipulate some constants at runtime. This is what Specialization Constants are for. A Specialization Constant is a way of injecting an integer, Boolean, uint, float, or double constant into a *halfway-compiled* version of a shader right before the *rest-of-the-way* compilation.

Specialization Constants

↓

Pipeline Shader Stage

That final compilation happens when you call **vkCreateComputePipelines()**

Without Specialization Constants, you would have to commit to a final value before the SPIR-V compile was done, which could have been a long time ago

Shader Source

↓


SPIR-V Compile

↓

.spv File

↓

Final Compile




mjb - December 29, 2022

Why Do We Need Specialization Constants?

Specialization Constants could be used for:

- Setting the work-items per work-group in a compute shader
- Setting a Boolean flag and then eliminating the if-test that used it
- Setting an integer constant and then eliminating the switch-statement that looked for it
- Making a decision to unroll a for-loop because the number of passes through it are small enough
- Collapsing arithmetic expressions into a single value
- Collapsing trivial simplifications, such as adding zero or multiplying by 1



mjb - December 29, 2022

Specialization Constants are Described in the Compute Pipeline

Oregon State University Computer Graphics
mjb - December 29, 2022

Specialization Constant Example -- Setting an Array Size

In the compute shader

```
layout( constant_id = 7 ) const int ASIZE = 32;
int array[ASIZE];
```

In the Vulkan C/C++ program:

```
int asize = 64;
VkSpecializationMapEntry vsme[1]; // one array element for each // Specialization Constant
vsme[0].constantID = 7; // # bytes into the Specialization Constant // array this one item is
vsme[0].offset = 0; // size of just this Specialization Constant
vsme[0].size = sizeof(asize);
VkSpecializationInfo vsi;
vsi.mapEntryCount = 1;
vsi.pMapEntries = &vsme[0];
vsi.dataSize = sizeof(asize); // size of all the Specialization Constants together
vsi.pData = &asize; // array of all the Specialization Constants
```

Oregon State University Computer Graphics
mjb - December 29, 2022

Linking the Specialization Constants into the Compute Pipeline

```
int asize = 64;
VkSpecializationMapEntry vsme[1];
vsme[0].constantID = 7;
vsme[0].offset = 0;
vsme[0].size = sizeof(asize);

VkSpecializationInfo vsi;
vsi.mapEntryCount = 1;
vsi.pMapEntries = &vsme[0];
vsi.dataSize = sizeof(asize);
vsi.pData = &asize;

VkPipelineShaderStageCreateInfo vpssci;
vpssci.sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
vpssci.pNext = nullptr;
vpssci.flags = 0;
vpssci.stage = VK_SHADER_STAGE_COMPUTE_BIT;
vpssci.module = computeShader;
vpssci.pName = "main";
vpssci.pSpecializationInfo = &vsi;

VkComputePipelineCreateInfo vcpcci[1];
vcpcci[0].sType = VK_STRUCTURE_TYPE_COMPUTE_PIPELINE_CREATE_INFO;
vcpcci[0].pNext = nullptr;
vcpcci[0].flags = 0;
vcpcci[0].stage = vpssci;
vcpcci[0].layout = ComputePipelineLayout;
vcpcci[0].basePipelineHandle = VK_NULL_HANDLE;
vcpcci[0].basePipelineIndex = 0;

result = vkCreateComputePipelines( LogicalDevice, VK_NULL_HANDLE, 1, &vcpcci[0], PALLOCATOR, OUT &ComputePipeline );
```

Oregon State University Computer Graphics
mjb - December 29, 2022

Specialization Constant Example -- Setting Multiple Constants

In the compute shader

```
layout( constant_id = 9 ) const int a = 1;
layout( constant_id = 10 ) const int b = 2;
layout( constant_id = 11 ) const float c = 3.14;
```

In the C/C++ program:

```
struct abc { int a, int b, float c; } abc;
VkSpecializationMapEntry vsme[3];
vsme[0].constantID = 9;
vsme[0].offset = offsetof( abc, a );
vsme[0].size = sizeof(abc.a);
vsme[1].constantID = 10;
vsme[1].offset = offsetof( abc, b );
vsme[1].size = sizeof(abc.b);
vsme[2].constantID = 11;
vsme[2].offset = offsetof( abc, c );
vsme[2].size = sizeof(abc.c);

VkSpecializationInfo vsi;
vsi.mapEntryCount = 3;
vsi.pMapEntries = &vsme[0];
vsi.dataSize = sizeof(abc);
vsi.pData = &abc;
```

It's important to use sizeof() and offsetof() instead of hardcoding numbers!

Oregon State University Computer Graphics
mjb - December 29, 2022

9

Specialization Constants – Setting the Number of Work-items Per Work-Group in the Compute Shader

In the compute shader

```
layout( local_size_x_id=12 ) in;
layout( local_size_x = 32, local_size_y = 1, local_size_z = 1 ) in;
```

In the C/C++ program:

```
int numXworkItems = 64;
VkSpecializationMapEntry vsme[1];
vsme[0].constantID = 12;
vsme[0].offset = 0;
vsme[0].size = sizeof(int);
VkSpecializationInfo vsi;
vsi.mapEntryCount = 1;
vsi.pMapEntries = &vsme[0];
vsi.dataSize = sizeof(int);
vsi.pData = &numXworkItems;
```

Oregon State University
Computer Graphics

mjb - December 29, 2022