Vulkanised Webinar

Reduce Draw Time Hitching with Vulkan Graphics Pipeline Library Extension

May 18, 2022 / VIRTUAL
Improving hitching with
VK_EXT_graphics_pipeline_library

Chris Glover
May 2022
Agenda

- Who am I?
- Pipeline creation problem review
- Introducing VK_EXT_graphics_pipeline_library
- Examples
- Caveats and special considerations
- Panel & Questions
Who’s this Chris Glover person?

- Software engineer @ Google
  - I work on Stadia

- Games industry veteran
  - 15 years in games studios prior to Google
  - Primarily as a rendering programmer

- Chair of the Khronos “Improving Pipeline Creation” TSG
  - Started in May 2020 to address the pipeline creation problem

- Linked in: https://www.linkedin.com/in/gloverchris/
Problem Review
In the beginning...

- Rewind 10 years (OpenGL, DX11, etc)
- Stateful graphics APIs
  - Set, set, set all your things -> Draw
  - Easy to use, burns CPU time
    - Forces a lot of work onto the driver to manage all the state
- Explicit APIs emerged (Vulkan, DX12, etc)
  - - State tracking
  - + Command buffers
  - + Pipelines
- Explicit is good
  - Should also open up GPU optimisation opportunities
- This is going to be great!
Problem!

- Pipeline creation is expensive
  - Burns CPU time
  - Causes hitching or missed draws

- Applications create many pipelines
  - Legacy just-in-time decision making
  - Artist controlled assets, situational shaders, user generated content
    - Permutation explosion
    - API limitations (static states, render pass compatibility)

- Combined, we have a big problem
So many pipelines

- **Permutations**
  - 5 vertex formats
  - 10 vertex shaders
  - 500 fragment shaders
  - 4 output formats
  - $= 5 \times 10 \times 500 \times 4$
  - $= 100,000$ pipelines

- **More likely**
  - 20 $\times$ 5000 $\times$ 30,000 $\times$ 10
  - $= 30$ billion possible pipelines

- **Can be improved with dynamic state**
  - 5 $\times$ 500 $\times$ 10,000 $\times$ 4
  - $= 100$ million possible pipelines
So many pipelines

• Compiling the same code over and over and over again

• Previous attempts to address
  - Caching!
    - Requires high coverage and at least one run :-(
  - Dynamic state!
    - Reduces the problem, but not drastically
  - Non-trivial API impedance
  - Pre-compiling
    - Long load times
    - Not always possible
      - Do you know all your permutations up front?
Enter VK_EXT_graphics_pipeline_library

https://www.khronos.org/registry/vulkan/specs/1.3-extensions/man/html/VK_EXT_graphics_pipeline_library.html
VK_EXT_graphics_pipeline_library | Intro

- **What it is:**
  - Stage level separate compilation
  - Allows separation of vertex and fragment shader compilation
    - No longer requiring full pipeline state up front
    - Create the pieces you need up front, then cheaply assemble them at draw time
  - Plus a few other bits we’ll get to

- **What it is not:**
  - Not function level separate compilation
    - Can not compile parts of a shader and then link them
  - Not shader-only compilation
    - We still require some pipeline state
VK_EXT_graphics_pipeline_library | Overview

- Splits the pipeline into 4 stages
  - Vertex-input
  - Pre-rasterisation shader
  - Fragment shader
  - Fragment-output
- Each stage can be created independently
  - Or in arbitrary groups
- Existing pipeline state is distributed amongst the stages
  - See `VkGraphicsPipelineCreateInfo` spec for split
- Stages are explicitly linked into a final pipeline
  - See `VkGraphicsPipelineLibraryCreateInfoEXT` for interface
- Designed to facilitate pre-compilation of stages
  - $20 \times 5000 \times 30,000 \times 10 \rightarrow 20 + 5000 + 30,000 + 10$
Creating libraries

- **Reuses** vkCreateGraphicsPipelines
  - Chain VkGraphicsPipelineLibraryCreateInfoEXT to pNext
  - VkGraphicsPipelineLibraryCreateInfoEXT::flags is one or more of:
    - VERTEX_INPUT_INTERFACE_BIT_EXT
    - PRE_RASTERIZATION_SHADERS_BIT_EXT
    - FRAGMENT_SHADER_BIT_EXT
    - FRAGMENT_OUTPUT_INTERFACE_BIT_EXT
- If a bit is set, corresponding states(s) must also be set
VK_EXT_graphics_pipeline_library | Details

Linking libraries

- **Reuses** `vkCreateGraphicsPipelines`
  - Chain `VkPipelineLibraryCreateInfoKHR` to `pNext`
  - `VkPipelineLibraryCreateInfoKHR::pLibraries` points to a list of `VkPipeline` handles previously created with `VkGraphicsPipelineLibraryCreateInfoEXT`

- **Linking can generate partial pipelines**
  - Full graphics pipeline must contain all four stages
    - Unless rasterization is disabled, then only need the top half
Full State

```c
VkGraphicsPipelineCreateInfo info;
info.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
info.pNext = nullptr;
info.flags = 0;
info.stageCount = stages.size();
info.pStages = stages.data();
info.pVertexInputState = &viInfo;
info.pInputAssemblyState = &iaInfo;
info.pTessellationState = tsInfo.patchControlPoints ? &tsInfo : nullptr;
info.pViewportState = &vpInfo;
info.pRasterizationState = &rsInfo;
info.pMultisampleState = &msInfo;
info.pDepthStencilState = &dsInfo;
info.pColorBlendState = &cbInfo;
info.pDynamicState = &dyInfo;
info.layout = &layout;
info.renderPass = &renderpass;
info.subpass = 0;
info.basePipelineHandle = VK_NULL_HANDLE;
info.basePipelineIndex = -1;
```
**Vertex-input State**

```c
VkGraphicsPipelineCreateInfo info;
info.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
info.pNext = nullptr;
info.flags = 0;
info.stageCount = 0;
info.pStages = nullptr;
info.pVertexInputState = &viInfo;
info.pInputAssemblyState = &iaInfo;
info.pTessellationState = nullptr;
info.pViewportState = nullptr;
info.pRasterizationState = nullptr;
info.pMultisampleState = nullptr;
info.pDepthStencilState = nullptr;
info.pColorBlendState = nullptr;
info.pDynamicState = &dyInfo;
info.layout = nullptr;
info.renderPass = nullptr;
info.subpass = 0;
info.basePipelineHandle = VK_NULL_HANDLE;
info.basePipelineIndex = -1;
```
Pre-rasterisation State

```c
VkGraphicsPipelineCreateInfo info;
info.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
info.pNext = nullptr;
info.flags = 0;
info.stageCount = 1;
info.pStages = vertexShader; // Or tesselation, geometry, etc.
info.pVertexInputState = nullptr;
info.pInputAssemblyState = nullptr;
info.pTessellationState = tsInfo.patchControlPoints ? &tsInfo : nullptr;
info.pViewportState = &vpInfo;
info.pRasterizationState = &rsInfo;
info.pMultisampleState = nullptr;
info.pDepthStencilState = nullptr;
info.pColorBlendState = nullptr;
info.pDynamicState = &dyInfo;
info.layout = layout;
info.renderPass = renderpass;
info.subpass = 0;
info.basePipelineHandle = VK_NULL_HANDLE;
info.basePipelineIndex = -1;
```
VK_EXT_graphics_pipeline_library | Example

Post-rasterisation State

```c
VkGraphicsPipelineCreateInfo info;
info.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
info.pNext = nullptr;
info.flags = 0;
info.stageCount = 1;
info.pStages = fragmentShader
info.pVertexInputState = nullptr;
info.pInputAssemblyState = nullptr;
info.pTessellationState = nullptr;
info.pViewportState = nullptr;
info.pRasterizationState = nullptr;
info.pMultisampleState = &msInfo;
info.pDepthStencilState = &dsInfo;
info.pColorBlendState = nullptr;
info.pDynamicState = &dyInfo;
info.layout = layout;
info.renderPass = renderpass
info.subpass = 0;
info.basePipelineHandle = VK_NULL_HANDLE;
info.basePipelineIndex = -1;
```
Fragment-output State

```c
VkGraphicsPipelineCreateInfo info;

info.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
info.pNext = nullptr;
info.flags = 0;
info.stageCount = 0;
info.pStages = nullptr;
info.pVertexInputState = nullptr;
info.pInputAssemblyState = nullptr;
info.pTessellationState = nullptr;
info.pViewportState = nullptr;
info.pRasterizationState = nullptr;
info.pMultisampleState = &msInfo;
info.pDepthStencilState = nullptr;
info.pColorBlendState = &cbInfo;
info.pDynamicState = &dyInfo;
info.layout = nullptr;
info.renderPass = renderpass;
info.subpass = 0;
info.basePipelineHandle = VK_NULL_HANDLE;
info.basePipelineIndex = -1;
```
Linking

```cpp
std::array<VkPipeline, 4> stages;
stages[0] = createVertexInputPipeline(context);
stages[1] = createPreRasterizationPipeline(context);
stages[2] = createFragmentShaderPipeline(context);
stages[3] = createFragmentOutputPipeline(context);
VkPipelineLibraryCreateInfoKHR libraryInfo;
libraryInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LIBRARY_CREATE_INFO_KHR;
libraryInfo.pNext = nullptr;
libraryInfo.libraryCount = stages.size();
libraryInfo.pLibraries = stages.data();
VkGraphicsPipelineCreateInfo info;
info.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
info.pNext = &libraryInfo;
info.flags = 0;
info.stageCount = 0;
info.pStages = nullptr;
... = nullptr;
info.subpass = 0;
info.basePipelineHandle = VK_NULL_HANDLE;
info.basePipelineIndex = -1;
```
Caveats, tips, and special considerations
Performance

- There’s probably a GPU performance cost
  - Reduced cross stage optimizations
- Create optimised pipelines on a thread and swap when ready
  - Create stages using
    VK_PIPELINE_CREATE_RETAIN_LINK_TIME_OPTIMIZATION_INFO_BIT_EXT
  - Link optimised pipeline using
    VK_PIPELINE_CREATE_LINK_TIME_OPTIMIZATION_BIT_EXT
- Use a heuristic to prioritise swap
  - Number of draw calls
  - Shader complexity
  - Hard coded priority
Use independent sets

- **Use** VK_PIPELINE_LAYOUT_CREATE_INDEPENDENT_SETS_BIT_EXT
  - Separate layouts for the top and bottom halves of the pipe
  - Otherwise compiled tops and bottoms might be incompatible
Careful with Multisample State

- pMultisampleState can be omitted from the fragment stage if:
  - Sample shading is not enabled
  - No input variables are decorated with Sample
Careful with device features

VkPhysicalDeviceGraphicsPipelineLibraryPropertiesEXT

- Linking isn’t guaranteed to be fast on all hardware
  - Check graphicsPipelineLibraryFastLinking == VK_TRUE

- Interpolation decoration might need to match between Pre-rasterization and fragment stages
  - Check
    graphicsPipelineLibraryIndependentInterpolationDecoration == VK_TRUE

- Both are VK_TRUE for at least NVIDIA, AMD, and Intel
Example integration strategy

- Precompute all variants of each stage independently
  - Or most variants, maybe all your streamed content
- Change existing full pipeline creation to pipeline linking
  - Requires a way to associate state with a pipeline library
- Enqueue a background job to compile an optimised pipeline
  - Might be optional, or limited to expensive pipelines
    - Consider a heuristic
Wrapping up

- **VK_EXT_graphics_pipeline_library** provides a strong optimisation opportunity
  - But it’s not free; some work needs to be done to integrate it
    - Precompile the stages, then link at draw time. Don’t wait until draw time to compile.
- **Still room for improvement**
  - The Pipeline Creation TSG isn’t done yet. We’re working on more ideas to further improve pipeline creation in Vulkan
    - Performance, consistency, ease of use
  - Get involved!

- See Dan Ginsburg’s blog post for a case study:
Discussion & Panel

- Dan Ginsburg | Valve
- Chris Glover | Google
- Tobias Hector | AMD