

Introduction: NVIDIA Nsight Graphics

Di Lu
CIS 5650 - GPU Programming
Fall 2024



Who am I?

Name: Di Lu

DMD: May 2022

CGGT: May 2023; Took GPU in
Fall 2022

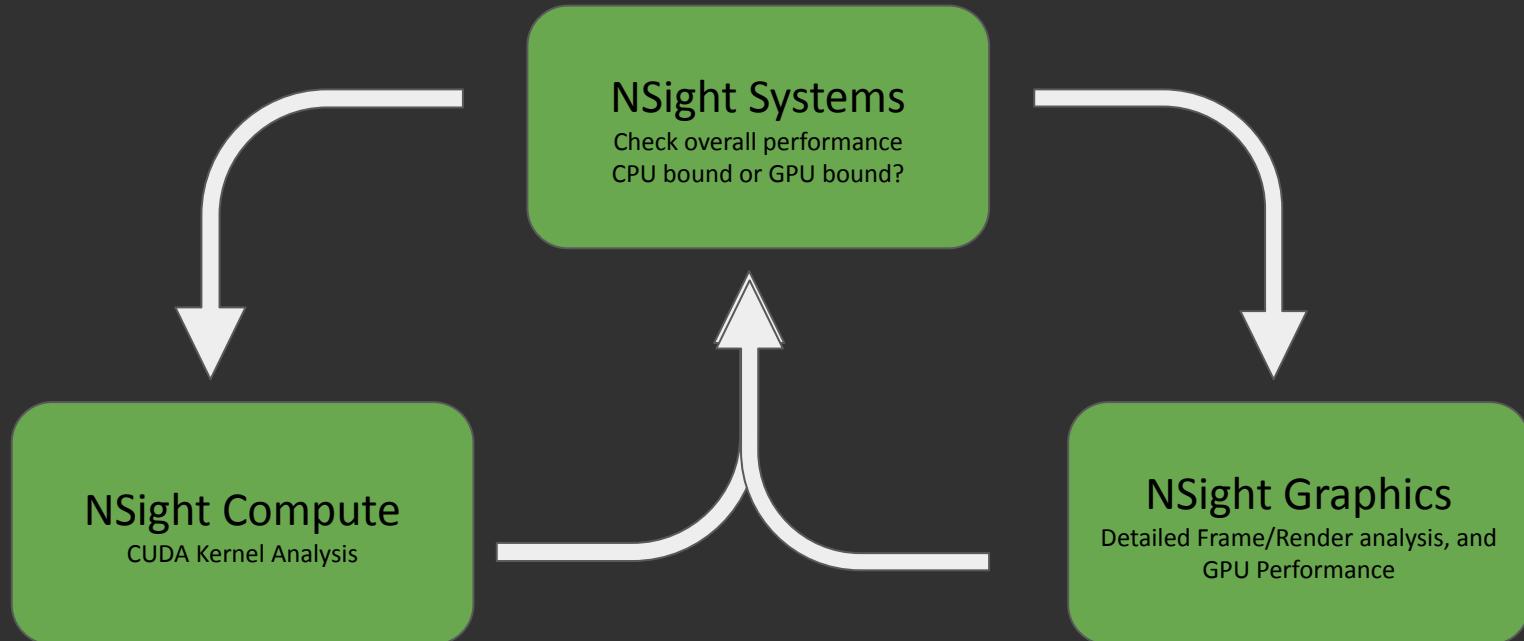


Overview

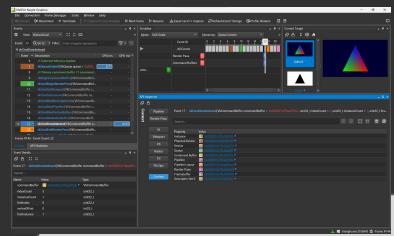
- Goal: Improve your life and understand your GPU by using tools!
- What is NSight Graphics?
- Walkthrough
 - Frame Debugger How-To
 - GPU Trace How-To & Perf Methods
- Conclusion



Workflow/Ecosystem



Workflow/Ecosystem



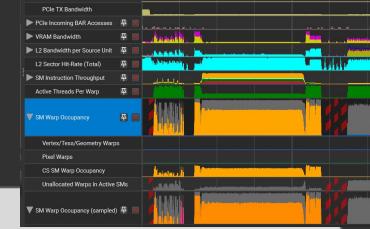
Frame Debugger

- Rendering bugs
- Examine API resources and states of change
- Draw-by-draw analysis of a frame
- Live Shader Debugging

NSight Graphics

Detailed Frame/Render performance

Shader Debugger (Beta)



GPU Trace

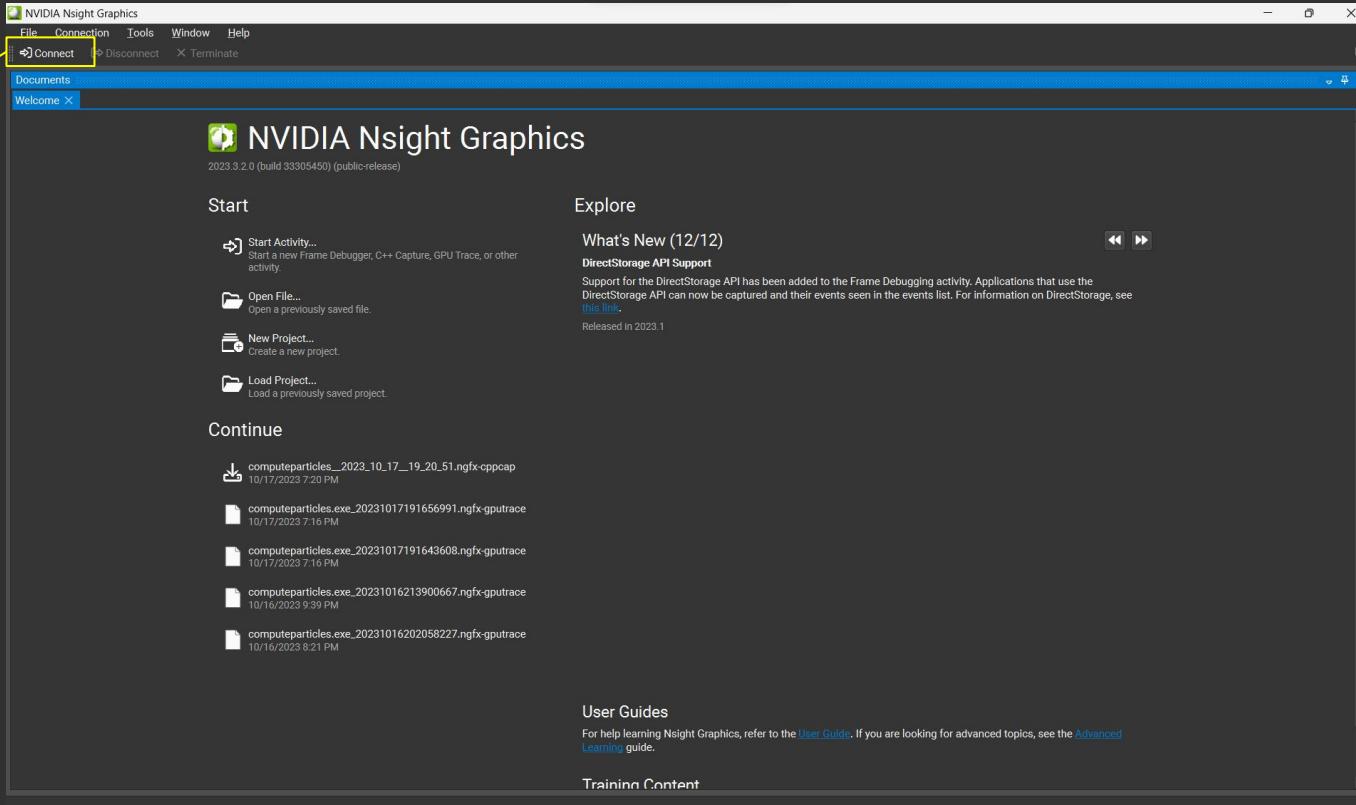
- HW Units throughput analysis
- Shader Profiling

Part 1

Frame Debugger

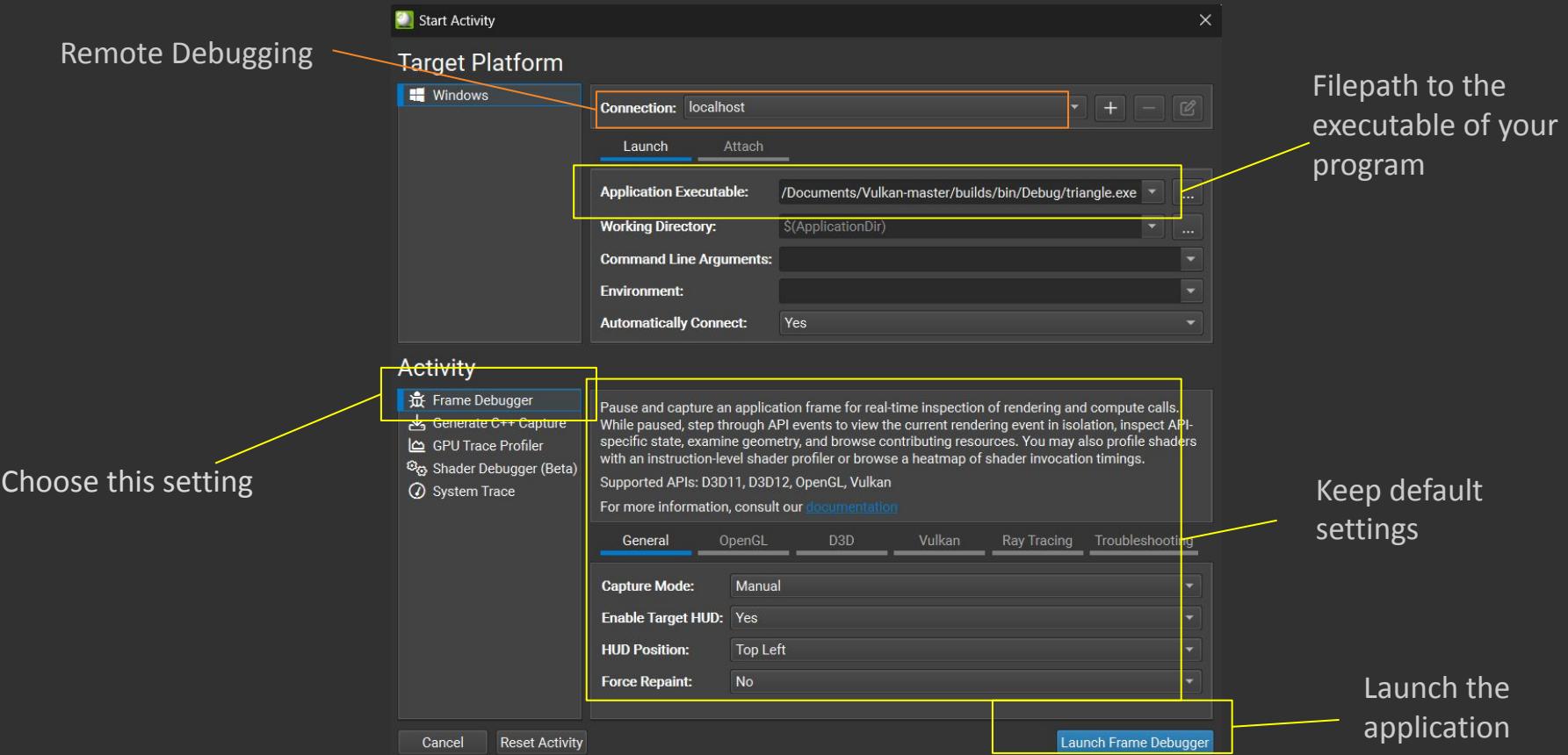
Render bugs, API resource states, draw-by-draw analysis of a
frame

1. Frame Debugger - Getting Started



Start a new session

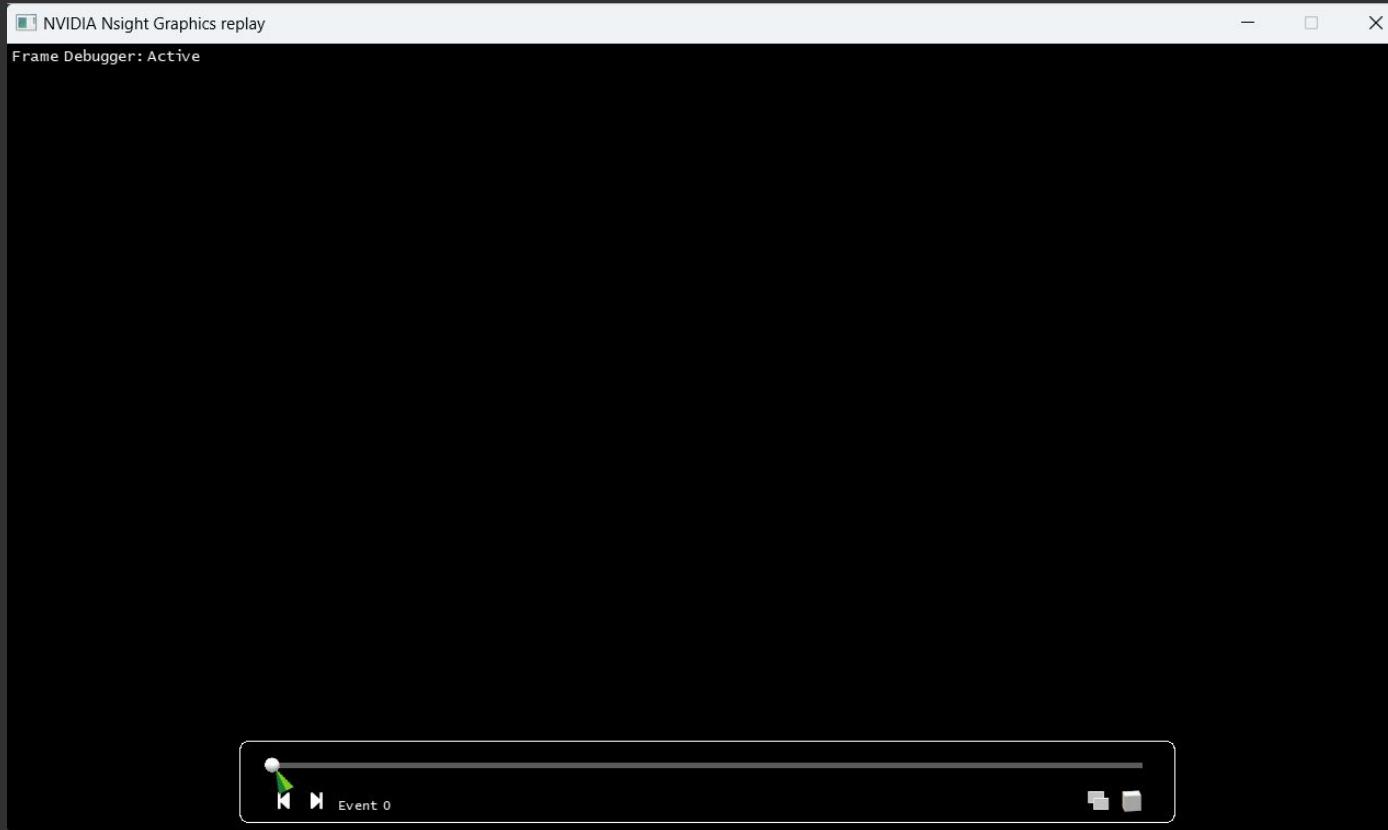
1. Frame Debugger - Getting Started



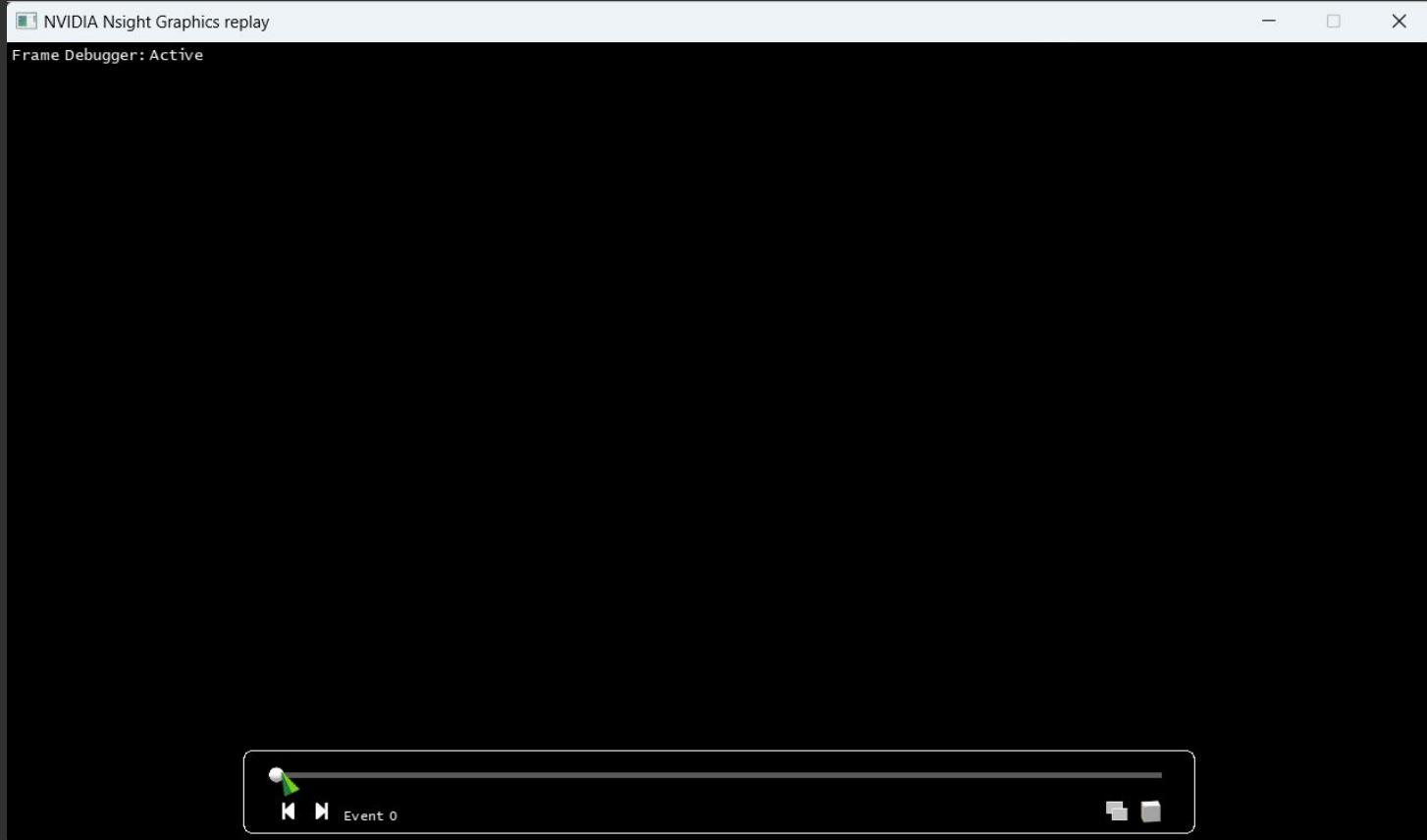
1. Frame Debugger - Getting Started



1. Frame Debugger - Getting Started



1. Frame Debugger - Getting Started



1. Frame Debugger

The screenshot shows the Frame Debugger interface with the following details:

- Events Panel:** Displays a list of events in a flat view. The current event is highlighted at index 18, which is a `vkCmdEndRenderPass` call. Other events include `vkBeginCommandBuffer`, `vkCmdBeginRenderPass`, `vkCmdSetViewport`, `vkCmdSetScissor`, `vkCmdBindDescriptorSets`, `vkCmdBindPipeline`, `vkCmdBindVertexBuffers`, `vkCmdBindIndexBuffer`, `vkCmdDrawIndexed`, and `vkEndCommandBuffer`. The timeline shows CPU ms and GPU ms.
- Scrubber Panel:** A timeline view showing the sequence of events. The timeline is labeled from 0 to 20. The scrubber highlights the `vkCmdEndRenderPass` event at index 18. Other markers include `All Events`, `Render Pass`, and `Command Buffers`.
- API Inspector Panel:** Shows details for the selected event (Event 18).
 - Event Details:** Event 18 is a `vkCmdEndRenderPass` call with a command buffer value of `0x000001d78ea0f520`.
 - Context:** Provides context for the event, showing properties like Instance, Physical Device, Device, Queue, and Command Buffer, all associated with the value `0x000001d78ea0f520`.

1. Frame Debugger - General Settings

The screenshot shows the NVIDIA Nsight Graphics interface, specifically the Frame Debugger tab. A yellow box highlights the top navigation bar and the 'Events' pane.

Events Pane:

- Event ID: 17
- Description: // Coherent Memory Update
- CPU ms: 0.77
- GPU ms: -
- Event ID: 7
- Description: vkQueueSubmit(VkQueue queue = 0x000001d78ea0f520, VkSubmitInfo const& info, VkFence fence, VkSemaphore signalSemaphore, VkSemaphore waitSemaphore)
- CPU ms: 0.77
- GPU ms: -
- Event ID: 8
- Description: // Primary command buffer 11 command...
- CPU ms: -
- GPU ms: -
- Event ID: 9
- Description: vkBeginCommandBuffer(VkCommandBuffer commandBuffer, VkCommandBufferBeginInfo const& beginInfo)
- CPU ms: -
- GPU ms: -
- Event ID: 10
- Description: vkCmdBeginRenderPass(VkCommandBuffer commandBuffer, VkRenderPass const& renderPass, VkSubpassDescription const& subpassDescription, VkSubpassContents subpassContents, VkImageLayout finalImageLayout, VkImageLayout initialImageLayout)
- CPU ms: -
- GPU ms: -
- Event ID: 11
- Description: vkCmdSetViewport(VkCommandBuffer commandBuffer, uint32_t firstViewport, uint32_t viewportCount, const VkViewport* pViewports)
- CPU ms: -
- GPU ms: -
- Event ID: 12
- Description: vkCmdSetScissor(VkCommandBuffer commandBuffer, uint32_t firstScissor, uint32_t scissorCount, const VkScissor* pScissiors)
- CPU ms: -
- GPU ms: -
- Event ID: 13
- Description: vkCmdBindDescriptorSets(VkCommandBuffer commandBuffer, VkPipelineBindPoint bindPoint, VkDescriptorSetLayout descriptorSetLayout, uint32_t firstDescriptorSet, uint32_t descriptorSetCount, const VkDescriptorSet* pDescriptorSets, const VkDynamicState* pDynamicStates)
- CPU ms: -
- GPU ms: -
- Event ID: 14
- Description: vkCmdBindPipeline(VkCommandBuffer commandBuffer, VkPipelineBindPoint bindPoint, VkPipeline pipeline)
- CPU ms: -
- GPU ms: -
- Event ID: 15
- Description: vkCmdBindVertexBuffers(VkCommandBuffer commandBuffer, uint32_t firstBinding, uint32_t bindingCount, const VkBuffer* pBuffers, const VkDeviceSize* pOffset)
- CPU ms: -
- GPU ms: -
- Event ID: 16
- Description: vkCmdBindIndexBuffer(VkCommandBuffer commandBuffer, VkIndexType indexType, VkBuffer indexBuffer, VkDeviceSize offset)
- CPU ms: -
- GPU ms: -
- Event ID: 17
- Description: vkCmdDrawIndexed(VkCommandBuffer commandBuffer, uint32_t firstIndex, int32_t vertexOffset, uint32_t indexCount, uint32_t instanceCount, uint32_t firstInstance)
- CPU ms: <0.01
- GPU ms: -
- Event ID: 18
- Description: vkCmdEndRenderPass(VkCommandBuffer commandBuffer)
- CPU ms: -
- GPU ms: -

Frame #146 - Event Count: 22

Scrubber Pane:

- Mode: Unit Scale
- Hierarchy: Queue Centric
- Event ID: 0, 2, 4, 6, 8, 10, 12, 14, 16, 17, 18, 20
- All Events
- Render Pass
- Command Buffers

API Inspector:

Context: Pipeline

Event 17 - vkCmdDrawIndexed(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, uint32_t indexCount = 3, uint32_t instanceCount = 1, uint32_t firstIndex = 0, int32_t vertexOffset = 0, uint32_t firstInstance = 1)

Search...

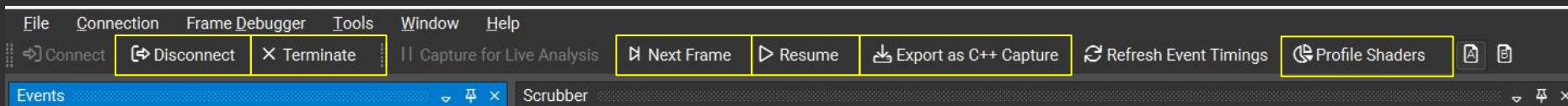
Property	Value
Instance	0x000001d78e96fa50
Physical Device	0x000001d78d940df0
Device	0x000001d78e3d0670
Queue	0x000001d78e5a8a10
Command Buffer	0x000001d78ea0f520
Pipeline	0x000001d78ea63fe0
Pipeline Layout	0x000001d78ea46bc0
Render Pass	0x000001d78e250c0
Framebuffer	0x000001d78e960500
Descriptor Set 0	0x000001d78ea0f6a0

Sascha Willems - Triangle.exe

triangle.exe [15584] Frame #146

1. Frame Debugger - General Settings

- **Disconnect:** Stop Frame Debugger, but your program continues to run
- **Terminate:** To stop program and stop Frame Debugger session.
- **Next Frame:** Analyze the next frame
- **Resume:** Application resume so you can make another capture manually
- **Export as C++ Capture:** Isolate the frame as a C++ application
- **Profile Shaders:** [You will see soon...]



1. Frame Debugger - Event Viewer

The screenshot shows the NVIDIA Nsight Graphics interface, specifically the Event Viewer tab. The main window is divided into several panels:

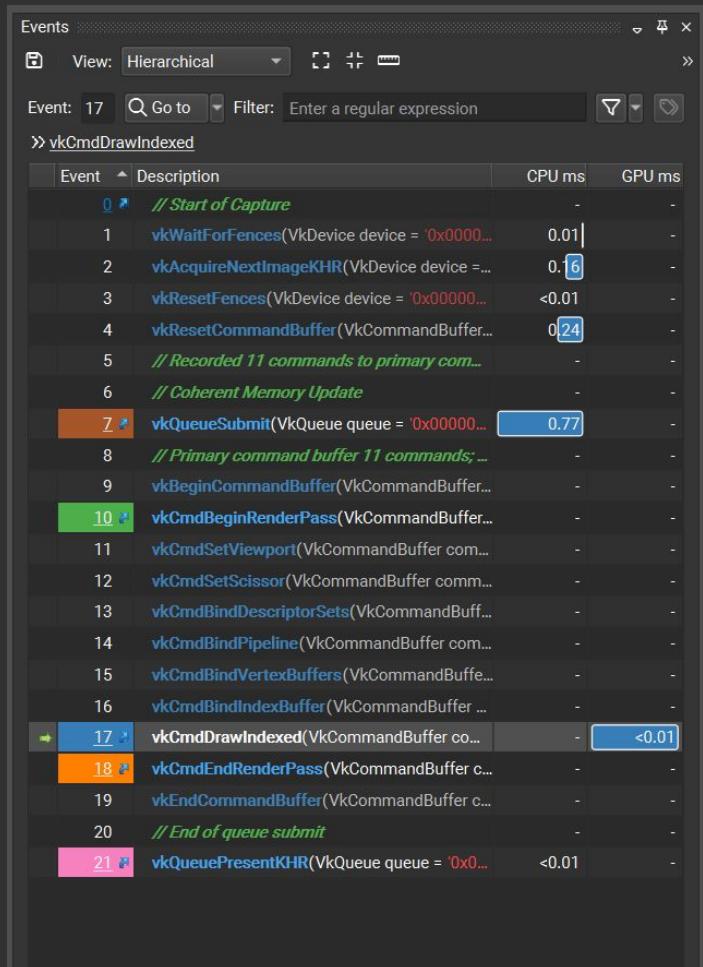
- Events Panel:** Shows a hierarchical list of events. The event at index 17 is selected, highlighted with a yellow border. The details for this event are shown in the API Inspector panel.
- Scrubber Panel:** A timeline showing the sequence of events. The event at index 17 is highlighted with an orange box.
- Current Target Panel:** Displays two render targets: "Color 0" and "Depth". Both show a single blue triangle.
- API Inspector Panel:** Provides detailed information about the selected event (Event 17). The "Context" tab is active, showing properties like Instance, Physical Device, Device, Queue, Command Buffer, Pipeline, Pipeline Layout, Render Pass, Framebuffer, and Descriptor Set 0, each with a corresponding memory dump icon.
- Event Details Panel:** Shows the specific API call: `vkCmdDrawIndexed(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, uint32_t indexCount = 3, uint32_t instanceCount = 1, uint32_t firstIndex = 0, int32_t vertexOffset = 0, uint32_t firstInstance = 1)`.
- Event Statistics Panel:** Shows the count of events in the current frame.
- Event Details Table:** A table showing the values for the parameters of the selected API call.

At the bottom left, the text "Sascha Willems - Triangle.exe" is visible. At the bottom right, the system tray shows "triangle.exe [15584] Frame #146".

Name	Type	Value
commandBuffer	VkCommandBuffer	0x000001d78ea0f520
indexCount	uint32_t	3
instanceCount	uint32_t	1
firstIndex	uint32_t	0
vertexOffset	int32_t	0
firstInstance	uint32_t	1

1. Frame Debugger - Event Viewer

- Can navigate or control the Frame Debugger session
- A list view of all the commands in order of execution with respect to the frame



1. Frame Debugger - Event Viewer

1:1 match with the render() function from Vulkan's Triangle sample

```

virtual void render()
{
    if ([prepared]
        return;

    // Use a fence to wait until the command buffer has finished execution before using it again
    vWaitForFences(device, 1, &waitFences[currentFrame], VK_TRUE, UINT64_MAX);

    // Get the next swap chain image from the implementation
    // Note that the implementation is free to return the images in any order, so we must use the acquire function and can't just cycle
    uint32_t imageIndex;
    VkResult result = vImageAcquireNextImageKHR(device, swapChain.swapChain, UINT64_MAX, presentCompleteSemaphores[currentFrame], VK_NULL_HANDLE);
    if (result == VK_ERROR_OUT_OF_DATE_KHR) {
        windowResize();
        return;
    } else if ((result != VK_SUCCESS) && (result != VK_SUBOPTIMAL_KHR)) {
        throw "Could not acquire the next swap chain image!";
    }

    // Update the uniform buffer for the next frame
    ShaderData shaderData;
    shaderData.projectionMatrix = camera.matrices.perspective;
    shaderData.viewMatrix = camera.matrices.view;
    shaderData.modelMatrix = glm::mat4(1.0f);

    // Copy the current matrices to the current frame's uniform buffer
    // Note: Since we requested a host coherent memory type for the uniform buffer, the write is instantly visible to the GPU
    memcpy(uniformBuffers[currentBuffer].mapped, &shaderData, sizeof(shaderData));

    VK_CHECK_RESULT(vkResetFences(device, 1, &waitFences[currentFrame]));

    // Build the command buffer
    // Unlike in OpenGL all rendering commands are recorded into command buffers that are then submitted to the queue
    // This allows to generate work upfront in a separate thread
    // For basic command buffers (like in this sample), recording is so fast that there is no need to offload this

    vkResetCommandBuffer(commandBuffers[currentBuffer], 0);

    VkCommandBufferBeginInfo cmdBufInfo{};

    cmdBufInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;

    // Set clear values for all framebuffer attachments with loadOp set to clear
    // We use two attachments (color and depth) that are cleared at the start of the subpass and as such we need to set clear values for both
    VkClearValue clearValues[2];
    clearValues[0].color = { { 0.0f, 0.0f, 0.2f, 1.0f } };
    clearValues[1].depthStencil = { 1.0f, 0 };

    VkRenderPassBeginInfo renderPassBeginInfo{};
    renderPassBeginInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO;
    renderPassBeginInfo.pNext = nullptr;
    renderPassBeginInfo.renderPass = renderPass;
    renderPassBeginInfo.renderArea.offset.x = 0;
    renderPassBeginInfo.renderArea.offset.y = 0;
    renderPassBeginInfo.renderArea.extent.width = width;
    renderPassBeginInfo.renderArea.extent.height = height;
    renderPassBeginInfo.clearValueCount = 2;
    renderPassBeginInfo.pClearValues = clearValues;
    renderPassBeginInfo.pSubpasses = &subpassInfo.imageIndex;

    VK_CHECK_RESULT(vkBeginCommandBuffer(commandBuffers[currentBuffer], cmdBufInfo));

```

Event	View:	Hierarchical			
Event:	17	Q Go to	Filter:	Enter a regular expression	
» vkCmdDrawIndexed					
Event	Description	CPU ms	GPU ms		
0	// Start of Capture	-	-	-	-
1	vkWaitForFences(VkDevice device = '0x0000...)	0.01	-	-	-
2	vkAcquireNextImageKHR(VkDevice device = ...)	0.6	-	-	-
3	vkResetFences(VkDevice device = '0x0000...)	<0.01	-	-	-
4	vkResetCommandBuffer(VkCommandBuffer...)	0.24	-	-	-
5	// Recorded 11 commands to primary com...	-	-	-	-
6	// Coherent Memory Update	-	-	-	-
7	vkQueueSubmit(VkQueue queue = '0x0000...)	-	0.77	-	-
8	// Primary command buffer 11 commands; ...	-	-	-	-
9	vkBeginCommandBuffer(VkCommandBuffer...	-	-	-	-
10	vkCmdBeginRenderPass(VkCommandBuffer...	-	-	-	-
11	vkCmdSetViewport(VkCommandBuffer com...	-	-	-	-
12	vkCmdSetScissor(VkCommandBuffer comm...	-	-	-	-
13	vkCmdBindDescriptorSets(VkCommandBuff...	-	-	-	-
14	vkCmdBindPipeline(VkCommandBuffer com...	-	-	-	-
15	vkCmdBindVertexBuffers(VkCommandBuff...	-	-	-	-
16	vkCmdBindIndexBuffer(VkCommandBuffer ...	-	-	-	-
17	vkCmdDrawIndexed(VkCommandBuffer co...	-	<0.01	-	-
18	vkCmdEndRenderPass(VkCommandBuffer c...	-	-	-	-
19	vkEndCommandBuffer(VkCommandBuffer c...	-	-	-	-
20	// End of queue submit	-	-	-	-
21	vkQueuePresentKHR(VkQueue queue = '0x0...	-	<0.01	-	-

1. Frame Debugger - Event Viewer

1:1 match with the render() function from Vulkan's Triangle sample

```
    vkCmdBeginRenderPass(commandBuffers[currentBuffer],
// Update dynamic viewport state
VkViewport viewport{};
viewport.height = (float)height;
viewport.width = (float)width;
viewport.minDepth = (float)0.0f;
viewport.maxDepth = (float)1.0f;
vkCmdSetViewport(commandBuffers[currentBuffer],
// Update dynamic scissor state
VkRect2D scissor{};
scissor.extent.width = width;
scissor.extent.height = height;
scissor.offset.x = 0;
scissor.offset.y = 0;
vkCmdSetScissor(commandBuffers[currentBuffer], 0,
// Bind descriptor set for the current frame's
vkCmdBindDescriptorSets(commandBuffers[currentBuffer],
// Bind the rendering pipeline
// The pipeline (state object) contains all static
vkCmdBindPipeline(commandBuffers[currentBuffer],
// Bind triangle vertex buffer (contains position
VkDeviceSize offsets[1]{ 0 };
vkCmdBindVertexBuffers(commandBuffers[currentBuffer],
// Bind triangle index buffer
vkCmdBindIndexBuffer(commandBuffers[currentBuffer],
// Draw indexed triangle
vkCmdDrawIndexed(commandBuffers[currentBuffer],
vkCmdEndRenderPass(commandBuffers[currentBuffer])
// Ending the render pass will add an implicit barrier
// VK_IMAGE_LAYOUT_PRESENT_SRC_KHR for presenting
VK_CHECK_RESULT(vkEndCommandBuffer(commandBuffer))
```

Event	Description	CPU ms	GPU ms
0	// Start of Capture	-	-
1	vkWaitForFences(VkDevice device = 0x0000...)	0.01	-
2	vkAcquireNextImageKHR(VkDevice device = ...)	0.16	-
3	vkResetFences(VkDevice device = 0x0000...)	<0.01	-
4	vkResetCommandBuffer(VkCommandBuffer...	0.24	-
5	// Recorded 11 commands to primary com...	-	-
6	// Coherent Memory Update	-	-
7	vkQueueSubmit(VkQueue queue = 0x0000...)	0.77	-
8	// Primary command buffer 11 commands;	-	-
9	vkBeginCommandBuffer(VkCommandBuffer...	-	-
10	vkCmdBeginRenderPass(VkCommandBuffer...	-	-
11	vkCmdSetViewport(VkCommandBuffer com...	-	-
12	vkCmdSetScissor(VkCommandBuffer comm...	-	-
13	vkCmdBindDescriptorSets(VkCommandBuff...	-	-
14	vkCmdBindPipeline(VkCommandBuffer com...	-	-
15	vkCmdBindVertexBuffers(VkCommandBuff...	-	-
16	vkCmdBindIndexBuffer(VkCommandBuffer ...	-	-
17	vkCmdDrawIndexed(VkCommandBuffer co...	-	<0.01
18	vkCmdEndRenderPass(VkCommandBuffer c...	-	-
19	vkEndCommandBuffer(VkCommandBuffer c...	-	-
20	// End of queue submit	-	-
21	vkQueuePresentKHR(VkQueue queue = 0x0...	<0.01	-

1. Frame Debugger - Event Viewer

1:1 match with the render() function from Vulkan's Triangle sample

```
// Submit to the graphics queue passing a wait fence
VK_CHECK_RESULT(vkQueueSubmit(queue, 1, &submitInfo, wai

// Present the current frame buffer to the swap chain
// Pass the semaphore signaled by the command buffer sub
// This ensures that the image is not presented to the w

VkPresentInfoKHR presentInfo{};
presentInfo.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
presentInfo.waitSemaphoreCount = 1;
presentInfo.pWaitSemaphores = &renderCompleteSemaphores[0];
presentInfo.swapchainCount = 1;
presentInfo.pSwapchains = &swapChain.swapChain;
presentInfo.pImageIndices = &imageIndex;
result = vkQueuePresentKHR(queue, &presentInfo);

if ((result == VK_ERROR_OUT_OF_DATE_KHR) || (result == VK_ERROR_INCOMPATIBLE_DISPLAY_KHR)) {
    windowResize();
}

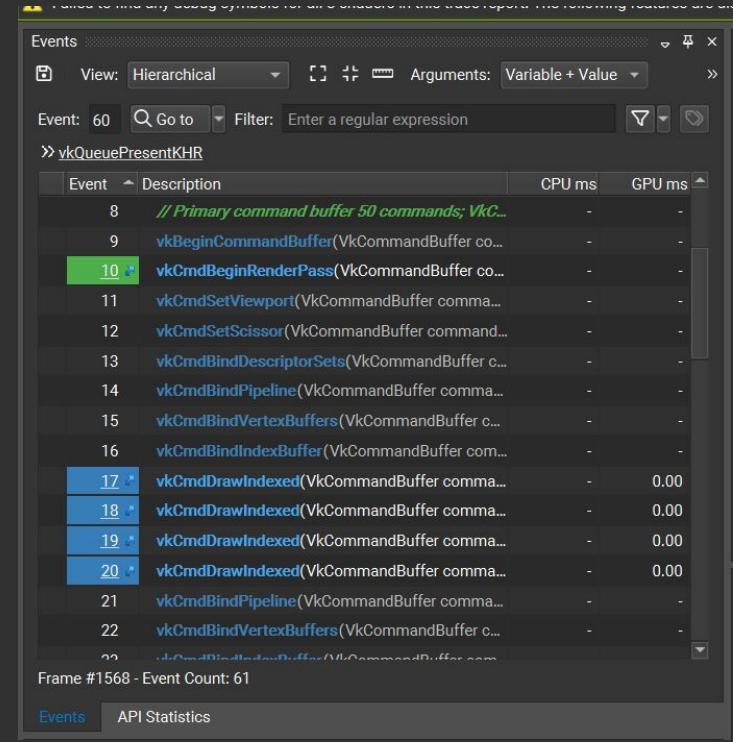
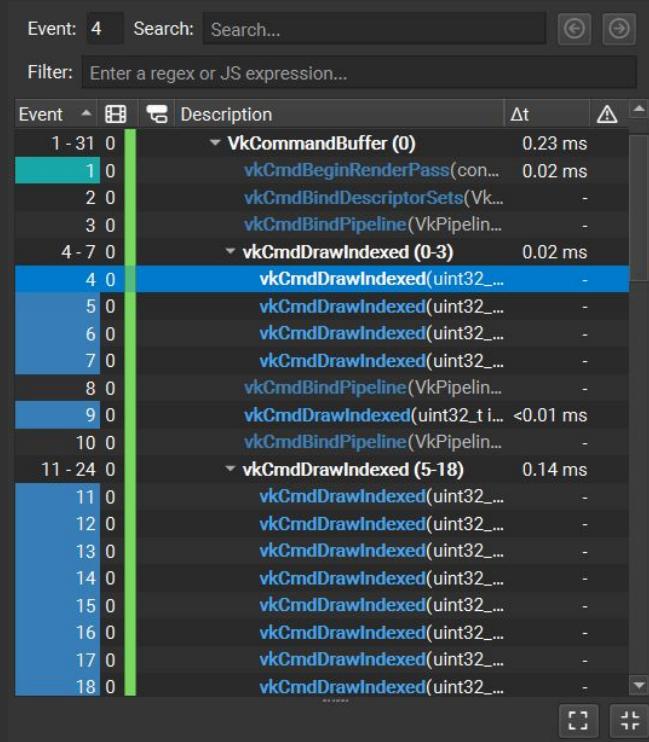
else if (result != VK_SUCCESS) {
    throw "Could not present the image to the swap chain";
}
```

Event: 17 Filter: Enter a regular expression

» **vkCmdDrawIndexed**

Event	Description	CPU ms	GPU ms
0	// Start of Capture	-	-
1	vkWaitForFences(VkDevice device = 0x0000...)	0.01	-
2	vkAcquireNextImageKHR(VkDevice device = ...)	0.16	-
3	vkResetFences(VkDevice device = 0x0000...)	<0.01	-
4	vkResetCommandBuffer(VkCommandBuffer...)	0.24	-
5	// Recorded 11 commands to primary com...	-	-
6	// Coherent Memory Update	-	-
7	vkQueueSubmit(VkQueue queue = 0x0000...)	0.77	-
8	// Primary command buffer 11 commands; ...	-	-
9	vkBeginCommandBuffer(VkCommandBuffer...)	-	-
10	vkCmdBeginRenderPass(VkCommandBuffer...)	-	-
11	vkCmdSetViewport(VkCommandBuffer com...	-	-
12	vkCmdSetScissor(VkCommandBuffer comm...	-	-
13	vkCmdBindDescriptorSets(VkCommandBuff...	-	-
14	vkCmdBindPipeline(VkCommandBuffer com...	-	-
15	vkCmdBindVertexBuffers(VkCommandBuff...	-	-
16	vkCmdBindIndexBuffer(VkCommandBuffer ...)	-	-
17	vkCmdDrawIndexed(VkCommandBuffer co...)	-	<0.01
18	vkCmdEndRenderPass(VkCommandBuffer c...)	-	-
19	vkEndCommandBuffer(VkCommandBuffer c...)	-	-
20	// End of queue submit	-	-
21	vkQueuePresentKHR(VkQueue queue = 0x0...	<0.01	-

1. Frame Debugger - Event Viewer



1. Frame Debugger - Scrubber

The screenshot shows the NVIDIA Nsight Graphics interface with the Scrubber tool highlighted by a yellow box. The Scrubber panel displays a timeline of events, with event 17 selected. The timeline shows various stages of a render pass, including command buffers and memory updates. To the right of the timeline, the Current Target window shows a 3D triangle rendered with a color gradient and depth information. Below the timeline, the API Inspector panel provides detailed context for the selected event, listing properties like Instance, Physical Device, and Command Buffer. The Events panel on the left lists all captured events, and the Event Details panel shows the specific parameters for event 17.

Events

Event: 17 | Go to | Filter: Enter a regular expression

Event	Description	CPU ms	GPU ms
6	// Coherent Memory Update	-	-
7	vkQueueSubmit(VkQueue queue = 0x0000000000000000, VkSubmitInfo const& submitInfo, VkFence fence = 0x0000000000000000, VkSemaphore semaphore = 0x0000000000000000, uint32_t waitSemaphoreCount = 0, VkSemaphore const& pWaitSemaphores = 0x0000000000000000, VkSemaphore signalSemaphore = 0x0000000000000000, VkSemaphore const& pSignalSemaphore = 0x0000000000000000, uint32_t waitSemaphoreMask = 0, uint32_t commandBufferCount = 1, VkCommandBuffer const& pCommandBuffers = 0x0000000000000000, uint32_t signalSemaphoreMask = 0, VkFence const& pFence = 0x0000000000000000, VkFence const& pNext = 0x0000000000000000)	0.77	-
8	// Primary command buffer 11 command...	-	-
9	-	-	-
10	vkCmdBeginRenderPass(VkCommandBuf...	-	-
11	vkCmdSetViewport(VkCommandBuffer co...	-	-
12	vkCmdSetScissor(VkCommandBuffer co...	-	-
13	vkCmdBindDescriptorSets(VkCommandB...	-	-
14	vkCmdBindPipeline(VkCommandBuffer c...	-	-
15	vkCmdBindVertexBuffers(VkCommandBu...	-	-
16	vkCmdBindIndexBuffer(VkCommandBuff...	-	-
17	vkCmdDrawIndexed(VkCommandBuffer c...	<0.01	-
18	vkCmdEndRenderPass(VkCommandBuff...	-	-
19	-	-	-
20	-	-	-

Frame #146 - Event Count: 22

Events | API Statistics

Event Details

Event 17 - `vkCmdDrawIndexed(VkCommandBuffer commandBuffer = 0x0000000000000000, uint32_t indexCount = 3, uint32_t instanceCount = 1, uint32_t firstIndex = 0, int32_t vertexOffset = 0, uint32_t firstInstance = 1)`

Search...

Name	Value	Type
commandBuffer	0x0000000000000000	VkCommandBuffer
indexCount	3	uint32_t
instanceCount	1	uint32_t
firstIndex	0	uint32_t
vertexOffset	0	int32_t
firstInstance	1	uint32_t

Scrubber

Mode: Unit Scale | Hierarchy: Queue Centric

Event ID	0	2	4	6	8	10	12	14	16	17	18	20
All Events	■	■	■	■	■	■	■	■	■	■	■	■
Render Pass	■	-	-	-	-	-	-	-	-	-	-	-
Command Buffers	■	-	-	-	-	-	-	-	-	■	-	-

Current Target

API Inspector

Context

Event 17 - `vkCmdDrawIndexed(VkCommandBuffer commandBuffer = 0x0000000000000000, uint32_t indexCount = 3, uint32_t instanceCount = 1, uint32_t firstIndex = 0, int32_t vertexOffset = 0, uint32_t firstInstance = 1)`

Property Value

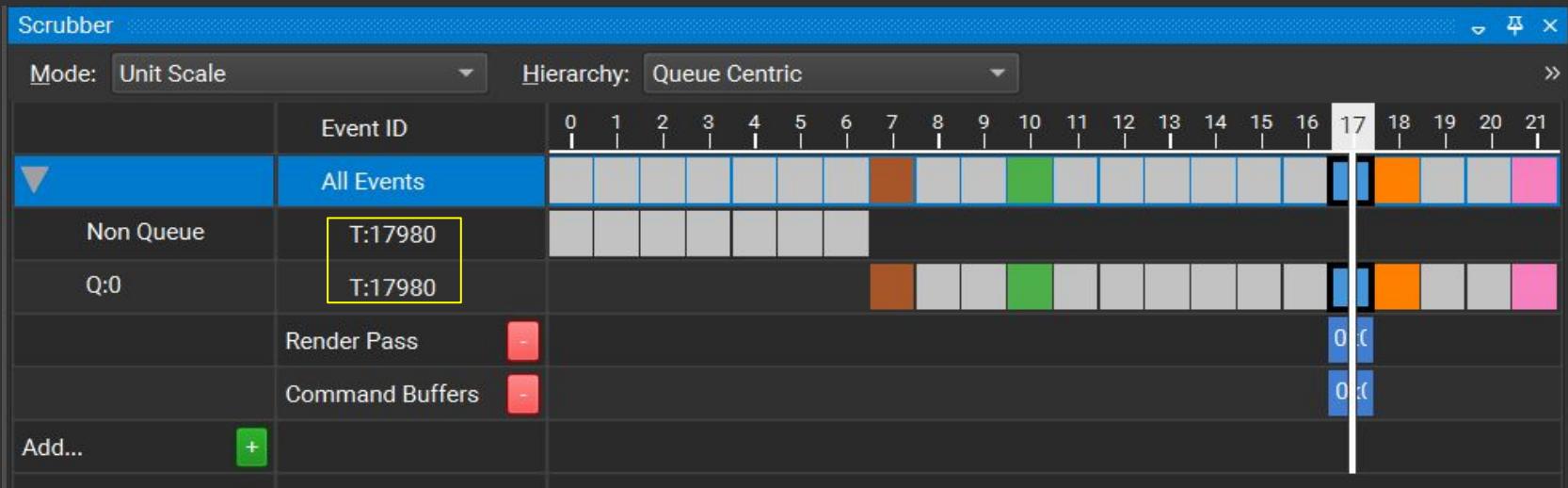
Instance	0x0000000000000000
Physical Device	0x0000000000000000
Device	0x0000000000000000
Queue	0x0000000000000000
Command Buffer	0x0000000000000000
Pipeline	0x0000000000000000
Pipeline Layout	0x0000000000000000
Render Pass	0x0000000000000000
Framebuffer	0x0000000000000000
Descriptor Set 0	0x0000000000000000

Sascha Willems - Triangle.exe

triangle.exe [15584] Frame #146

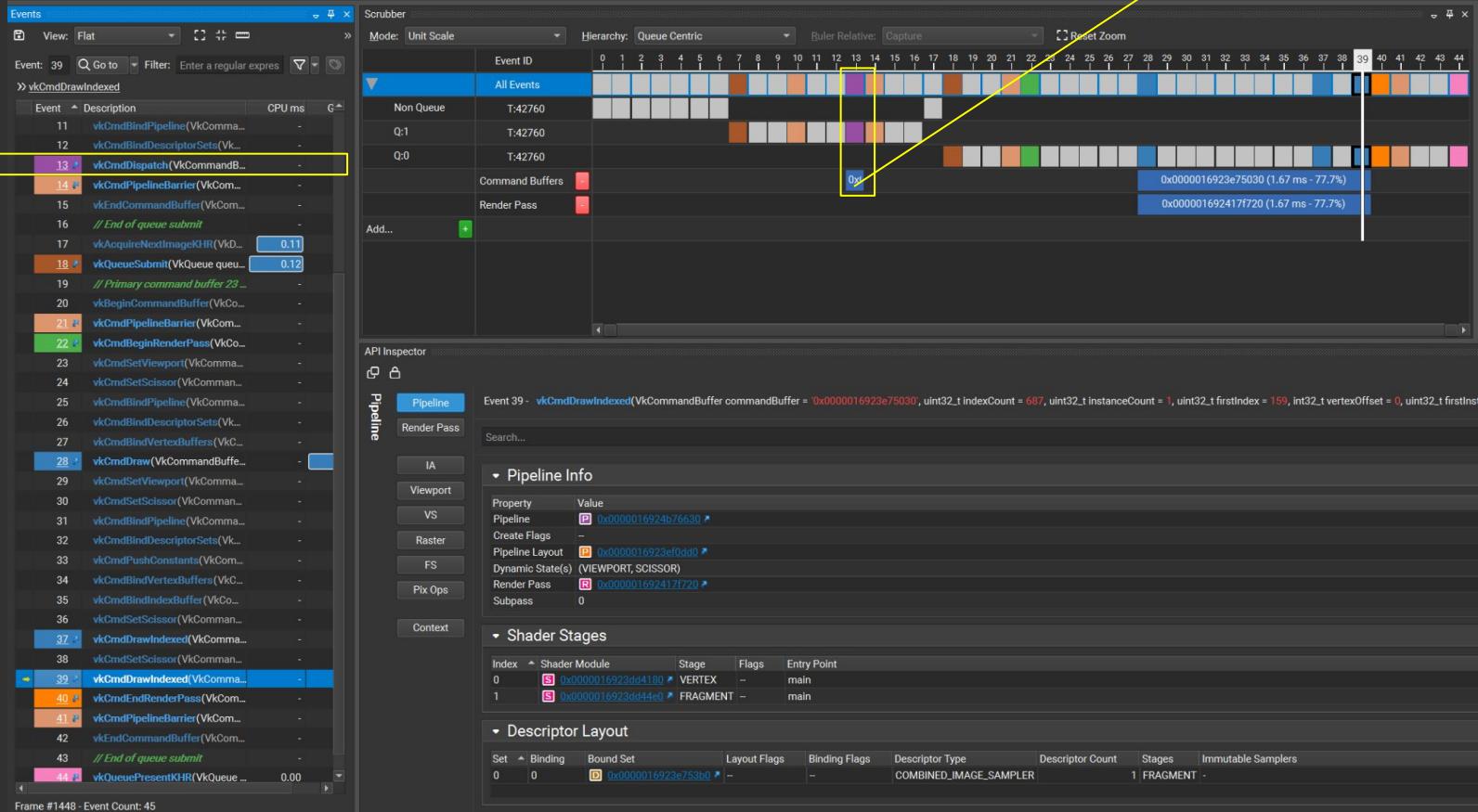
1. Frame Debugger - Scrubber

- Can navigate or control the Frame Debugger session
- Queue Centric Hierarchy: Splits events by queue
- Thread Centric Hierarchy: Splits events by thread



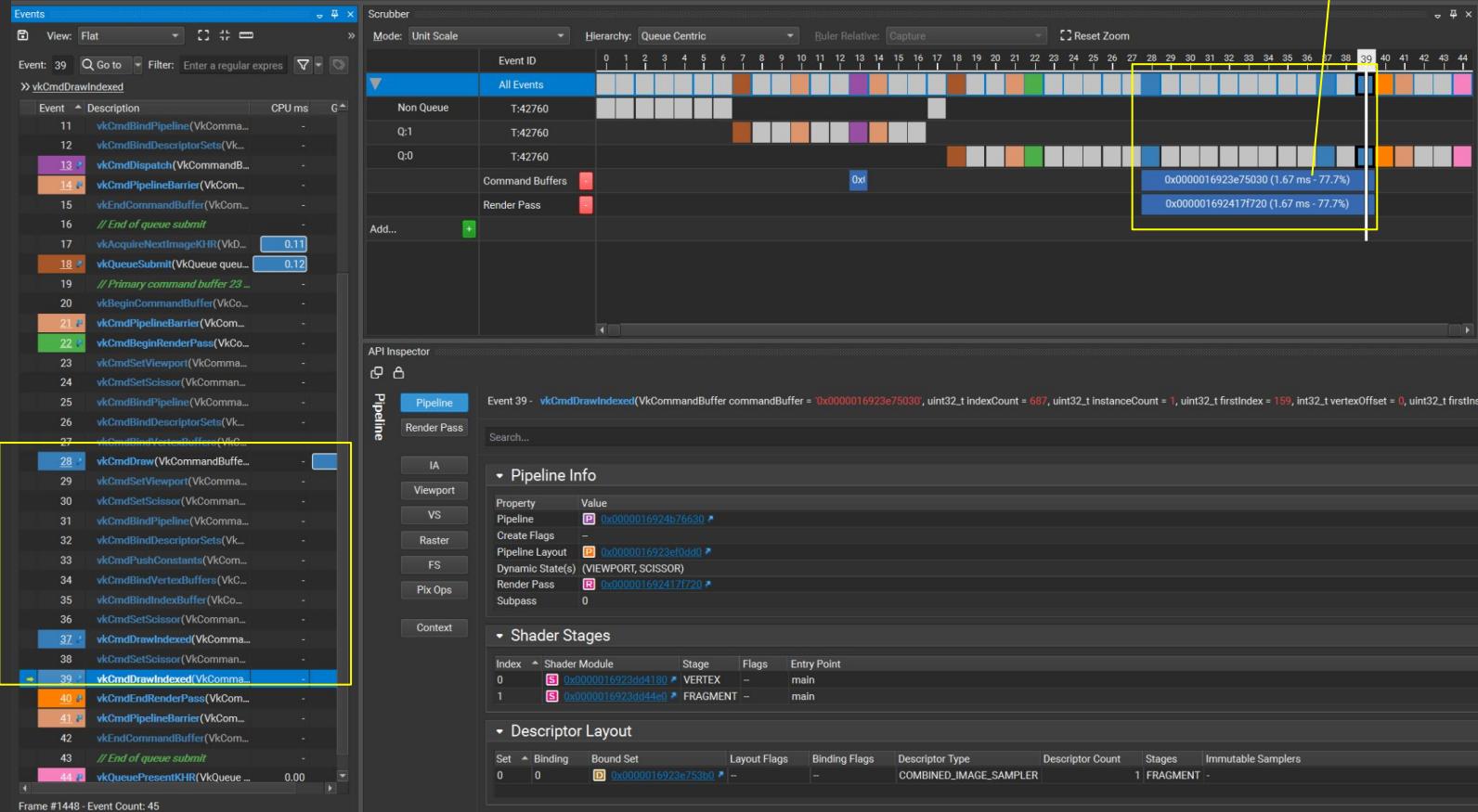
1. Frame Debugger - Scrubber

Compute Command Buffer



1. Frame Debugger - Scrubber

Graphics Command Buffer



1. Frame Debugger - Current Target

The screenshot shows the NVIDIA Nsight Graphics interface with the "Frame Debugger" tab selected. The main window displays the "Events" panel, which lists GPU events with their descriptions, CPU ms, and GPU ms. Event 17, "vkCmdDrawIndexed", is highlighted. The "Scrubber" panel shows a timeline of events from 0 to 20, with event 17 selected. The "Current Target" panel on the right displays a rendered triangle with "Color 0" and "Depth" information. The "API Inspector" panel provides detailed context for event 17, listing properties like Instance, Physical Device, and Command Buffer. The "Event Details" panel at the bottom shows the parameters for the selected event.

Events

Event	Description	CPU ms	GPU ms
6	// Coherent Memory Update	-	-
7	vkQueueSubmit(VkQueue queue = 0x000001d78ea0f520, VkSubmitInfo const& info = {0}, VkSemaphore signalSemaphore = 0x0000000000000000, VkSemaphore waitSemaphore = 0x0000000000000000, uint32_t waitSemaphoreCount = 0, uint32_t submitCount = 1)	0.77	-
8	// Primary command buffer 11 command...	-	-
9	vkBeginCommandBuffer(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, VkCommandBufferLevel level = VK_COMMAND_BUFFER_LEVEL_PRIMARY)	-	-
10	vkCmdBeginRenderPass(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, VkRenderPass renderPass = 0x0000000000000000, VkSubpassDescription subpass = {0}, VkSubpassContents contents = VK_SUBPASS_CONTENTS_INLINE)	-	-
11	vkCmdSetViewport(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, VkViewport viewport = {0.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f})	-	-
12	vkCmdSetScissor(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, VkRect scissor = {0, 0, 100, 100})	-	-
13	vkCmdBindDescriptorSets(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, VkPipelineBindPoint bindPoint = VK_DESCRIPTOR_SETS_BEGIN, VkDescriptorSet descriptorSets = 0x0000000000000000, uint32_t startingBinding = 0, uint32_t descriptorSetCount = 1)	-	-
14	vkCmdBindPipeline(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, VkPipeline pipeline = 0x0000000000000000, VkPipelineBindPoint bindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS)	-	-
15	vkCmdBindVertexBuffers(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, uint32_t firstBinding = 0, VkBuffer buffers = 0x0000000000000000, VkDeviceSize offsets = 0x0000000000000000, uint32_t count = 1)	-	-
16	vkCmdBindIndexBuffer(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, VkBuffer buffer = 0x0000000000000000, VkIndexType indexType = VK_INDEX_TYPE_UINT32, int32_t firstIndex = 0, int32_t indexCount = 3)	-	-
17	vkCmdDrawIndexed(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, uint32_t indexCount = 3, uint32_t instanceCount = 1, uint32_t firstIndex = 0, int32_t vertexOffset = 0, uint32_t firstInstance = 1)	<0.01	-
18	vkCmdEndRenderPass(VkCommandBuffer commandBuffer = 0x000001d78ea0f520)	-	-

Frame #146 - Event Count: 22

Events API Statistics

Event Details

Event 17 - `vkCmdDrawIndexed(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, uint32_t indexCount = 3, uint32_t instanceCount = 1, uint32_t firstIndex = 0, int32_t vertexOffset = 0, uint32_t firstInstance = 1)`

Name	Value	Type
commandBuffer	0x000001d78ea0f520	VkCommandBuffer
indexCount	3	uint32_t
instanceCount	1	uint32_t
firstIndex	0	uint32_t
vertexOffset	0	int32_t
firstInstance	1	uint32_t

Scrubber

Current Target

API Inspector

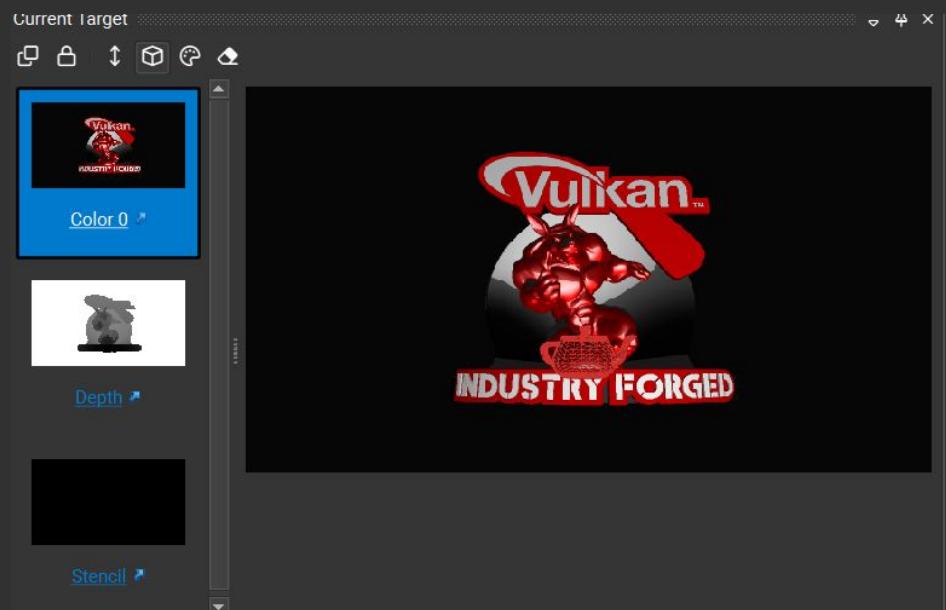
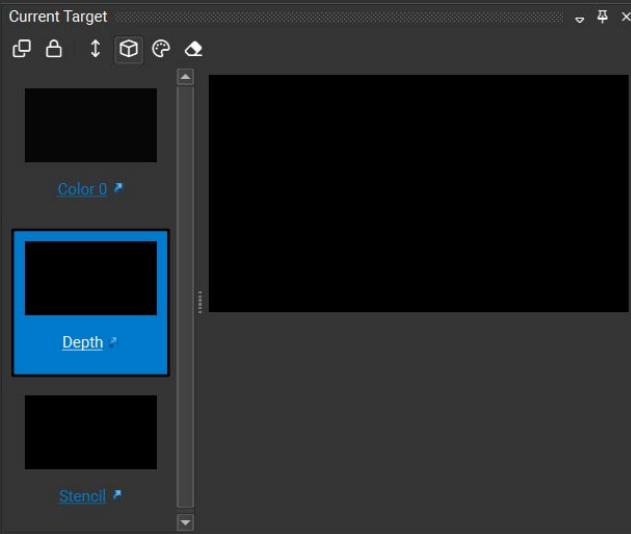
Context

Sascha Willems - Triangle.exe

triangle.exe [15584] Frame #146

1. Frame Debugger - Current Target

- Currently bound output targets
- Can show color, depth, and stencil buffers



1. Frame Debugger - Event Details

The screenshot shows the NVIDIA Nsight Graphics Frame Debugger interface. The main window is divided into several panels:

- Events Panel:** Shows a hierarchical list of events. The current event is highlighted as "Event 17" with the description "vkCmdDrawIndexed(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, uint32_t indexCount = 3, uint32_t instanceCount = 1, uint32_t firstIndex = 0, int32_t vertexOffset = 0, uint32_t firstInstance = 1)".
- Scrubber Panel:** A timeline showing the sequence of events. The event "vkCmdDrawIndexed" is highlighted at frame 17.
- Current Target Panel:** Displays a 3D rendering of a triangle colored by depth (Color 0) and a depth map.
- API Inspector Panel:** Provides detailed information about the selected event, including the pipeline, render pass, and context properties.
- Context Panel:** Shows the context properties for the selected event, such as command buffer, pipeline, and descriptor sets.
- Event Details Panel:** A table showing the parameters of the vkCmdDrawIndexed call. The table is highlighted with a yellow border.

Event Details Table:

Name	Value	Type
commandBuffer	0x000001d78ea0f520	VkCommandBuffer
indexCount	3	uint32_t
instanceCount	1	uint32_t
firstIndex	0	uint32_t
vertexOffset	0	int32_t
firstInstance	1	uint32_t

Sascha Willems - Triangle.exe

1. Frame Debugger - Event Details

- Info about parameters to the call
- Links to the objects (Clicking the links leads to the Object Browser)

Event Details		
Event 16 - vkCmdBindIndexBuffer (VkCommandBuffer commandBuffer = 0x000001d78ea0f520...)		
Search...		
Name	Value	Type
commandBuffer	C 0x000001d78ea0f520	VkCommandBuffer
buffer	B 0x000001d78ea1f910	VkBuffer
offset	0	VkDeviceSize
indexType	VK_INDEX_TYPE_UINT32	VkIndexType

1. Frame Debugger - API Inspector

The screenshot shows the NVIDIA Nsight Graphics application interface. The main window is divided into several panels:

- Events Panel:** Shows a hierarchical list of GPU events. Event 17, "vkCmdDrawIndexed", is selected and highlighted in orange. Other events listed include "vkQueueSubmit", "vkBeginCommandBuffer", "vkCmdBeginRenderPass", "vkCmdSetViewport", "vkCmdSetScissor", "vkCmdBindDescriptorSets", "vkCmdBindPipeline", "vkCmdBindVertexBuffers", "vkCmdBindIndexBuffer", and "vkCmdEndRenderPass".
- Scrubber Panel:** A timeline view showing the sequence of events. The event "vkCmdDrawIndexed" (Event 17) is highlighted in blue.
- Current Target Panel:** Displays a 3D rendering of a triangle colored with a gradient from red to blue. Below it is a depth map of the same triangle.
- API Inspector Panel (highlighted with a yellow border):** This panel provides detailed information about the selected event (Event 17). It includes tabs for Pipeline, Render Pass, IA, Viewport, VS, Raster, FS, Pix Ops, and Context. The Context tab is active, showing properties for the command buffer:

Property	Value
Instance	0x000001d78c96fa50
Physical Device	0x000001d78d940df0
Device	0x000001d78e3d0670
Queue	0x000001d78e5a8a10
Command Buffer	0x000001d78ea0f520
Pipeline	0x000001d78ea63fe0
Pipeline Layout	0x000001d78ea46bc0
Render Pass	0x000001d78e2250c0
Framebuffer	0x000001d78e960500
Descriptor Set 0	0x000001d78ea0f6a0
- Bottom Status Bar:** Shows the process name "triangle.exe [15584]", the frame number "Frame #146", and other system information.

1. Frame Debugger - API Inspector

The screenshot shows the API Inspector window for a Vulkan command buffer. The left sidebar lists pipeline components: Pipeline, Render Pass, Vertex Shader (selected), IA, Viewport, VS (highlighted in blue), Raster, FS, Pix Ops, and Context. The main area displays an event for `vkCmdDrawIndexed`. The Pipeline Shader Stage is identified as VERTEX. It has two inputs: `inPos` (vec3f*) and `inColor` (vec3f*). It has one output: `outColor` (vec3f*). Under Uniform & Storage Buffers, there is one entry for a UBO named `ubo`, which is a struct `UBO*` located at offset 0 with a range of 192 bytes. It is bound to descriptor index 0 and is a UNIFORM_BUFFER.

Event 17 - `vkCmdDrawIndexed`(`VkCommandBuffer commandBuffer = '0x000001d78ea0f520'`, `uint32_t indexCount = 3`, `uint32_t instanceCount = 1`, `uint32_t firstIndex = 0`, `int32_t vertex...`)

Search...

Flags: -- Stage: VERTEX Module: 0x000001d78e960860 Code: View Source

Inputs

Location	Name	Type
0	inPos	vec3f*
1	inColor	vec3f*

Outputs

Location	Name	Type
0	outColor	vec3f*

Uniform & Storage Buffers

Width: 0 Precision: 2 Rep: Scientific Hex:

Name	Value	Type	Offset	Range	Set	Binding	Descriptor Type	Buffer
ubo	[...]	struct UBO*	0	192	0	0	UNIFORM_BUFFER	01 0x000001d78ea20090 @ 0.B

1. Frame Debugger - API Inspector

The screenshot shows the API Inspector interface with the "Pipeline" tab selected (highlighted with a yellow box). The main pane displays an event for `vkCmdDrawIndexed` with the following details:

Event 17 - `vkCmdDrawIndexed(VkCommandBuffer commandBuffer = '0x000001d78ea0f520', uint32_t indexCount = 3, uint32_t instanceCount = 1, uint32_t firstIndex = 0, int32_t vertex...`

Pipeline Info

Property	Value
Pipeline	<code>P 0x000001d78ea63fe0</code>
Create Flags	-
Pipeline Layout	<code>P 0x000001d78ea46bc0</code>
Dynamic State(s)	(VIEWPORT, SCISSOR)
Render Pass	<code>R 0x000001d78e2250c0</code>
Subpass	0

Shader Stages

Index	Shader Module	Stage	Flags	Entry Point
0	<code>S 0x000001d78e960860</code>	VERTEX	-	main
1	<code>S 0x000001d78e960980</code>	FRAGMENT	--	main

Descriptor Layout

Set	Binding	Bound Set	Layout Flags	Binding Flags	Descriptor Type	Descriptor Count	Stages	Immutable Samplers
0	0	<code>D 0x000001d78ea0f6a0</code>	-	-	UNIFORM_BUFFER	1	VERTEX	-

A code editor window on the right shows the C++ code for creating a pipeline:

```
void createPipeline() {
    // Shaders
    std::array<VkPipelineShaderStageCreateInfo, 2> shaderStages{};

    // Vertex shader
    shaderStages[0].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
    shaderStages[0].stage = VK_SHADER_STAGE_VERTEX_BIT;
    // Load binary SPIR-V shader
    shaderStages[0].module = loadSPIRVShader(getShadersPath() + "triangle/triangle.vert.spv");
    // Main entry point for the shader
    shaderStages[0].pName = "main";
    assert(shaderStages[0].module != VK_NULL_HANDLE);

    // Fragment shader
    shaderStages[1].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
    shaderStages[1].stage = VK_SHADER_STAGE_FRAGMENT_BIT;
    // Load binary SPIR-V shader
    shaderStages[1].module = loadSPIRVShader(getShadersPath() + "triangle/triangle.frag.spv");
    // Main entry point for the shader
    shaderStages[1].pName = "main";
    assert(shaderStages[1].module != VK_NULL_HANDLE);

    // Set pipeline shader stage info
    pipelineCI.stageCount = static_cast<uint32_t>(shaderStages.size());
    pipelineCI.pStages = shaderStages.data();
}
```

1. Frame Debugger - API Inspector

The screenshot shows the API Inspector interface. On the left, there is a sidebar with several tabs: Pipeline, Render Pass (which is highlighted with a yellow box), IA, Viewport, VS, Raster, FS, Pix Ops, and Context. The main area displays an event log entry for 'vkCmdDrawIndexed'. The event details show a Render Pass (0x000001d78e2250c0) with Subpass 0 and Framebuffer (0x000001d78e960500). Below this, a table lists properties like Width (1280), Height (720), and Layers (1). A section titled 'Attachments' is expanded, showing two attachments. Attachment 0 uses an image at address 0x000001d78ea04050 with format B8G8R8A8_UNORM. Attachment 1 uses an image at address 0x000001d78ea049f0 with format B8G8R8A8_UNORM. The right side of the interface shows a code editor with C++ code for setting up the render pass, which includes descriptions for color and depth attachments.

Property	Value
Width	1280
Height	720
Layers	1

Attachments

Image	Index	Property	Value
0x000001d78ea04050	0	Aspect Mask	COLOR
0x000001d78ea049f0	1	Format	B8G8R8A8_UNORM
		View Type	2D
		Components	(R: R, G: G, B: B, A: A)
		Flags	--
		Mip Level(s)	[0 - 0]
		Layer(s)	[0 - 0]
		Load Op.	CLEAR
		Store Op.	STORE
		Clear Value	float32 = {R: 0, G: 0, B: 0.2, A: 1}

```
void setupRenderPass()
{
    // This example will use a single render pass with one subpass

    // Descriptors for the attachments used by this renderpass
    std::array<VkAttachmentDescription, 2> attachments{};

    // Color attachment
    attachments[0].format = swapChain.colorFormat;
    attachments[0].samples = VK_SAMPLE_COUNT_1_BIT;
    attachments[0].loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
    attachments[0].storeOp = VK_ATTACHMENT_STORE_OP_STORE;
    attachments[0].stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
    attachments[0].stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
    attachments[0].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
    attachments[0].finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR;

    // Depth attachment
    attachments[1].format = depthFormat;
    attachments[1].samples = VK_SAMPLE_COUNT_1_BIT;
    attachments[1].loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
    attachments[1].storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
    attachments[1].stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
    attachments[1].stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
    attachments[1].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
    attachments[1].finalLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;

    // Setup attachment references
    VkAttachmentReference colorReference{};
}
```

1. Frame Debugger - API Inspector

The screenshot shows the API Inspector interface for a Vulkan application. The left sidebar has tabs for Pipeline, Render Pass, IA (Input Assembler, highlighted with a yellow box), Viewport, VS (Vertex Shader), Raster, FS (Fragment Shader), Pix Ops, and Context. The main area displays an event for `vkCmdDrawIndexed`. The `Index Buffer` is listed as `0x000001d78ea1f910 @ 0 B` and the `Index Type` is `UINT32`. The `Flags` include `Topology: TRIANGLE_LIST` and `Primitive Restart Enable: FALSE`. The `Input Assembly` section contains code for creating an `VkPipelineInputAssemblyStateCreateInfo` structure:

```
// Input assembly state describes how primitives are assembled
// This pipeline will assemble vertex data as triangle lists (though we only use one triangle)
VkPipelineInputAssemblyStateCreateInfo inputAssemblyStateCI{};
inputAssemblyStateCI.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
inputAssemblyStateCI.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
```

The `Vertex Attributes` table shows two attributes:

Location	Name	Binding	Format	Offset
0	inPos	0	R32G32B32_SFLOAT	0
1	inColor	0	R32G32B32_SFLOAT	12

The `Vertex Bindings` table shows one binding:

Binding	Stride	Input Rate	Divisor	Buffer
0	24	VERTEX	1	81 10 0x000001d78ea1ff10 @ 0 B

A callout box highlights the buffer address `0x000001d78ea1ff10 @ 0 B` in the `vertexInputBinding.buffer` field of the table, with the following explanatory code:

```
// Vertex input binding
// This example uses a single vertex input binding at binding point 0 (see
VkVertexInputBindingDescription vertexInputBinding{};
vertexInputBinding.binding = 0;
vertexInputBinding.stride = sizeof(Vertex);
vertexInputBinding.inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
```

1. Frame Debugger - Resource View

Resource - 0x000001dce58118c0

Memory

Axis: Address ▾ View: 0x0 - 48 Width: 0 Precision: 4 Rep: Decimal ▾ Hex: Transpose Configure

	inPos[0]	inPos[1]	inPos[2]	inColor[0]	inColor[1]	inColor[2]
0x00000000	1.0000	1.0000	0.0000	1.0000	0.0000	0.0000
0x00000018	-1.0000	1.0000	0.0000	0.0000	1.0000	0.0000
0x00000030	0.0000	-1.0000	0.0000	0.0000	0.0000	1.0000

Cell: 0 0 Source Data Size: 72 | View Hash: 0xcde1ccd8

▼ Resource Info (1 revision)

Type:	Buffer
Name:	B 0x0...
Revision:	0
Memory Pool:	0x0000...
Size:	72 B
# Consumptions:	1

01
10
Rev: 0 Evt: 0

1. Frame Debugger - API Inspector

The screenshot shows the API Inspector interface. On the left, there is a sidebar with several tabs: Pipeline, Render Pass, IA, Viewport (which is highlighted with a yellow border), VS, Raster, FS, Pix Ops, and Context. The main area displays an event log for "Event 17 - vkCmdDrawIndexed". The log includes a search bar and icons for zooming and filtering. Below the search bar, there are two sections: "Viewports" and "Scissors". The "Viewports" section shows a table with one row:

Index	X	Y	Width	Height	Depth
0	0.00	0.00	1280.00	720.00	[0.00 - 1.00]

The "Scissors" section shows a table with one row:

Index	X	Y	Width	Height
0	0	0	1280	720

To the right of the tables, a code block shows the corresponding Vulkan command code:

```
// Update dynamic viewport state
VkViewport viewport{};
viewport.height = (float)height;
viewport.width = (float)width;
viewport.minDepth = (float)0.0f;
viewport.maxDepth = (float)1.0f;
vkCmdSetViewport(commandBuffers[currentBuffer], 0, 1, &viewport);
// Update dynamic scissor state
VkRect2D scissor{};
scissor.extent.width = width;
scissor.extent.height = height;
scissor.offset.x = 0;
scissor.offset.y = 0;
vkCmdSetScissor(commandBuffers[currentBuffer], 0, 1, &scissor);
```

1. Frame Debugger - API Inspector

The screenshot shows the API Inspector interface for a Vulkan command buffer. The left sidebar lists various stages: Pipeline, Render Pass, IA, Viewport, VS (Vertex Shader, highlighted with a yellow box), Raster, FS (Fragment Shader), Pix Ops, and Context. The main pane displays an event for `vkCmdDrawIndexed`. The Pipeline Shader Stage is expanded, showing inputs for `inPos` and `inColor`, and an output for `outColor`. The Uniform & Storage Buffers section shows a UBO named `ubo` at offset 0 with a value of `[...]`. A yellow box highlights the `Buffer` column entry, which is `0x000001d78ea20090 @ 0 B`.

Event 17 - `vkCmdDrawIndexed`(`VkCommandBuffer commandBuffer = '0x000001d78ea0f520'`, `uint32_t indexCount = 3`, `uint32_t instanceCount = 1`, `uint32_t firstIndex = 0`, `int32_t ver`)

Search...

▼ Pipeline Shader Stage

Flags: -- Stage: VERTEX Module: [0x000001d78e960860](#) Code: [View Source](#)

▼ Inputs

Location	Name	Type
0	<code>inPos</code>	<code>vec3f*</code>
1	<code>inColor</code>	<code>vec3f*</code>

▼ Outputs

Location	Name	Type
0	<code>outColor</code>	<code>vec3f*</code>

▼ Uniform & Storage Buffers

Width: 0 Precision: 2 Rep: Scientific Hex:

Name	Value	Type	Offset	Range	Set	Binding	Descriptor Type	Buffer
ubo	[...]	struct UBO*	0	192	0	0	UNIFORM_BUFFER	<code>0x000001d78ea20090 @ 0 B</code>

1. Frame Debugger - API Inspector

Resource - 0x000001e201586e00

Memory

Axis: Address View: 0x0 - 0xc0 Width: 0 Precision: 4 Rep: Decimal Hex: Transpose Configure

	ubo.projectionMatrix	ubo.modelMatrix	ubo.viewMatrix
0x00000000	[0.9743, 0.0000, 0.0000, 0.0000] [0.0000, 1.7321, 0.0000, 0.0000] [0.0000, 0.0000, -1.0039, -1.0000] [0.0000, 0.0000, -1.0039, 0.0000]	[1.0000, 0.0000, 0.0000, 0.0000] [0.0000, 1.0000, 0.0000, 0.0000] [0.0000, 0.0000, 1.0000, 0.0000] [0.0000, 0.0000, 0.0000, 1.0000]	[1.0000, 0.0000, 0.0000, 0.0000] [0.0000, 1.0000, 0.0000, 0.0000] [0.0000, 0.0000, 1.0000, 0.0000] [0.0000, 0.0000, -2.5000, 1.0000]

Cell: 0 0 0 Source Data Size: 192 | View Hash: 0xe06c48f5

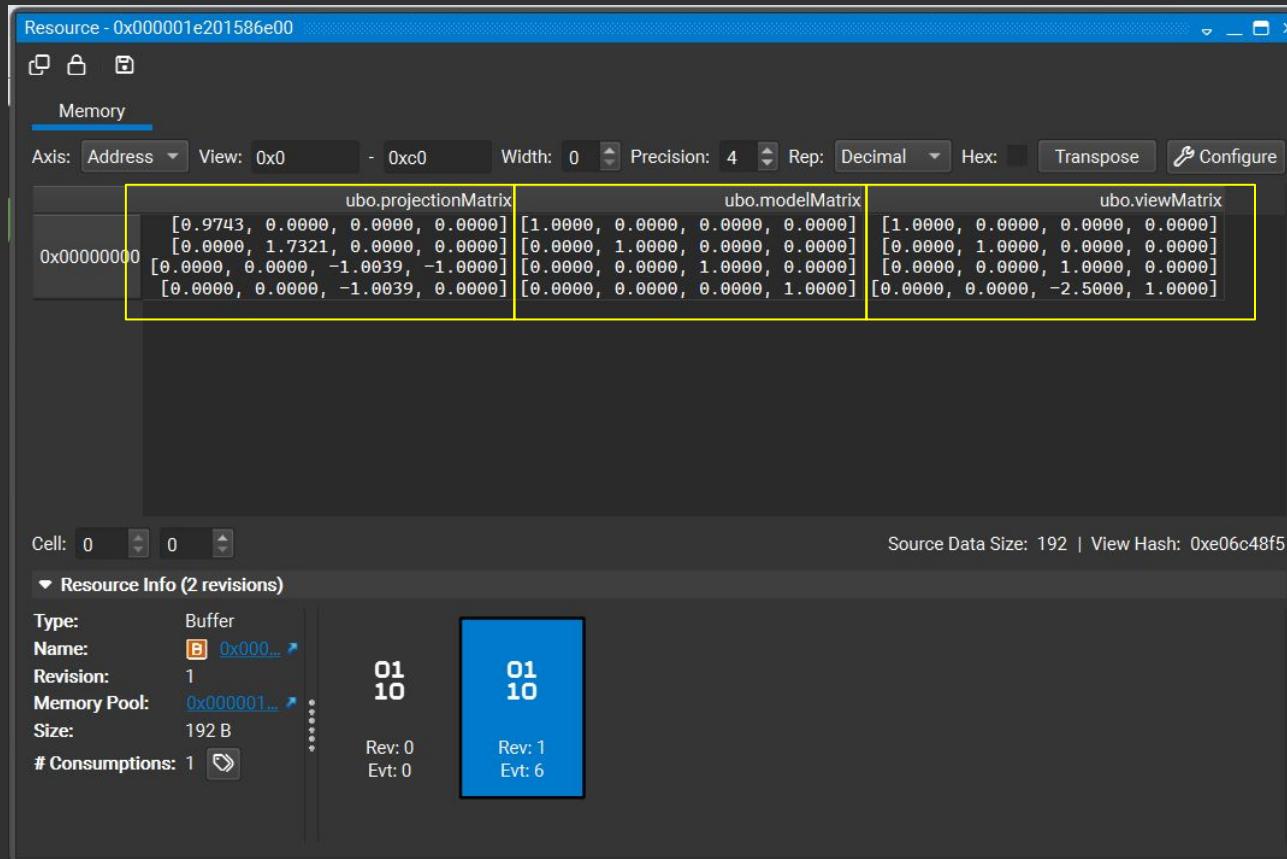
Resource Info (2 revisions)

Type:	Buffer
Name:	0x000...
Revision:	1
Memory Pool:	0x000001...
Size:	192 B
# Consumptions:	1

01
10

01
10

Rev: 0 Evt: 0 Rev: 1 Evt: 6



1. Frame Debugger - API Inspector

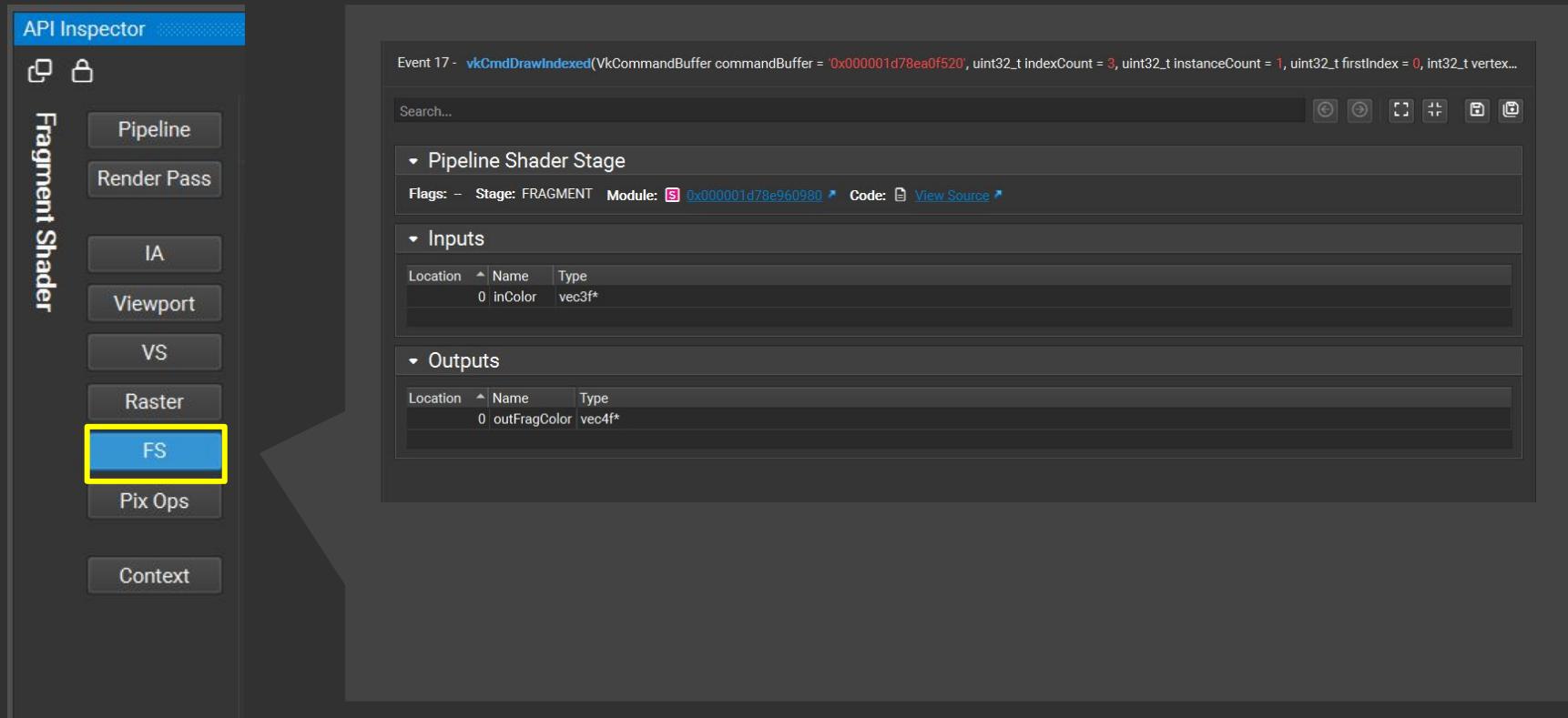
The screenshot shows the API Inspector interface with the "Rasterization" tab selected (indicated by a yellow border). The main pane displays a table of rasterization properties and their values, with a code snippet below it.

Event 17 - `vkCmdDrawIndexed`

Property	Value
Depth Clamp Enable	FALSE
Rasterizer Discard Enable	FALSE
Polygon Mode	FILL
Cull Mode	-
Front Face	COUNTER_CLOCKWISE
Depth Bias Enable	FALSE
Depth Bias Constant Factor	0.00
Depth Bias Clamp	0.00
Depth Bias Slope Factor	0.00
Line Width	1.00

```
// Rasterization state
VkPipelineRasterizationStateCreateInfo rasterizationStateCI{};
rasterizationStateCI.sType = VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO;
rasterizationStateCI.polygonMode = VK_POLYGON_MODE_FILL;
rasterizationStateCI.cullMode = VK_CULL_MODE_NONE;
rasterizationStateCI.frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
rasterizationStateCI.depthClampEnable = VK_FALSE;
rasterizationStateCI.rasterizerDiscardEnable = VK_FALSE;
rasterizationStateCI.depthBiasEnable = VK_FALSE;
rasterizationStateCI.lineWidth = 1.0f;
```

1. Frame Debugger - API Inspector



1. Frame Debugger - API Inspector

The screenshot shows the API Inspector interface with the 'Pix Ops' tab selected (highlighted with a yellow box). The main pane displays an event for `vkCmdDrawIndexed` with the following details:

Event 17 - `vkCmdDrawIndexed(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, uint32_t indexCount = 3, uint32_t instanceCount = 1, uint32_t firstIndex = 0, int32_t vertex...`

Depth Stencil State

Property	Value
Depth Test Enable	TRUE
Depth Write Enable	TRUE
Depth Compare Op.	LESS_OR_EQUAL
Depth Bounds Test Enable	FALSE
Stencil Test Enable	FALSE
Stencil Fail Op. (Front)	KEEP
Stencil Pass Op. (Front)	KEEP
Stencil Depth Fail Op. (Front)	KEEP
Stencil Compare Op. (Front)	ALWAYS
Stencil Compare Mask (Front)	0x00000000
Stencil Write Mask (Front)	0x00000000
Stencil Reference (Front)	0x00000000
Stencil Fail Op. (Back)	KEEP
Stencil Pass Op. (Back)	KEEP
Stencil Depth Fail Op. (Back)	KEEP
Stencil Compare Op. (Back)	ALWAYS
Stencil Compare Mask (Back)	0x00000000
Stencil Write Mask (Back)	0x00000000
Stencil Reference (Back)	0x00000000
Depth Bounds	[0.00 - 0.00]

Code block (highlighted in green):

```
// Depth and stencil state containing depth and stencil compare and test operations
// We only use depth tests and want depth tests and writes to be enabled and compare with
VkPipelineDepthStencilStateCreateInfo depthStencilStateCI{};
depthStencilStateCI.sType = VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO;
depthStencilStateCI.depthTestEnable = VK_TRUE;
depthStencilStateCI.depthWriteEnable = VK_TRUE;
depthStencilStateCI.depthCompareOp = VK_COMPARE_OP_LESS_OR_EQUAL;
depthStencilStateCI.depthBoundsTestEnable = VK_FALSE;
depthStencilStateCI.back.failOp = VK_STENCIL_OP_KEEP;
depthStencilStateCI.back.passOp = VK_STENCIL_OP_KEEP;
depthStencilStateCI.back.compareOp = VK_COMPARE_OP_ALWAYS;
depthStencilStateCI.stencilTestEnable = VK_FALSE;
depthStencilStateCI.front = depthStencilStateCI.back;
```

1. Frame Debugger - API Inspector

The screenshot shows the API Inspector interface. On the left, there is a sidebar with several tabs: Pipeline, Render Pass, IA, Viewport, VS, Raster, FS, Pix Ops, and Context. The Context tab is highlighted with a yellow border. The main area displays an event log and a properties table.

Event 17 - `vkCmdDrawIndexed(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, uint32_t indexCount = 3, uint32_t instanceCount = 1, uint32_t firstIndex = 0, int32_t vertex...`

Search...

Property	Value
Instance	I 0x000001d78c96fa50 ↗
Physical Device	P 0x000001d78d940df0 ↗
Device	D 0x000001d78e3d0670 ↗
Queue	Q 0x000001d78e5a8af0 ↗
Command Buffer	C 0x000001d78ea0f520 ↗
Pipeline	P 0x000001d78ea63fe0 ↗
Pipeline Layout	O 0x000001d78ea46bc0 ↗
Render Pass	R 0x000001d78e2250c0 ↗
Framebuffer	F 0x000001d78e960500 ↗
Descriptor Set 0	D 0x000001d78ea0f6a0 ↗

1. Frame Debugger - API Inspector

The screenshot shows the API Inspector interface. On the left, there's a sidebar with tabs: Pipeline, Context, and CS (which is highlighted with a yellow box). The main area displays an event for `vkCmdDispatch`. The event details are as follows:

Event 13 - `vkCmdDispatch(VkCommandBuffer commandBuffer = 0x000001ea28be0460, uint32_t groupCountX = 1024, uint32_t groupCountY = 1, uint32_t groupCountZ = 1)`

Flags: - Stage: COMPUTE Module: 0x000001ea28cfa760 Code: [View Source](#)

Pipeline Shader Stage

Uniform & Storage Buffers

Name	Value	Type	Offset	Range	Set	Binding	Descriptor Type	Buffer
%99	[...]	struct Pos*	0	VK_WHOLE_SIZE	0	0	STORAGE_BUFFER	0x000001ea28da4830 @ 0 B
ubo	[...]	struct UBO*	0	VK_WHOLE_SIZE	0	1	UNIFORM_BUFFER	0x000001ea26e7c490 @ 0 B

1. Frame Debugger - API Inspector

Tessellation Control Shader



The main window is titled "Event 22 - `vkCmdDrawIndexed`". The pipeline stage is identified as "Pipeline Shader Stage" with "Stage: TESSELLATION_CONTROL". The module is listed as "Module: 0x0000028f82b84cf0". The code section includes a "View Source" link. The "Inputs" section shows two inputs: "inNormal" (vec3f*) and "inUV" (vec2f*). The "Outputs" section shows two outputs: "outNormal" (vec3f*) and "outUV" (vec2f*).

Location	Name	Type
0	inNormal	vec3f*
1	inUV	vec2f*

Location	Name	Type
0	outNormal	vec3f*
1	outUV	vec2f*

1. Frame Debugger - API Inspector

Tessellation Evaluation Shader

Pipeline

Render Pass

IA

Viewport

VS

TCS

TES

Raster

FS

Pix Ops

Context

Event 22 - `vkCmdDrawIndexed`(`VkCommandBuffer commandBuffer = '0x0000028f81c008c0'`, `uint32_t indexCount = 10764`, `uint32_t instanceCount = 1`, `uint32_t firstIndex = 0`, `uint32_t vertexOffset = 0`, `const VkIndex* pIndices = '0x0000028f81c008d0'`)

Search...

▼ Pipeline Shader Stage

Flags: - Stage: TESSELLATION_EVALUATION Module: [0x0000028f82b85b90](#) Code: [View Source](#)

▼ Inputs

Location	Name	Type
0	inNormal	vec3f*
1	inUV	vec2f*

▼ Outputs

Location	Name	Type
0	outNormal	vec3f*
1	outUV	vec2f*

▼ Uniform & Storage Buffers

Width: 0 Precision: 2 Rep: Decimal Hex:

Name	Value	Type	Offset	Range	Set	Binding	Descriptor Type	Buffer
ubo	[...]	struct UBO*	0	VK_WHOLE_SIZE	0	1	UNIFORM_BUFFER	0x0000028f81df0e90 @ 0 B

1. Frame Debugger - API Statistics

The screenshot shows the NVIDIA Nsight Graphics interface during a live analysis session. The main window is divided into several panels:

- Events:** A table listing GPU events with columns for Event ID, Description, CPU ms, and GPU ms. Event 17, "vkCmdDrawIndexed", is highlighted.
- Scrubber:** A timeline showing the sequence of events. Event 17 is highlighted in orange.
- Current Target:** Displays two render targets: "Color 0" showing a colored triangle and "Depth" showing a depth map.
- API Inspector:** Provides detailed information for the selected event (Event 17). The "Context" tab is active, showing properties like Instance, Physical Device, Device, Queue, Command Buffer, Pipeline, Pipeline Layout, Render Pass, Framebuffer, and Descriptor Set 0, each with a corresponding memory address.
- API Statistics:** A panel at the bottom left showing the command buffer details for Event 17. It includes fields for commandBuffer, indexCount, instanceCount, firstIndex, vertexOffset, and firstInstance.

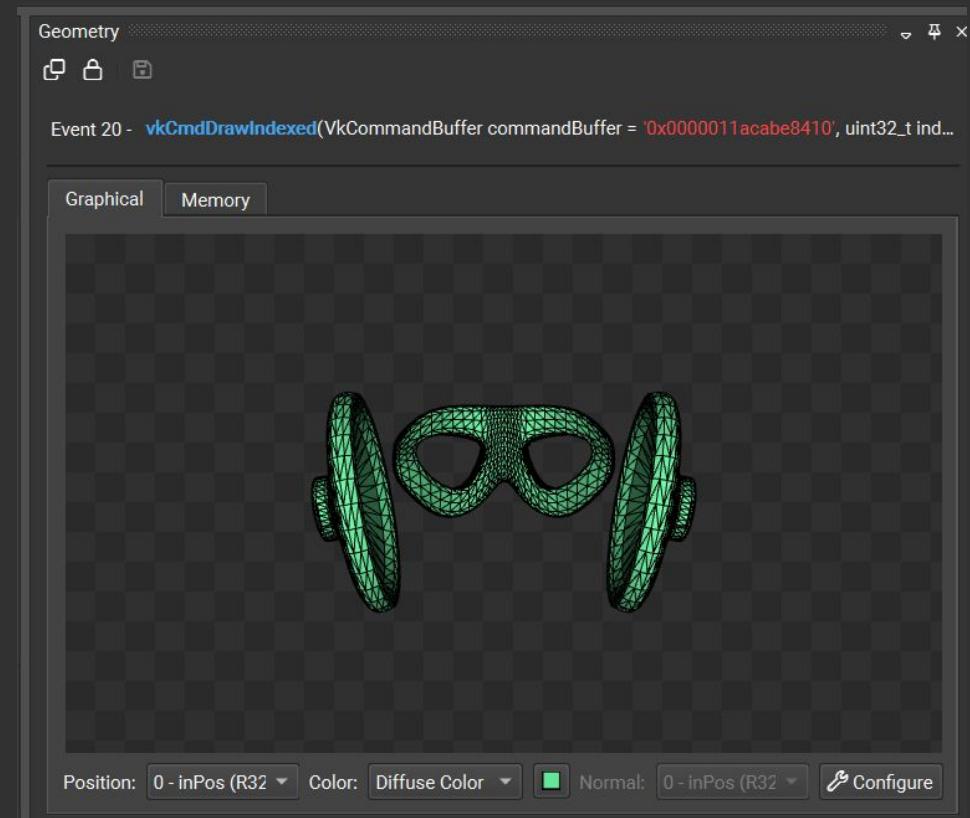
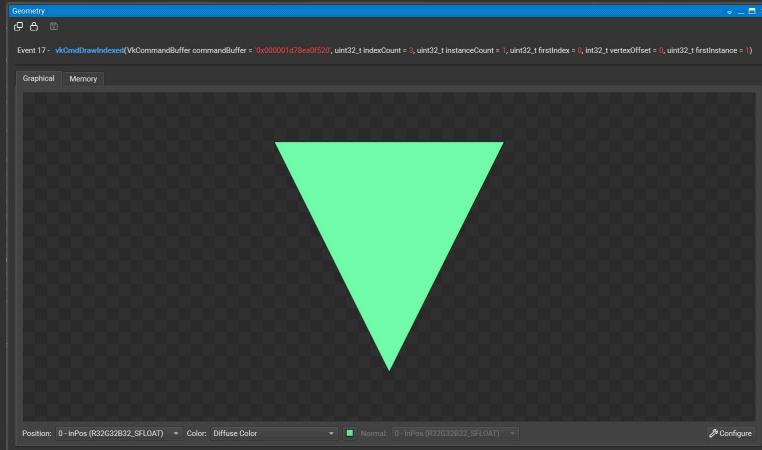
A yellow box highlights the "Events" and "API Statistics" tabs in the bottom-left corner. The status bar at the bottom right indicates the process name "triangle.exe [15584]" and frame number "Frame #146".

1. Frame Debugger - API Statistics

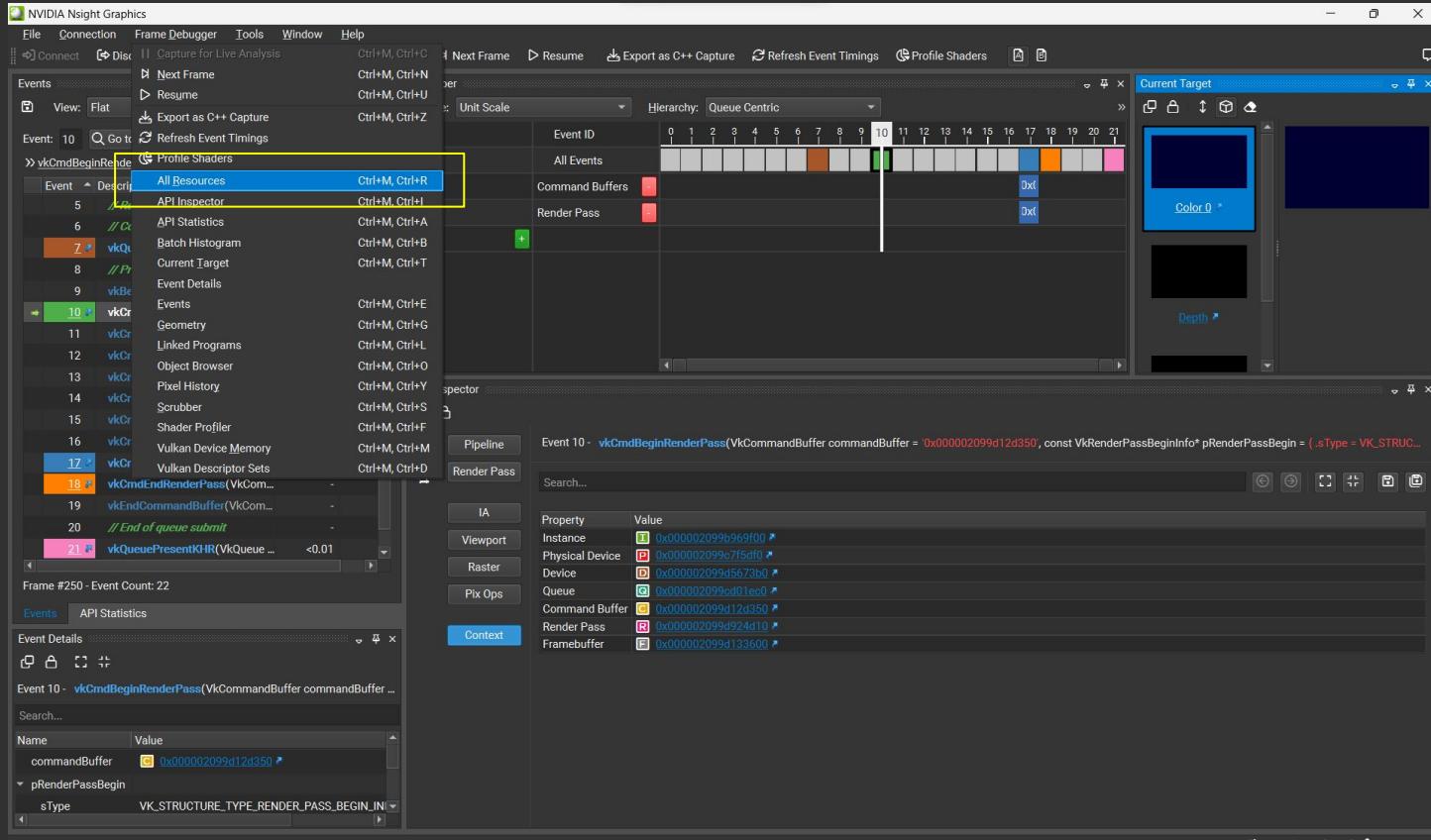
API Statistics					
Summary					
API Call	Count	Avg CPU ms	Σ CPU ms	Avg GPU ms	Σ GPU ms
vkWaitForFences()	1	0.01	0.01	0.00	0.00
vkAcquireNextImageKHR()	1	0.16	0.16	0.00	0.00
vkResetFences()	1	<0.01	<0.01	0.00	0.00
vkResetCommandBuffer()	1	0.24	0.24	0.00	0.00
vkQueueSubmit()	1	0.77	0.77	0.00	0.00
vkBeginCommandBuffer()	1	0.00	0.00	0.00	0.00
vkCmdBeginRenderPass()	1	0.00	0.00	0.00	0.00
vkCmdSetViewport()	1	0.00	0.00	0.00	0.00
vkCmdSetScissor()	1	0.00	0.00	0.00	0.00
vkCmdBindDescriptorSets()	1	0.00	0.00	0.00	0.00
vkCmdBindPipeline()	1	0.00	0.00	0.00	0.00
vkCmdBindVertexBuffers()	1	0.00	0.00	0.00	0.00
vkCmdBindIndexBuffer()	1	0.00	0.00	0.00	0.00
vkCmdDrawIndexed()	1	0.00	0.00	<0.01	<0.01
vkCmdEndRenderPass()	1	0.00	0.00	0.00	0.00
vkEndCommandBuffer()	1	0.00	0.00	0.00	0.00
vkQueuePresentKHR()	1	<0.01	<0.01	0.00	0.00

1. Frame Debugger - Geometry View

What the current draw call is trying to draw



1. Frame Debugger - All Resources



1. Frame Debugger - All Resources

All Resources

Drawables (3) Images (2) Buffers (4) Memory (5)

Filter: Enter a filter or select a predefined one on the right

Select a predefined filter

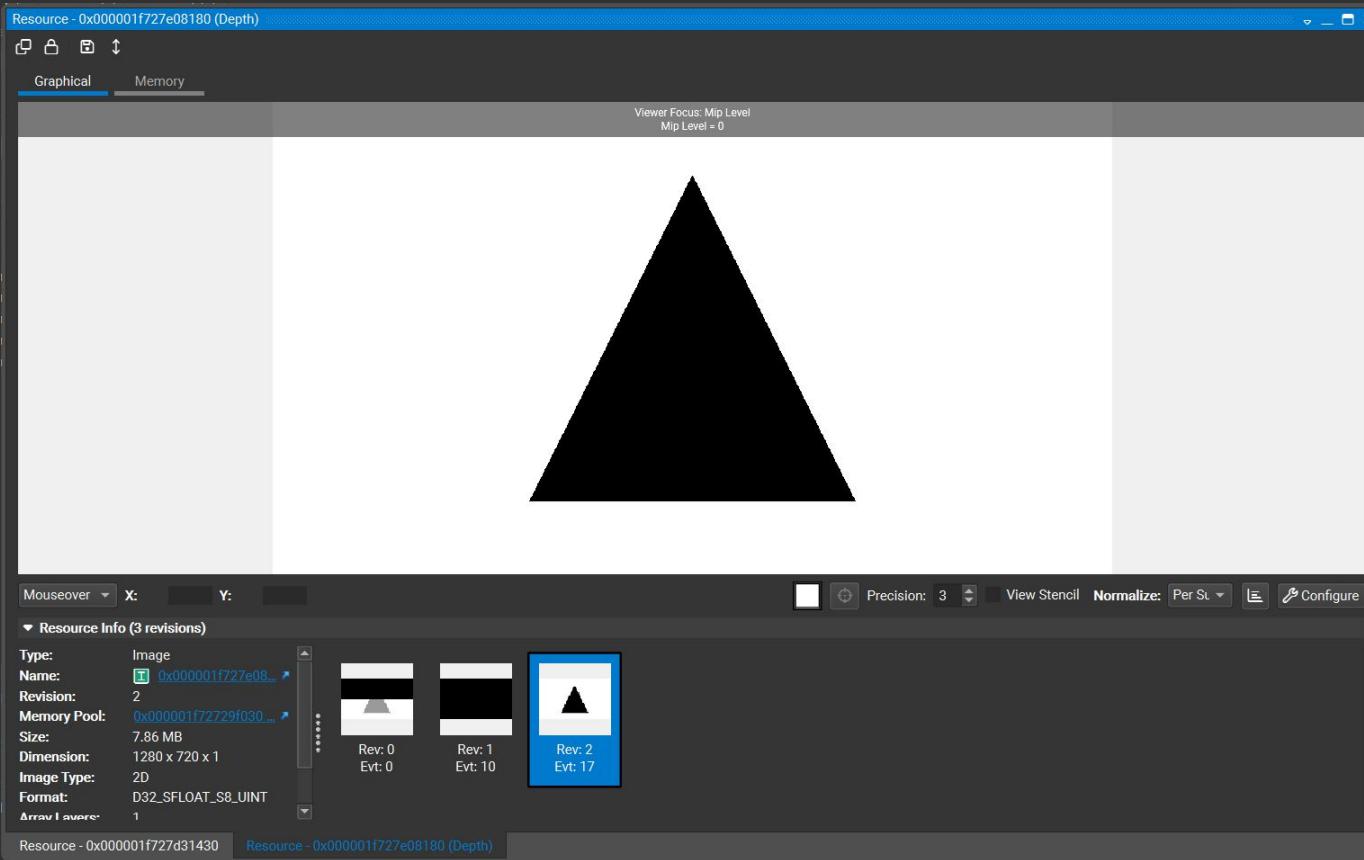
Name	Type	Revision	# Revisions	# Consumptions	Filter Group	Memory	Create Flags	Usage Flags	Sharing Mode	Queue Families	External Memory
0x000001d...	Swapchain Ima...	0	1	0	Drawables	3.93 MB	-	TRANSFER_SR...	EXCLUSIVE	-	-
0x000001d...	Swapchain Ima...	2	4	2	Drawables	3.93 MB	-	TRANSFER_SR...	EXCLUSIVE	-	-
0x000001d...	Swapchain Ima...	0	1	0	Drawables	3.93 MB	-	TRANSFER_SR...	EXCLUSIVE	-	-
0x000001d...	Image	2	3	2	Images	7.86 MB	-	DEPTH_STENCIL...	EXCLUSIVE	-	-
0x000001d...	Image	2	3	2	Images	7.86 MB	-	DEPTH_STENCIL...	EXCLUSIVE	-	-
0x000001d...	Buffer	0	1	0	Buffers	192 B	-	UNIFORM_BUF...	EXCLUSIVE	-	-
0x000001d...	Buffer	0	1	1	Buffers	12 B	-	TRANSFER_DS...	EXCLUSIVE	-	-
0x000001d...	Buffer	0	1	1	Buffers	72 B	-	TRANSFER_DS...	EXCLUSIVE	-	-
0x000001d...	Buffer	1	2	1	Buffers	192 B	-	UNIFORM_BUF...	EXCLUSIVE	-	-
0x000001d...	Memory Pool	0	1	0	Memory	192 B	-	-	-	-	-
0x000001d...	Memory Pool	0	1	1	Memory	80 B	-	-	-	-	-
0x000001d...	Memory Pool	0	1	1	Memory	12 B	-	-	-	-	-
0x000001d...	Memory Pool	2	3	2	Memory	7.86 MB	-	-	-	-	-
0x000001d...	Memory Pool	1	2	1	Memory	192 B	-	-	-	-	-

Resource Info (3 revisions)

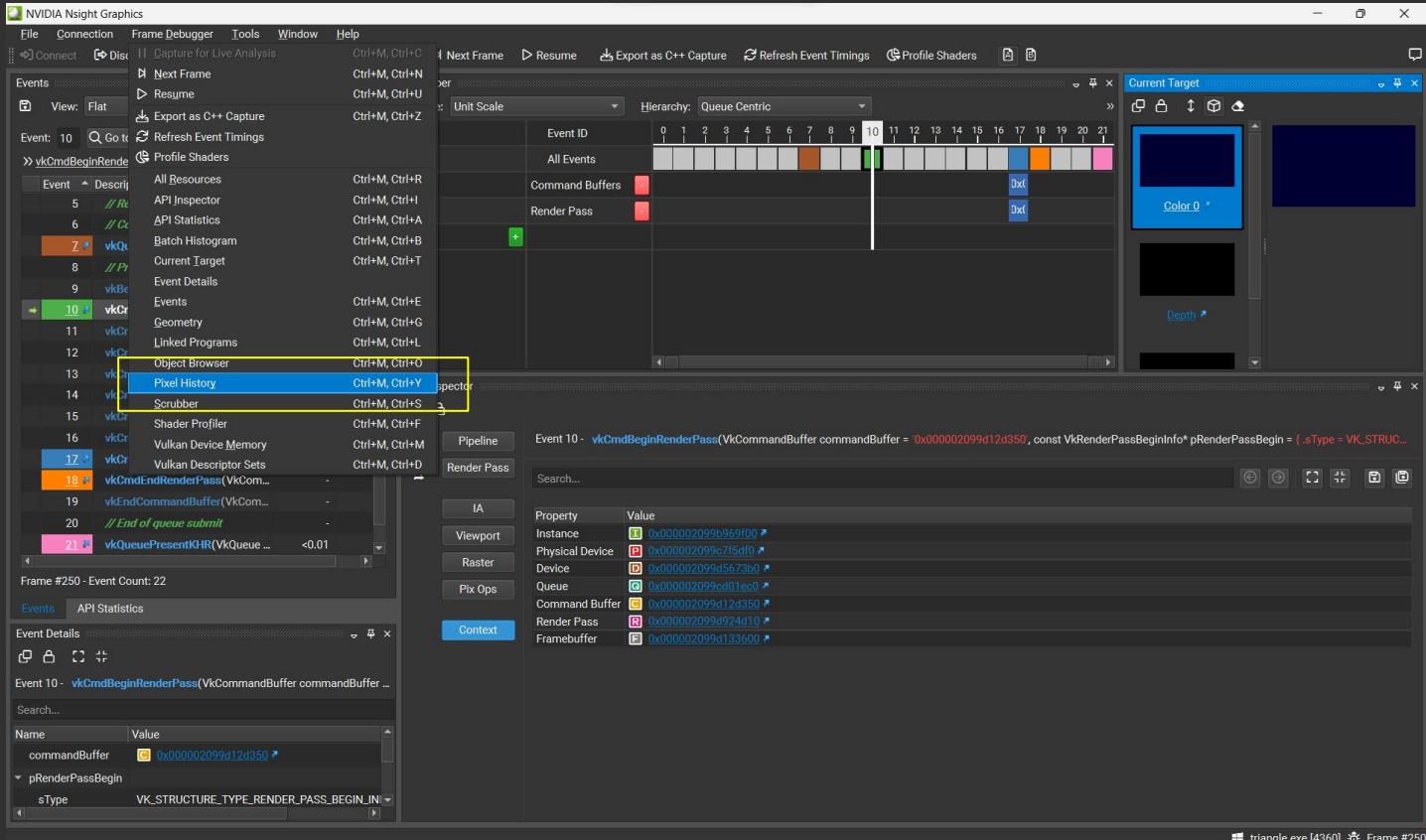
Type:	Image
Name:	0x000001d78ea04...
Revision:	2
Memory Pool:	0x000001d79da9c1e0...
Size:	7.86 MB
Dimension:	1280 x 720 x 1
Image Type:	2D
Format:	D32_SFLOAT_S8_UINT
Arrows aware:	1

Rev: 0 Evt: 0 Rev: 1 Evt: 10 Rev: 2 Evt: 17

1. Frame Debugger - All Resources



1. Frame Debugger - Pixel History



1. Frame Debugger - Pixel History

- Select the resource and revision you want to check
- Click the target button
- Select the pixel
- Check the pixel history below:

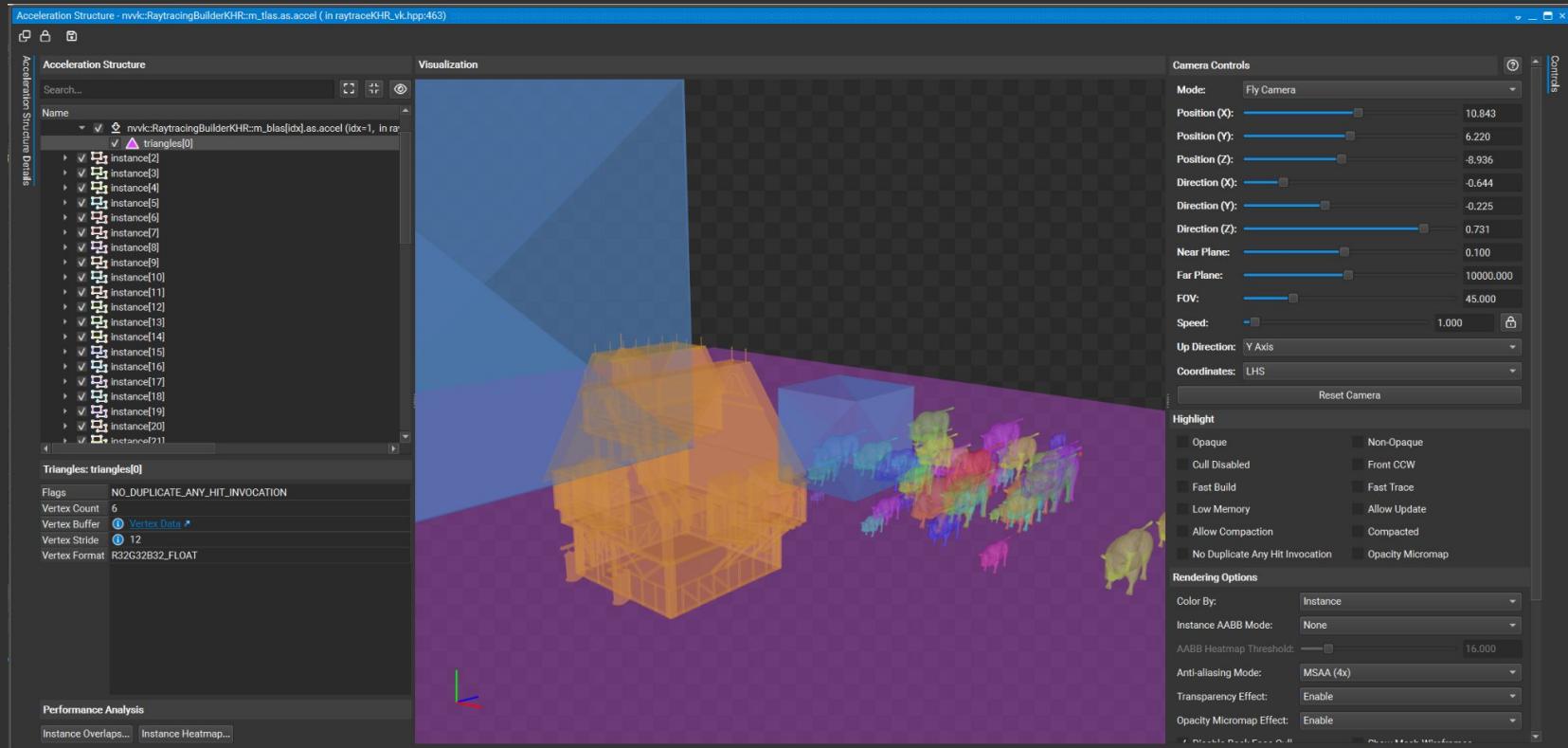
Pixel History								
		Description	Primitive	Pixel Before	Fragments	Pixel After	Status	Color Preview
Event				R: 0.576 G: 0.020 B: 0.404 A: 1.000	R: 0.000 G: 0.000 B: 0.200 A: 1.000	R: 0.000 G: 0.000 B: 0.200 A: 1.000	1 Passed	
10	+	vkCmdBeginRe...						
17	+	vkCmdDrawInd...	Prim: 0					

2. Frame Debugger - Acceleration Structure View

- If you have an acceleration structure in your scene, you can see it from your list of “All Resources”

Name	Type	Revision	# Revisions	# Consumptions	Filter Group	Memory	Create Flags	Usage Flags	Sharing Mode
Filter: Enter a filter or select a predefined one on the right									
0x000001c...	Acceleration Str...	0	1	1	AS	–	–	–	–
0x000001c...	Acceleration Str...	0	1	0	AS	–	–	–	–

2. Frame Debugger - Acceleration Structure View



1. Frame Debugger - Shader Profiler

NVIDIA Nsight Graphics

File Connection Frame Debugger Tools Window Help

Connect Disconnect Terminate Capture for Live Analysis Next Frame Resume Export as C++ Capture Refresh Event Timings Profile Shaders

Events

Event: 17 Go to Filter: Enter a regular expression

Event	Description	CPU ms	GPU ms
6	// Coherent Memory Update	-	-
7	vkQueueSubmit(VkQueue queue = 0x000...	0.77	-
8	// Primary command buffer 11 command...	-	-
9	vkBeginCommandBuffer(VkCommandBuf...	-	-
10	vkCmdBeginRenderPass(VkCommandBuf...	-	-
11	vkCmdSetViewport(VkCommandBuffer c...	-	-
12	vkCmdSetScissor(VkCommandBuffer co...	-	-
13	vkCmdBindDescriptorSets(VkCommandB...	-	-
14	vkCmdBindPipeline(VkCommandBuffer c...	-	-
15	vkCmdBindVertexBuffers(VkCommandBu...	-	-
16	vkCmdBindIndexBuffer(VkCommandBuff...	-	-
17	vkCmdDrawIndexed(VkCommandBuffer c...	<0.01	-
18	vkCmdEndRenderPass(VkCommandBuff...	-	-

Frame #146 - Event Count: 22

Events API Statistics

Event Details

Event 17 - vkCmdDrawIndexed(VkCommandBuffer commandBuffer = 0x000001d78ea0f520...

Search...

Name	Value	Type
commandBuffer	0x000001d78ea0f520	VkCommandBuffer
indexCount	3	uint32_t
instanceCount	1	uint32_t
firstIndex	0	uint32_t
vertexOffset	0	int32_t
firstInstance	1	uint32_t

Scrubber

Mode: Unit Scale Hierarchy: Queue Centric

Event ID 0 2 4 6 8 10 12 14 16 17 18 20

All Events

Render Pass Command Buffers

Add...

Current Target

Color 0 Depth

API Inspector

Context Pipeline Render Pass

Search...

IA Viewport VS Raster FS Pix Ops Context

Property Value

Property	Value
Instance	0x000001d78c96fa50
Physical Device	0x000001d78d940df0
Device	0x000001d78e3d0670
Queue	0x000001d78e5a8a10
Command Buffer	0x000001d78ea0f520
Pipeline	0x000001d78ea63fe0
Pipeline Layout	0x000001d78ea46bc0
Render Pass	0x000001d78e2250c0
Framebuffer	0x000001d78e960500
Descriptor Set 0	0x000001d78ea0f6a0

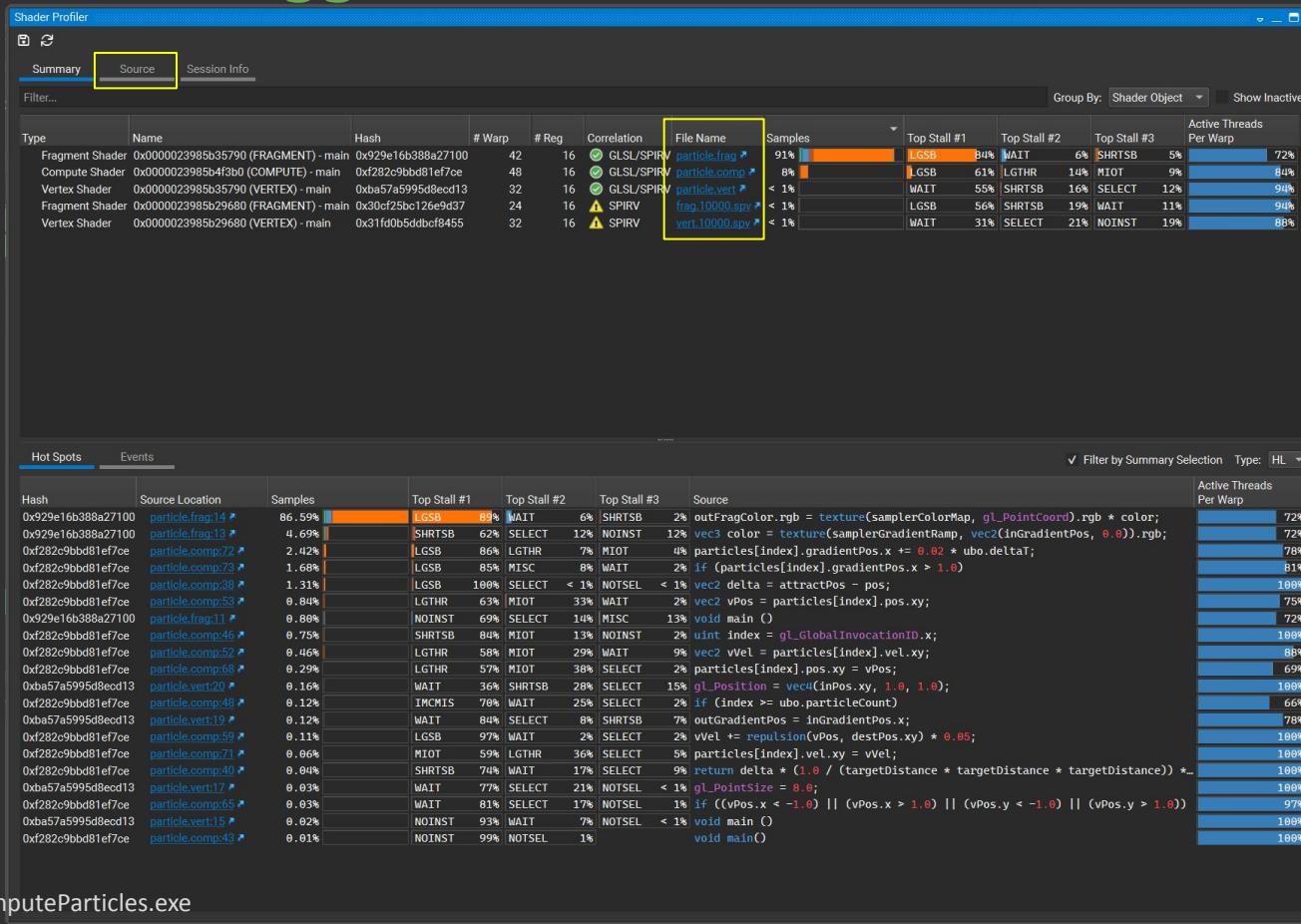
Sascha Willems - Triangle.exe

triangle.exe [15584] Frame #146

1. Frame Debugger - Shader Profiler

- Allows you to see how your shaders are performing
- Live compiling of shaders after editing!
- In order to see Shader correlation, you MUST compile your shaders with debug symbols:
https://docs.nvidia.com/nsight-graphics/UserGuide/index.html#gcd_aftermath_source_shader_debug_info
- In Vulkan, your shaders can be written in GLSL or HLSL, both get compiled to SPIR-V, but GLSL is more common.

1. Frame Debugger - Shader Profiler



1. Frame Debugger - Shader Profiler

The screenshot shows the NVIDIA Nsight Graphics Frame Debugger interface. At the top, there's a toolbar with tabs for "GLSL (Debug)", "Main*", "Edit Shader", "Compile Shader", "Promote Shader", "Compile Tool: Glslang (GLSL)", and some other icons.

The main area displays two versions of the same shader code:

```
0x000002f5201cd350 (FRAGMENT) - main - FS
Graphics Pipeline: 0x000002f5201cd350 Stage: Fragment Shader Module: 0x000002f5201cd350 (FRAGMENT) - main

#version 450
layout (binding = 0) uniform sampler2D samplerColorMap;
layout (binding = 1) uniform sampler2D samplerGradientRamp;
layout (location = 0) in vec4 inColor;
layout (location = 1) in float inGradientPos;
layout (location = 0) out vec4 outFragColor;
void main ()
{
    vec3 color = texture(samplerGradientRamp, vec2(inGradientPos, 0.0)).rgb;
    outFragColor.rgb = texture(samplerColorMap, gl_PointCoord).rgb * color;
}

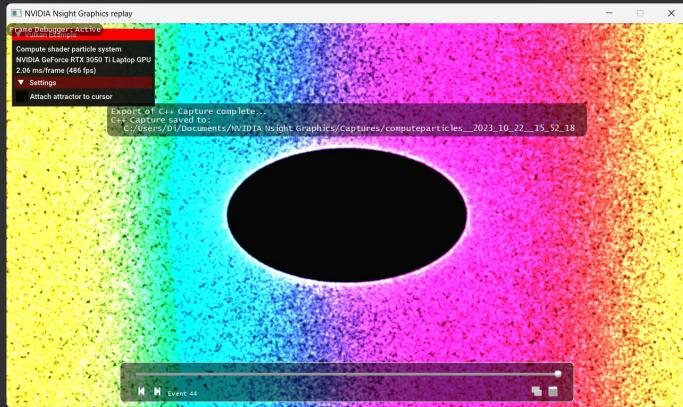
#version 450
layout (binding = 0) uniform sampler2D samplerColorMap;
layout (binding = 1) uniform sampler2D samplerGradientRamp;
layout (location = 0) in vec4 inColor;
layout (location = 1) in float inGradientPos;
layout (location = 0) out vec4 outFragColor;
void main ()
{
    vec3 color = texture(samplerGradientRamp, vec2(inGradientPos, 0.0)).rgb;
    outFragColor.rgb = vec3(0.0, 0.0, 1.0);
}
```

Below the code, there's a "Compile Results" section showing the compilation process:

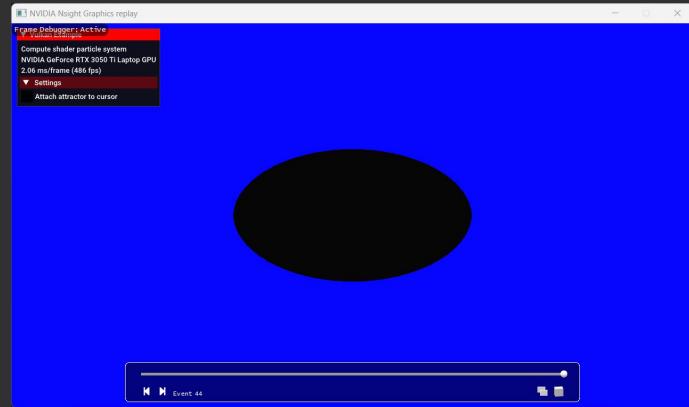
```
Compiling...
Running: cmd /c "C:/VulkanSDK/1.3.231.0/Bin/glslangValidator.exe" -g -V -o "C:/Users/Di/Documents/NVIDIA Nsight Graphics/vk_ray_tracing_advance_KHR/SpirvTemp/temp_output_file.spv" "C:/Users/Di/Documents/NVIDIA Nsight Graphics/vk_ray_tracing_advance_KHR/SpirvTemp/temp_input_file.frag"
Compile succeeded
Installing SPIR-V bytecode...
Install succeeded
```

A yellow box labeled "Original view" is positioned over the left side of the interface, and a yellow box labeled "Edit View" is positioned over the right side.

1. Frame Debugger - Shader Profiler



Original view



Edit View

Part 2

GPU Trace

Analyze the performance of your application, SM/Warp usage on
the GPU, Throughput, and Shaders.

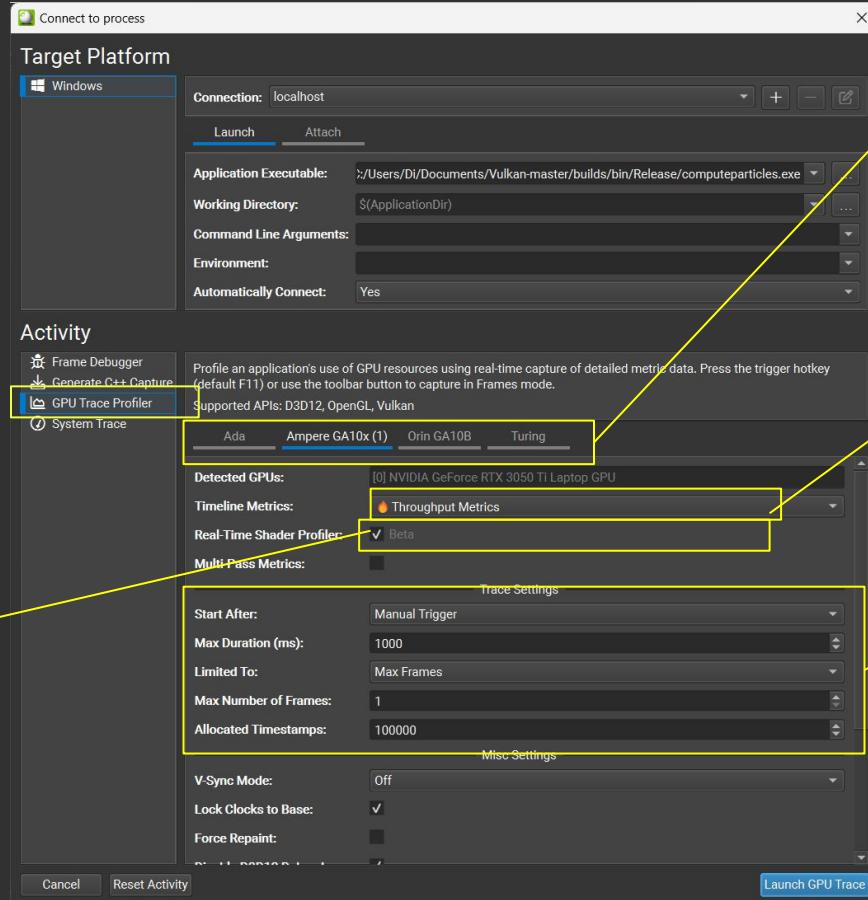
2. GPU Trace - Motivations

- WHAT can we do with GPU Trace?
 - Observe workloads running on the GPU over time
- WHERE
 - Identify which workloads are taking the most time and resources, and where in the frame they occur
- WHY
 - So we can determine the performance limiters: math-limited, bandwidth-limited, latency-limited, or starvation bound.
- End Goal: Decrease frame or workload runtime (ms, or increase fps)

2. GPU Trace - Overview

1. How to take a trace
2. Important Units/Metrics
3. Overview of Pixel Shader on the GPU
4. **Peak Perf % Method** - Intro to Triaging Workflows on the GPU
5. Case Study!
6. Walkthrough of UI and Features
 - a. Timeline Metrics
 - b. Shader Profiler

2. GPU Trace - Getting Started



Choose your GPU. Most of the time this is auto detected.

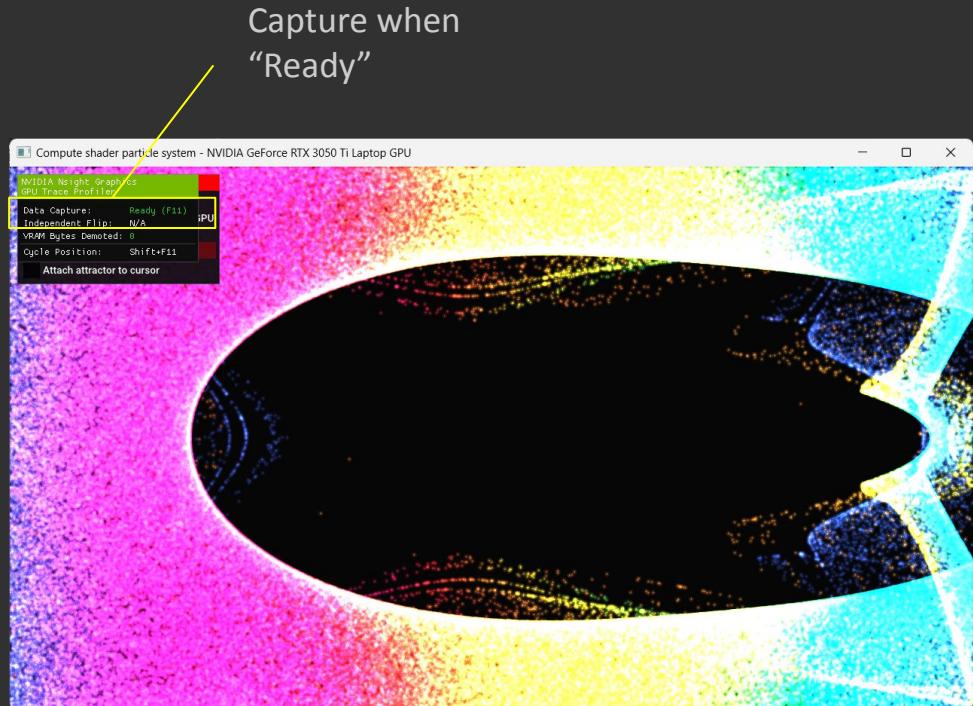
The type of metrics you want to measure (There are 2 options)

How do I want to trigger a capture?

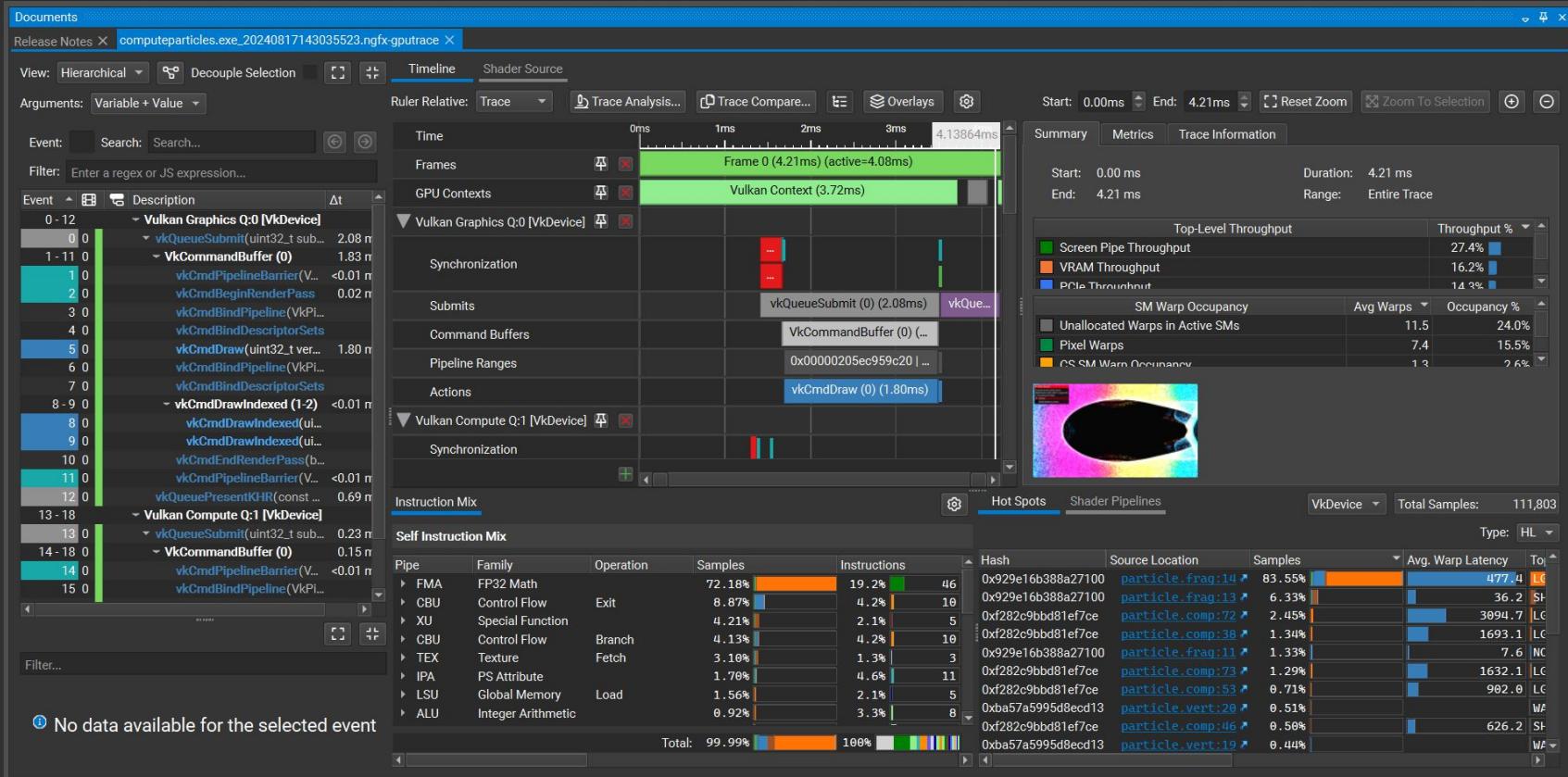
Turn off/on shader profiling information

2. GPU Trace - Getting Started

- If you configured manual capture, just press F11 when “Data Capture” is ready.
- Unlike Frame Debugger, no draw-by-draw panel will pop up.



2. GPU Trace



2. GPU Trace - Important Concepts

- GPU Starvation
 - You are not giving enough work to the GPU in general
- Latency Limitation
 - It's taking the GPU a very long time to complete operations.
 - **Examples:** Memory latency, pipeline stalls, context switching
- Throughput Limitation
 - The number of operations the unit can execute at any time has been exceeded
 - **Examples:** Memory bandwidth, instruction throughput, resource contests

2. GPU Trace - Important Metrics

Keep an eye on...

GR Active

Unit Throughput

SM Instruction
Throughput

SM Warp
Occupancy

Useful in detecting
Starvation.

GR = Graphics

Identify which HW
Units are the
performance limiters

What kind of math
operations are
limiters (if any),
sorted into pipes

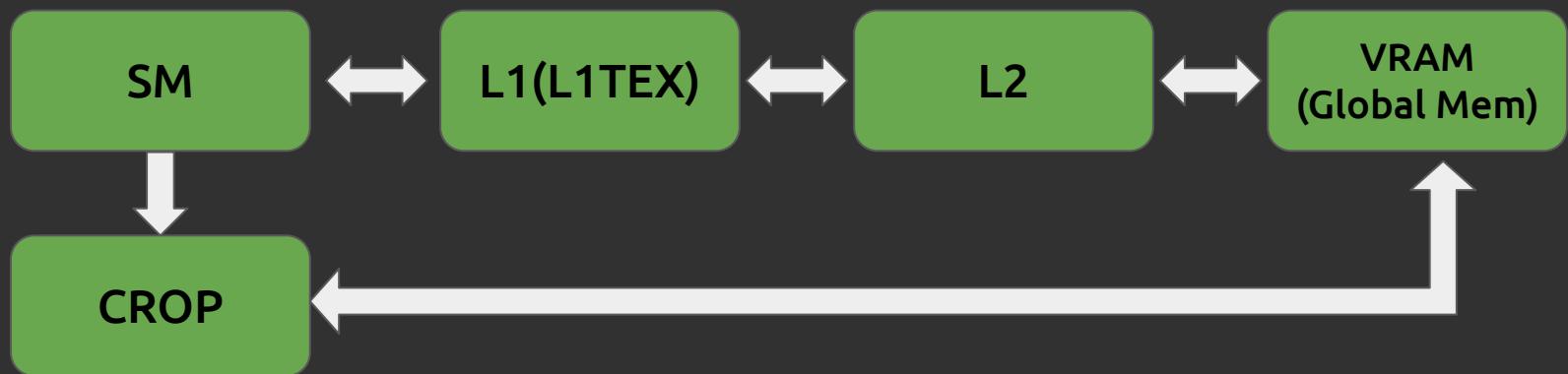
Goal: Get throughput
to 80%

On average, how
many warps were
active on an SM at
any moment?

What kind of warps
are they?

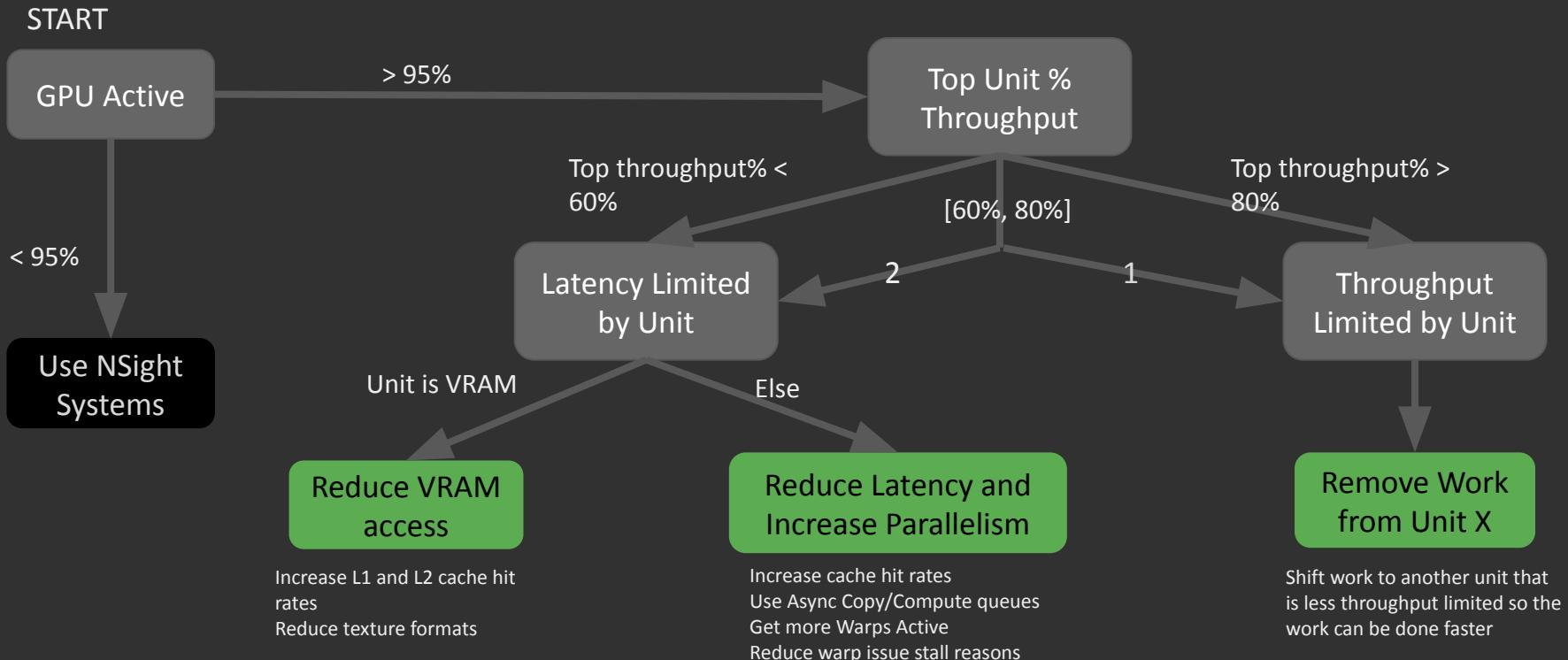
2. GPU Trace - Quick Review of Pixel Shaders on GPU

Unit Throughput % Metrics

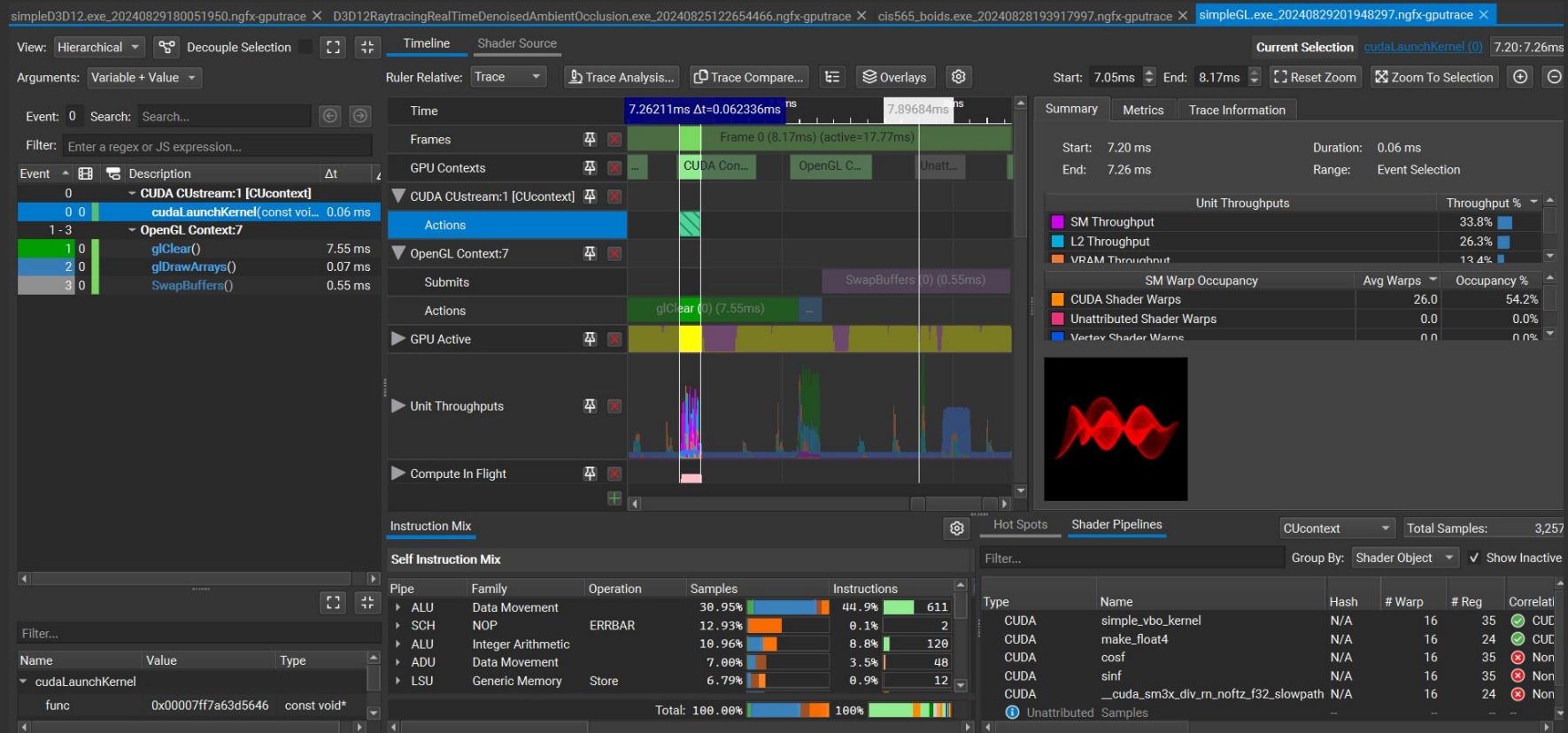


Throughput = % of maximum theoretical output achieved
Also known as “Speed of Light” or Peak-Performance % Metrics

2. GPU Trace - P3 (Peak Perf %) Method

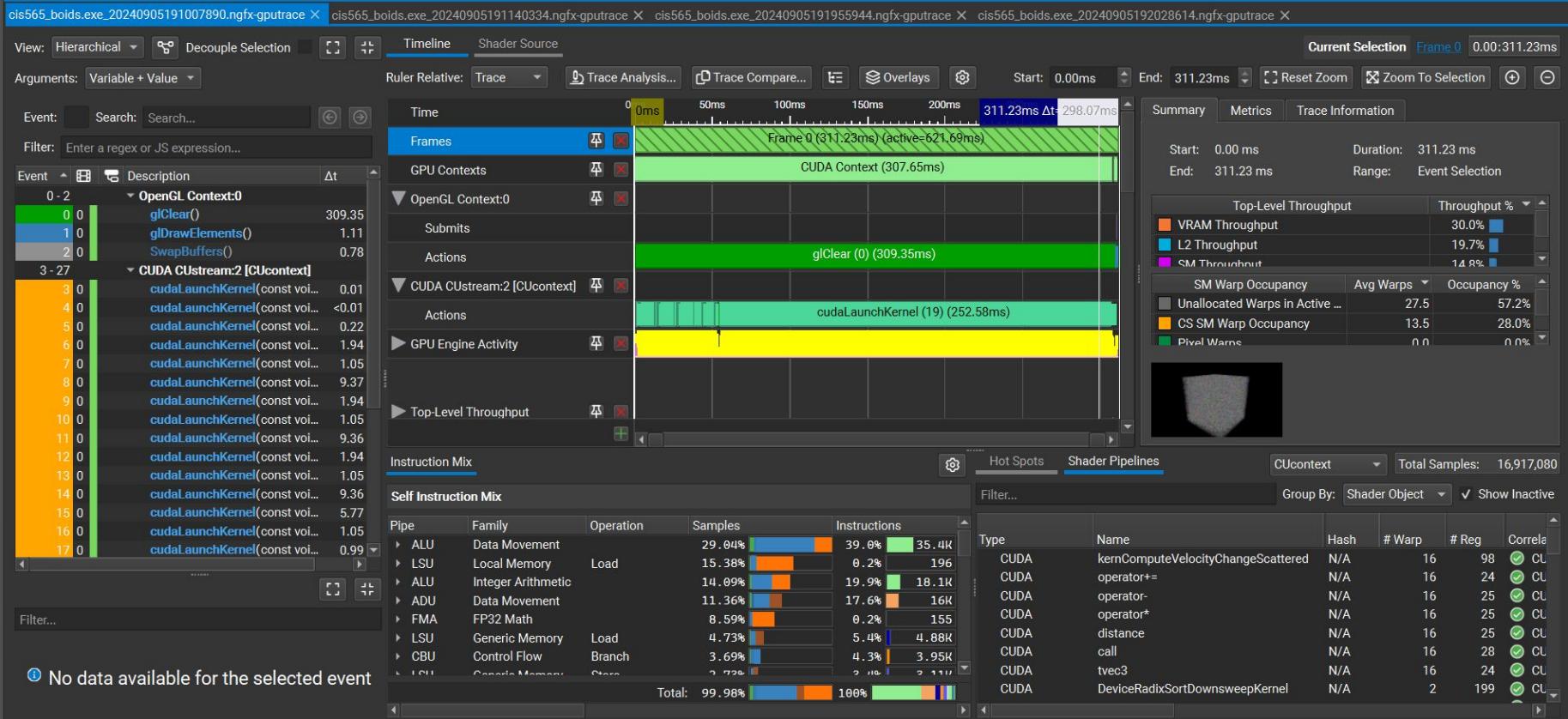


2. GPU Trace - CUDA Support

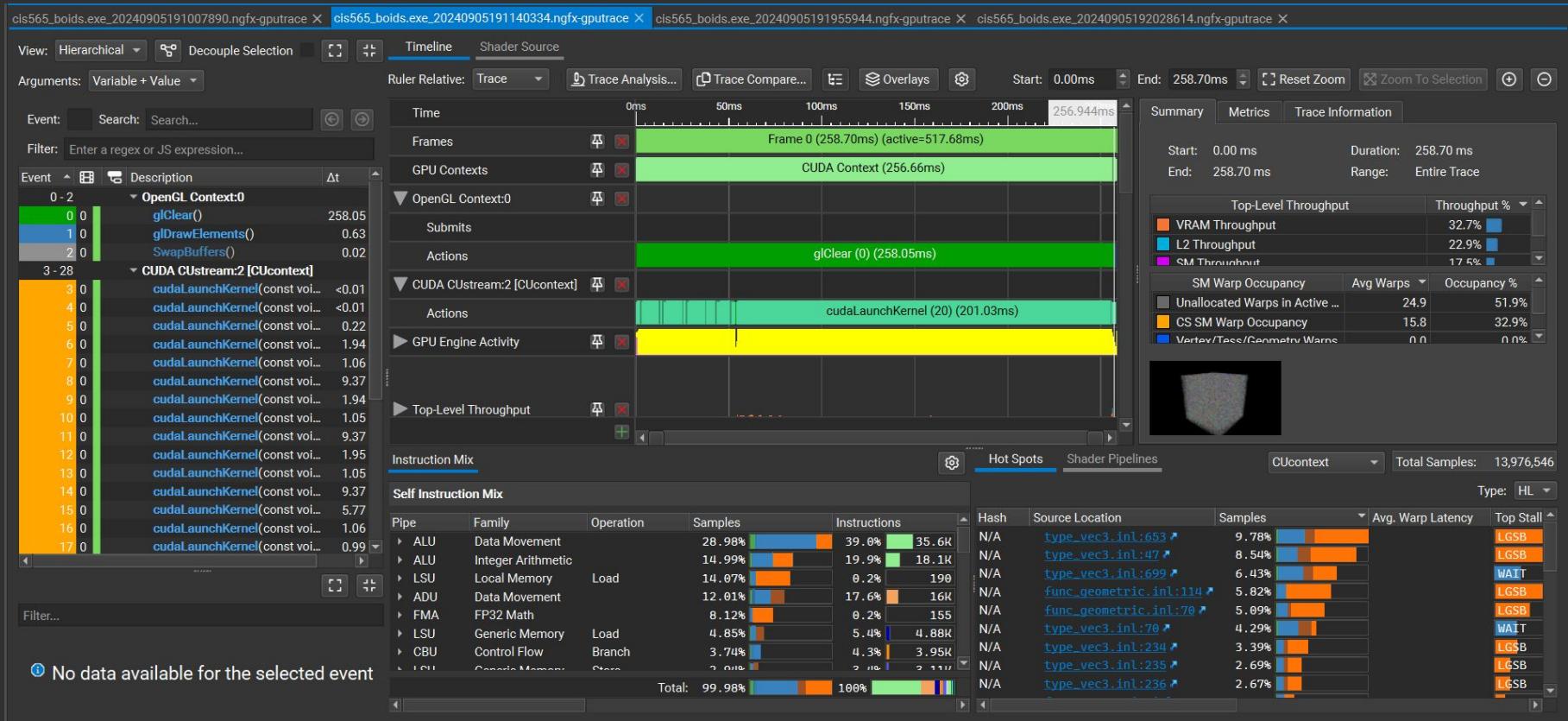


[Cuda-samples simpleGL.exe](#)

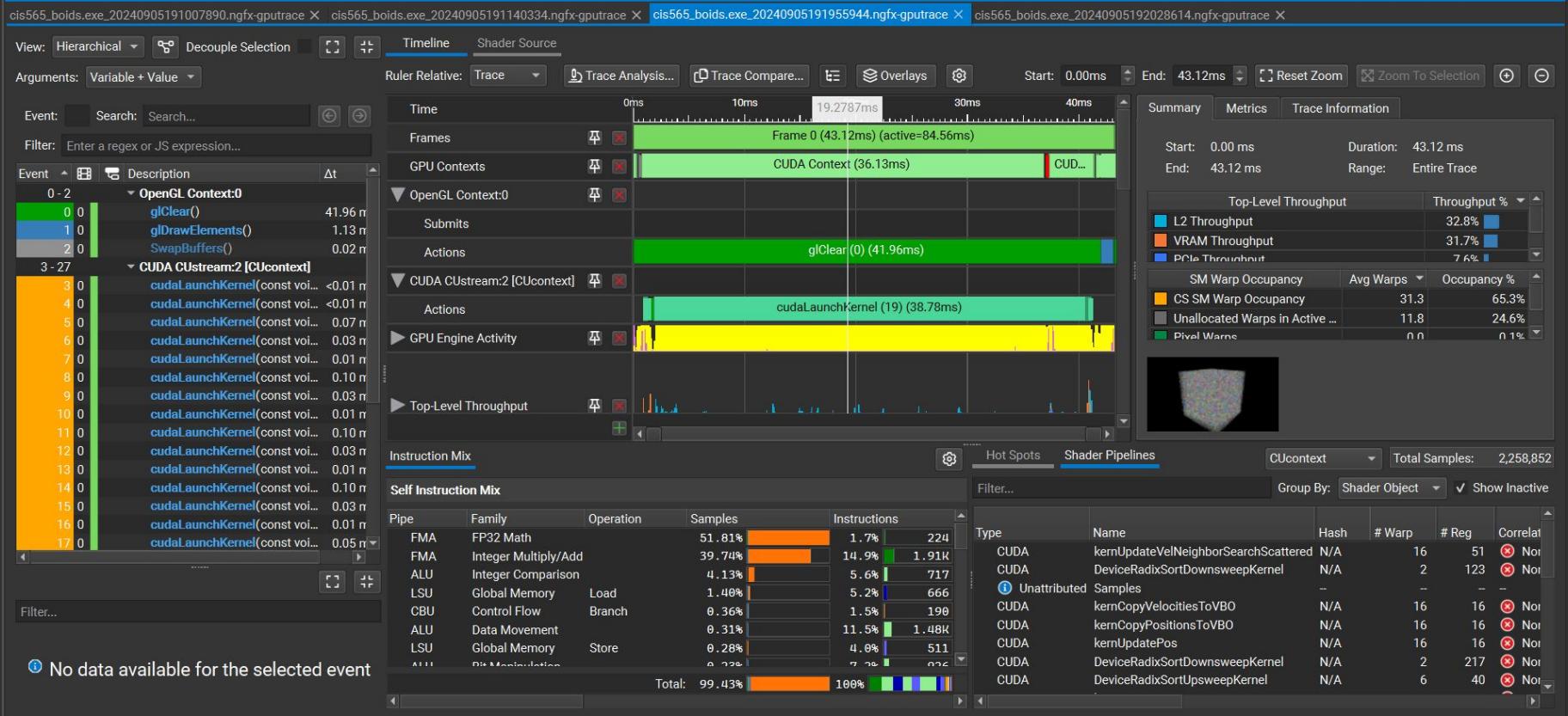
2. GPU Trace - Flocking (Uniform vs. Coherent)



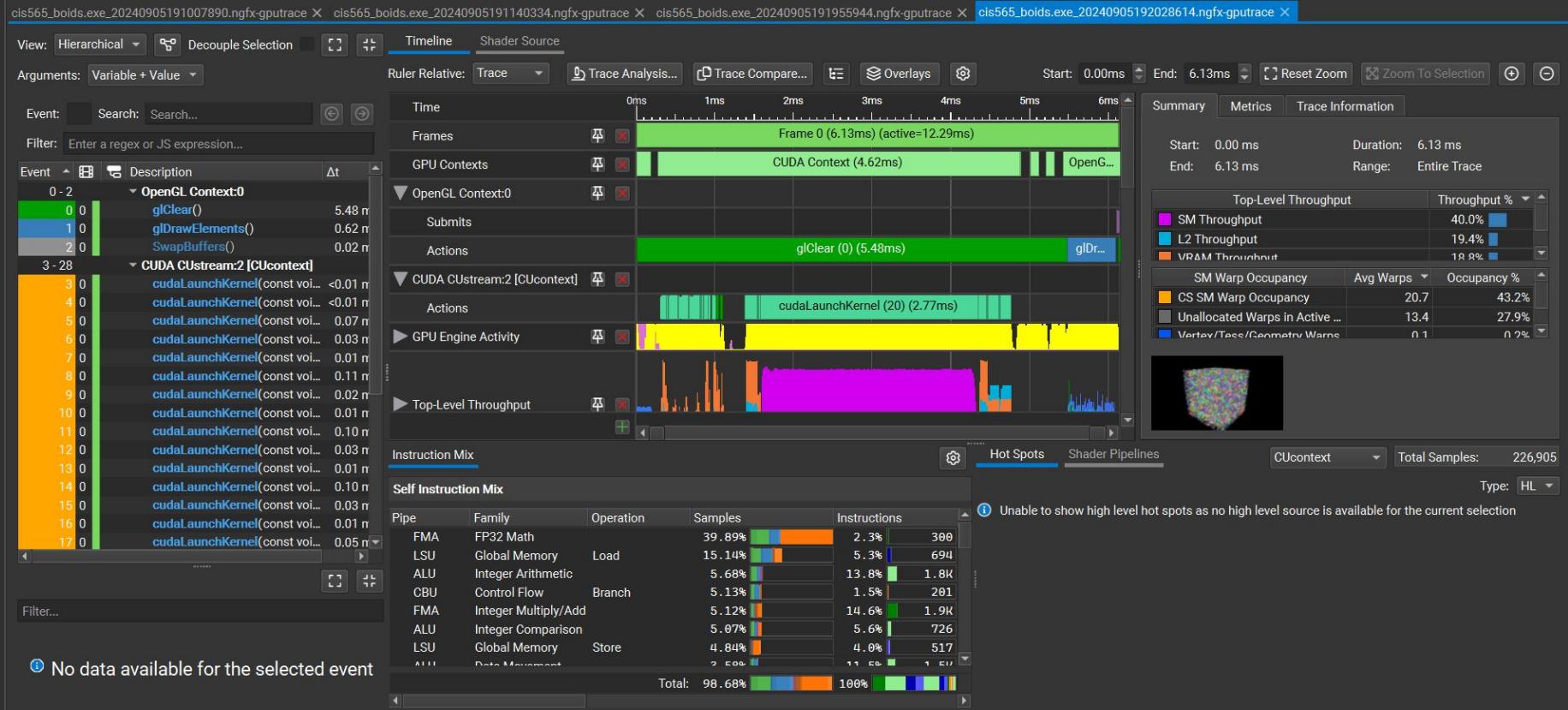
2. GPU Trace - Flocking



2. GPU Trace - Flocking



2. GPU Trace - Flocking



2. GPU Trace - Flocking

Live Demo

2. GPU Trace - Flocking (Process)

Observations:

Top Unit Throughput:

L2 and VRAM
(Suspiciously low SM Throughput)

Instead of executing instructions, we spend most of our time fetching data

Compute Warp Latency:

1,200,000 cycles

On avg, a single warp lifespan is high

Warp Stalled Reason:

Long Scoreboard

Instructions depending on data that might leave the SM (i.e TEX fetches, global/local mem access)

2. GPU Trace - Flocking (Causes)

Causes?

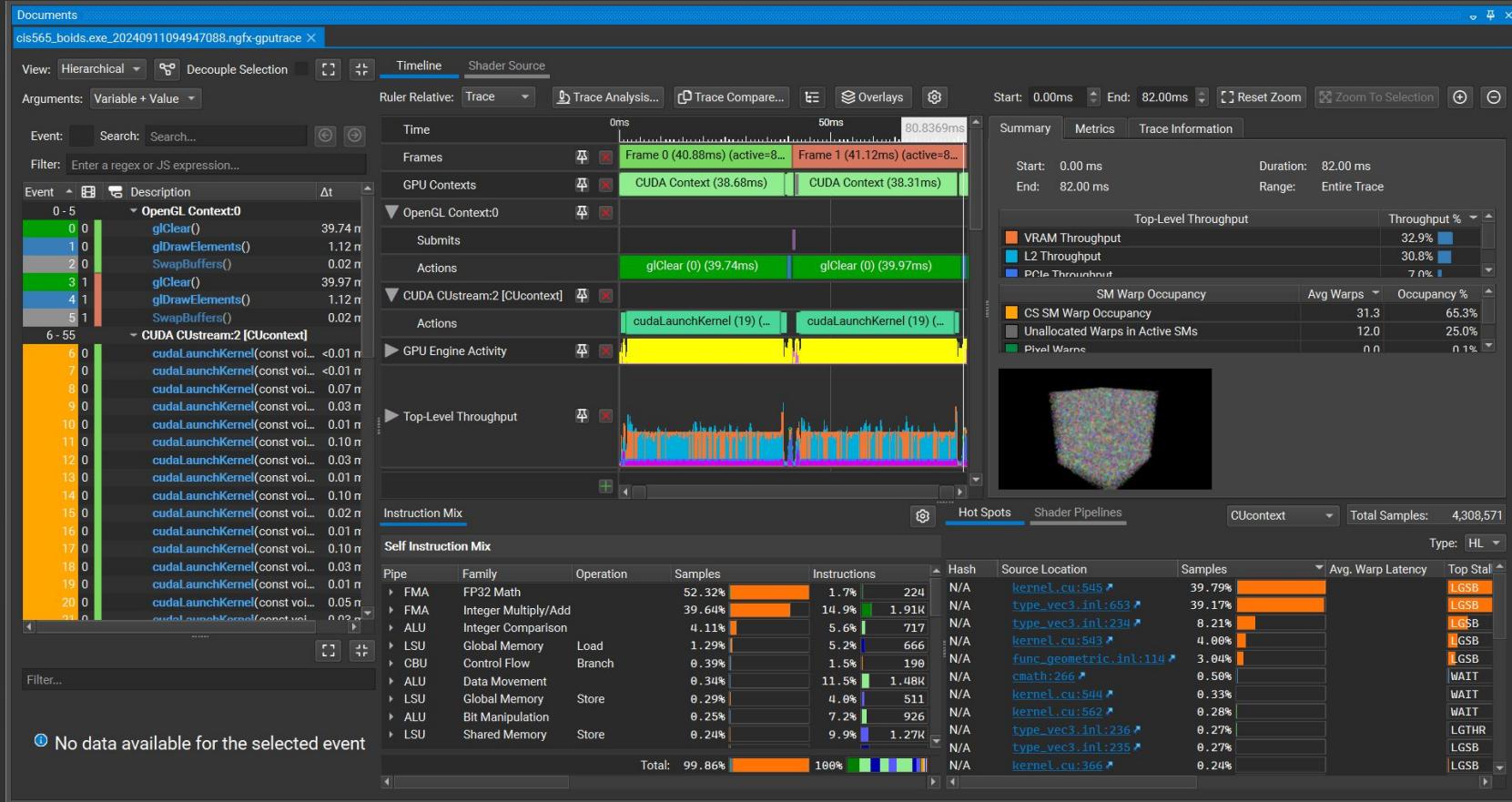
Cache Thrashing

When a thread accesses a neighbor's position or velocity, it brings in a cache line (~128 bytes, depending on the unit). However, if the next neighbor's data is not nearby in memory, this cache line might not be used for subsequent accesses. This means we have to make another request from global mem! This leads to cache thrashing, where cache lines are constantly being replaced without being fully utilized.

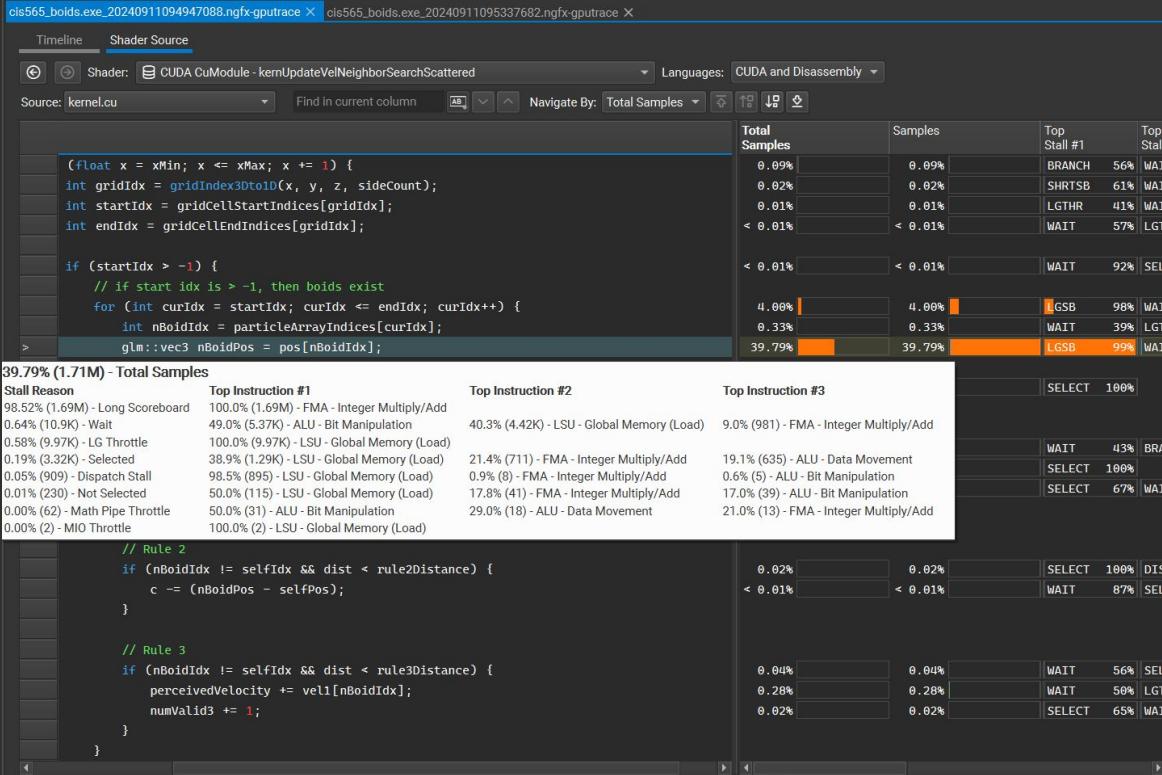
Memory Latency

Random memory accesses often result in higher latency compared to sequential accesses, as the memory controller cannot predict and prefetch data as effectively. (prefetching can be implemented by using hw counters, stream buffers, prefetch tables, etc.)

2. GPU Trace - Flocking; Per-Line Analysis

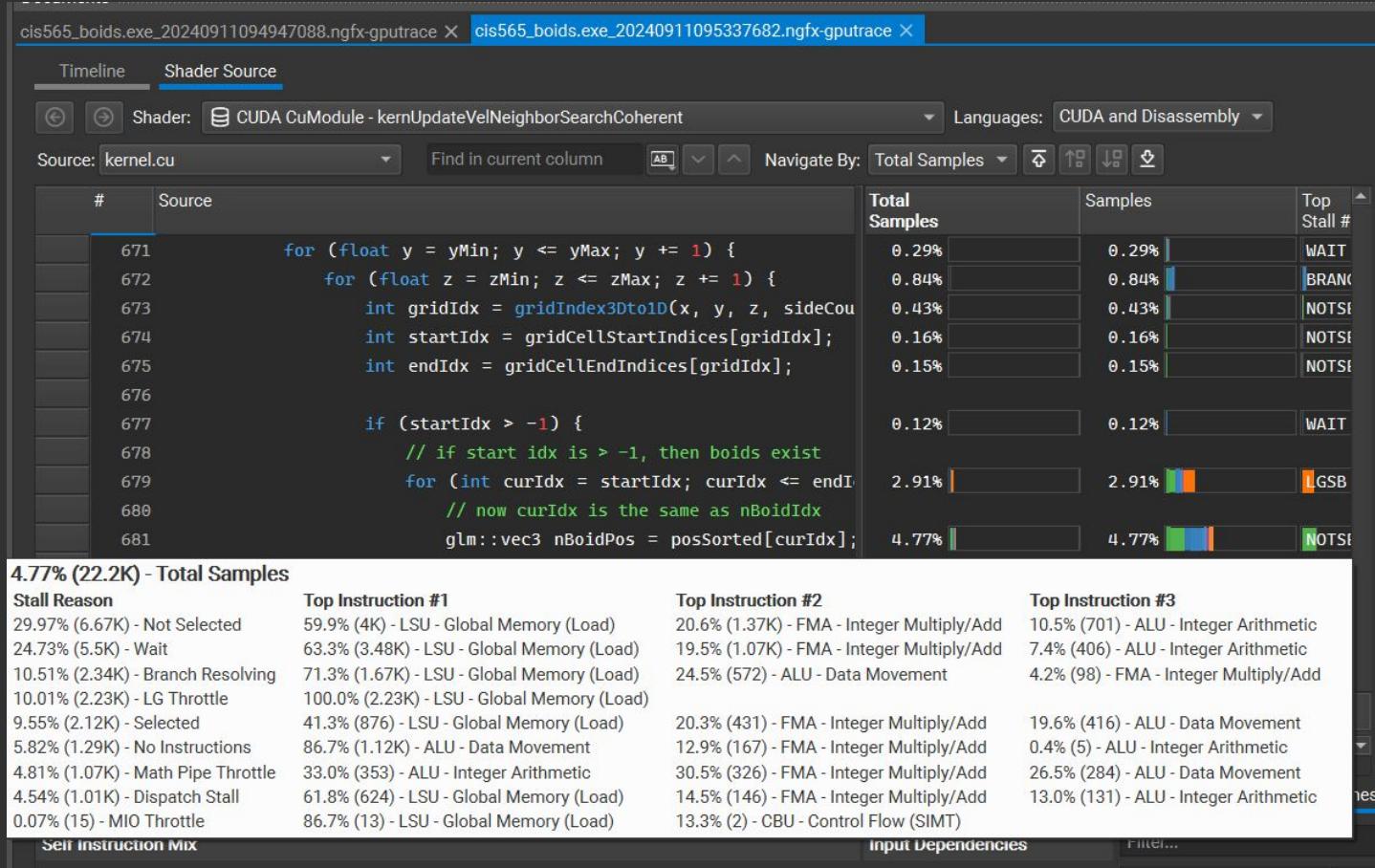


2. GPU Trace - Flocking; Per-Line Analysis

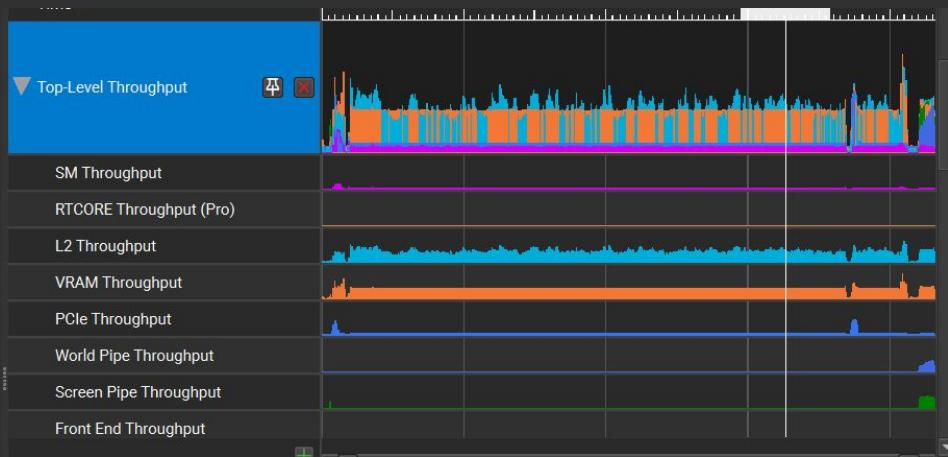


Rearranged POS and VEL, such that information from boids closer together will be physically to each other

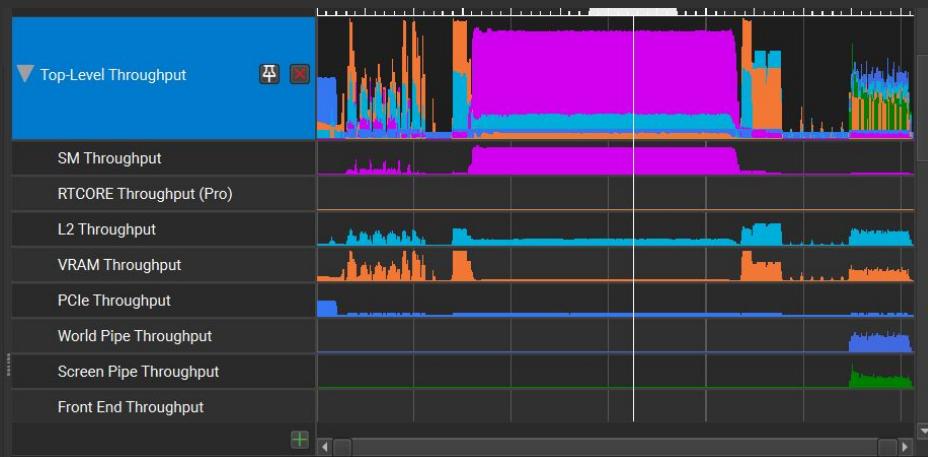
2. GPU Trace - Flocking; Per-Line Analysis



2. GPU Trace - Flocking (Results)



Uniform Grid



Coherent Grid

2. GPU Trace - Flocking (Results)

	Uniform	Coherent
Kernel Duration (ms)	38.78	2.77
SM Throughput %	4.7	82.4
VRAM Throughput %	32.3	3.8
L2 Throughput %	34.8	17.7
Avg. Warp Latency (cycles)	1,186,350	85,810
Warp Stalled - Long Scoreboard (warps)	33 (68.7%)	9.2 (19.2%)

2. GPU Trace - Case Study 1

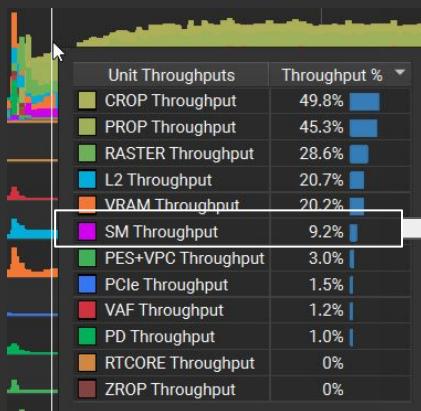
Example: SM Sub-Throughput Metrics on
D3D12RealtimeRaytraceDenoisedAmbientOcclusion >> Denoise pass

Example:

SM:	70%
L1Tex:	57%
L2:	5%
VRAM:	2%
CROP:	2%

Example:

SM FMA Pipe:	70%
SM SFU Pipe:	52%
SM ALU Pipe:	25%
SM FP16 Pipe:	0.0%



The figure shows a GPU Trace interface. On the left is a heatmap-style chart. To its right is a table titled "SM Instruction Throughputs" with a dropdown menu "Throughput %". The table lists various SM instruction types and their throughput percentages:

Instruction Type	Throughput %
SM Tensor Pipe Active	0%
SM SFU Pipe Throughput	4.1%
SM Issue Active	7.0%
SM FMA Pipe Throughput	2.1%
SM FMA Heavy Pipe Throu...	1.1%
SM ALU Pipe Throughput	1.3%

2. GPU Trace - Case Study 1

Example: SM Sub-Throughput Metrics on
D3D12RealtimeRaytraceDenoisedAmbientOcclusion >> Denoise pass

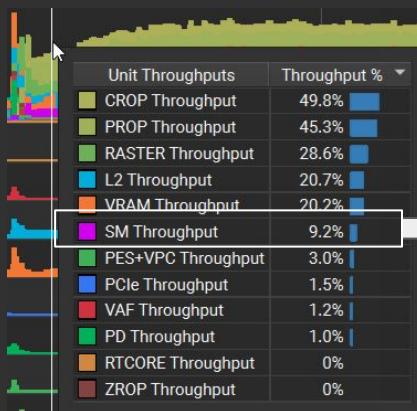
Example:

SM:	70%
L1Tex:	57%
L2:	5%
VRAM:	2%
CROP:	2%

Example:

SM FMA Pipe:	70%
SM SFU Pipe:	52%
SM ALU Pipe:	25%
SM FP16 Pipe:	0.0%

Action Item: Reduce the amount of FMA operations in your shader



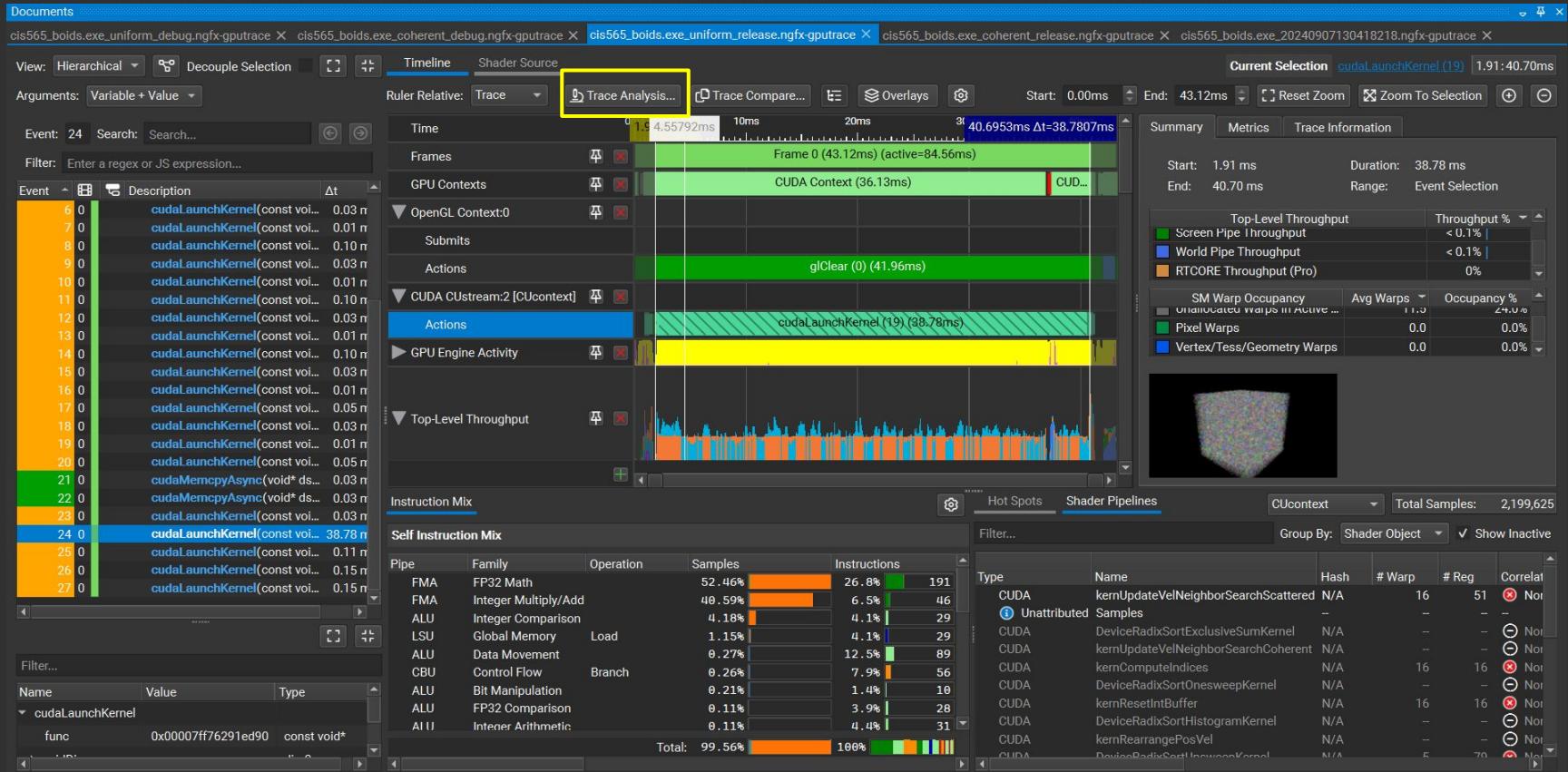
SM Instruction Throughputs	Throughput %
SM Tensor Pipe Active	0%
SM SFU Pipe Throughput	4.1%
SM Issue Active	7.0%
SM FMA Pipe Throughput	2.1%
SM FMA Heavy Pipe Throu...	1.1%
SM ALU Pipe Throughput	1.3%

2. GPU Trace - Other Common Patterns

If top SOL is NOT SM, then you might have TEX or L2 cache thrashing generated by poor GPU access patterns.

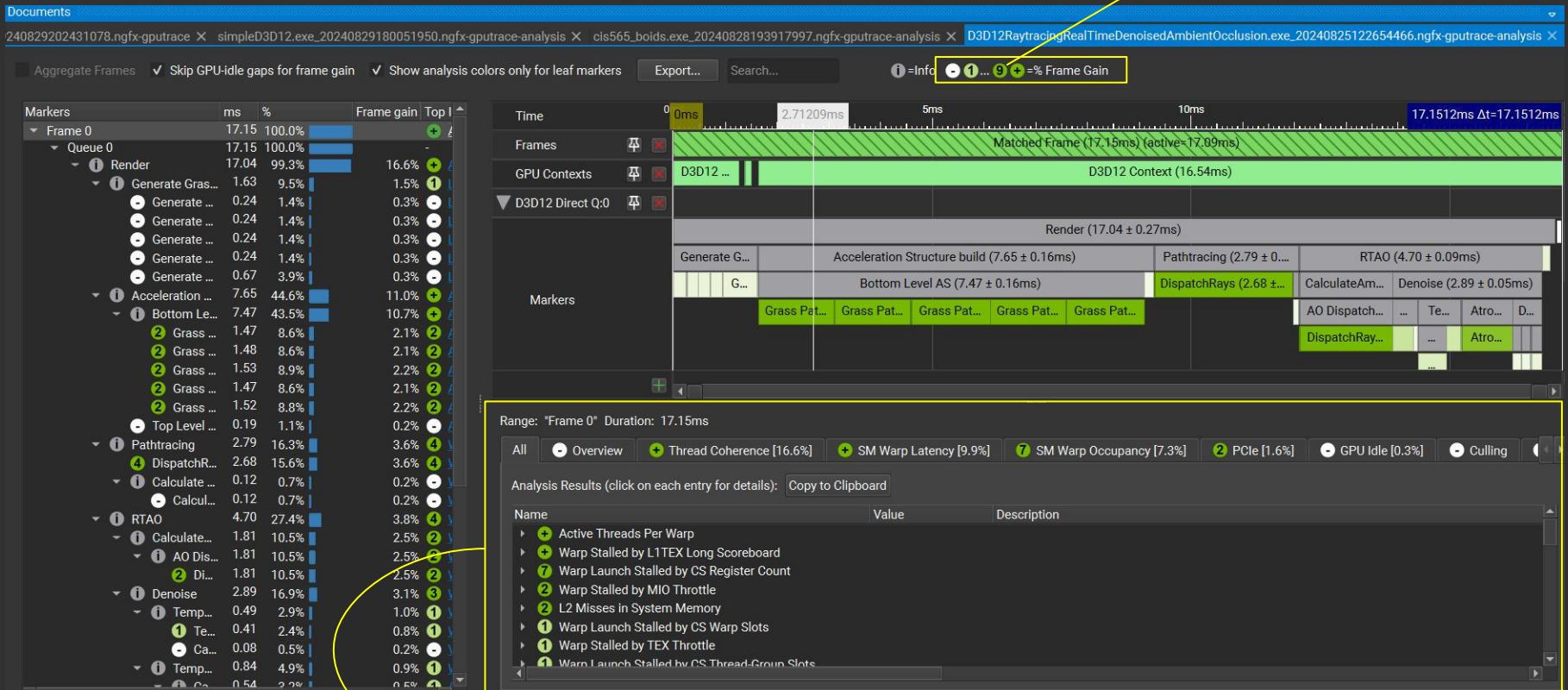
If CROP or ZROP, then drop pixels more aggressively or reducing number of render targets

2. GPU Trace - Trace Analysis



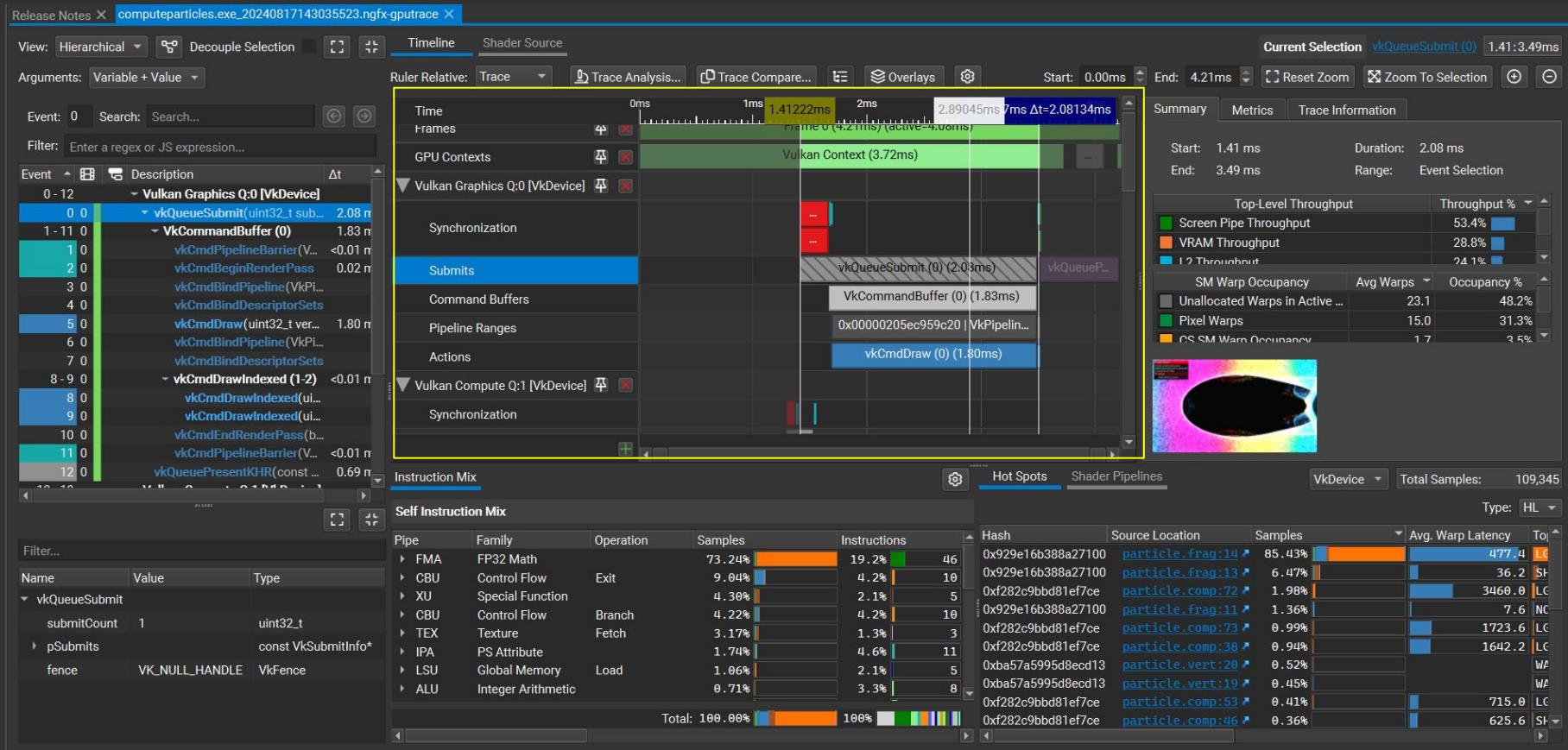
2. GPU Trace - Trace Analysis

Projected Frame gain % if you apply the suggested changes

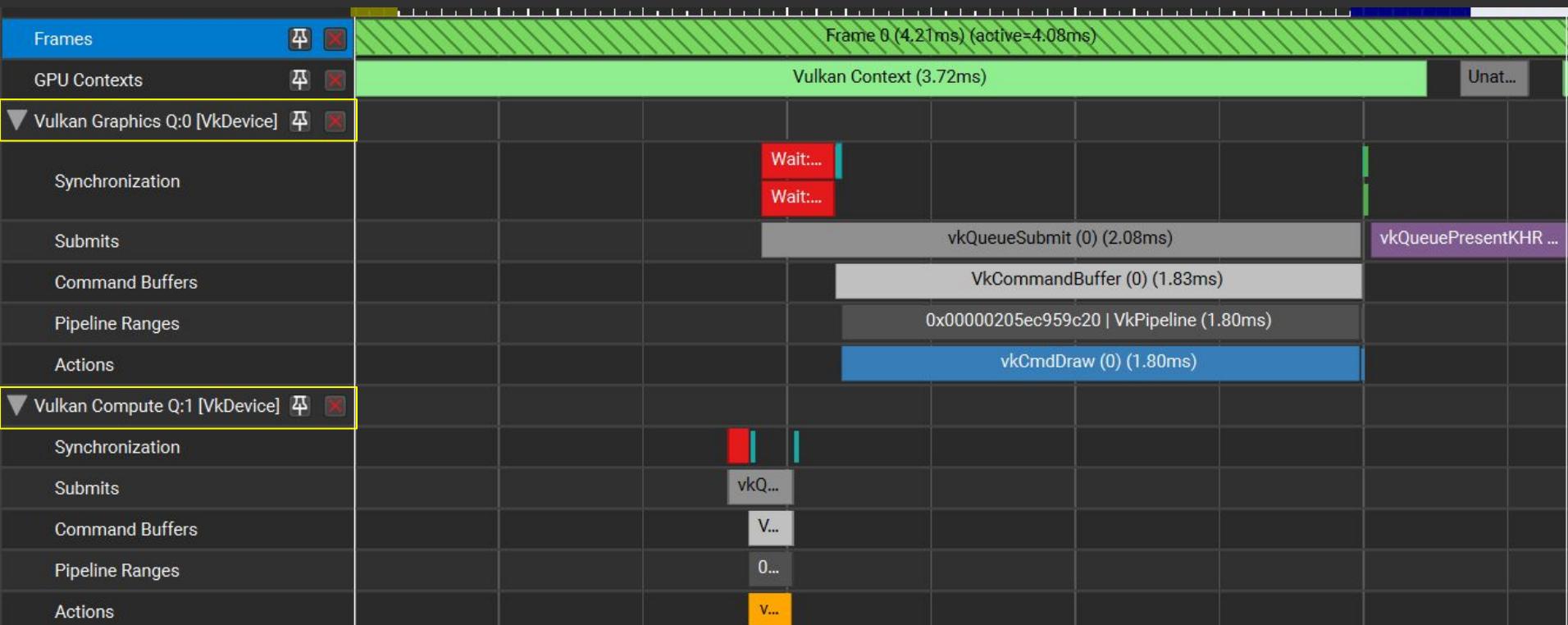


Potential problems, explanation of them, and solution suggestions

2. GPU Trace - Timeline

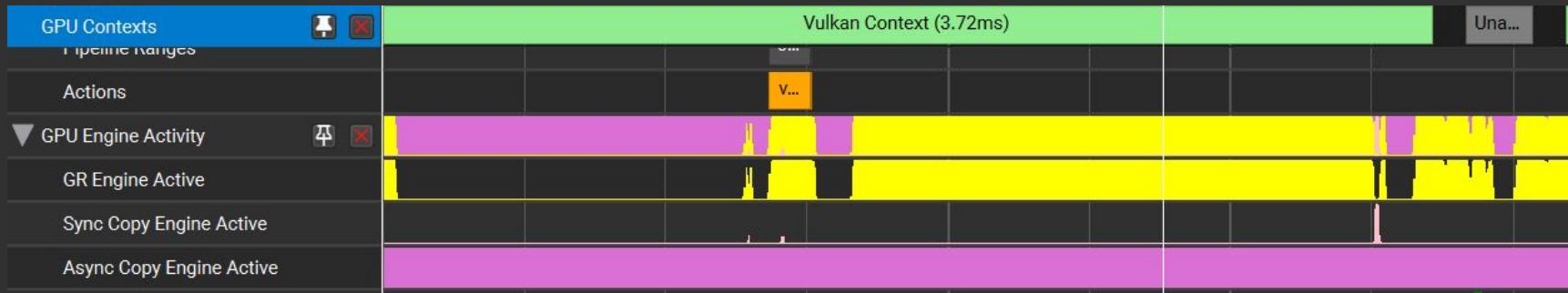


2. GPU Trace - Queues



Events from the command queues. If you have multiple queues, they will each get their own row.

2. GPU Trace - Timeline



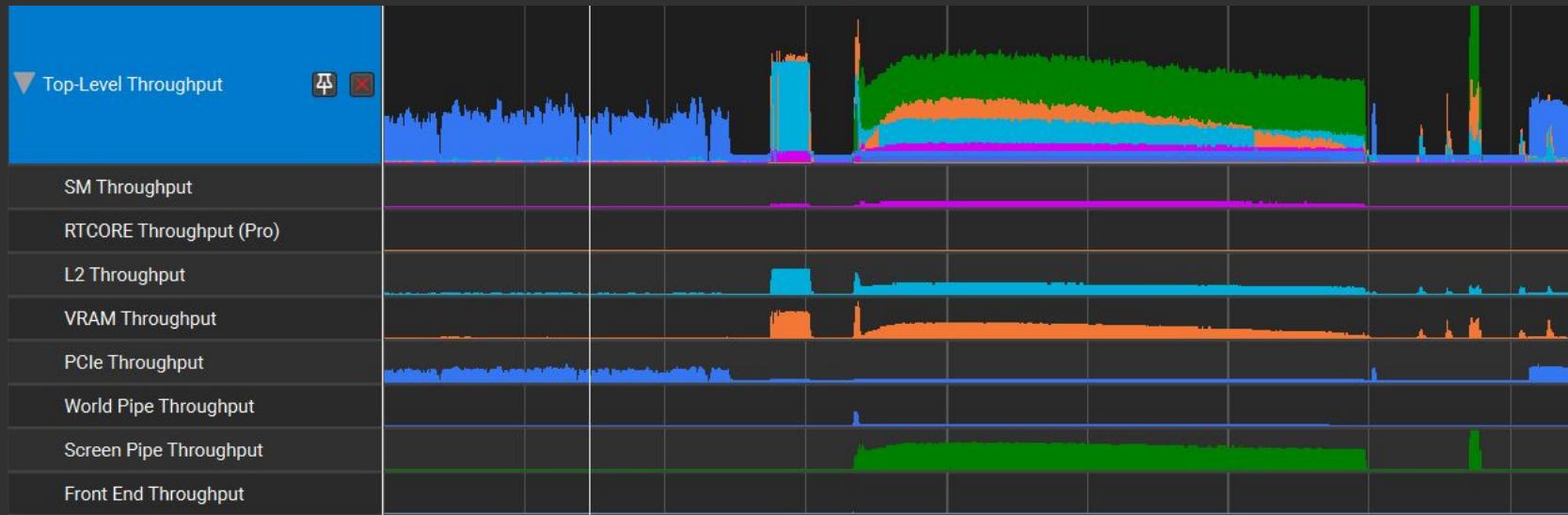
GPU Active - Duration during which any Graphics or Copy Engine were active

GR Active - Duration during which the Graphics Engine was active (on Ampere architecture, this means SMs, Texture Units, the Raster engine, ROPs, etc).

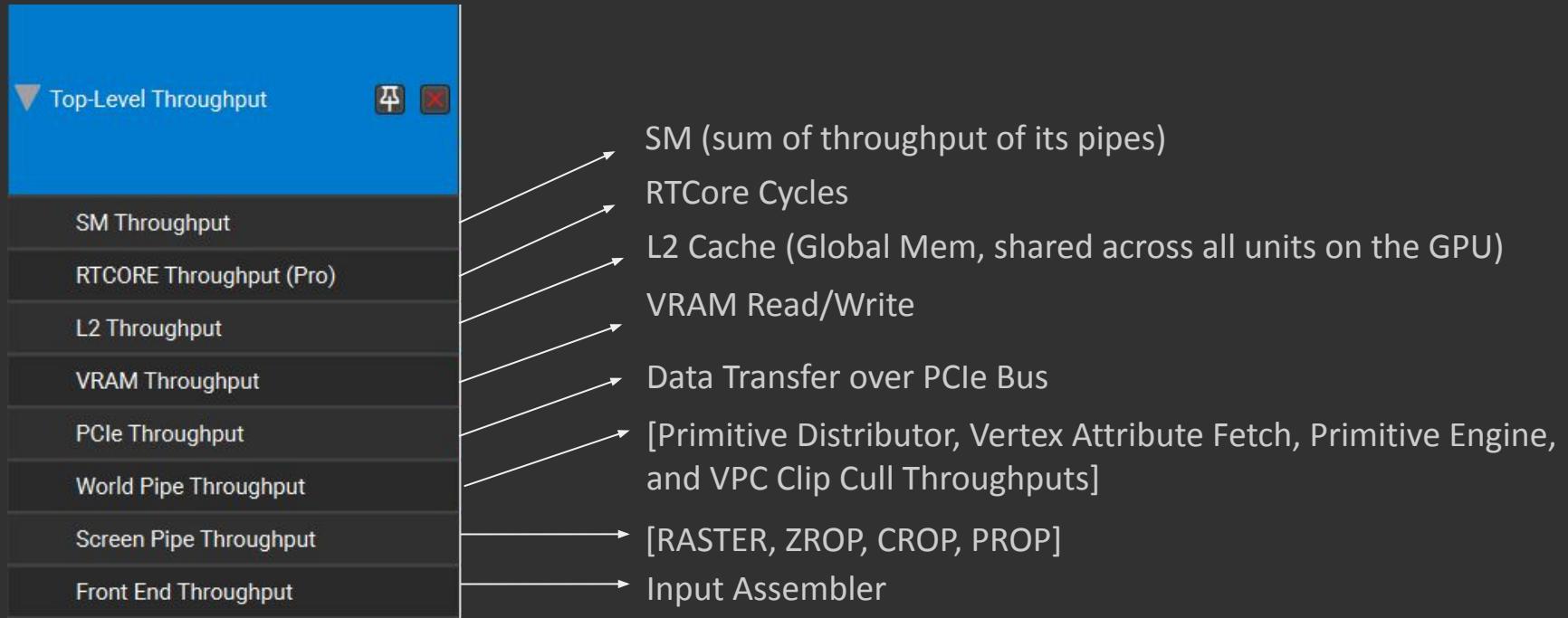
Sync Copy Engine - [CPU and GPU Memcpy] Blocking CPU until copy is done

Async Copy Engine - [CPU and GPU Memcpy] Does not block until copy is done, used with CUDA Streams (cudaMemcpyAsync)

2. GPU Trace - Timeline



2. GPU Trace - Timeline

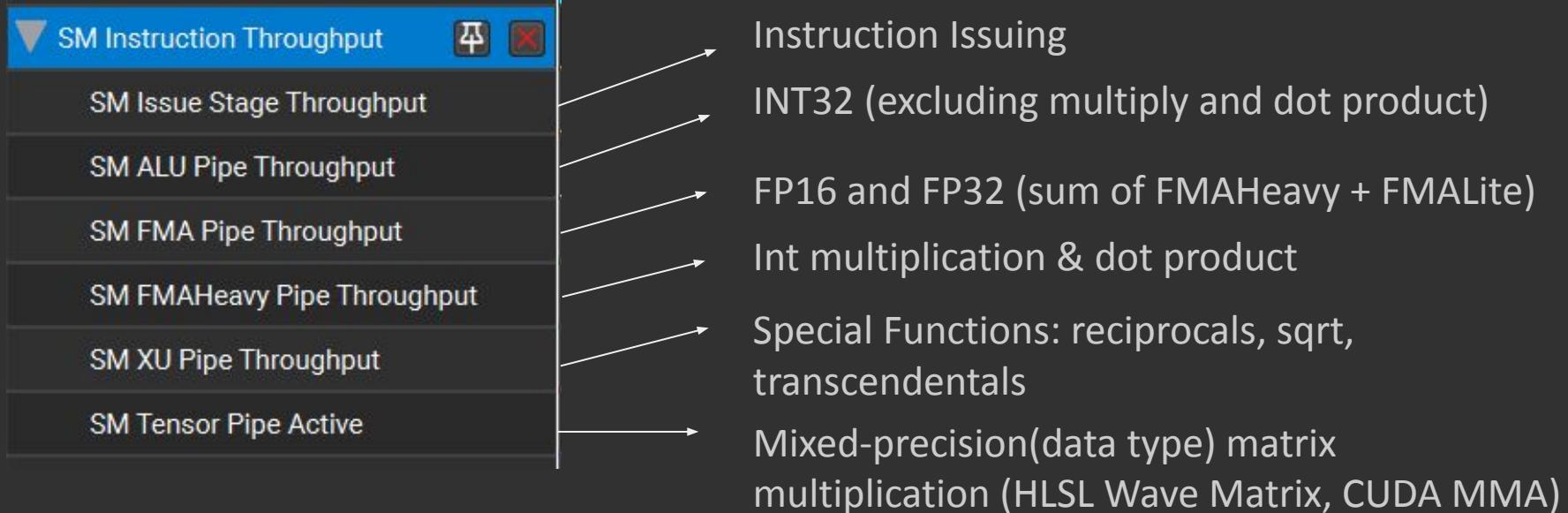


2. GPU Trace - Timeline

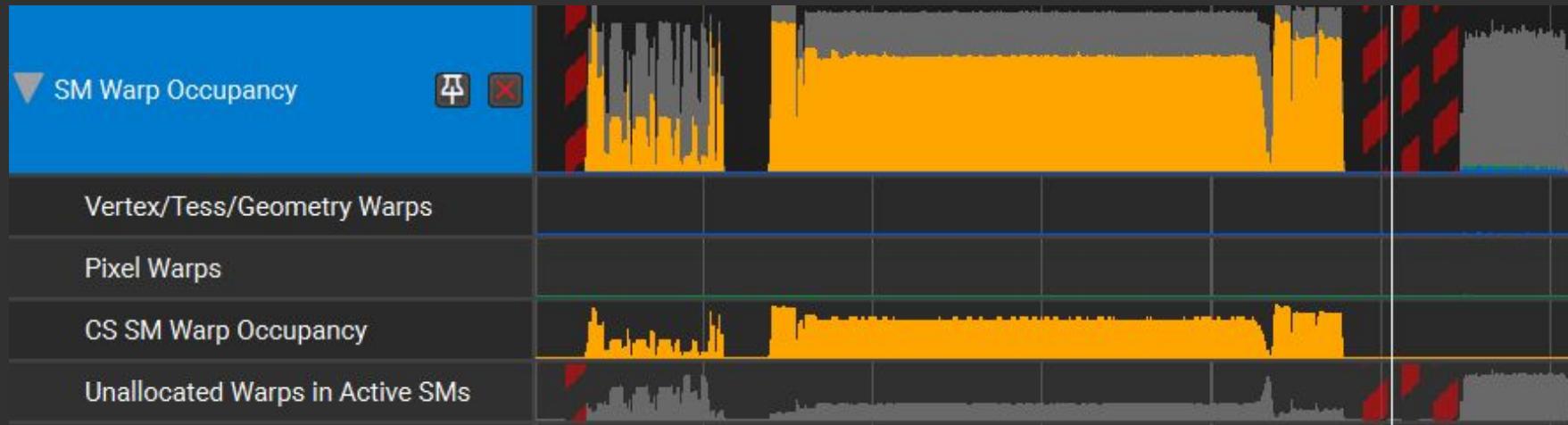


2. GPU Trace - Timeline

Shader cores' performance:



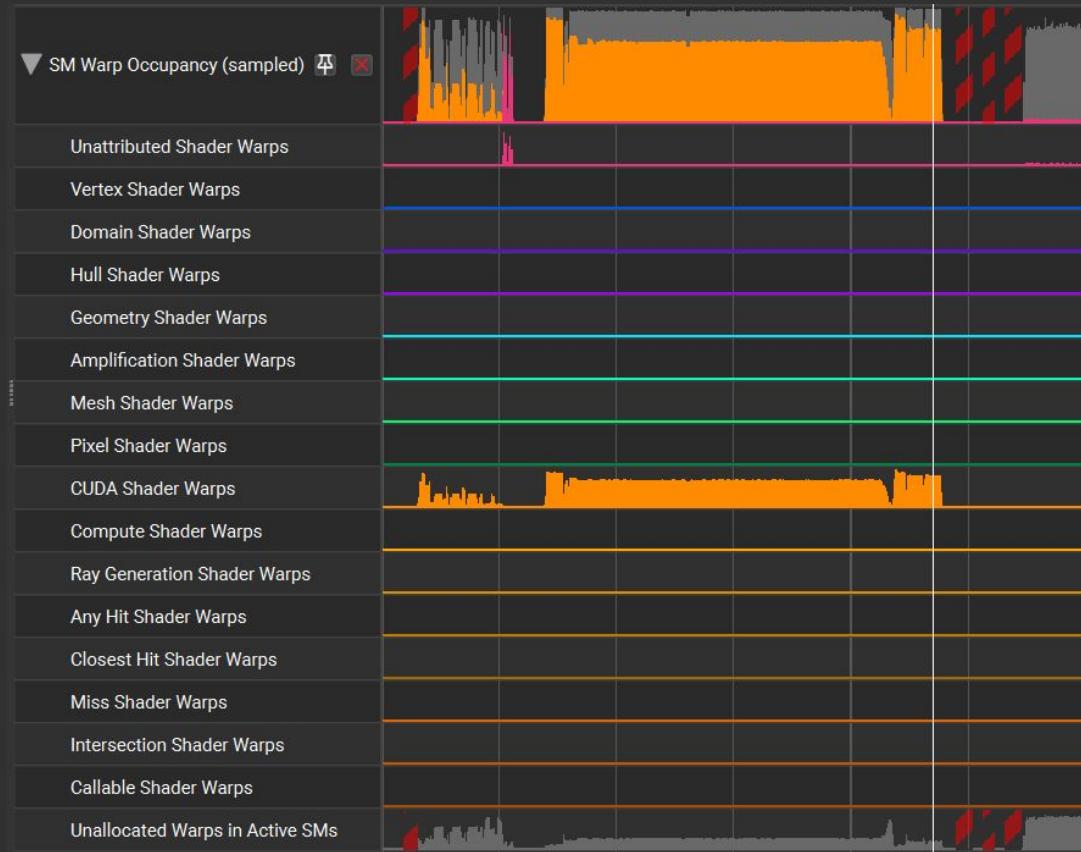
2. GPU Trace - Timeline



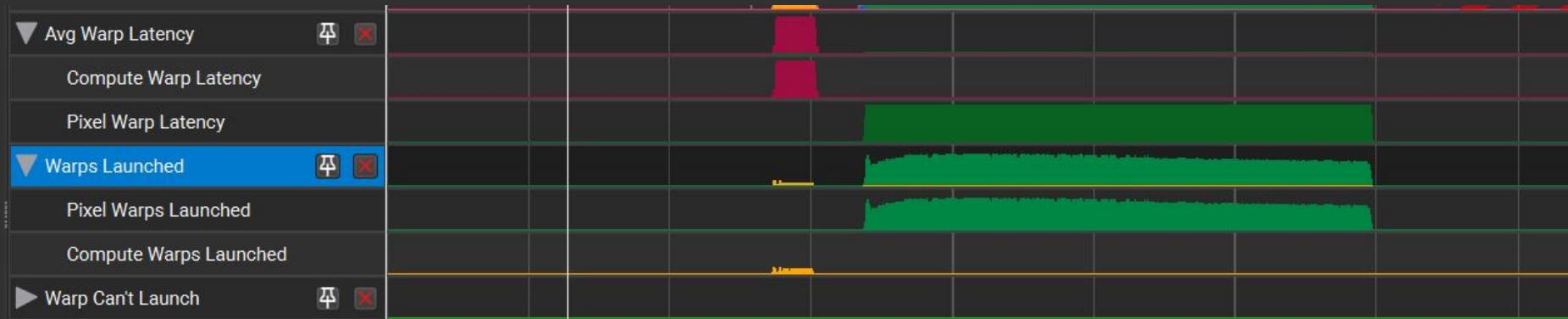
Active SM Unused Warp Slots:

- Active SM = SMs with at least 1 Active warp
- For an active SM, how many unused warps are there?

2. GPU Trace - Timeline



2. GPU Trace - Timeline



Average Warp Latency

- Absolute measure of shader performance, for how many cycles did a shader warp live across this shader workload?

Warps Launched

- Number of warps launched across all SMs of Pixel or Compute type

2. GPU Trace - Timeline

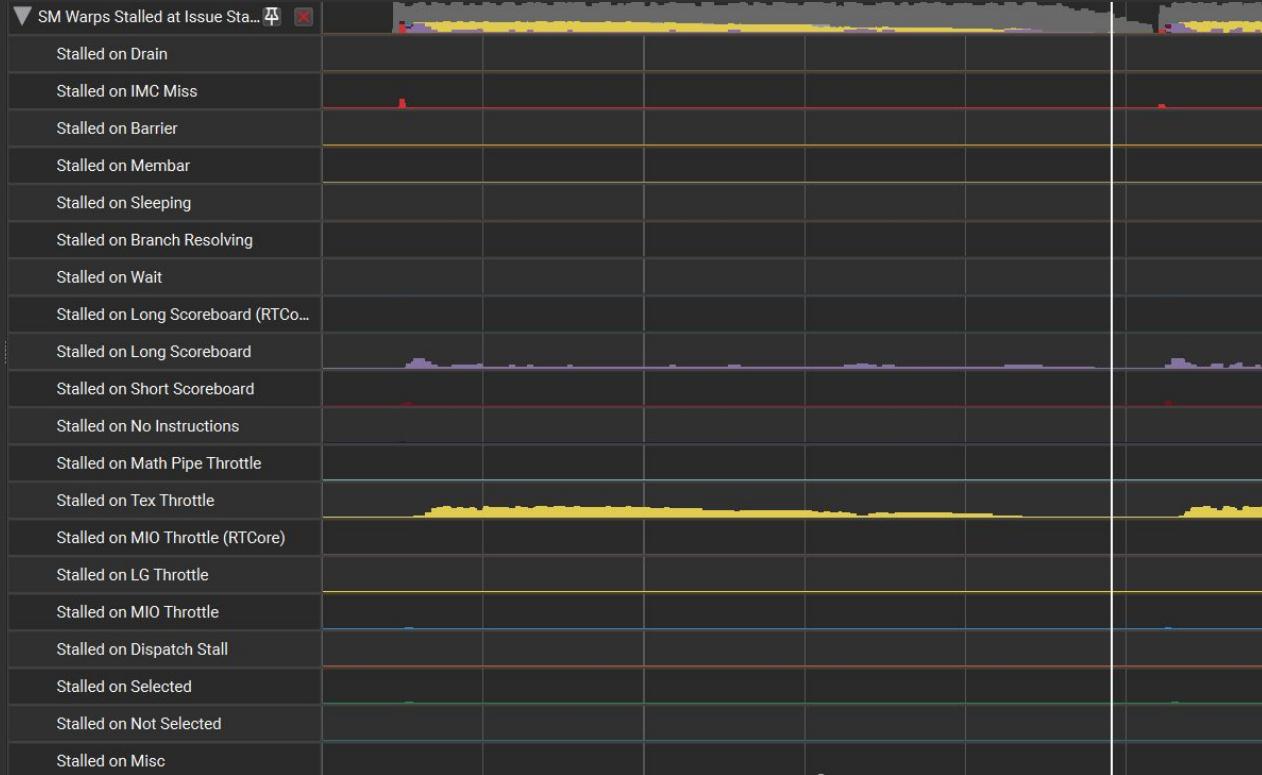
Warps Issue Stalled

- For example, if you realize that you are SM Instruction Throughput limited, and some SMs were unable to launch warps, why?

SM Warps Stalled at Issue Stage	Avg Warps	Occupancy %
Stalled on Barrier	0.0	0.0%
Stalled on Branch Resolving	0.2	0.3%
Stalled on Dispatch Stall	0.0	0.0%
Stalled on Drain	0.0	0.1%
Stalled on IMC Miss	0.0	0.0%
Stalled on LG Throttle	0.2	0.4%
Stalled on Long Scoreboard	6.5	13.5%
Stalled on Long Scoreboard (RTCORE)	0.0	0.0%
Stalled on Math Pipe Throttle	0.0	0.0%
Stalled on Membar	0.0	0.0%
Stalled on MIO Throttle	0.1	0.2%
Stalled on MIO Throttle (RTCORE)	0.0	0.0%
Stalled on Misc	0.1	0.2%
Stalled on No Instructions	0.1	0.1%
Stalled on Not Selected	0.0	0.0%
Stalled on Selected	0.2	0.4%
Stalled on Short Scoreboard	0.6	1.2%
Stalled on Sleeping	0.0	0.0%
Stalled on Tex Throttle	0.0	0.0%
Stalled on Wait	0.7	1.5%

2. GPU Trace - Timeline

Warps Issue Stalled



2. GPU Trace - Capture Information

Summary Metrics Capture Information

System

Computer Name	DESKTOP-53P2RL9
Operating System	Windows 11 (22H2)
Operating System Build	22621
Processor	12th Gen Intel(R) Core(TM) i7-12700H
RAM Usage	21,843 MiB / 32,439 MiB (67.3%)

GPU

Device Name	NVIDIA GeForce RTX 3050 Ti Laptop GPU
Chip Name	GA107
Driver Version	537.58
Hardware Scheduling	Enabled

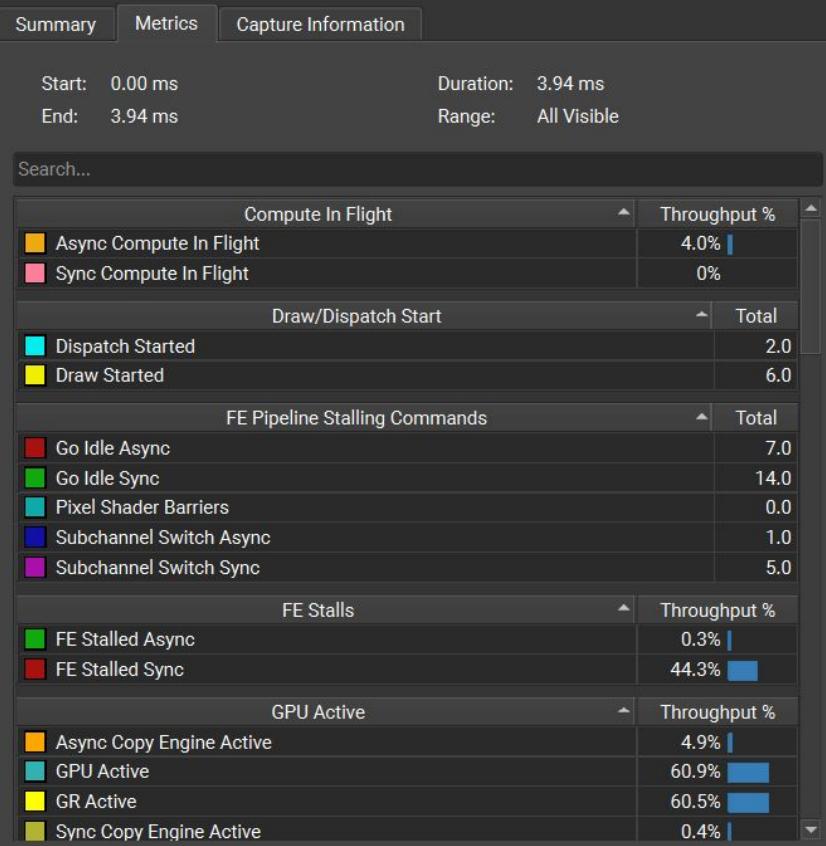
User Comments



Sascha Willems - computeParticles.exe

Sanity check for what the application is running against.

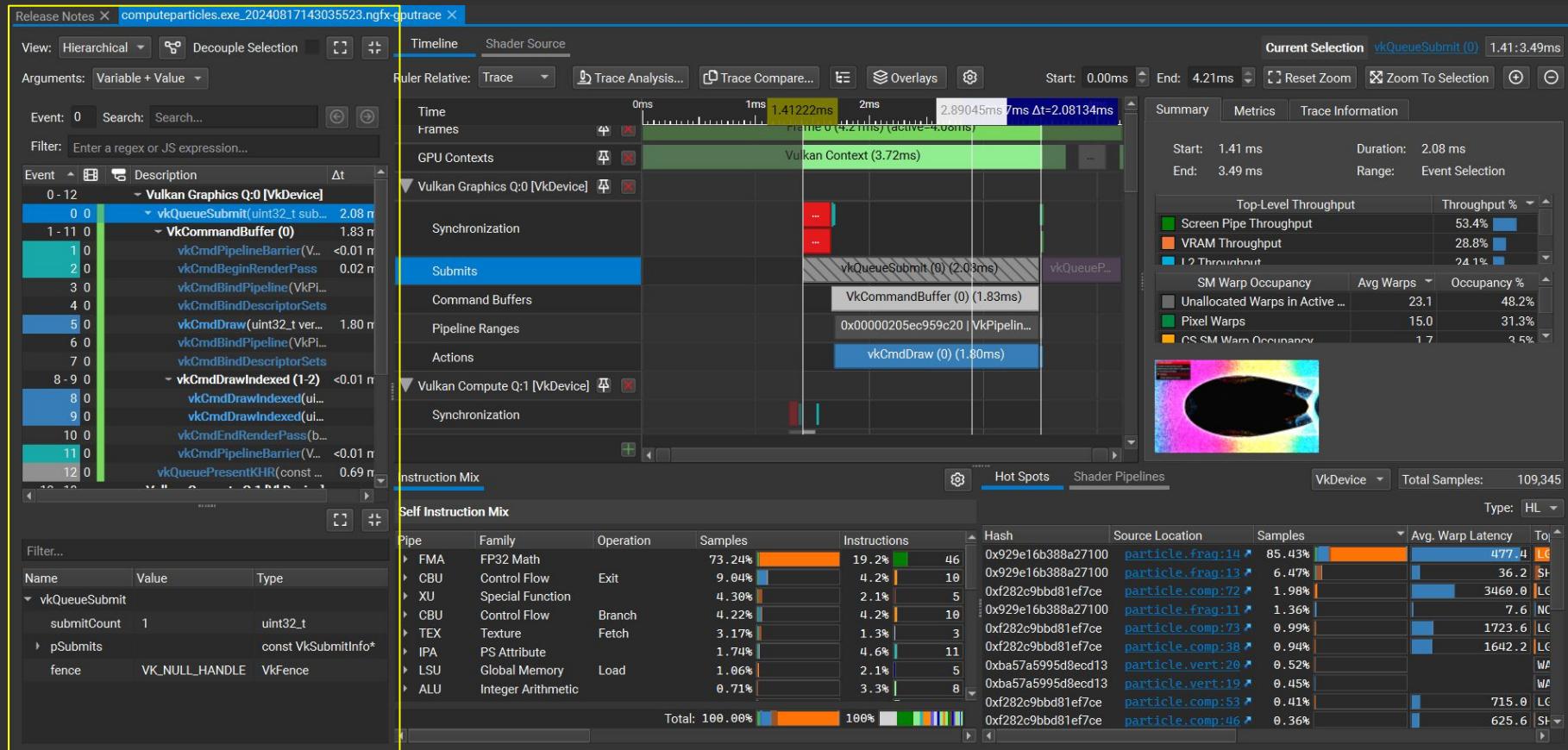
2. GPU Trace - Metrics Info



NOTE:

Your machine being plugged into power
will affect the duration of your frame

2. GPU Trace - API Event List



2. GPU Trace - Items Navigator

View: Hierarchical Decouple Selection

Arguments: Variable + Value

Event: 7 Search: Search...

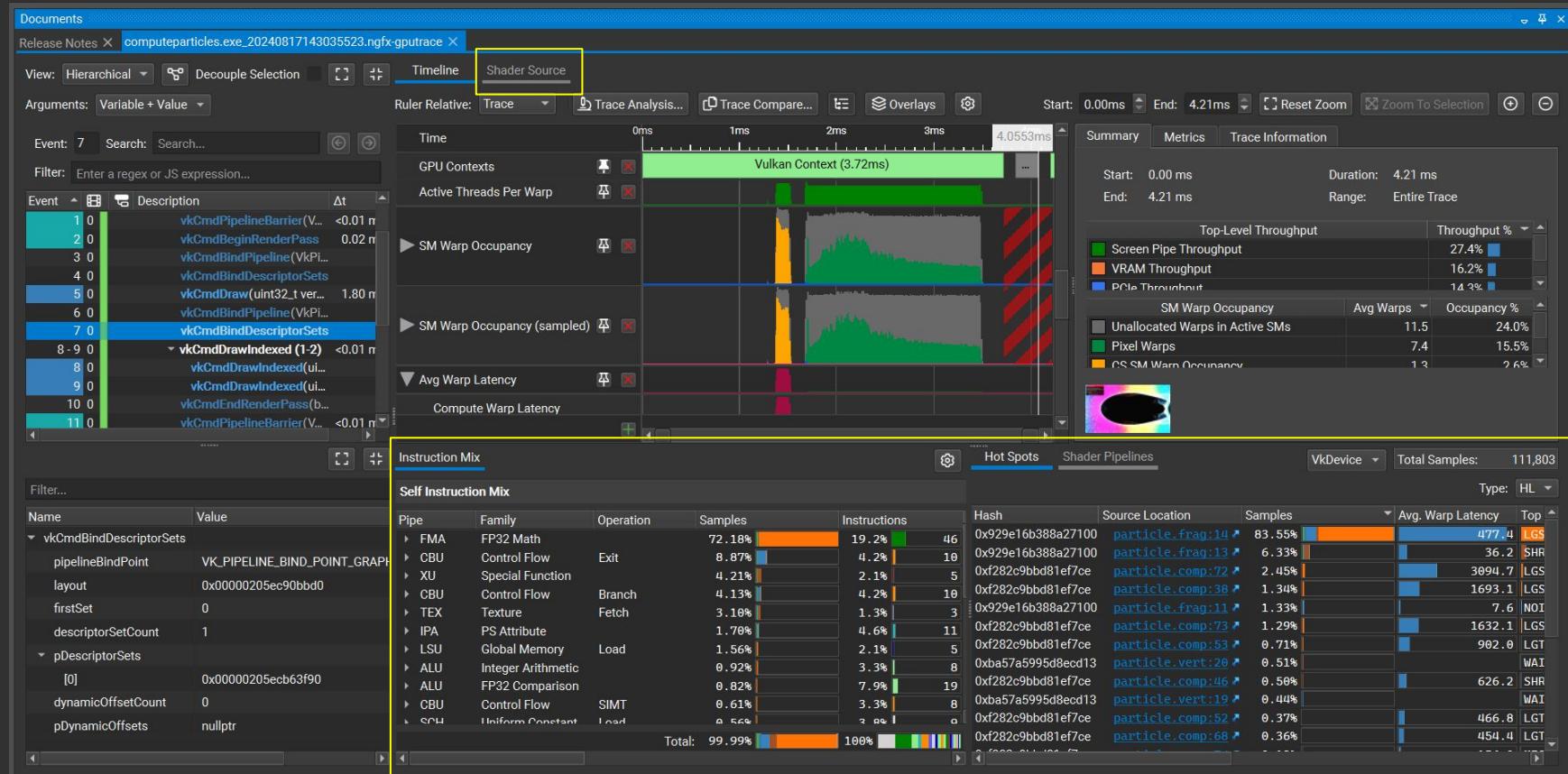
Filter: Enter a regex or JS expression...

Event	Description	Δt	Δ
1 0	vkCmdPipelineBarrier(VkPipelineStage...)	<0.01 ms	
2 0	vkCmdBeginRenderPass(const VkRend...	0.02 ms	
3 0	vkCmdBindPipeline(VkPipelineBindPoi...	-	
4 0	vkCmdBindDescriptorSets(VkPipelineB...	-	
5 0	vkCmdDraw(uint32_t vertexCount = 26...	1.80 ms	
6 0	vkCmdBindPipeline(VkPipelineBindPoi...	-	
7 0	vkCmdBindDescriptorSets(VkPipelineB...	-	
8-9 0	vkCmdDrawIndexed (1-2)	<0.01 ms	
8 0	vkCmdDrawIndexed(uint32_t indexCo...	-	
9 0	vkCmdDrawIndexed(uint32_t indexCo...	-	
10 0	vkCmdEndRenderPass(bool actsAsBar...	-	
11 0	vkCmdPipelineBarrier(VkPipelineStage...)	<0.01 ms	
12 0	vkQueuePresentV14D(const VkPresentInfo...	0.60 ms	

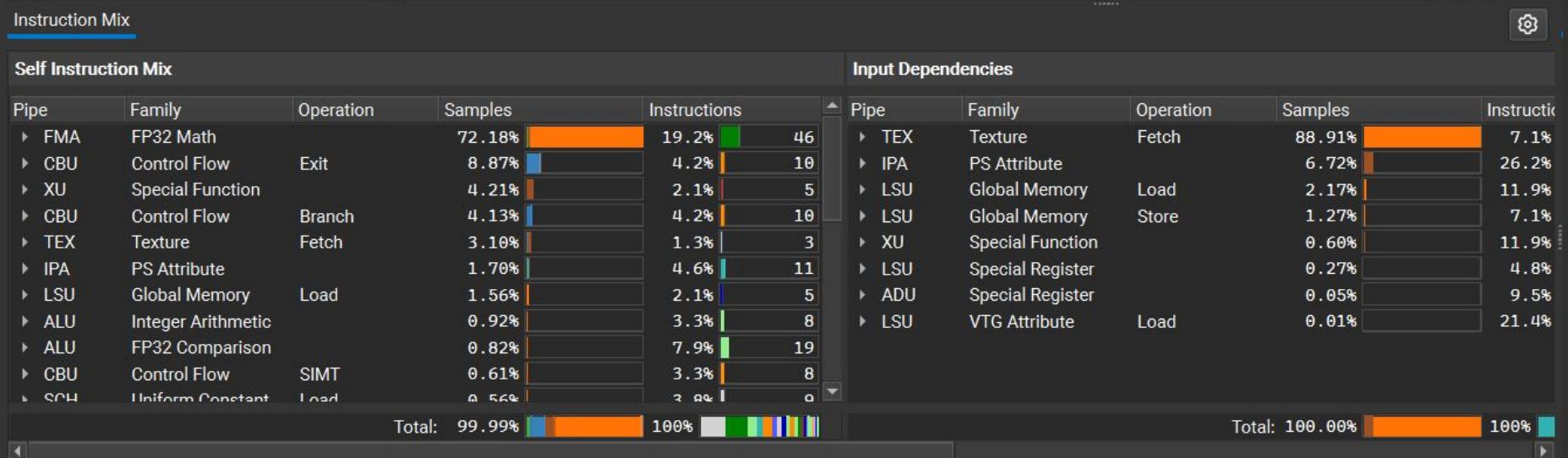
Filter...

Name	Value	Type
vkCmdBindDescriptorSets		
pipelineBindPoint	VK_PIPELINE_BIND_POINT_GRAPHICS	VkPipelineBin...
layout	0x00000205ec90bbd0	VkPipelineLay...
firstSet	0	uint32_t
descriptorSetCount	1	uint32_t
pDescriptorSets		const VkDesc...
[0]	0x00000205ecb63f90	VkDescriptorS...
dynamicOffsetCount	0	uint32_t
pDynamicOffsets	nullptr	const uint32_t

2. GPU Trace - Real-Time Shader Profiler

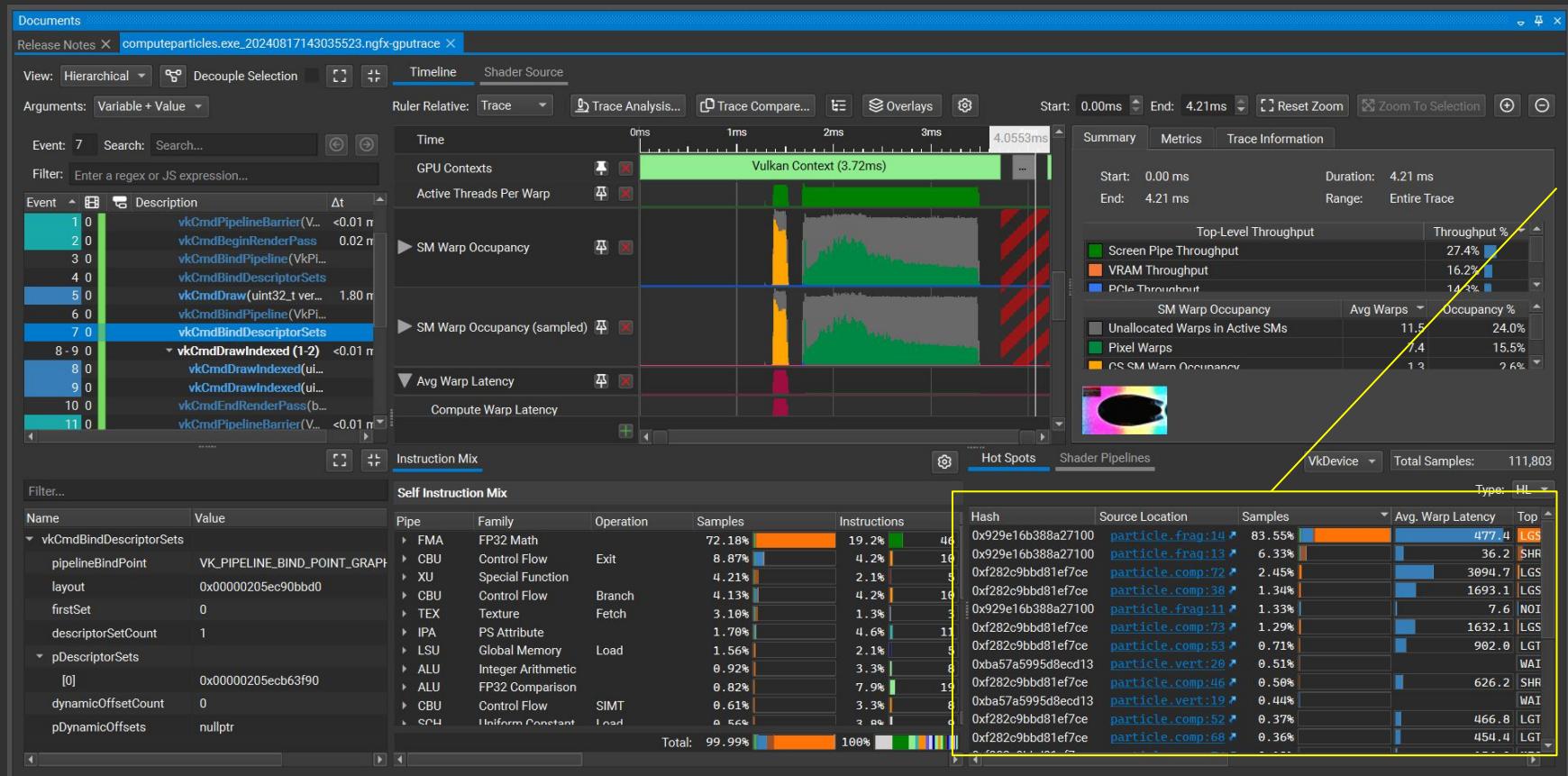


2. GPU Trace - Instruction Mix



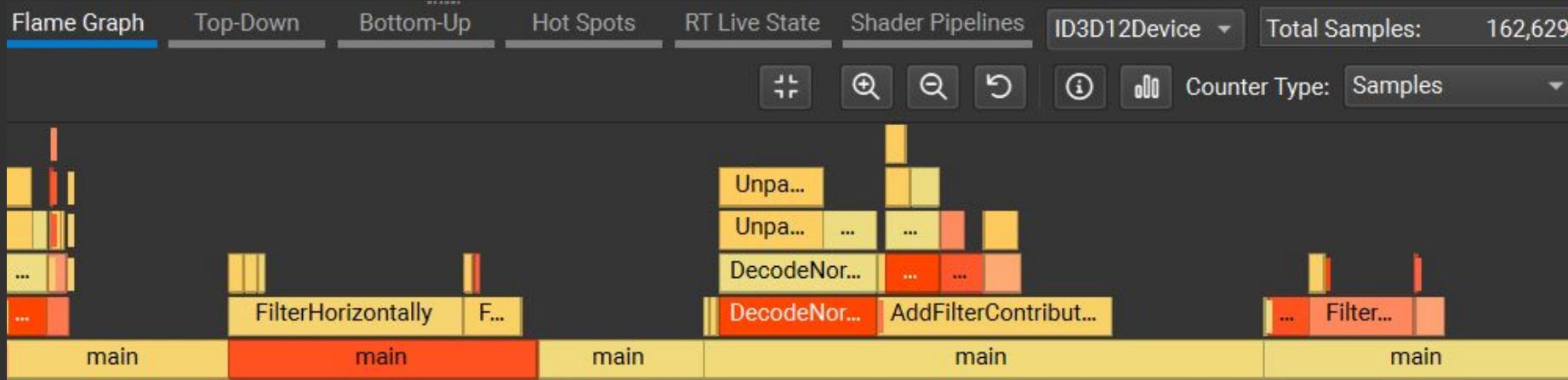
What types of instructions most commonly occur across your shaders?

2. GPU Trace - Hot Spots



https://com.s/Us.html_stal

2. GPU Trace - Flame Graph



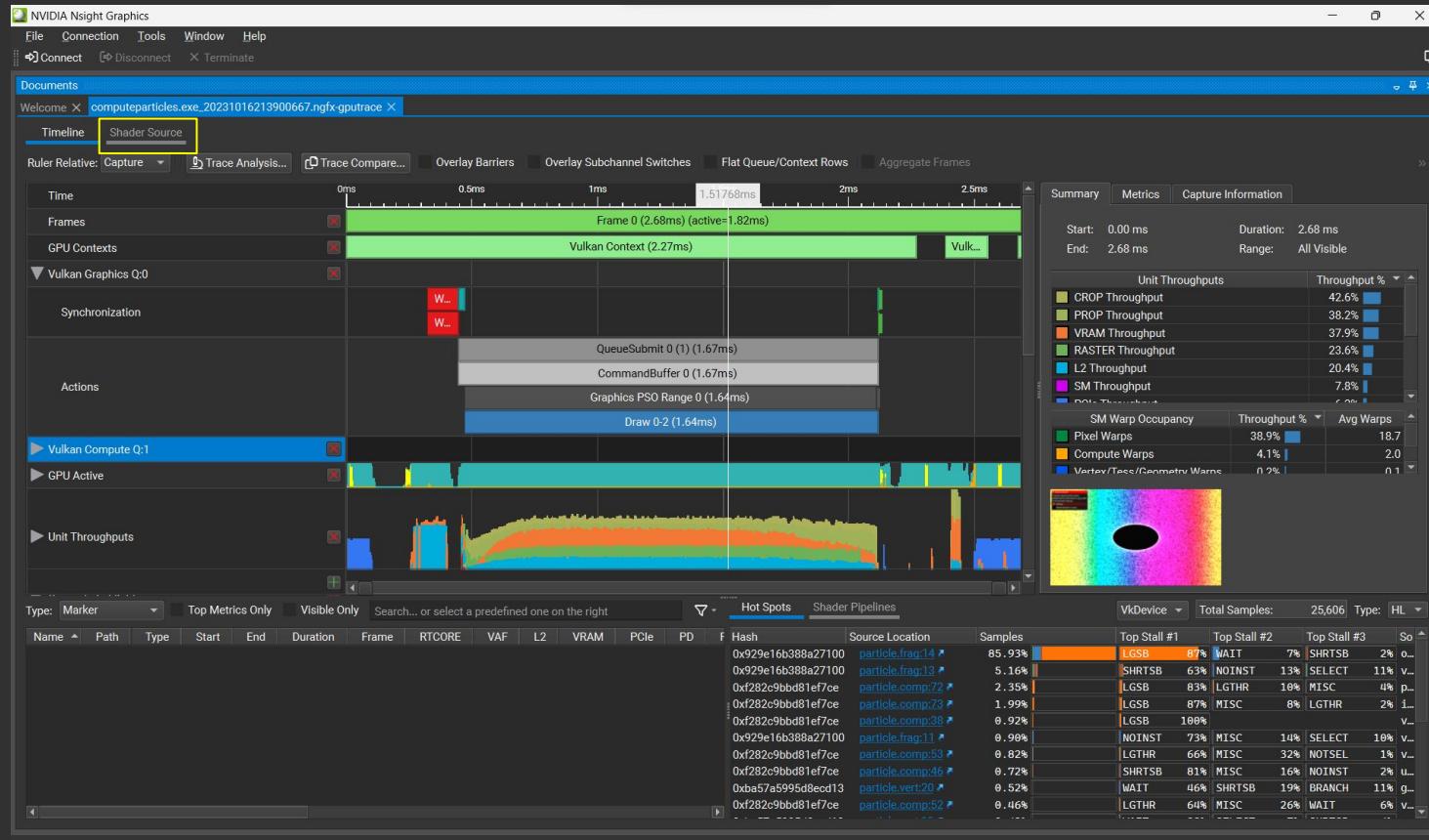
2. GPU Trace - Shader Pipelines/Shader Objects/Source

Hot Spots RT Live State **Shader Pipelines**

Filter... Group By: Pipeline Object ▾ Total Samples: 820,217 Show Inactive

Type	Name	Hash	# Warp	# Reg	Correlation	File Name	Samples	Avg. Warp Latency
Ray Tracing Shader	NVIDIA RTCore Runtime	--	--	--	--	--	63%	3
Ray Tracing Shader	Ray Tracing Acceleration Structure Build	--	--	--	ⓘ N/A	--	39%	2
Ray Tracing Shader	Ray Tracing Traversal	--	--	--	ⓘ N/A	--	19%	
Ray Tracing Shader	Ray Tracing Scheduler	--	--	--	ⓘ N/A	--	5%	
Ray Tracing Shader	Ray Tracing Internal Logic	--	--	--	ⓘ N/A	--	< 1%	
Ray Tracing Shader	Ray Tracing Geometry Construction	--	--	--	ⓘ N/A	--	0%	
ID3D12PipelineState	Pipeline state object: GenerateGrassPatch	--	32	54	--	--	8%	
Compute Shader	Compute	0x20c2c028fe66d731	32	54	⚠ DXIL	dxi ↗	8%	
ID3D12PipelineState	Pipeline state object: AtrousWaveletTransformCro...	--	--	48	38	--	7%	
Compute Shader	Compute	0x7205b73c5c3d2167	48	38	⚠ DXIL	dxi ↗	7%	
ID3D12StateObject		--	20-28	70-92	--	--	5%	
Closest Hit Shader	MyClosestHitShader_RadianceRay	0x5da0bd8ab6f71d99	20-28	70-92	⚠ DXIL	dxi ↗	5%	
Ray Generation Shader	MyRayGenShader_RadianceRay	0x5da0bd8ab6f71d99	28	70	⚠ DXIL	dxi ↗	< 1%	
Miss Shader	MyMissShader_RadianceRay	0x5da0bd8ab6f71d99	28	70	⚠ DXIL	dxi ↗	< 1%	
Miss Shader	MyMissShader_ShadowRay	0x5da0bd8ab6f71d99	--	--	✗ None	--	0%	
Closest Hit Shader	MyClosestHitShader_ShadowRay	0x5da0bd8ab6f71d99	--	--	✗ None	--	0%	
ID3D12PipelineState	Pipeline state object: DisocclusionBilateralFilter	--	32	29	--	--	4%	
Compute Shader	Compute	0xe21eeeca5054a55f6	32	29	⚠ DXIL	dxi ↗	4%	
ID3D12PipelineState	Pipeline state object: CalculateMeanVariance	--	--	32	24	--	4%	
Compute Shader	Compute	0xa7ba28f040f1278f	32	24	⚠ DXIL	dxi ↗	4%	
ID3D12PipelineState	Pipeline state object: TemporalSupersampling_Rev...	--	--	32	44	--	3%	
Compute Shader	Compute	0xc6f451bff7398a64	32	44	⚠ DXIL	dxi ↗	3%	
ID3D12PipelineState	Pipeline state object: TemporalSupersampling_Ble...	--	--	32	30	--	2%	
Compute Shader	Compute	0xbc6f079fba9c81a0	32	30	⚠ DXIL	dxi ↗	2%	
ID3D12StateObject		--	32-36	55-58	--	--	2%	
Ray Generation Shader	RayGenShader	0x4dbad77b6dd9cce5	32-36	55-58	⚠ DXIL	dxi ↗	2%	
Miss Shader	MissShader	0x4dbad77b6dd9cce5	36	55	⚠ DXIL	dxi ↗	< 1%	
Closest Hit Shader	ClosestHitShader	0x4dbad77b6dd9cce5	36	55	⚠ DXIL	dxi ↗	< 1%	
Ray Generation Shader	RayGenShader_sortedRays	0x4dbad77b6dd9cce5	--	--	✗ None	--	0%	
ⓘ Unattributed	Samples	--	--	--	--	--	1%	
ID3D12PipelineState	PSO: CompositionCS	--	--	32	24	--	1%	
Compute Shader	Compute	0xd1dae417e376d5f7	32	24	⚠ DXIL	dxi ↗	1%	

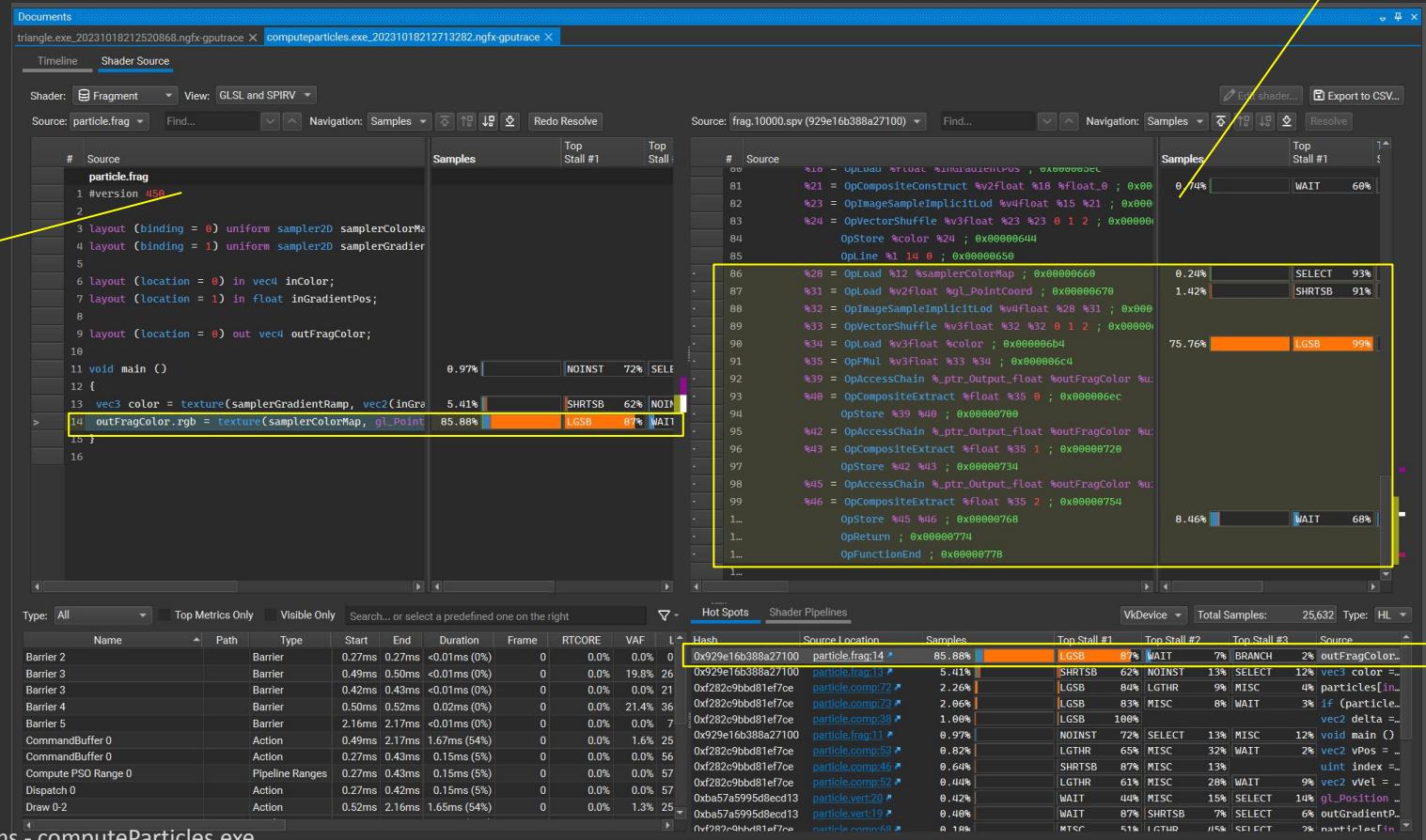
1. GPU Trace - Shader Source View



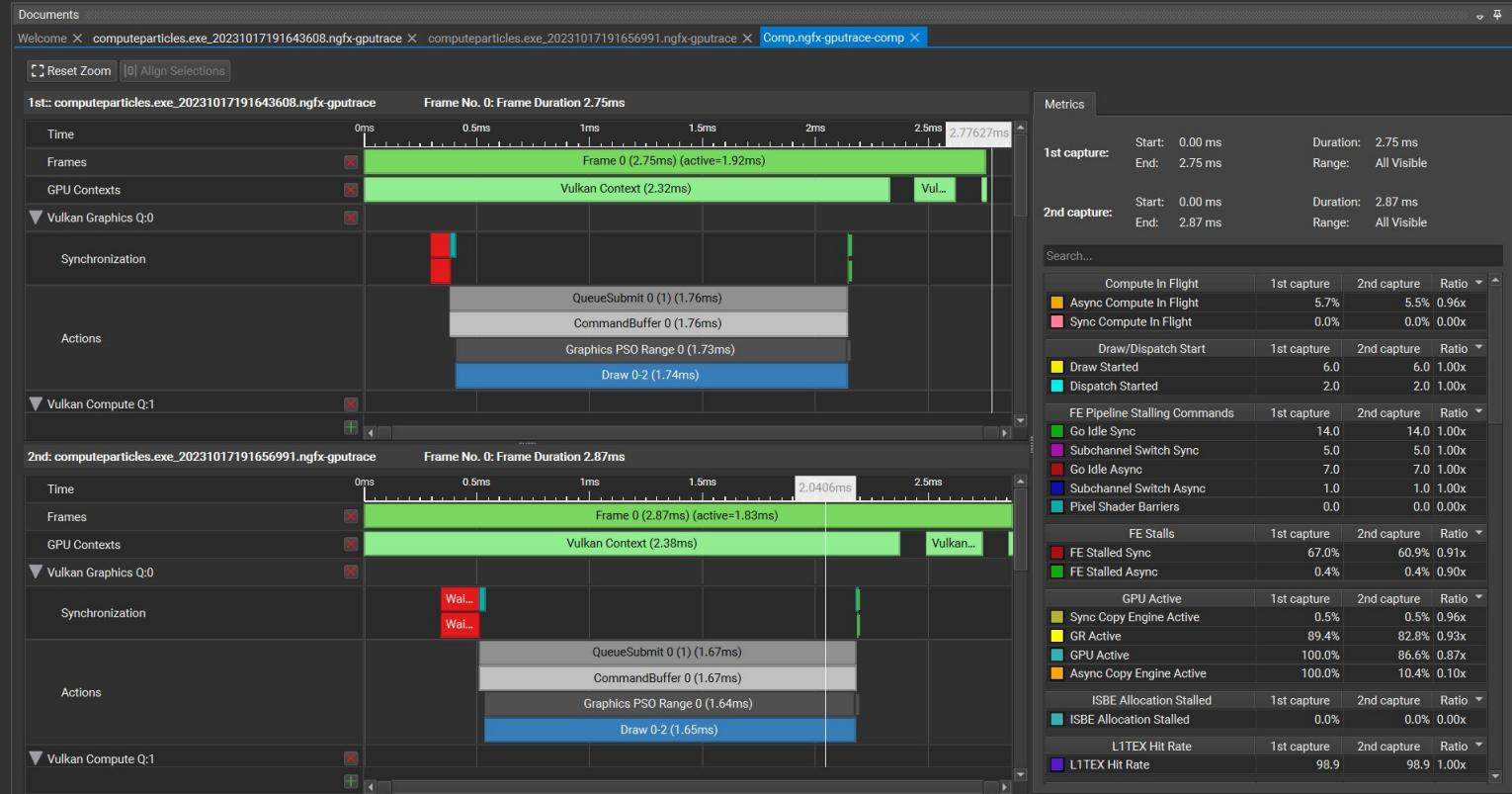
1. GPU Trace - Shader Source View

GLSL
View

SPIR-V View



2. GPU Trace - Trace Compare



Conclusion

FRAME DEBUGGER

- Render issues, pipeline issues, resource inspection, live shader editing

GPU TRACE

- GPU and Shader Profiling
- Use the P3 Method to analyze the performance of any GPU workload.
 - Start from “Top Throughput%” metrics
- Flocking Project (Uniform vs. Coherent)

Career Advice (?)



Build things from scratch



Spend time with your peers & random folks



Work smart and use tools!



Niche Knowledge



It's okay, Rocky.
You go when you feel like it.



There is no such thing as
“wasted time” in learning

Thanks!
Good luck!
Touch (vulkan) grass!

References & Resources

- What is the profiler telling you?

https://www.youtube.com/watch?v=kKANP0kL_hk&t=249s

- NSight Graphics Guide:

<https://docs.nvidia.com/nsight-graphics/UserGuide/>

- Louis Bavoil's GDC 2019 presentation:

https://developer.nvidia.com/video/GDC-19/NSIGHT_GPU_TRACE

- Louis Bavoil's P3 Method Explanation

<https://developer.nvidia.com/blog/the-peak-performance-analysis-method-for-optimizing-any-gpu-workload>

- NSight Graphics Advanced Learning

<https://docs.nvidia.com/nsight-graphics/AdvancedLearning/index.html>

- Life of a Triangle (Nvidia)

<https://developer.nvidia.com/content/life-triangle-nvidias-logical-pipeline>

- Cache Thrashing

<https://forums.developer.nvidia.com/t/difference-between-l2-read-write-transactions-and-l2-l1-read-write-transactions/80777/2>

Appendix

How to compile your project so GPU Trace can trace the CUDA kernels:

1. Open this file:

ProjectDirectory\build\CMakeFiles\cis565_boids.dir\src\cis565_boids_generated_kernel.cu.obj.<BUILDVER>.cmake

- a. Replace <BUILDVER> with Debug, Release, or any other build config you have

2. Edit line 75:

- a. `set(CUDA_NVCC_FLAGS -gencode arch=compute_86,code=sm_86 ;) # list`
- b. Remove `-gencode arch=compute_86,code=sm_86`

3. Edit line 84:

- a. `set(CUDA_NVCC_COMPILE_DEFINITIONS [=[CUDA_COMPUTE_86;GLEW_STATIC]==]) # list (needs to be in lua quotes see #16510).`
- b. Remove `CUDA_COMPUTE_86;`

4. Save file and rebuild project

Appendix

How to have cuda debug symbols on RelWithDebInfo Mode:

1. Open this file:

ProjectDirectory\build\CMakeFiles\cis565_boids.dir\src\cis565_boids_generated_kernel.cu.obj.RelWithDebInfo.cmake

2. Edit line 80:

- a. `set(CUDA_NVCC_FLAGS_RELWITHDEBINFO ;)`
- b. Add `--generate-line-info` right before semicolon ;

3. Save file and rebuild project on RelWithDebInfo mode