Introduction:
NVIDIA
NSight Graphics

Di Lu
CIS 5650 - GPU Programming
Fall 2024

Who am I?

Name: Di Lu

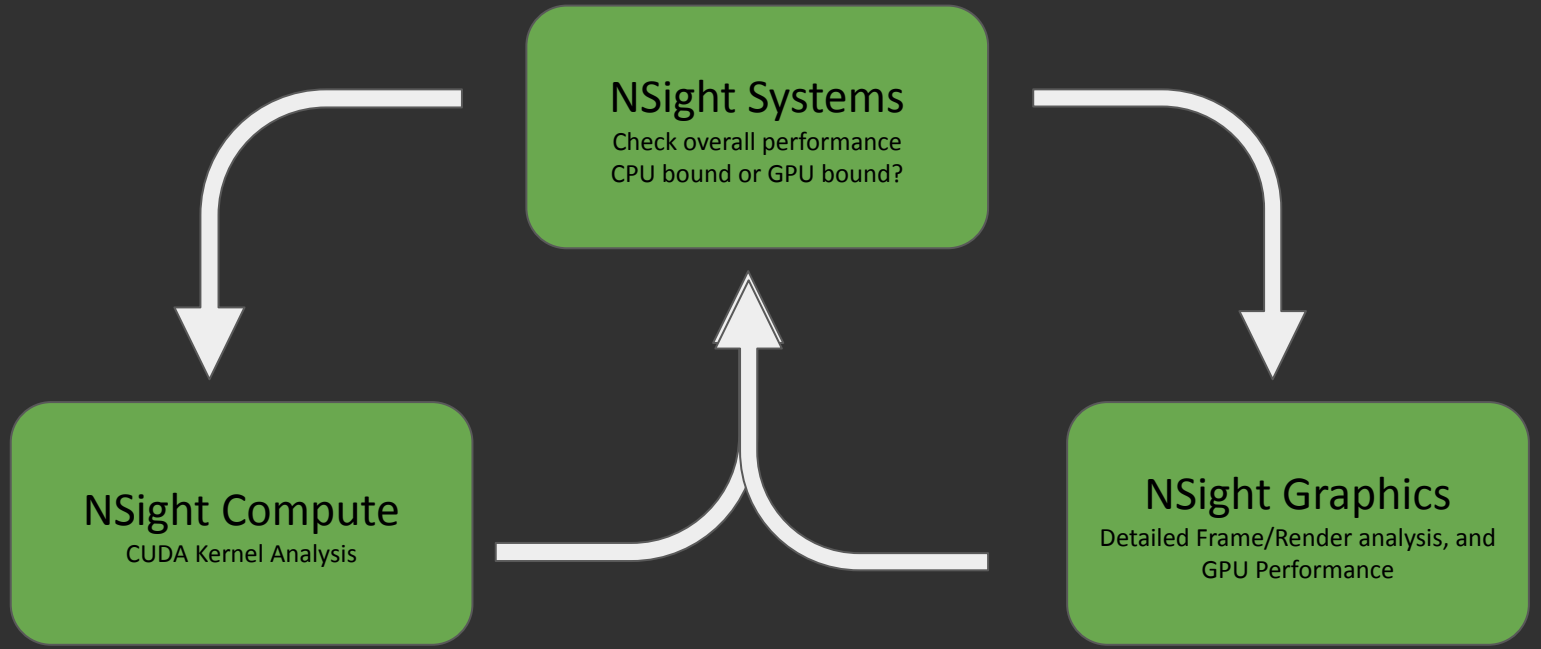
DMD: May 2022

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

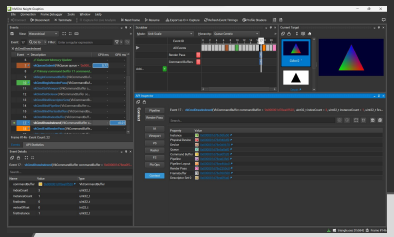
Workflow/Ecosystem



Workflow/Ecosystem

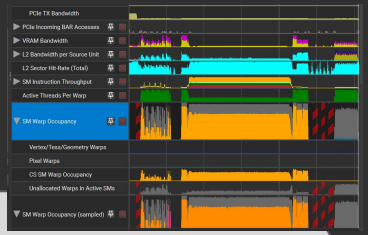
NSight Graphics
Detailed Frame/Render performance

Shader Debugger (Beta)



Frame Debugger

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



GPU Trace

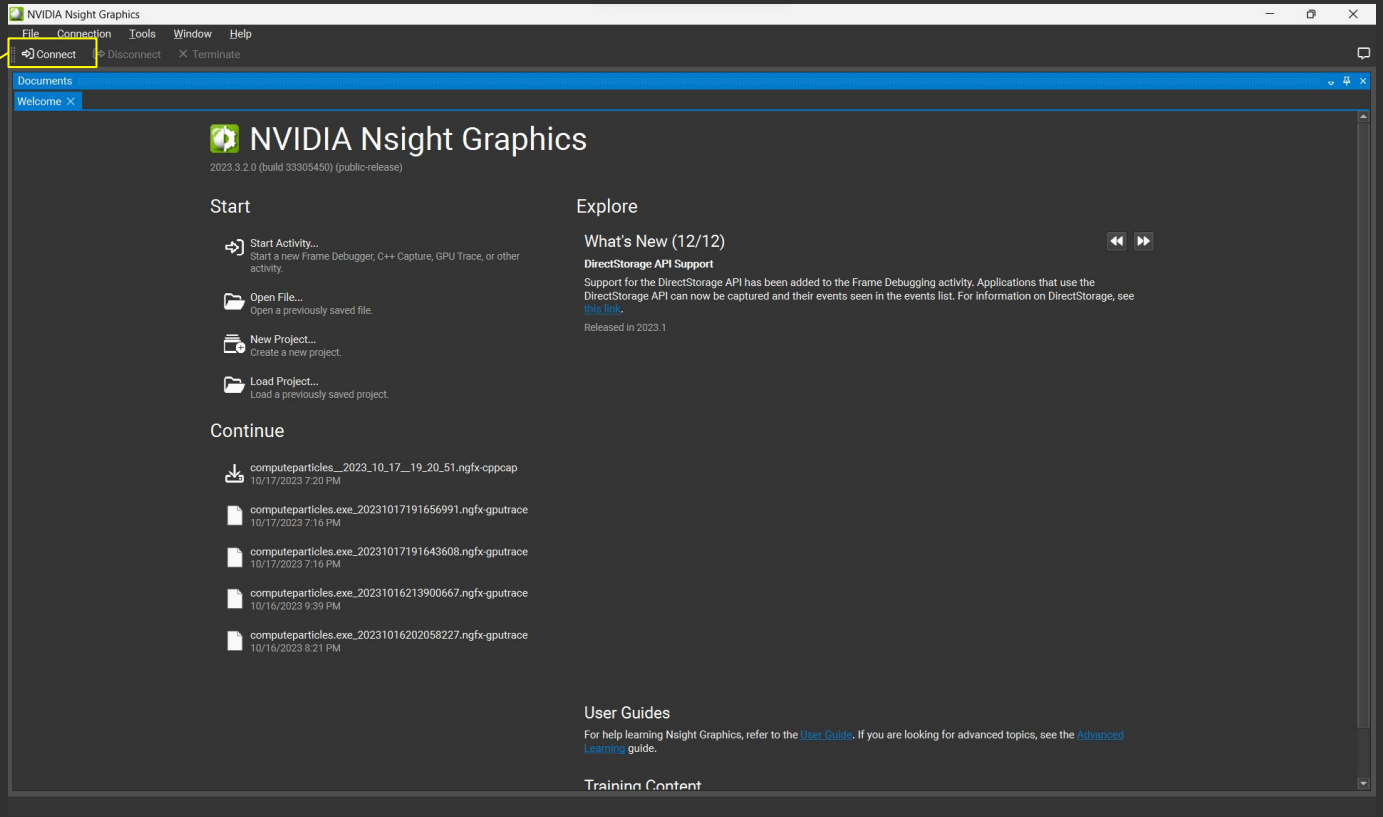
- HW Units throughput analysis
- Shader Profiling

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

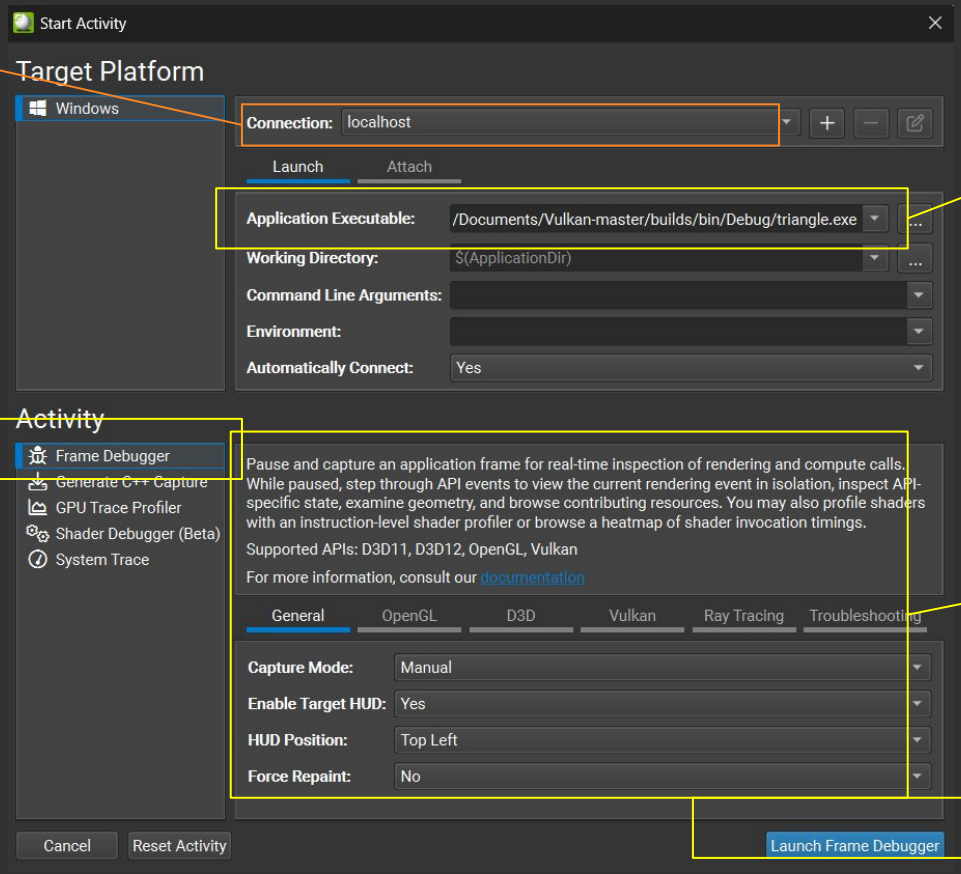
Remote Debugging

Filepath to the executable of your program

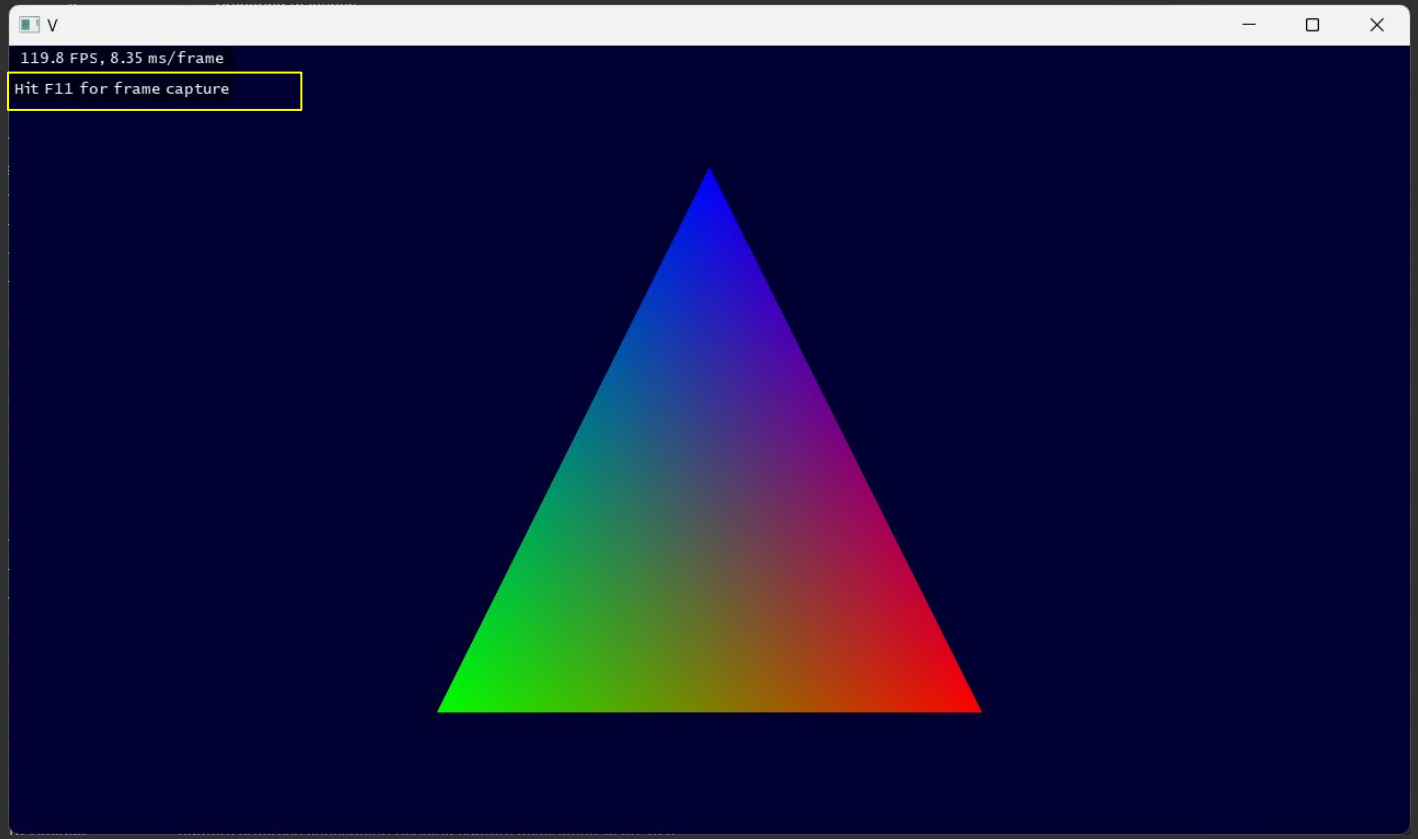
Choose this setting

Keep default settings

Launch the application

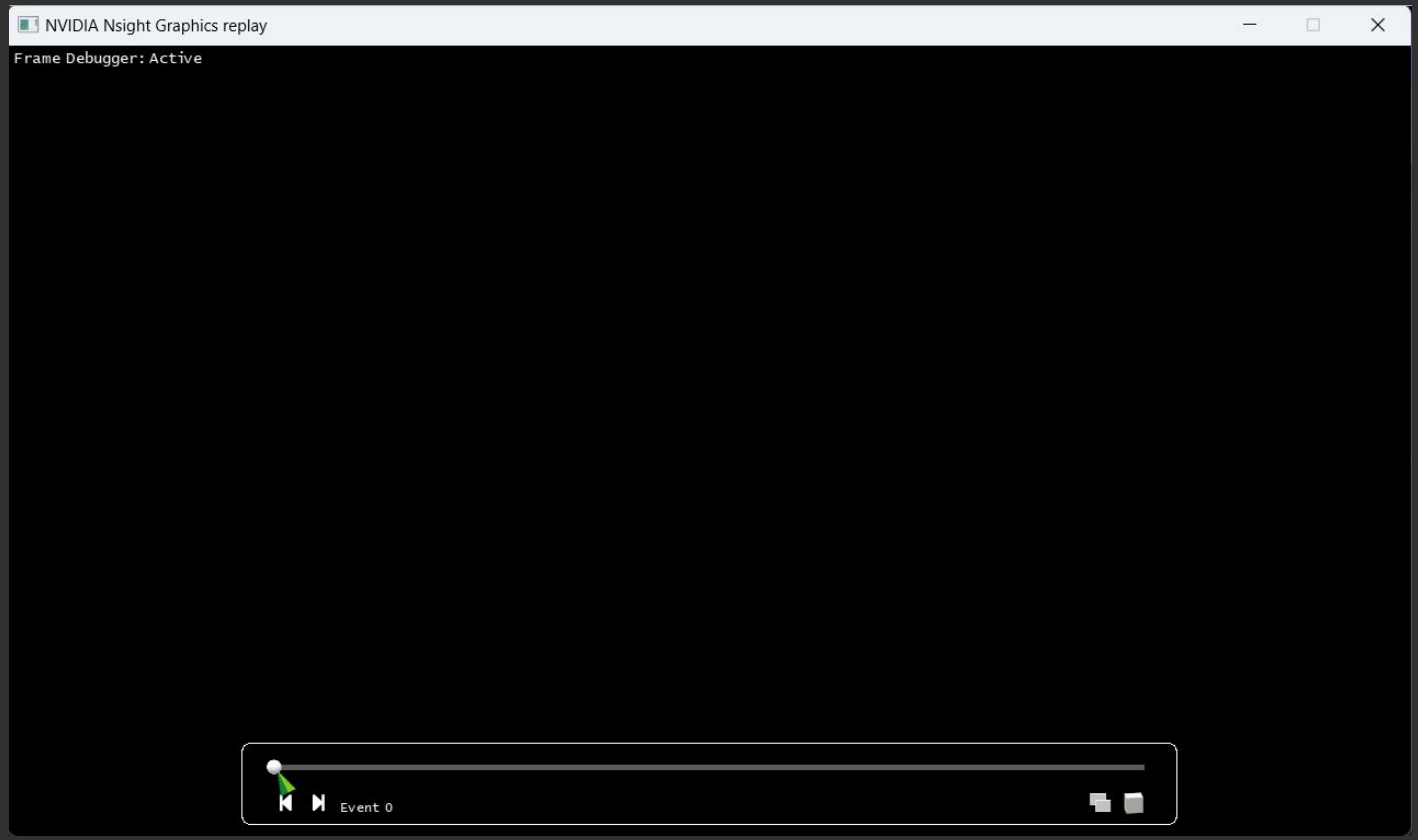


1. Frame Debugger - Getting Started

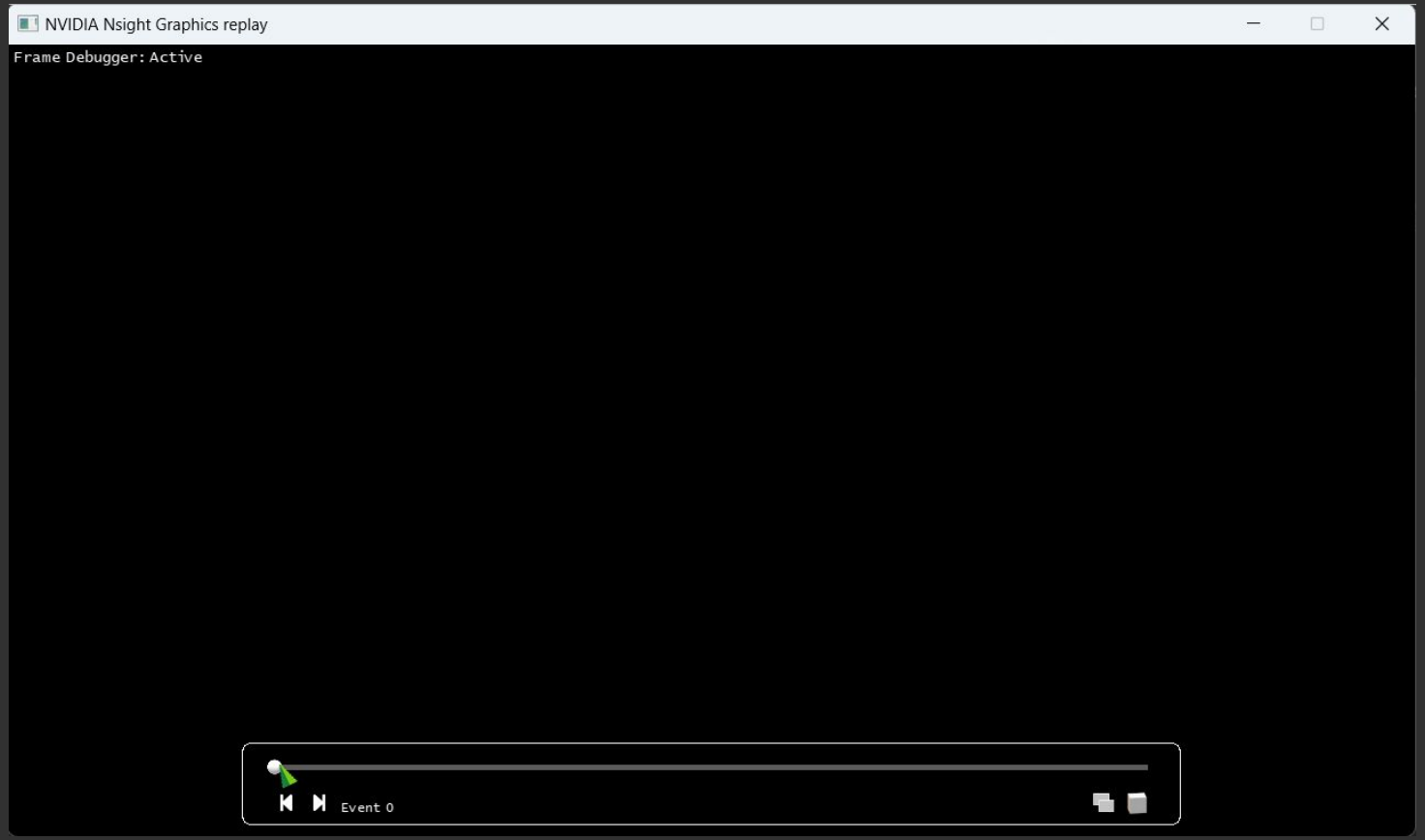


Sascha Willems - Triangle.exe

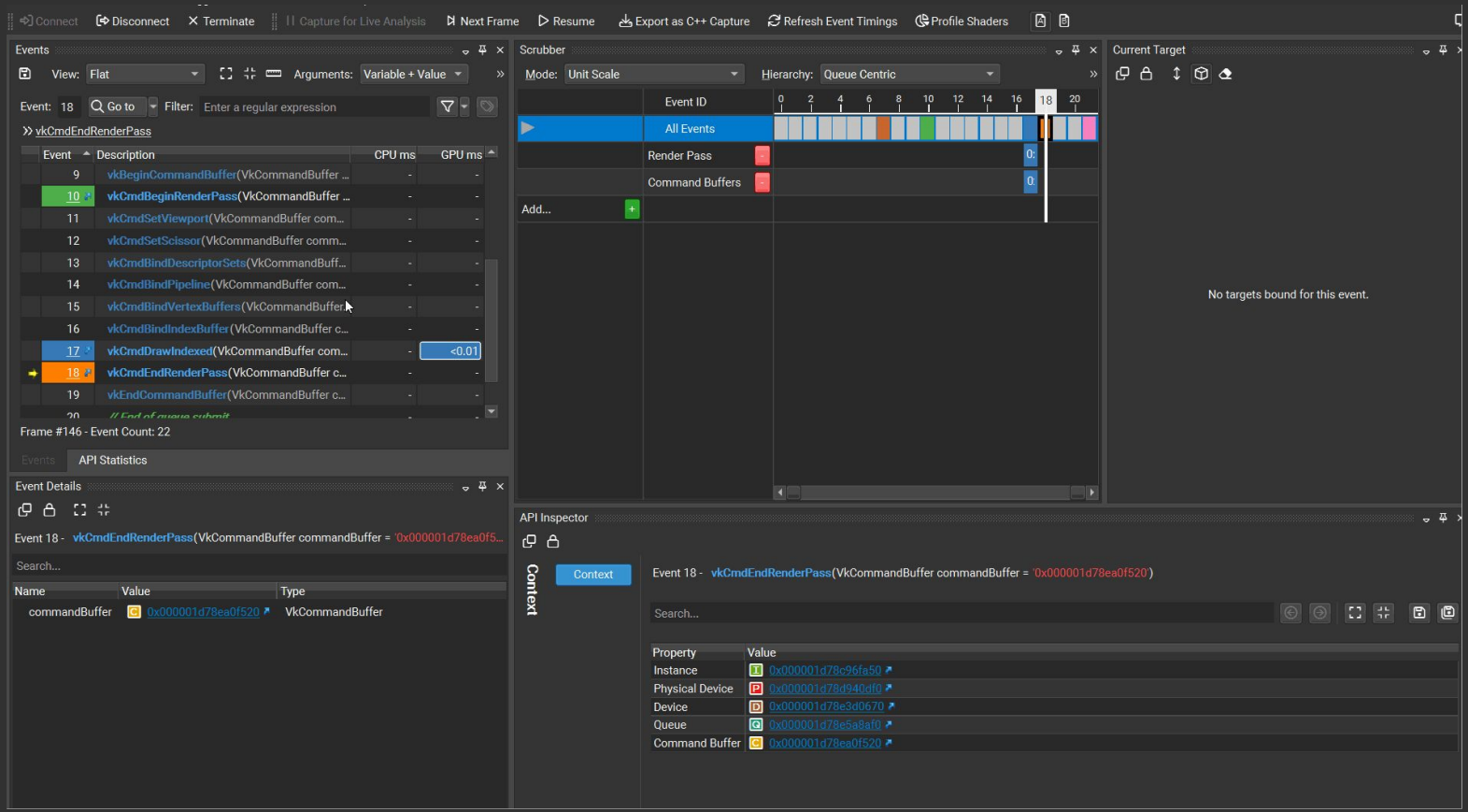
1. Frame Debugger - Getting Started



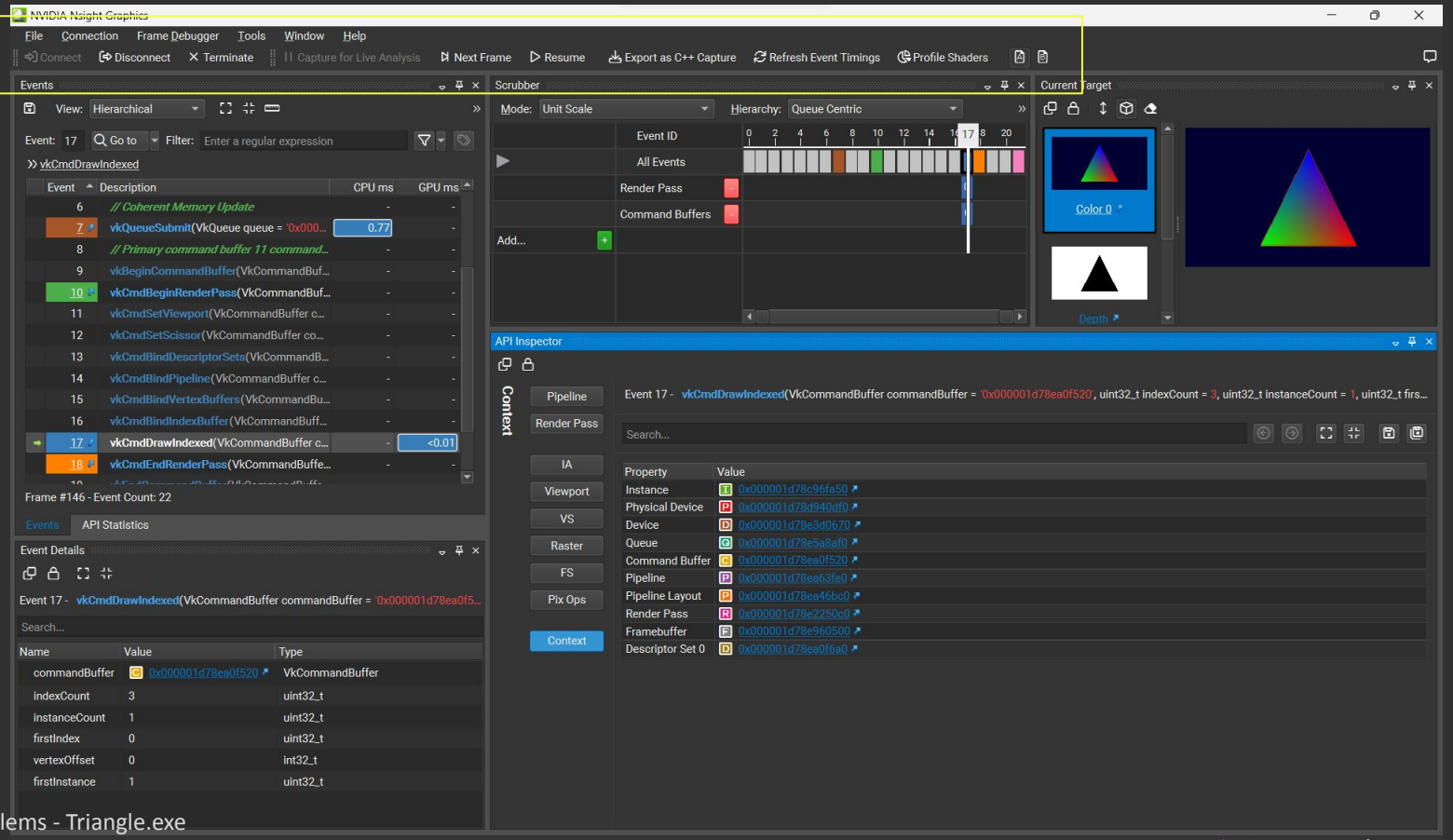
1. Frame Debugger - Getting Started



1. Frame Debugger



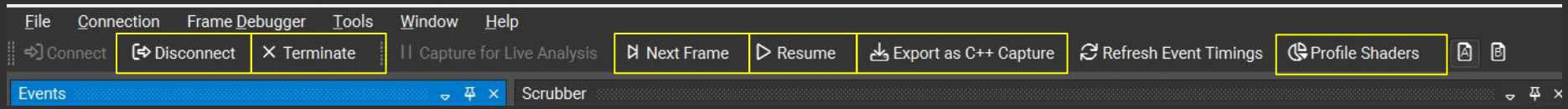
1. Frame Debugger - General Settings



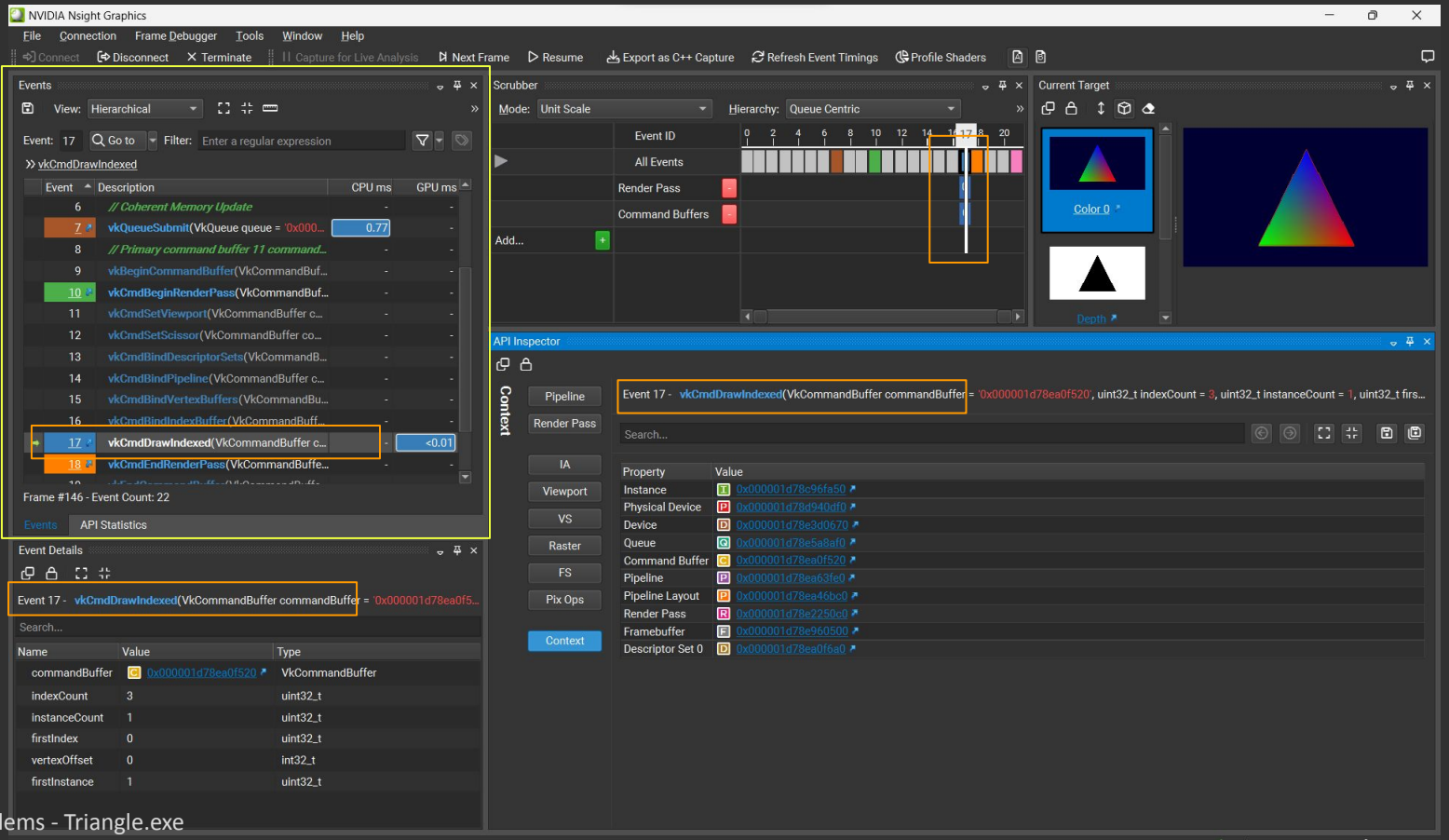
Sascha Willems - Triangle.exe

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...]



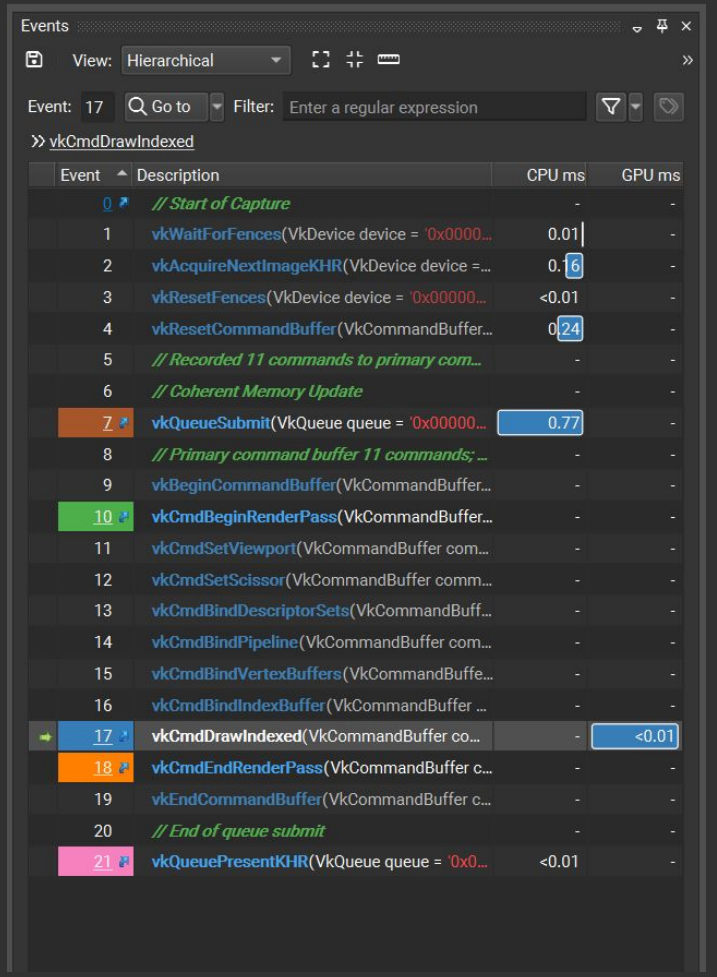
1. Frame Debugger - Event Viewer



Sascha Willems - Triangle.exe

1. Frame Debugger - Event Viewer

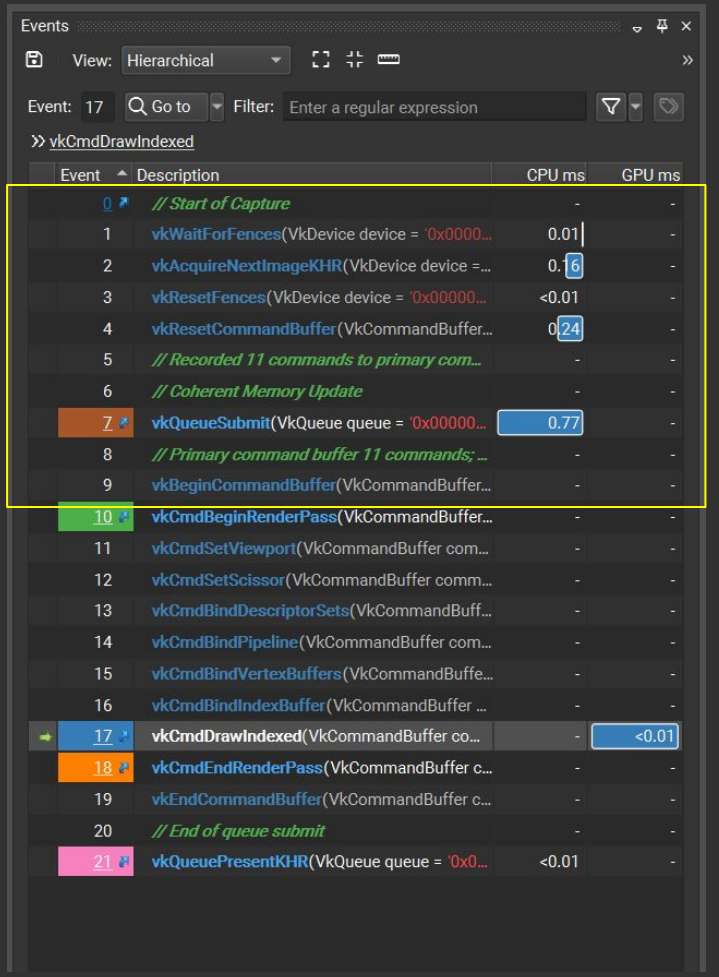
- **Can navigate or control the Frame Debugger session**
- A list view of all the commands in order of execution with respect to the frame



1. Frame Debugger - Event Viewer

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

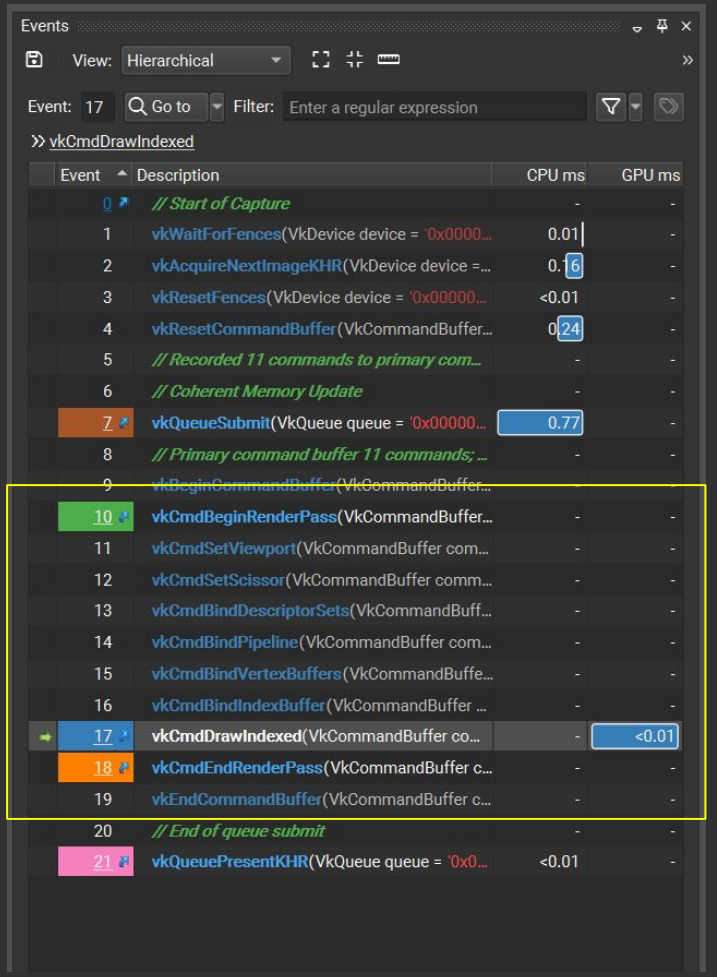
```
virtual void render()
{
    if (!prepared)
        return;

    // Use a fence to wait until the command buffer has finished execution before using it again
    vkWaitForFences(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 cycl
    uint32_t imageIndex;
    VkResult result = vkAcquireNextImageKHR(device, swapChain.swapChain, UINT64_MAX, presentCompleteSemaphores[currentFrame], VK_NULL_
    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 f
    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.framebuffer = frameBuffers[imageIndex];
    VK_CHECK_RESULT(vkBeginCommandBuffer(commandBuffers[currentBuffer], &cmdBufInfo));
```



1. Frame Debugger - Event Viewer

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

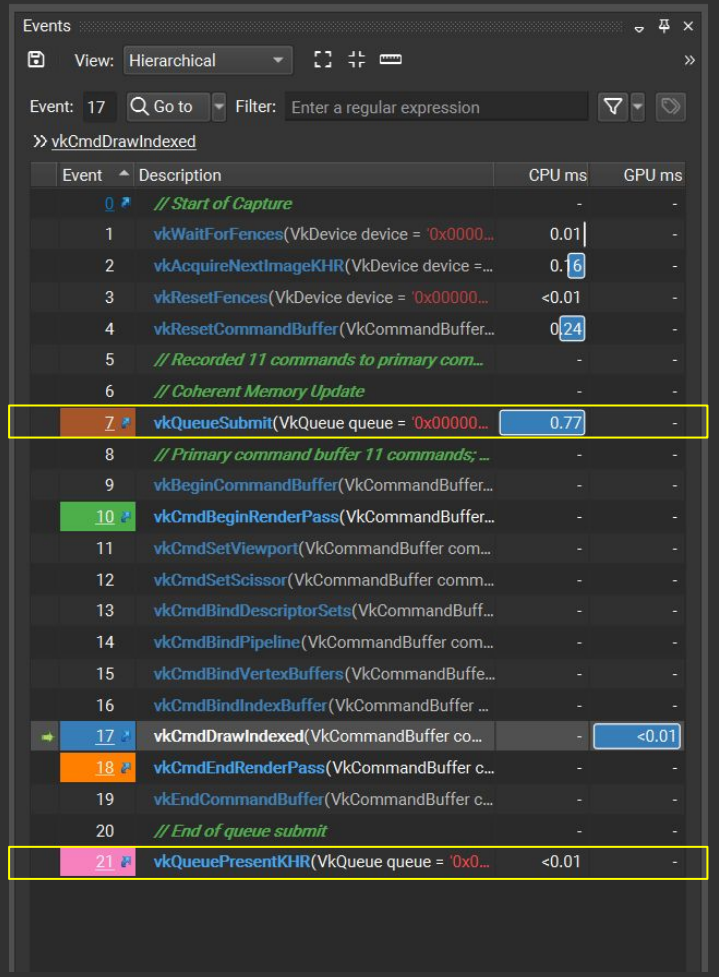
```
vkCmdBeginRenderPass(commandBuffers[currentBuffe
// Update dynamic viewport state
VkViewport viewport{};
viewport.height = (float)height;
viewport.width = (float)width;
viewport.minDepth = (float)0.0f;
viewport.maxDepth = (float)1.0f;
vkCmdSetViewport(commandBuffers[currentBuffer],
// Update dynamic 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[currentBu
// Bind the rendering pipeline
// The pipeline (state object) contains all stat
vkCmdBindPipeline(commandBuffers[currentBuffer],
// Bind triangle vertex buffer (contains positio
VkDeviceSize offsets[1]{ 0 };
vkCmdBindVertexBuffers(commandBuffers[currentBuf
// Bind triangle index buffer
vkCmdBindIndexBuffer(commandBuffers[currentBuffe
// Draw indexed triangle
vkCmdDrawIndexed(commandBuffers[currentBuffer],
vkCmdEndRenderPass(commandBuffers[currentBuffer]
// Ending the render pass will add an implicit b
// VK_IMAGE_LAYOUT_PRESENT_SRC_KHR for presentin
VK_CHECK_RESULT(vkEndCommandBuffer(commandBuffer
```



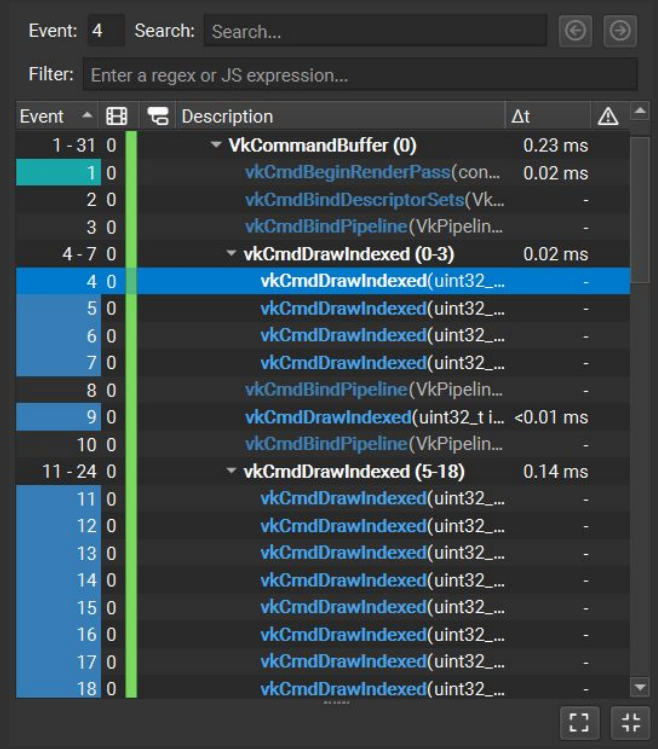
1. Frame Debugger - Event Viewer

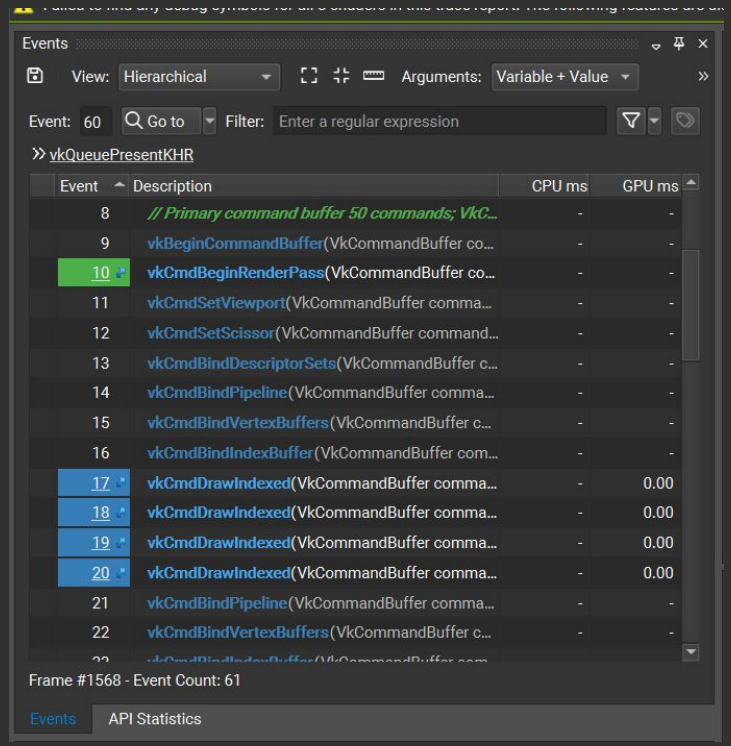
1:1 match with the render() function from Vulkan's Triangle sample

```
// 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[
presentInfo.swapchainCount = 1;
presentInfo.pSwapchains = &swapChain.swapChain;
presentInfo.pImageIndices = &imageIndex;
result = vkQueuePresentKHR(queue, &presentInfo);

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

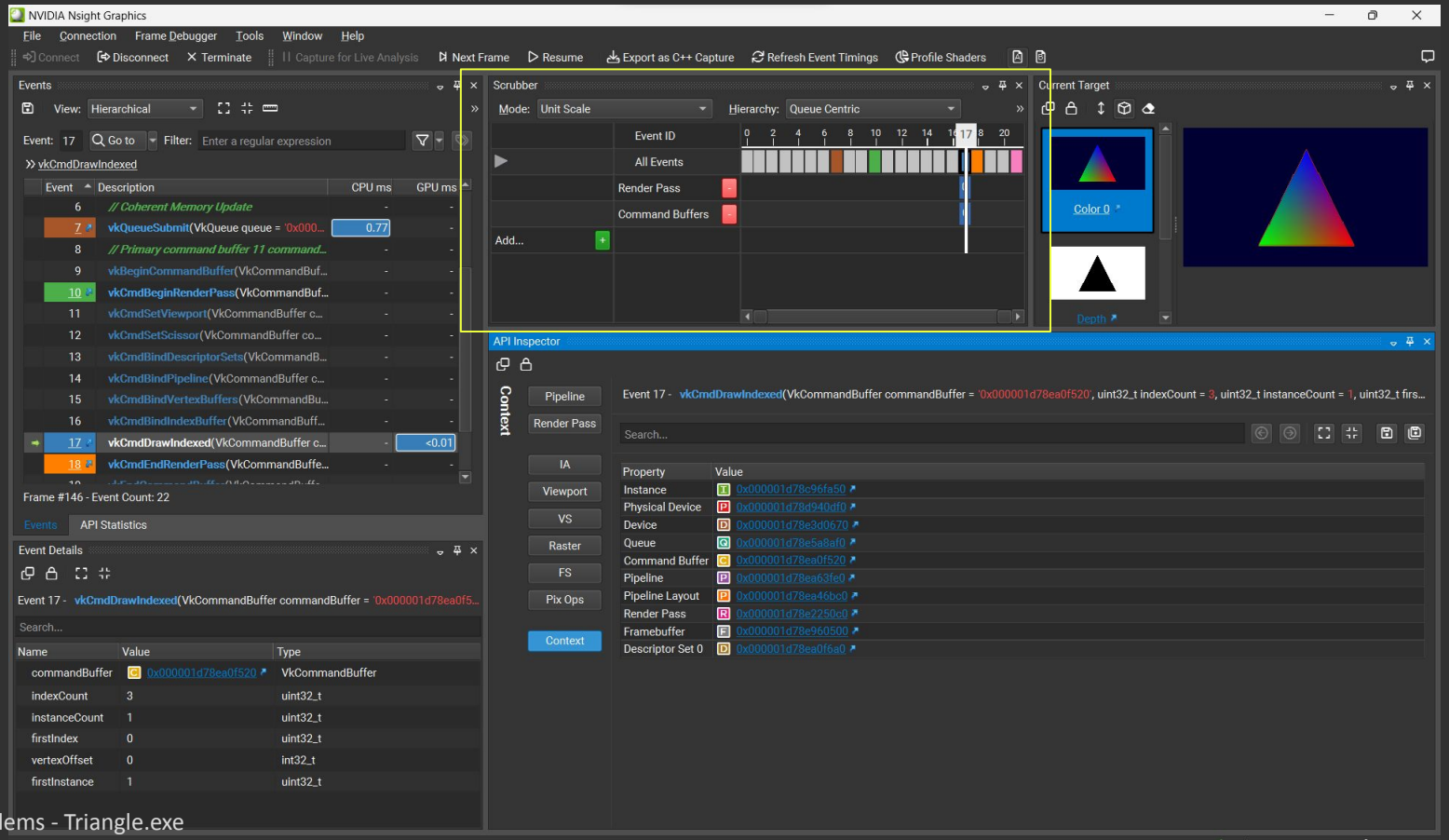


1. Frame Debugger - Event Viewer





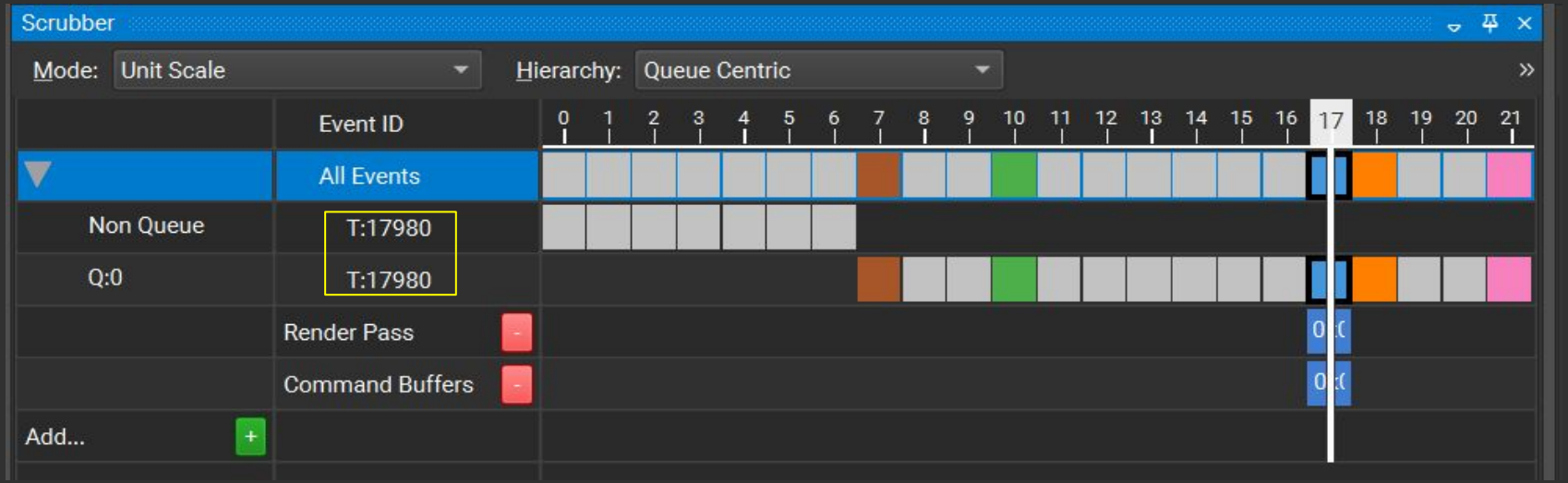
1. Frame Debugger - Scrubber



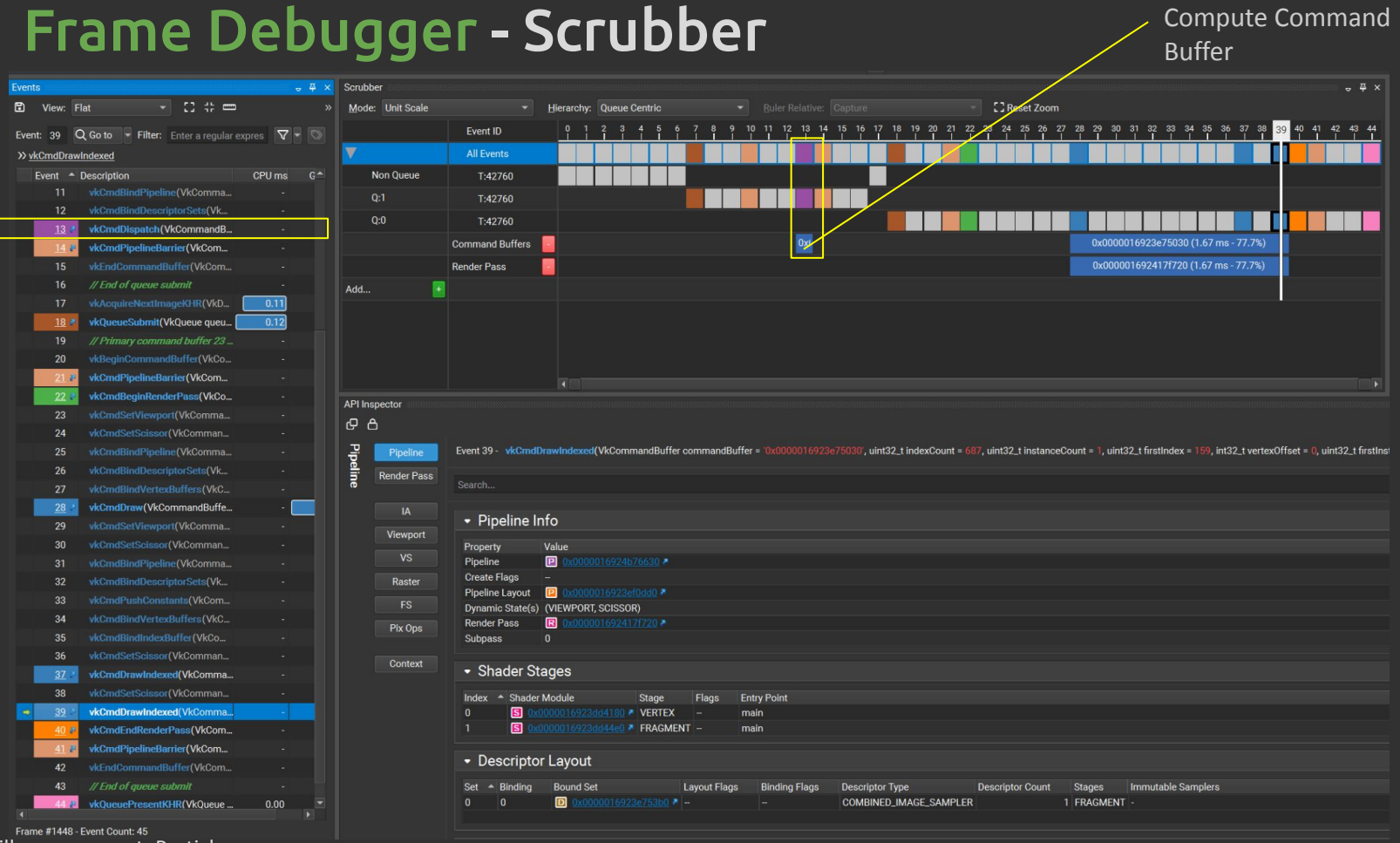
Sascha Willems - Triangle.exe

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



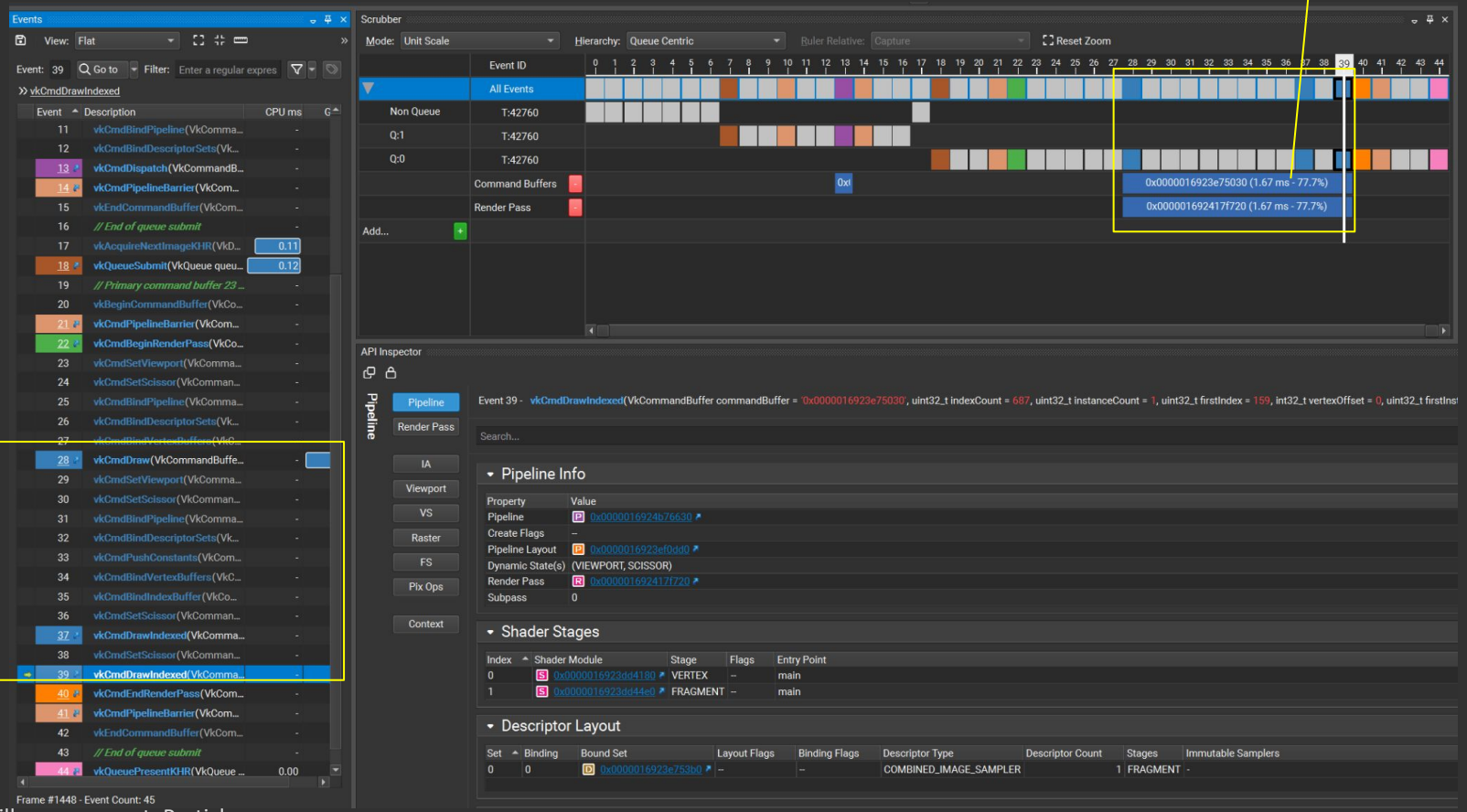
1. Frame Debugger - Scrubber



Sascha Willems - computeParticles.exe

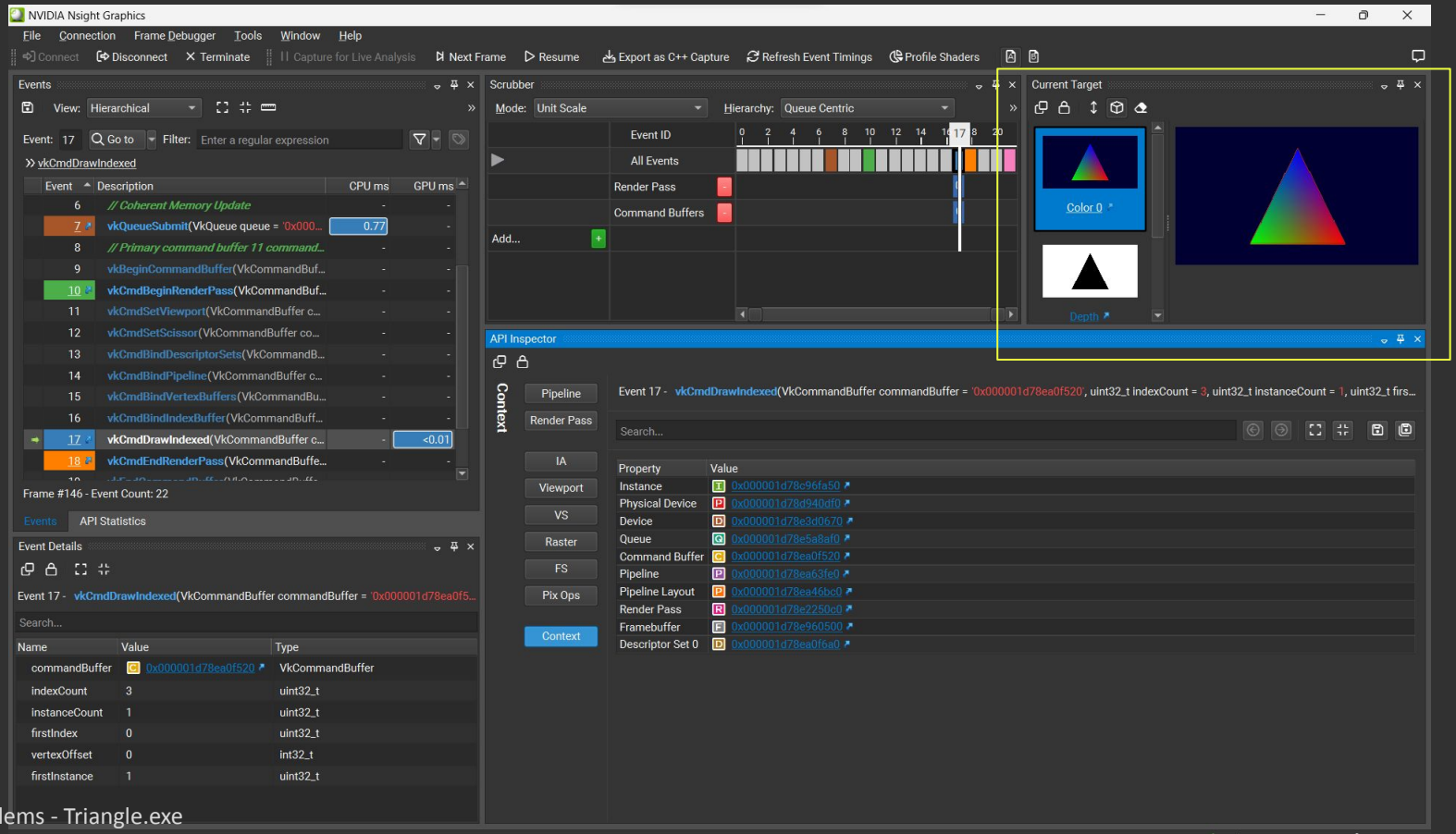
1. Frame Debugger - Scrubber

Graphics Command Buffer



Sascha Willems - computeParticles.exe

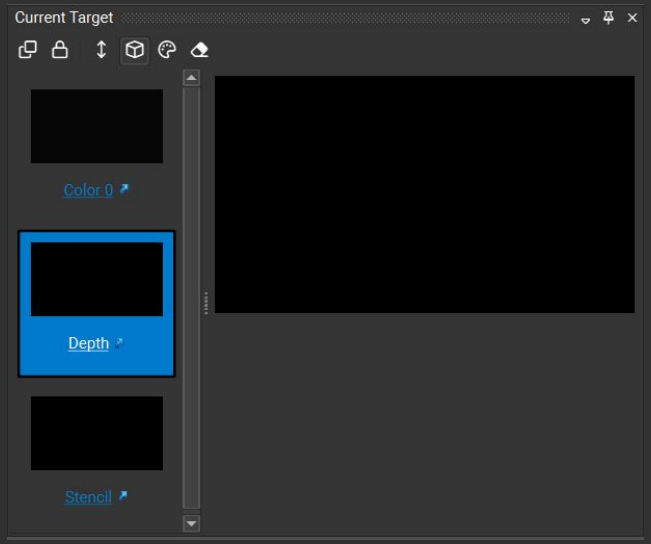
1. Frame Debugger - Current Target

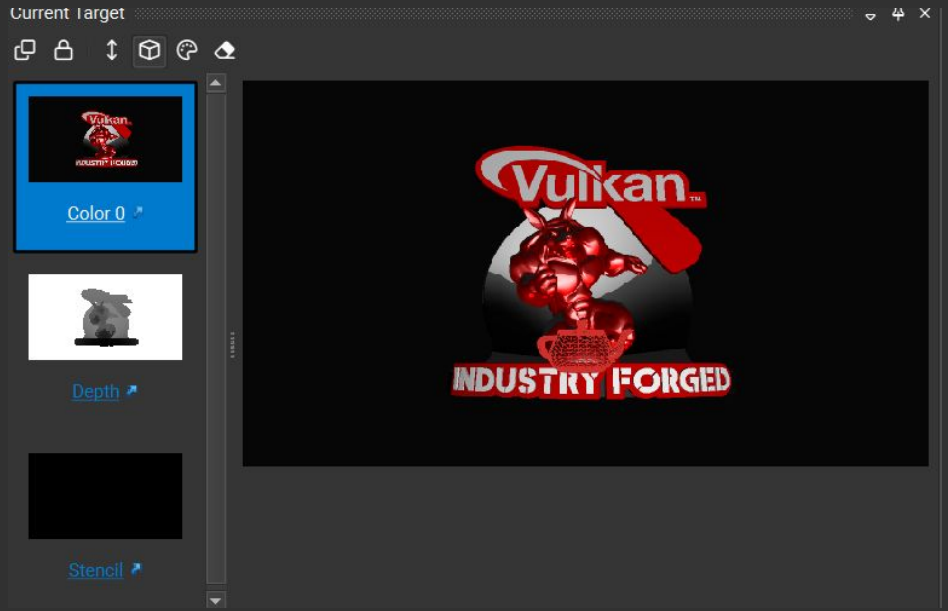


Sascha Willems - Triangle.exe

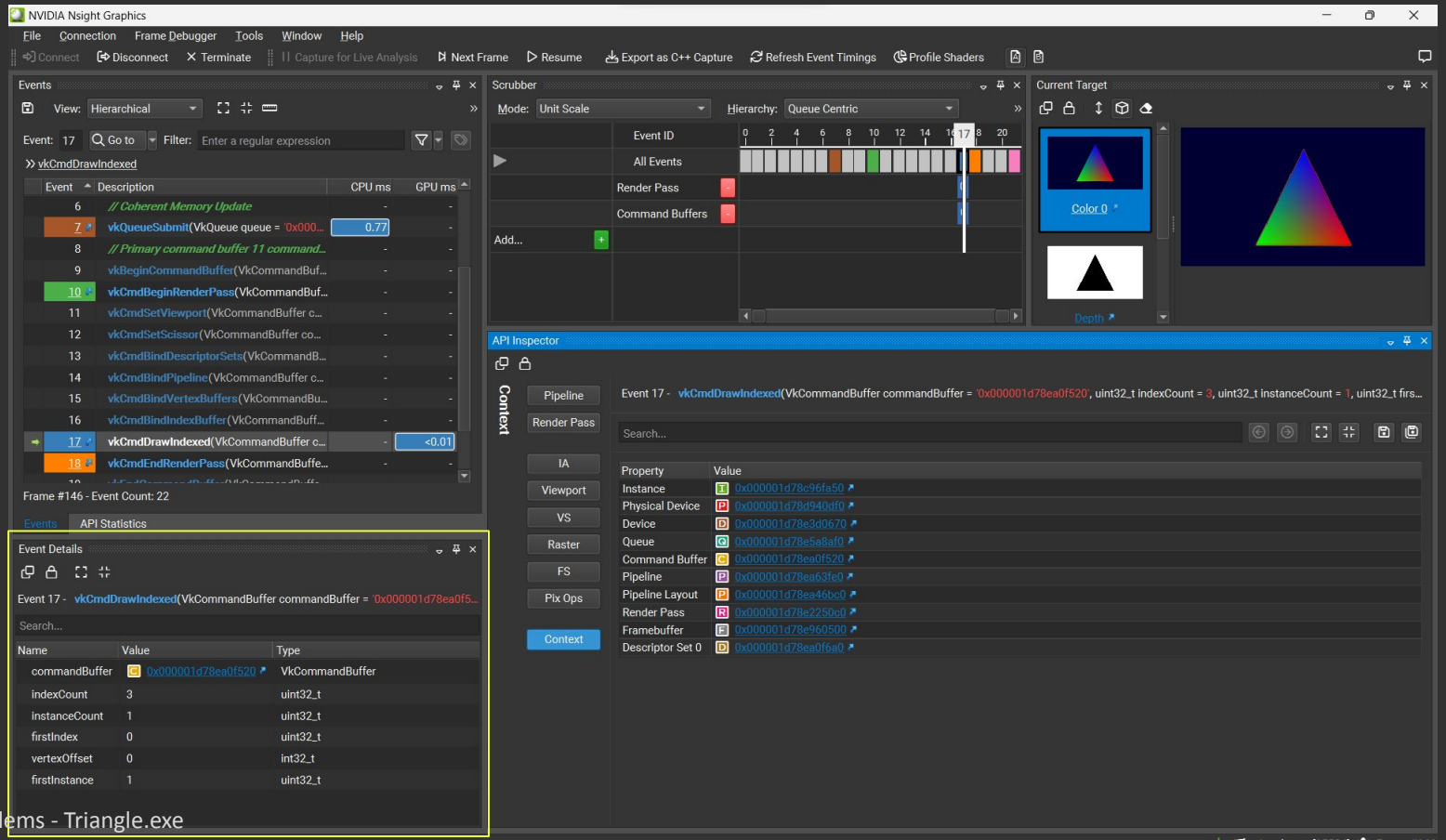
1. Frame Debugger - Current Target

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





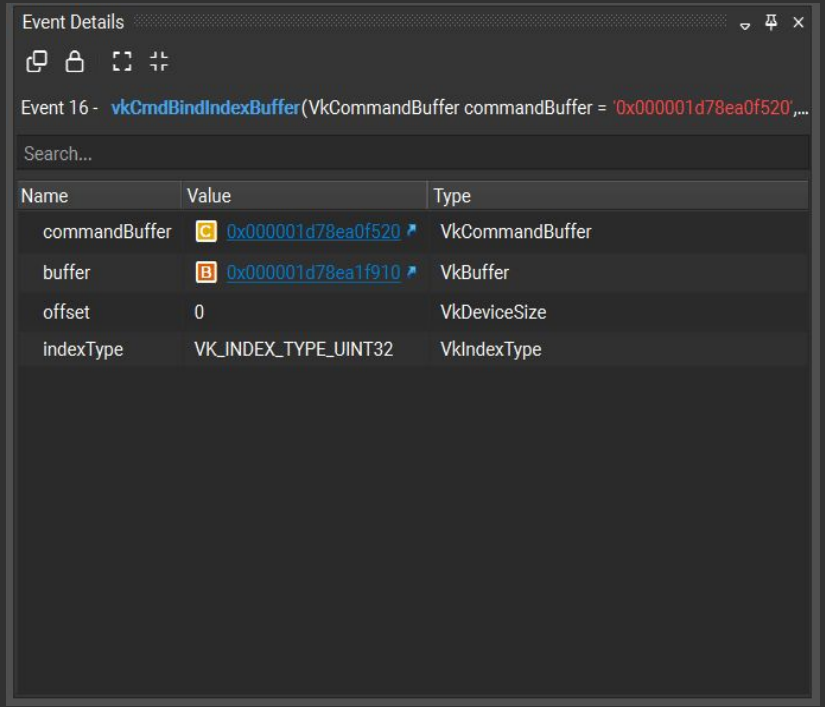
1. Frame Debugger - Event Details



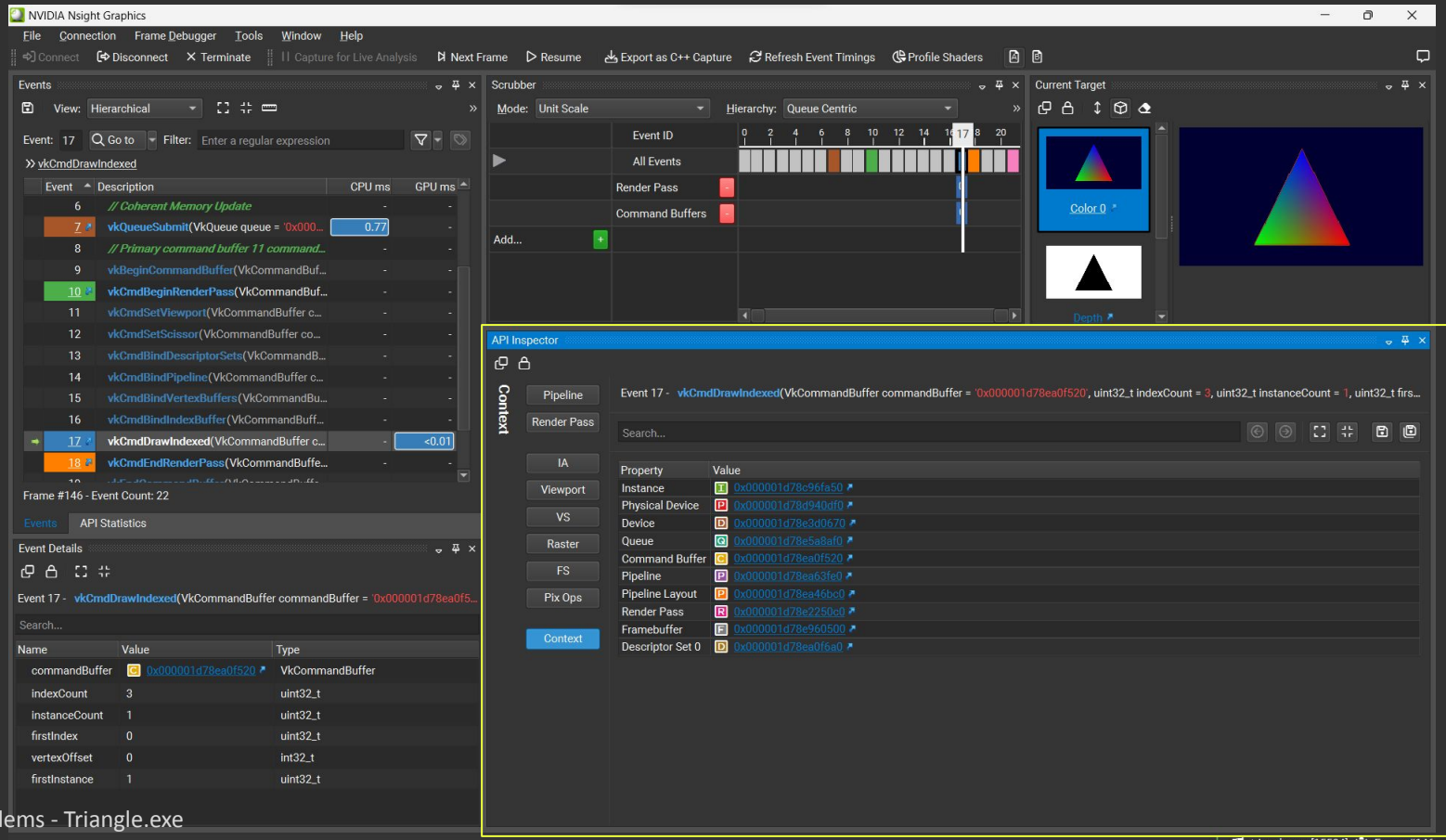
Sascha Willems - Triangle.exe

1. Frame Debugger - Event Details

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

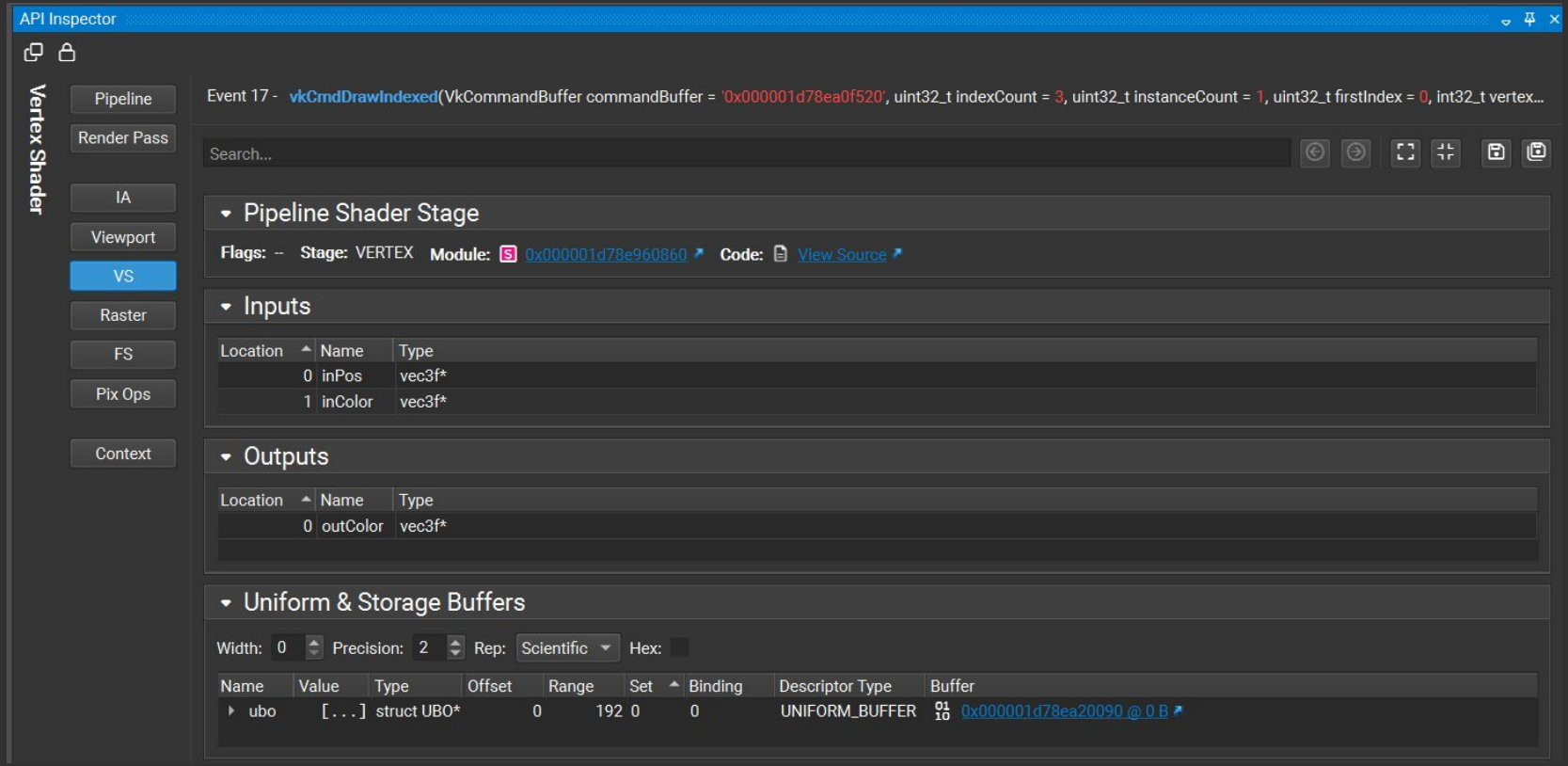


1. Frame Debugger - API Inspector

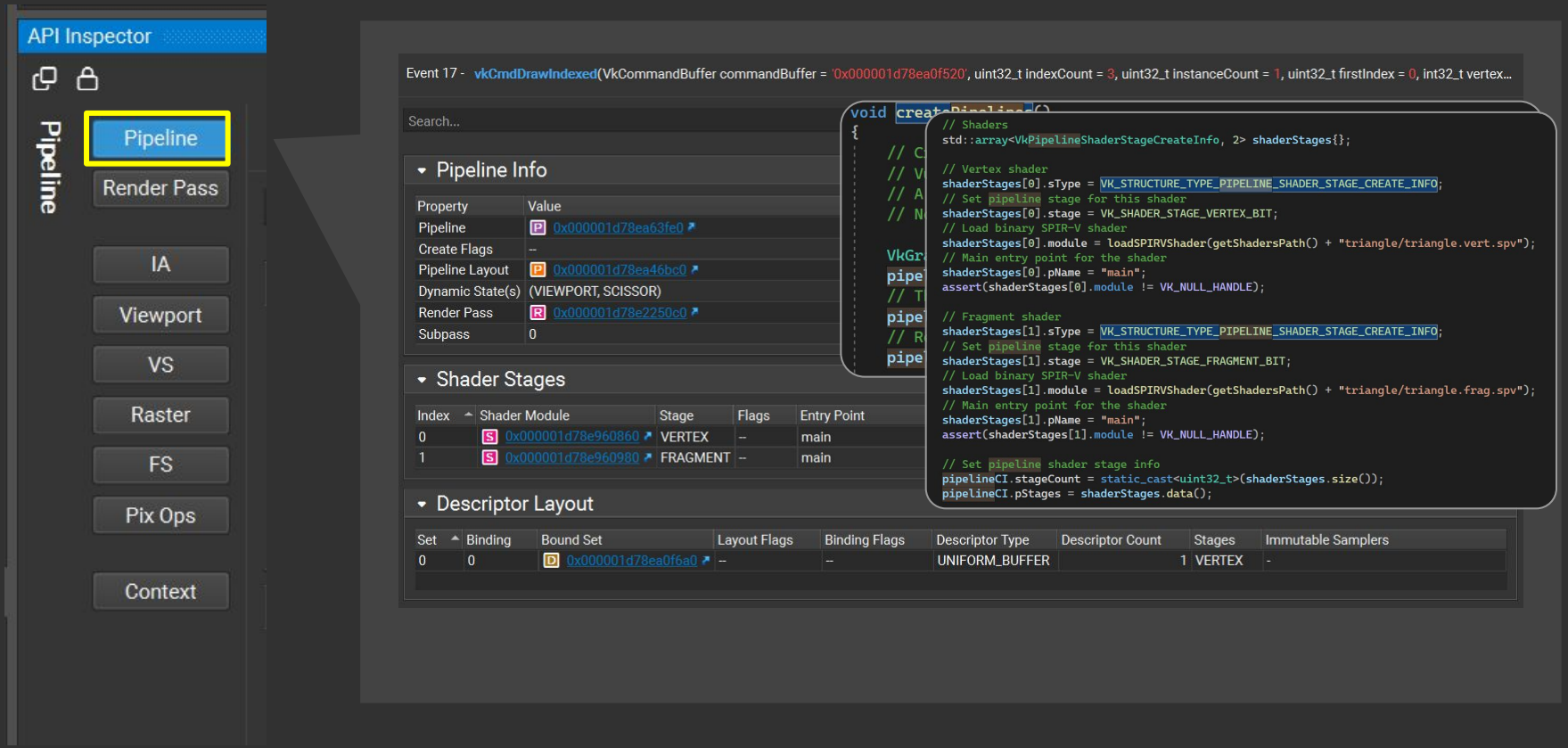


Sascha Willems - Triangle.exe

1. Frame Debugger - API Inspector

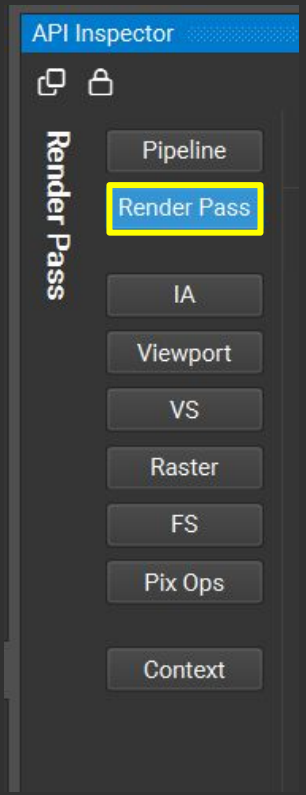


1. Frame Debugger - API Inspector



Sascha Willems - Triangle.exe

1. Frame Debugger - API Inspector



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

Search...

Render Pass: 0x000001d78e2250c0 Subpass: 0 Framebuffer: 0x000001d78e960500

Property	Value
Width	1280
Height	720
Layers	1

▾ Attachments

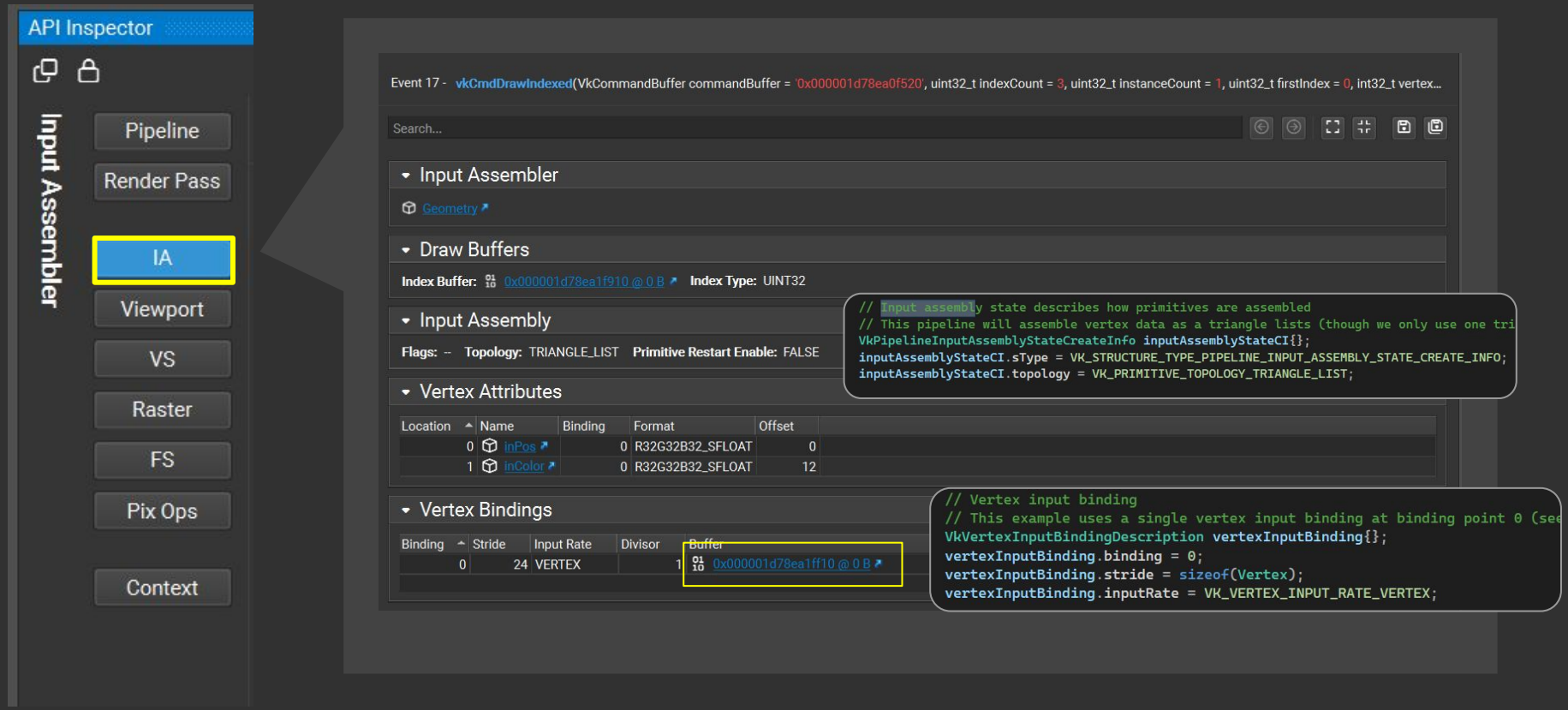
Filter...

Image	Index
0x000001d78ea04050	0
0x000001d78ea049f0	1

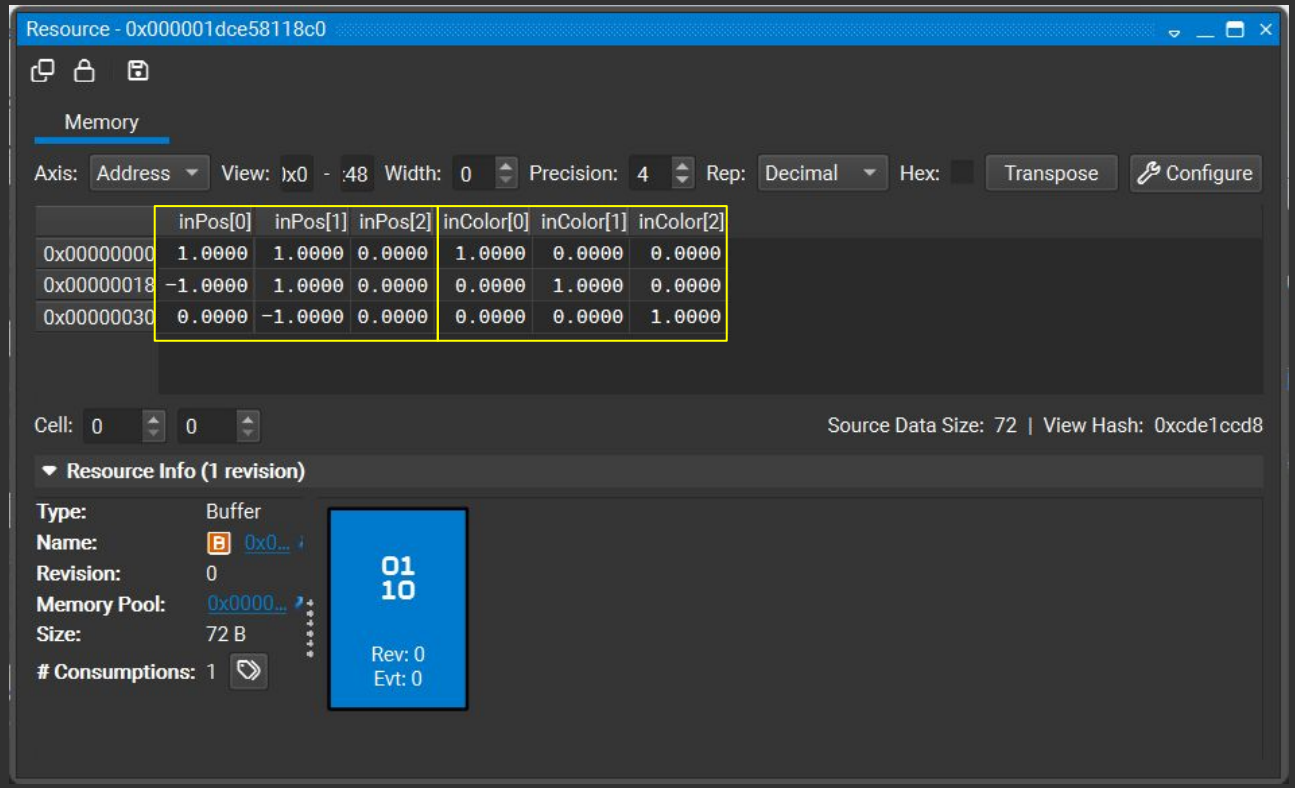
Property	Value
Aspect Mask	COLOR
Format	B8G8R8A8_UNORM
View Type	2D
Components	{R: R, G: G, B: B, A: A}
Flags	--
Mip Level(s)	[0 - 0]
Layer(s)	[0 - 0]
Load Op.	CLEAR
Store Op.	STORE
Clear Value	float32 = {R: 0, G: 0, B: 0.2, A: 1}

```
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_CA
    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_CA
    attachments[1].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
    attachments[1].finalLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATT

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

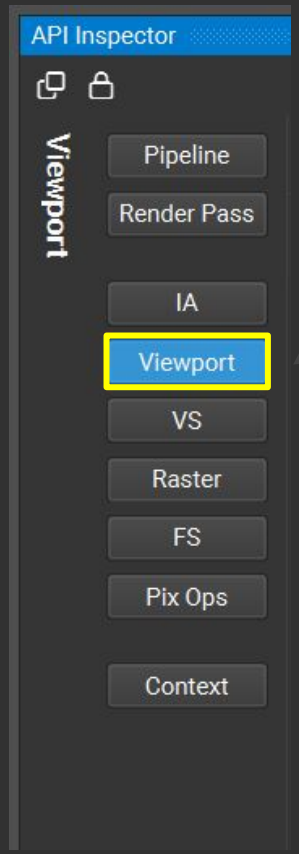
1. Frame Debugger - API Inspector



1. Frame Debugger - Resource View



1. Frame Debugger - API Inspector



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

Search...

Viewports

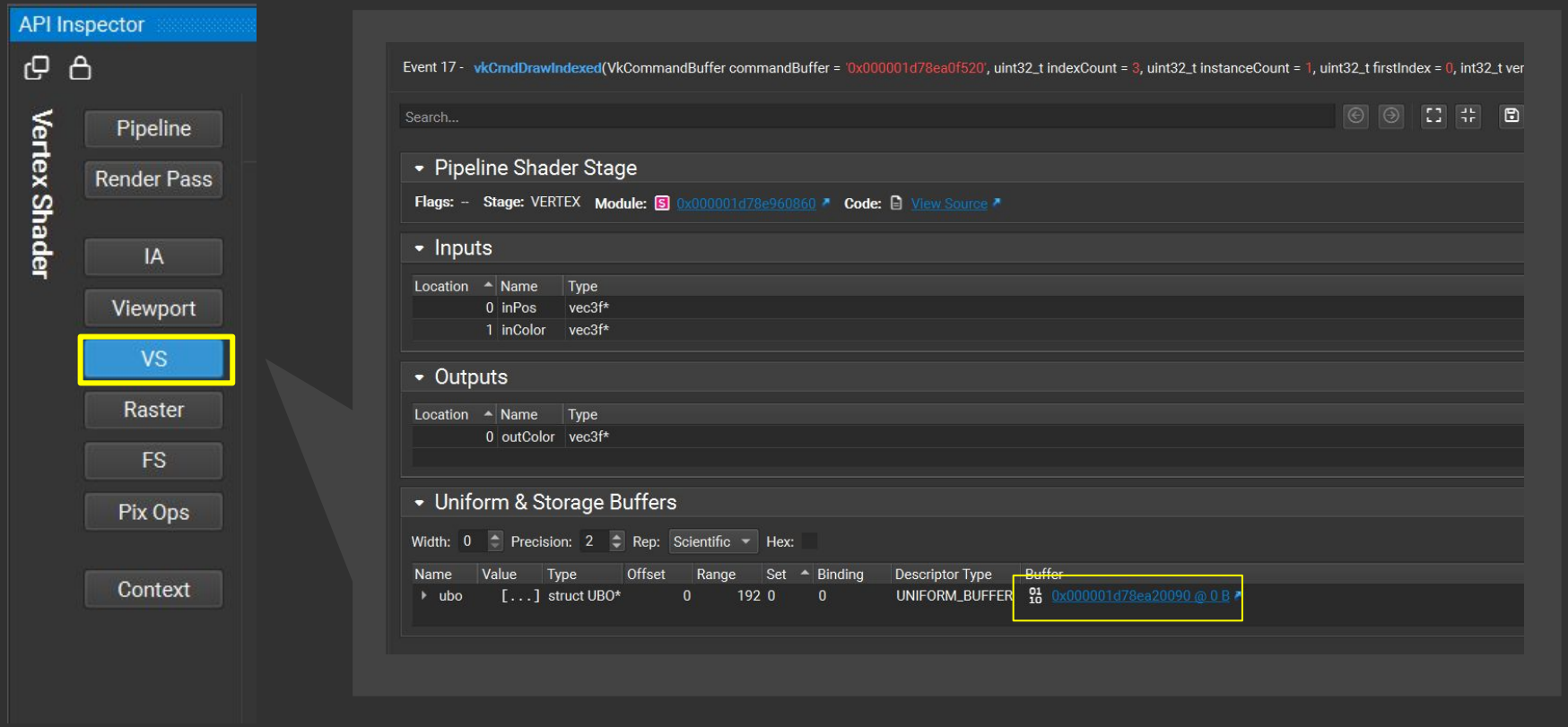
Index	X	Y	Width	Height	Depth
0	0.00	0.00	1280.00	720.00	[0.00 - 1.00]

Scissors

Index	X	Y	Width	Height
0	0	0	1280	720

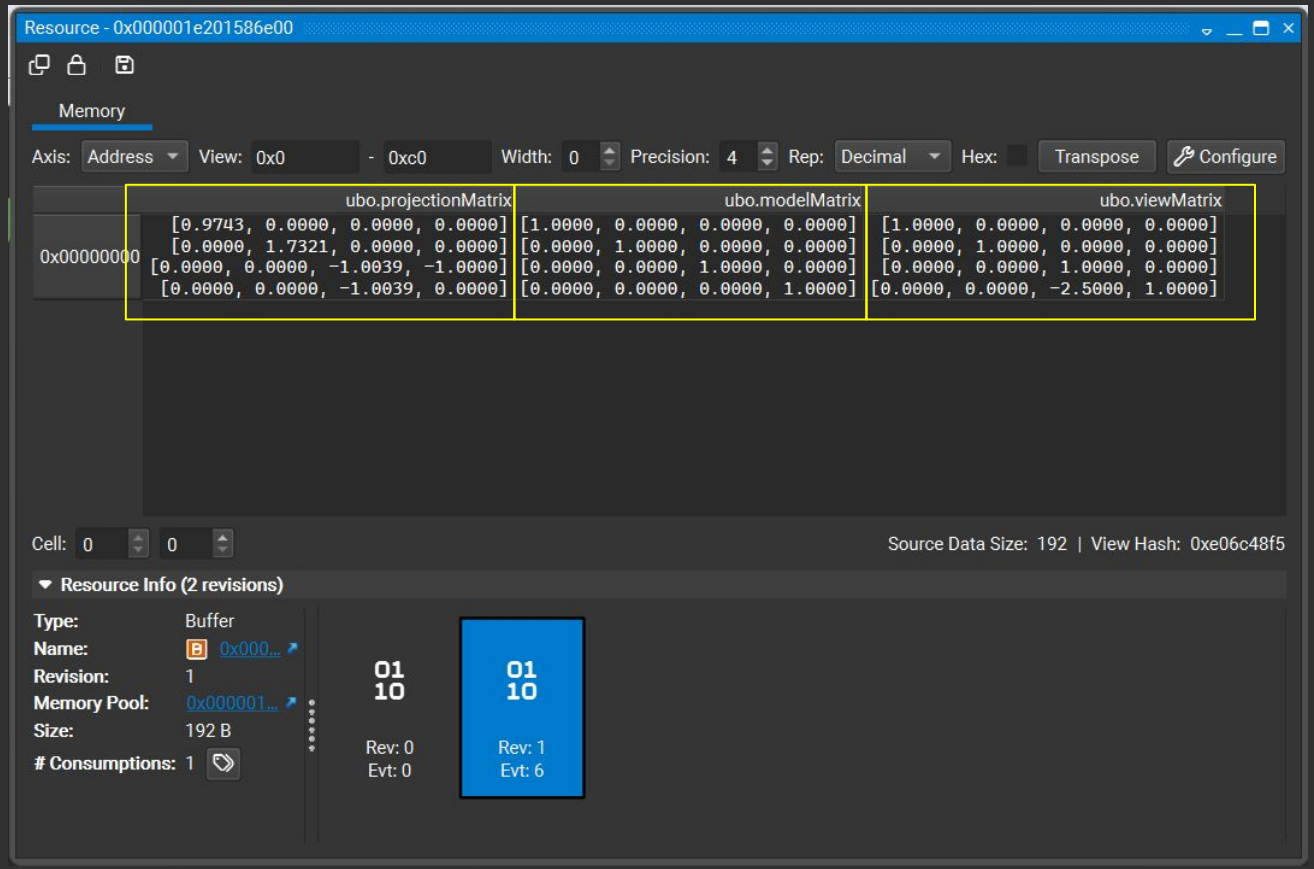
```
// Update dynamic viewport state
VkViewport viewport{};
viewport.height = (float)height;
viewport.width = (float)width;
viewport.minDepth = (float)0.0f;
viewport.maxDepth = (float)1.0f;
vkCmdSetViewport(commandBuffers[currentBuffer], 0, 1, &viewport);
// Update dynamic scissor state
VkRect2D scissor{};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

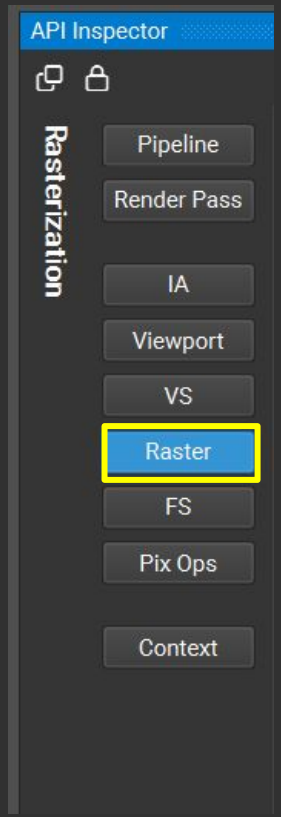


Sascha Willems - Triangle.exe

1. Frame Debugger - API Inspector



1. Frame Debugger - API Inspector



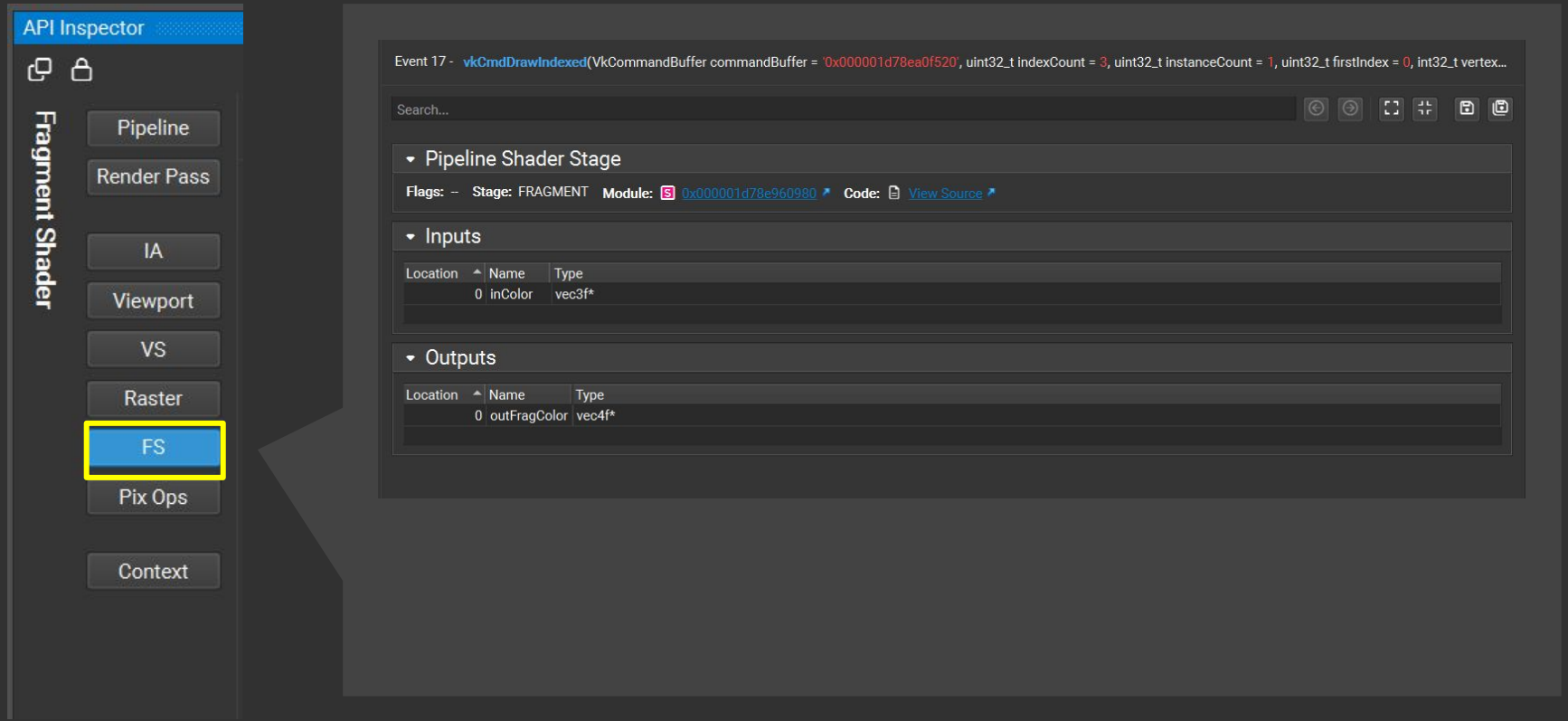
Event 17 - vkCmdDrawIndexed(VkCommandBuffer commandBuffer = 0x000001d78ea0f520, uint32_t indexCount = 3, uint32_t instanceCount = 1, uint32_t firstIndex = 0, int32_t vertex...

Search...

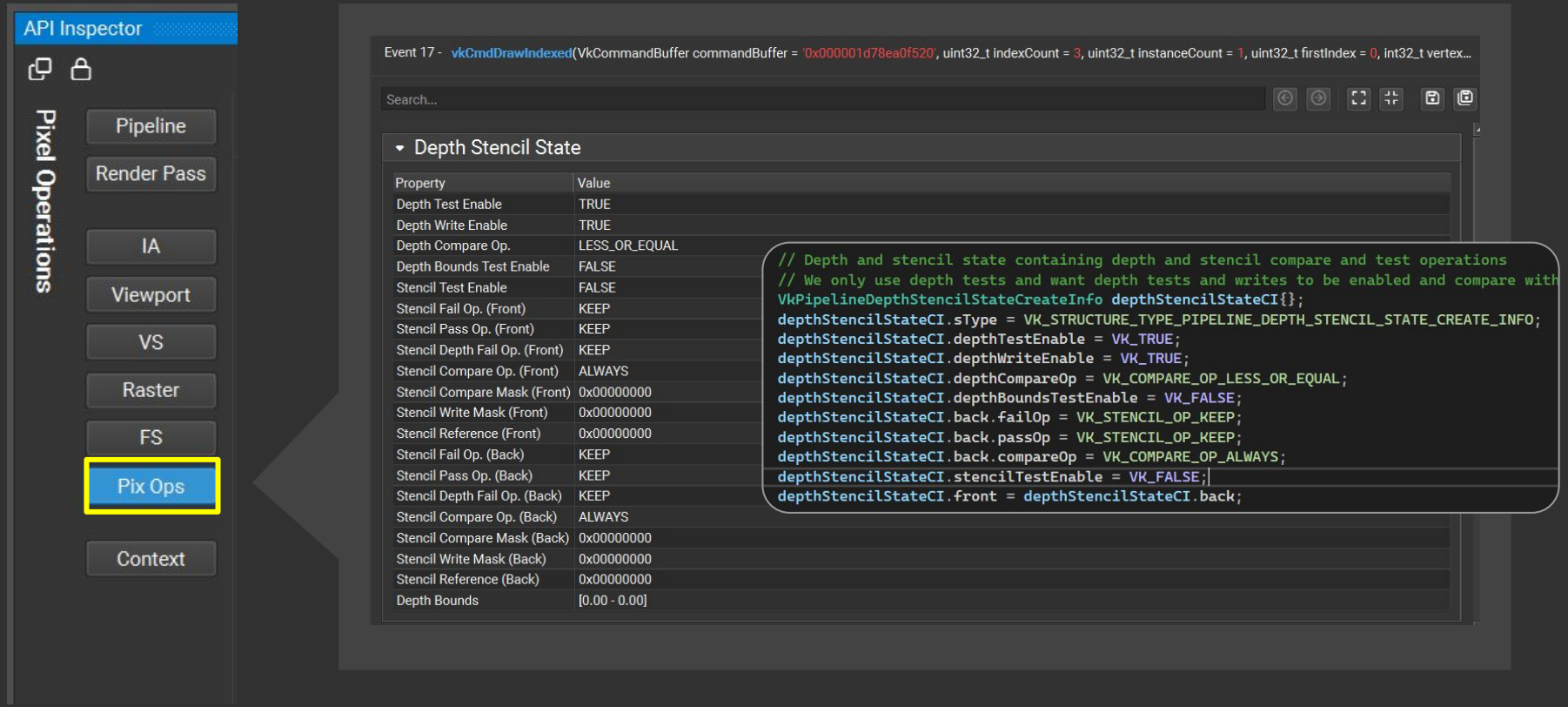
Property	Value
Depth Clamp Enable	FALSE
Rasterizer Discard Enable	FALSE
Polygon Mode	FILL
Cull Mode	--
Front Face	COUNTER_CLOCKWISE
Depth Bias Enable	FALSE
Depth Bias Constant Factor	0.00
Depth Bias Clamp	0.00
Depth Bias Slope Factor	0.00
Line Width	1.00

```
// 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;
```

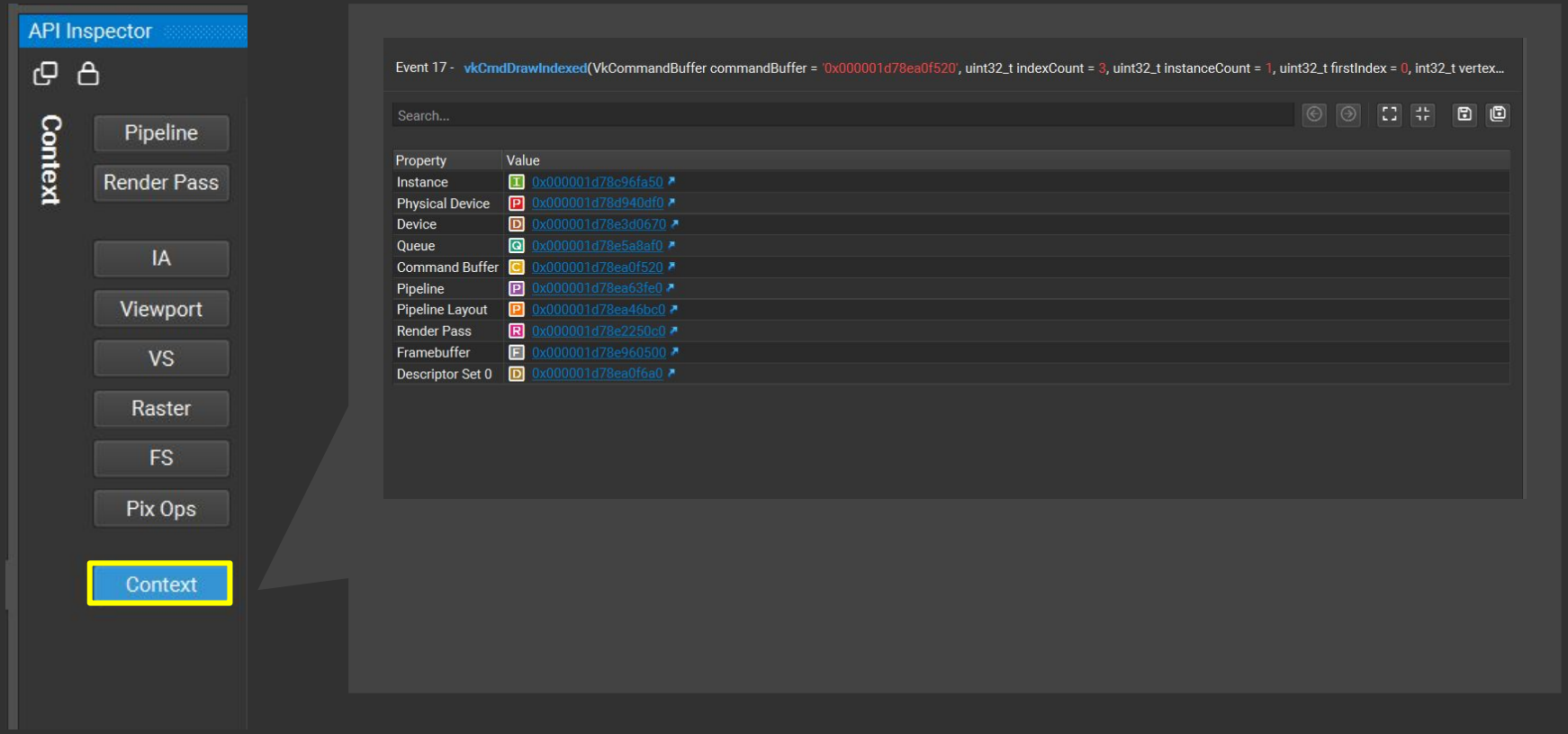
1. Frame Debugger - API Inspector



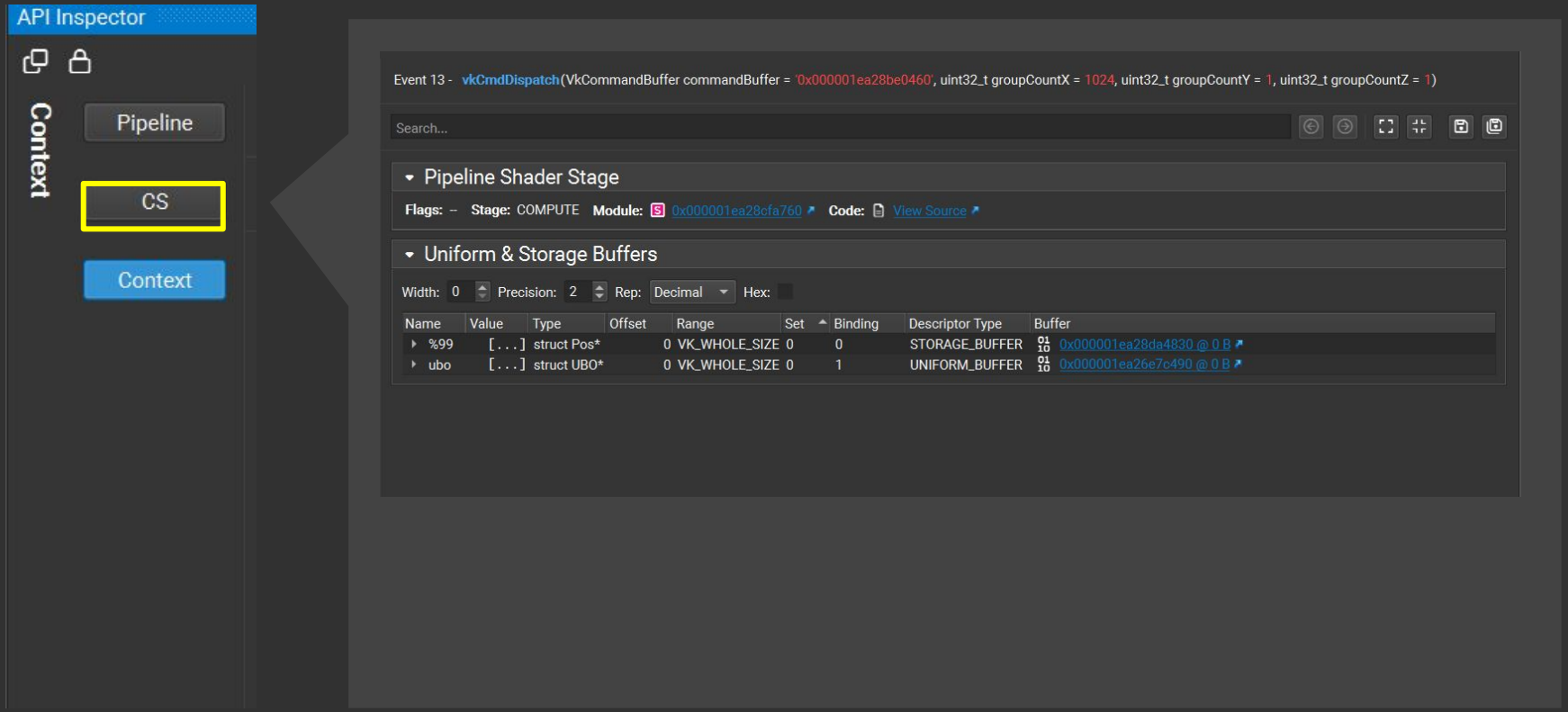
1. Frame Debugger - API Inspector



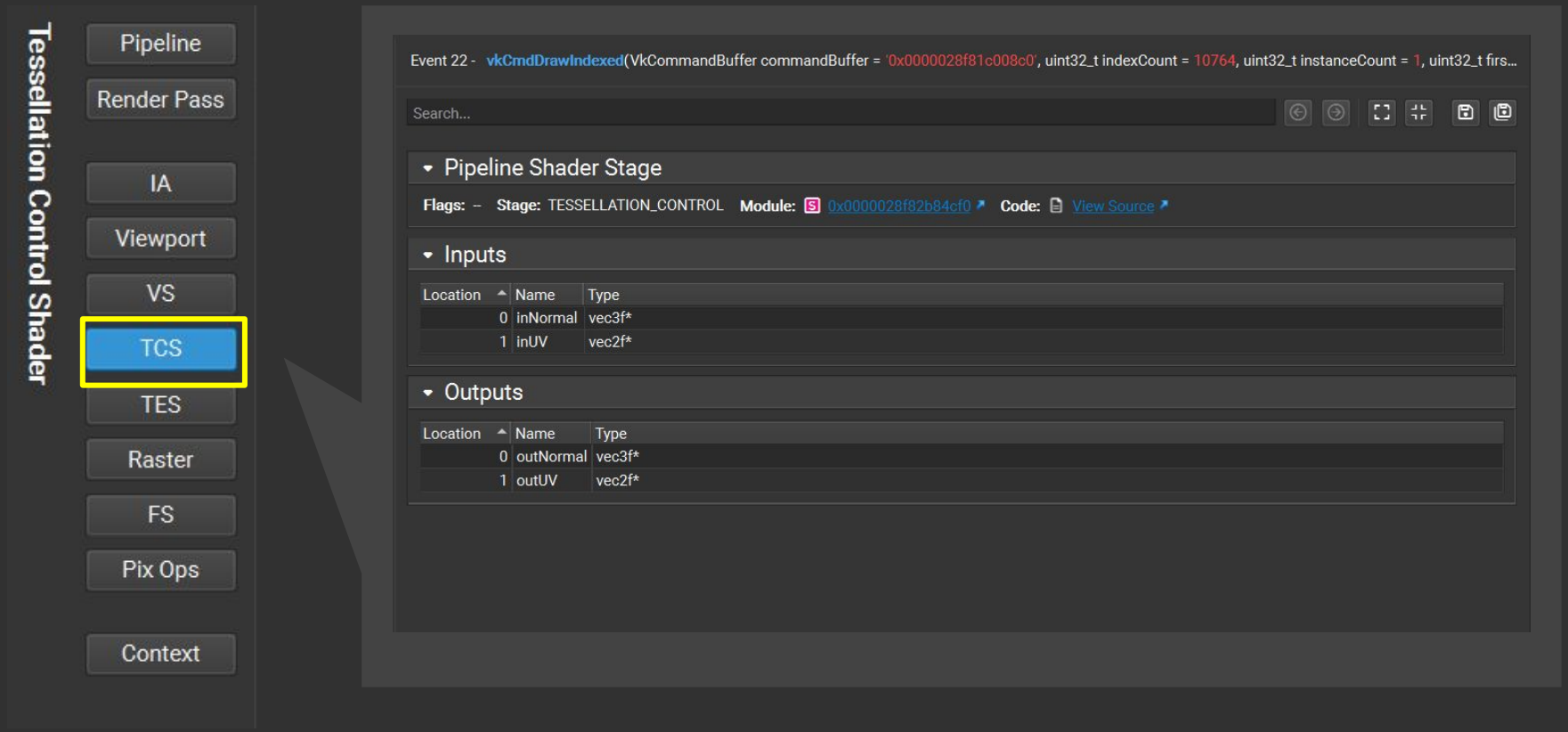
1. Frame Debugger - API Inspector



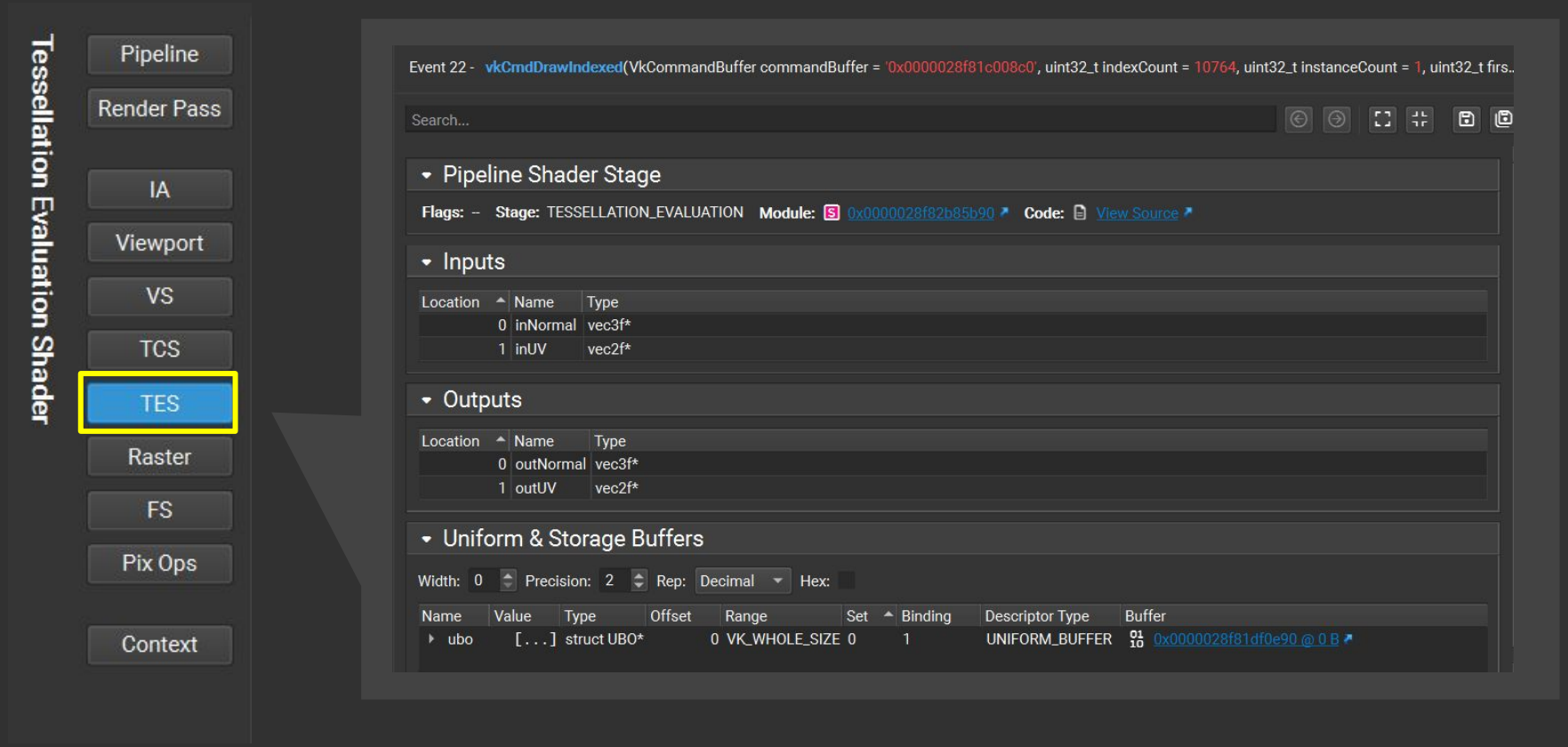
1. Frame Debugger - API Inspector



1. Frame Debugger - API Inspector

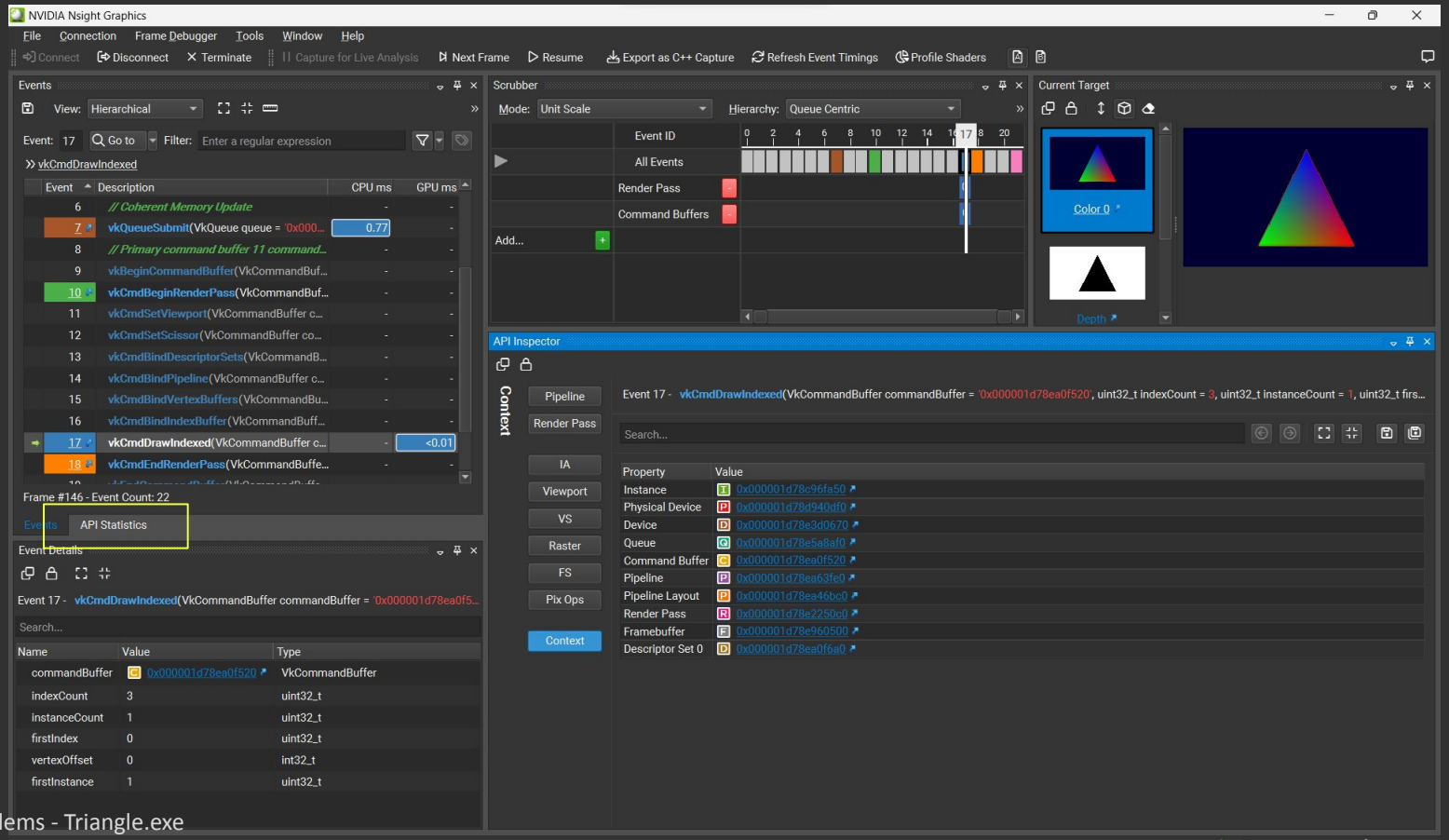


1. Frame Debugger - API Inspector



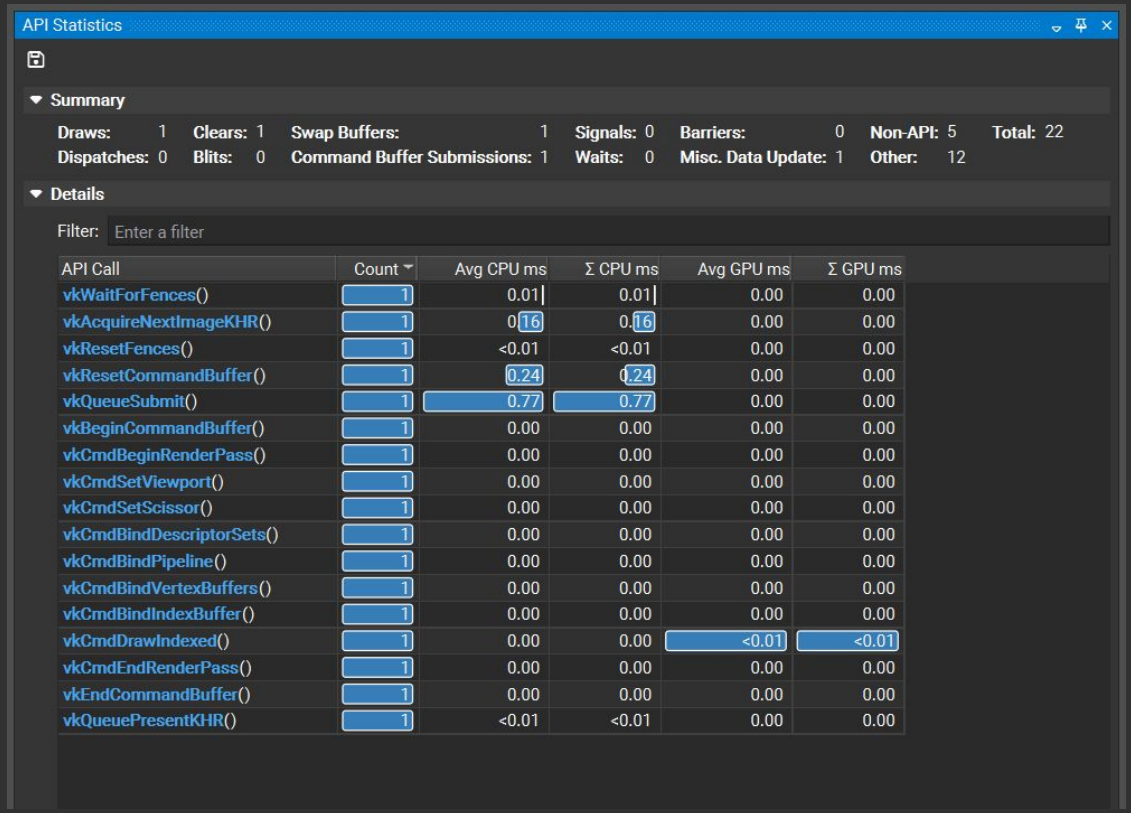
Sascha Willems - tessellation.exe

1. Frame Debugger - API Statistics



Sascha Willems - Triangle.exe

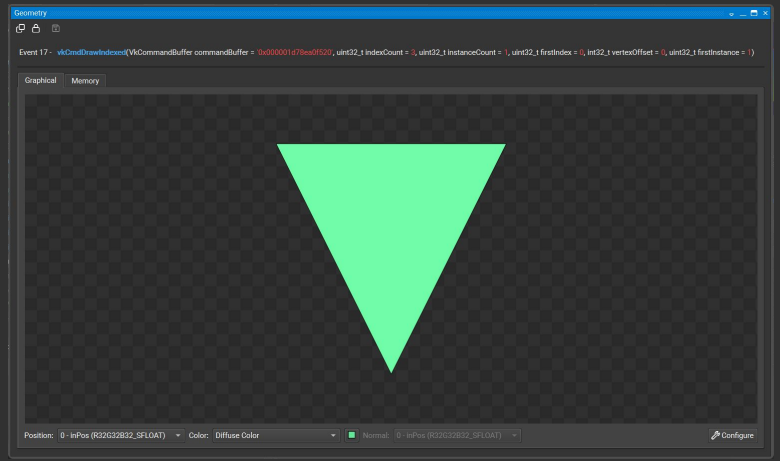
1. Frame Debugger - API Statistics

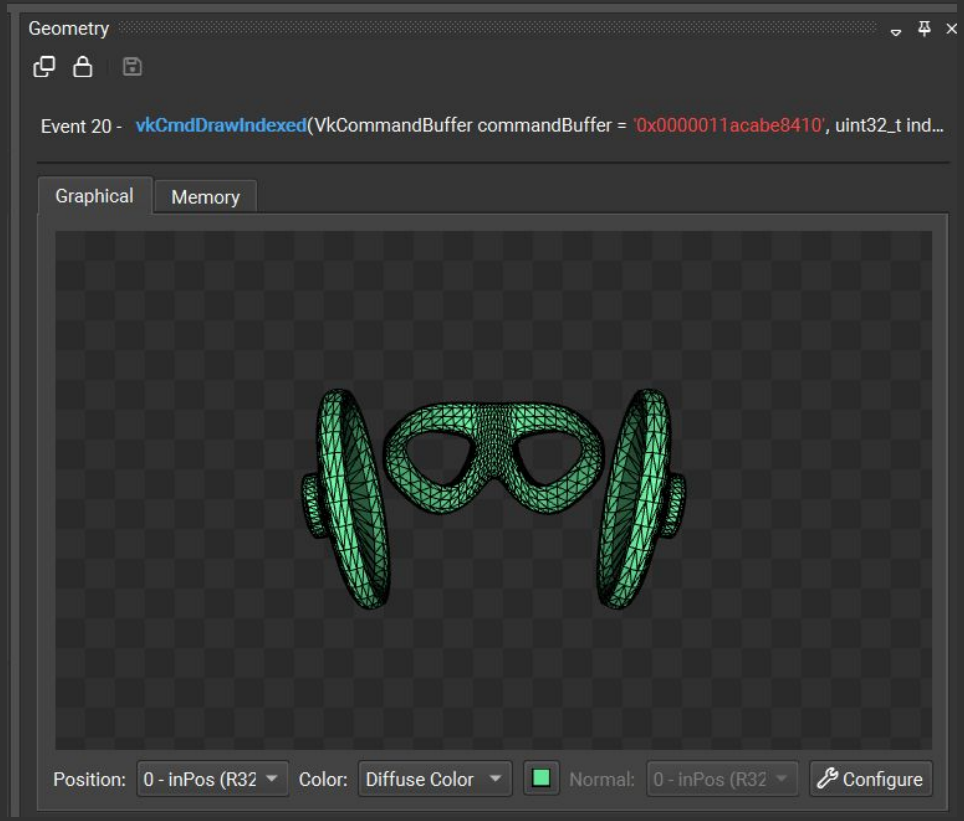


API Call	Count	Avg CPU ms	Σ CPU ms	Avg GPU ms	Σ GPU ms
vkWaitForFences()	1	0.01	0.01	0.00	0.00
vkAcquireNextImageKHR()	1	0.16	0.16	0.00	0.00
vkResetFences()	1	<0.01	<0.01	0.00	0.00
vkResetCommandBuffer()	1	0.24	0.24	0.00	0.00
vkQueueSubmit()	1	0.77	0.77	0.00	0.00
vkBeginCommandBuffer()	1	0.00	0.00	0.00	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

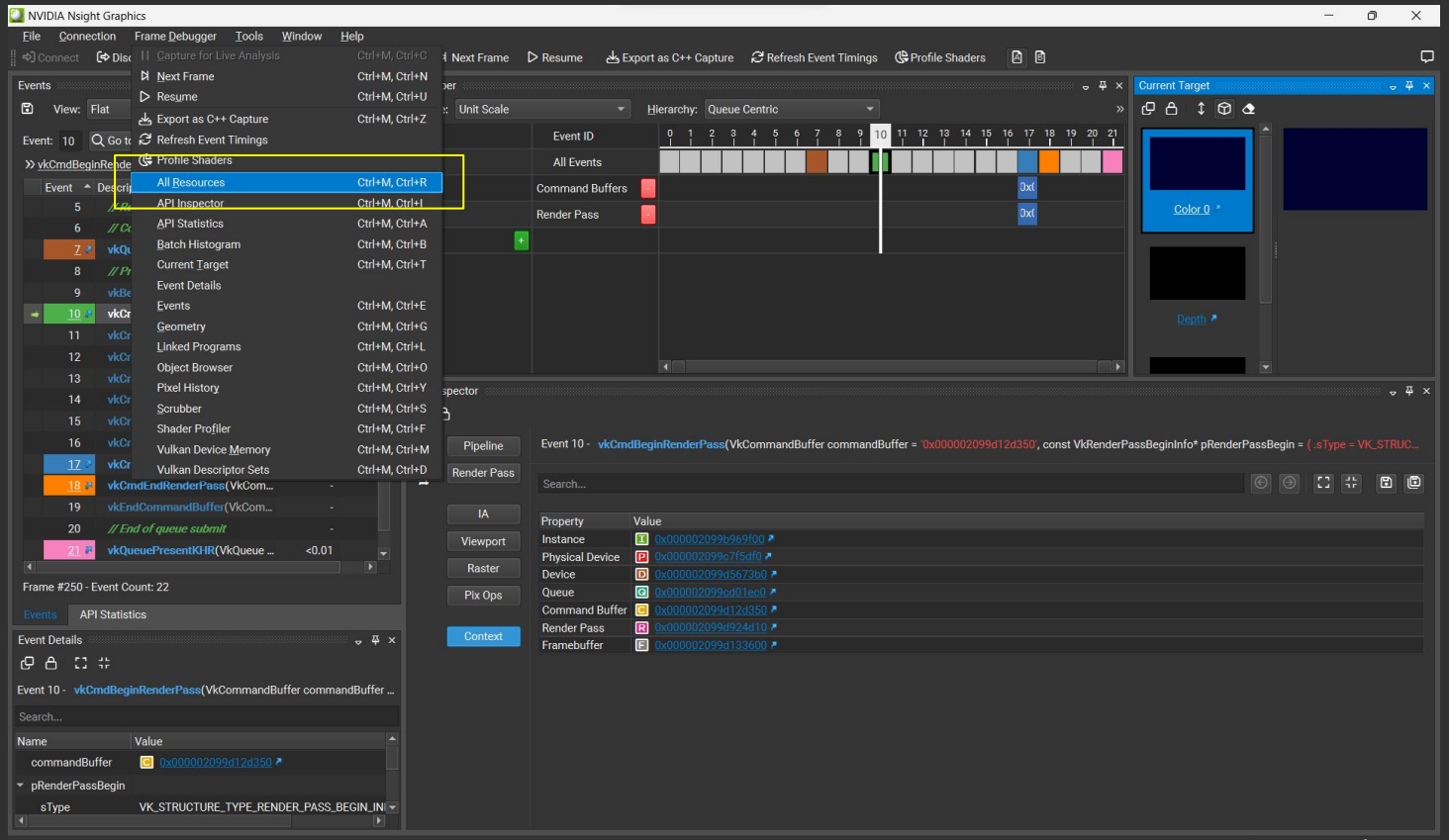
What the current draw call is trying to draw





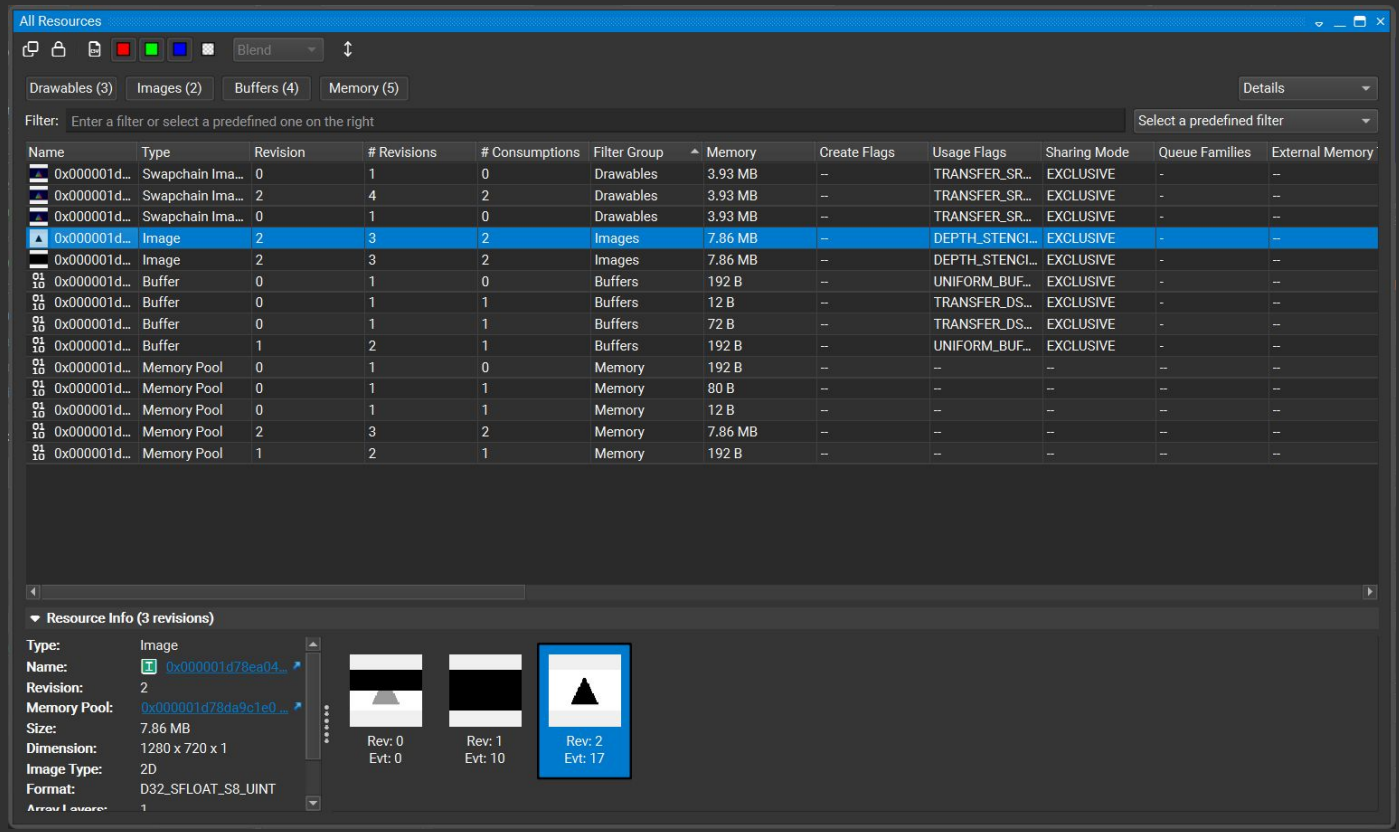
Sascha Willems - Triangle.exe, gltfloading.exe

1. Frame Debugger - All Resources

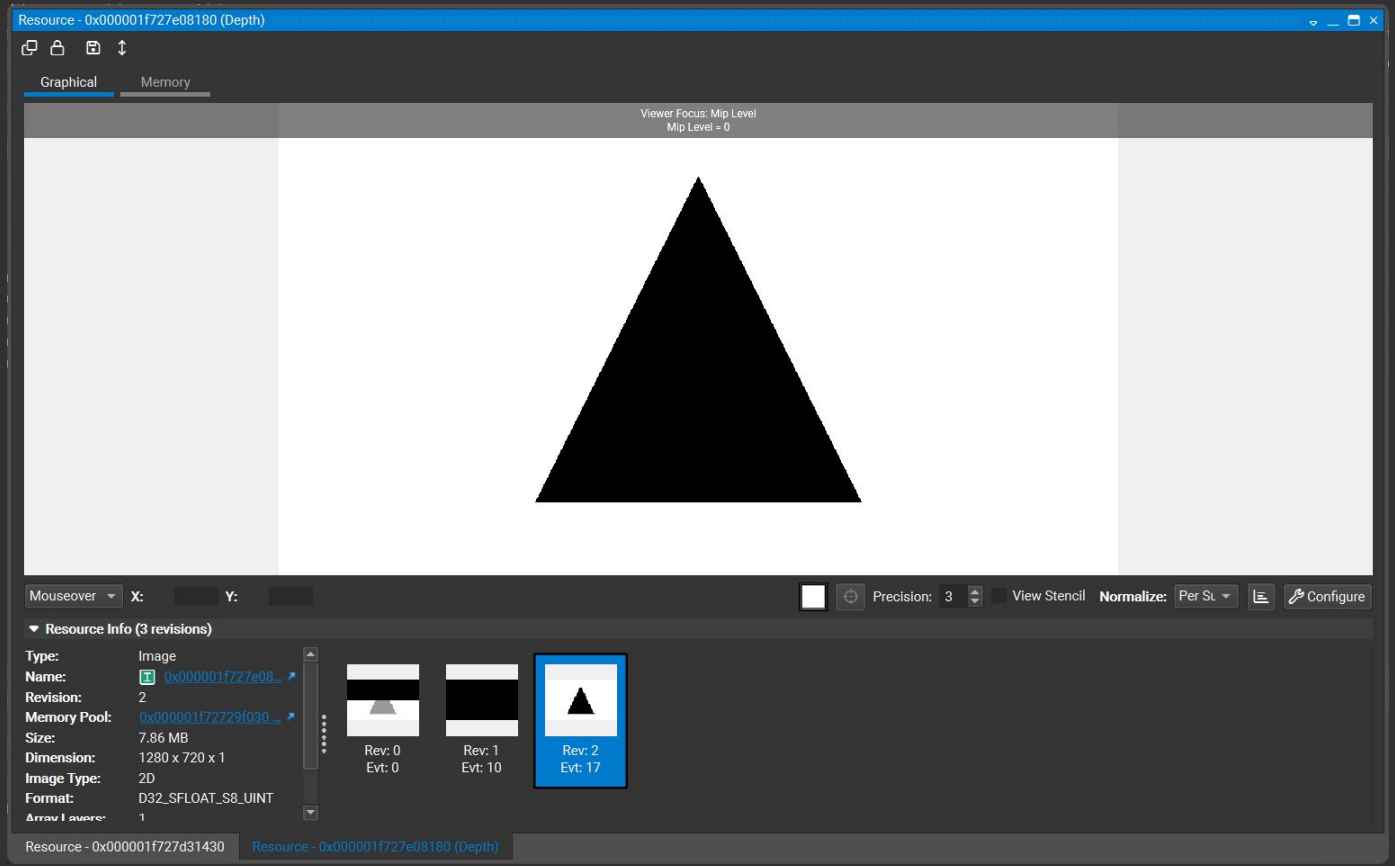


Sascha Willems - Triangle.exe

1. Frame Debugger - All Resources

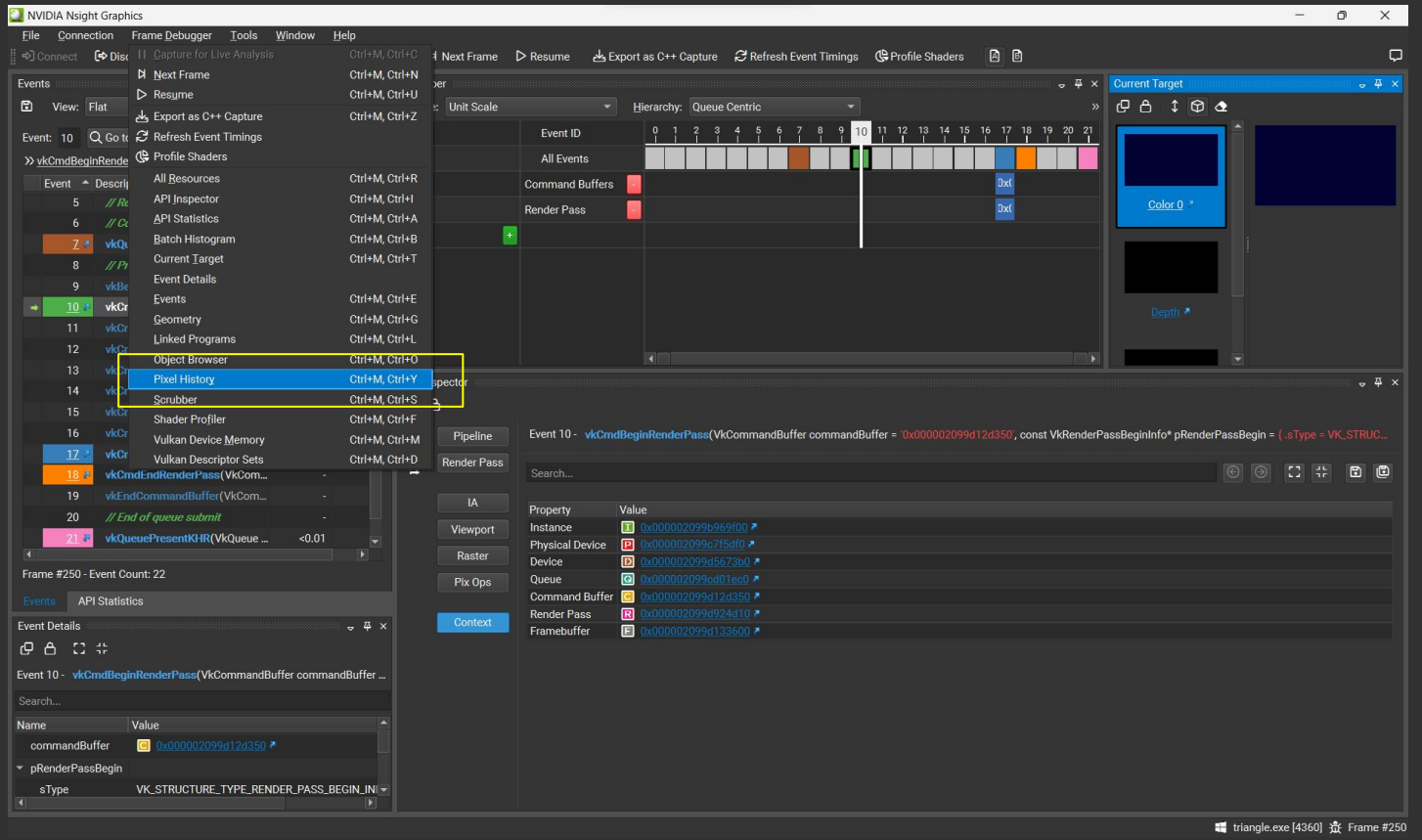


1. Frame Debugger - All Resources



Sascha Willems - Triangle.exe

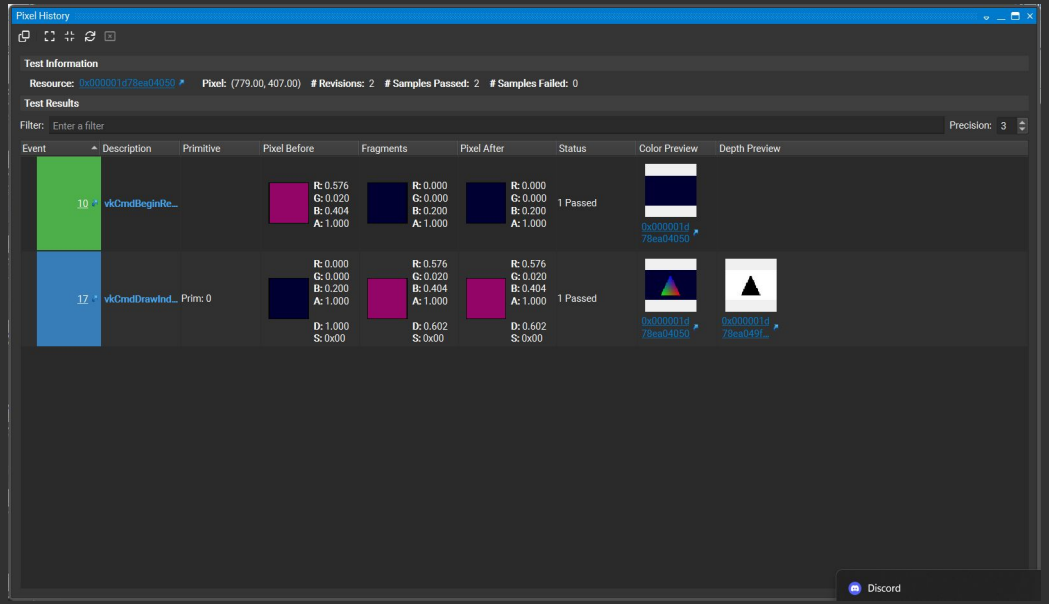
1. Frame Debugger - Pixel History



Sascha Willems - Triangle.exe

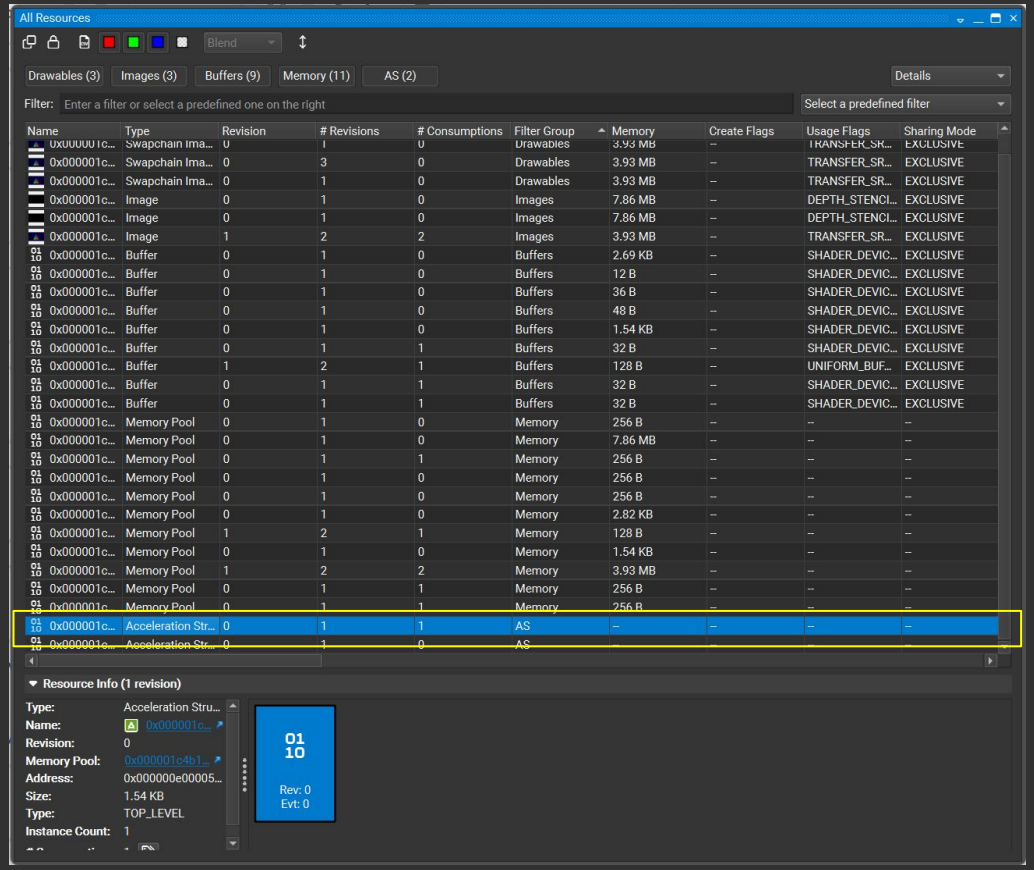
1. Frame Debugger - Pixel History

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

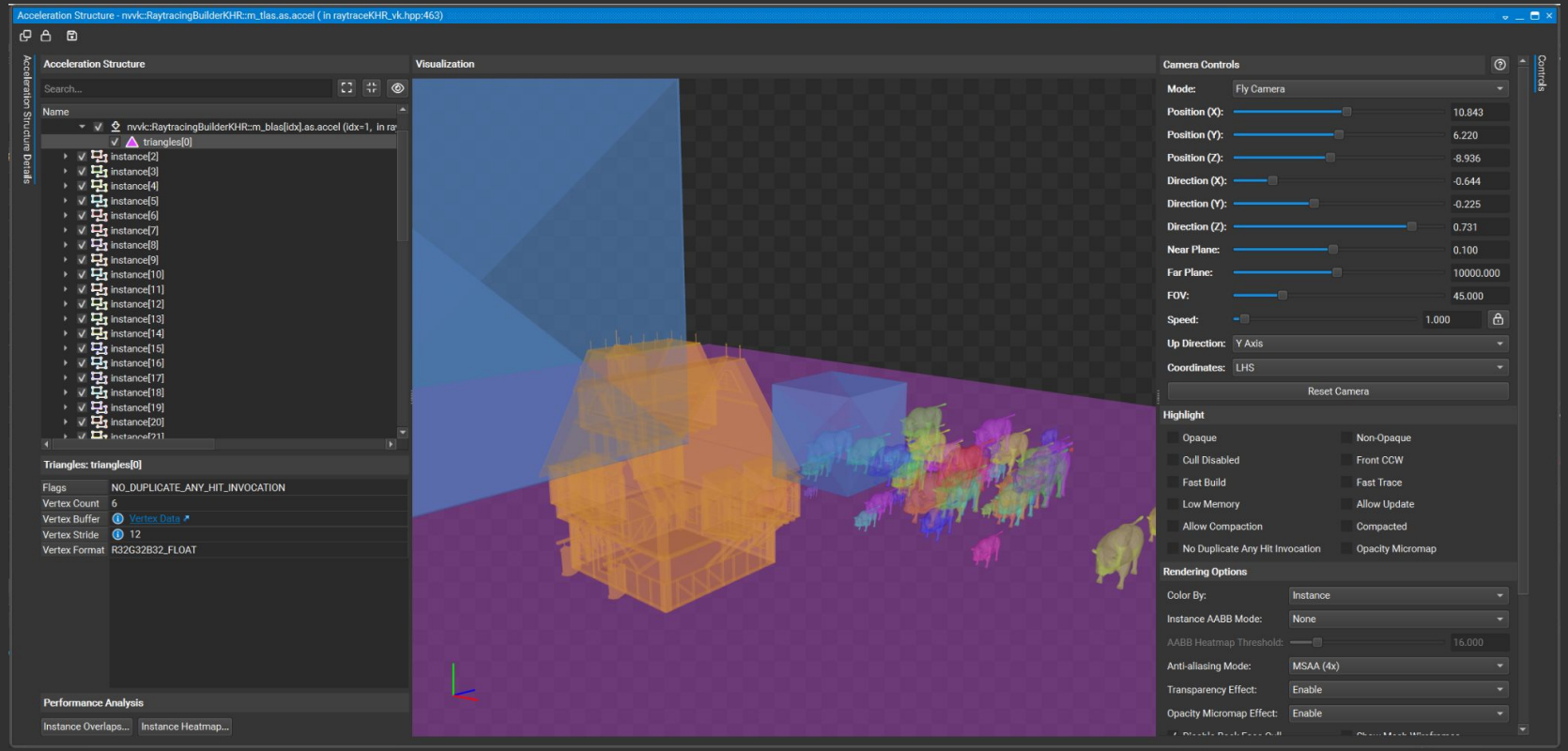


2. Frame Debugger - Acceleration Structure View

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

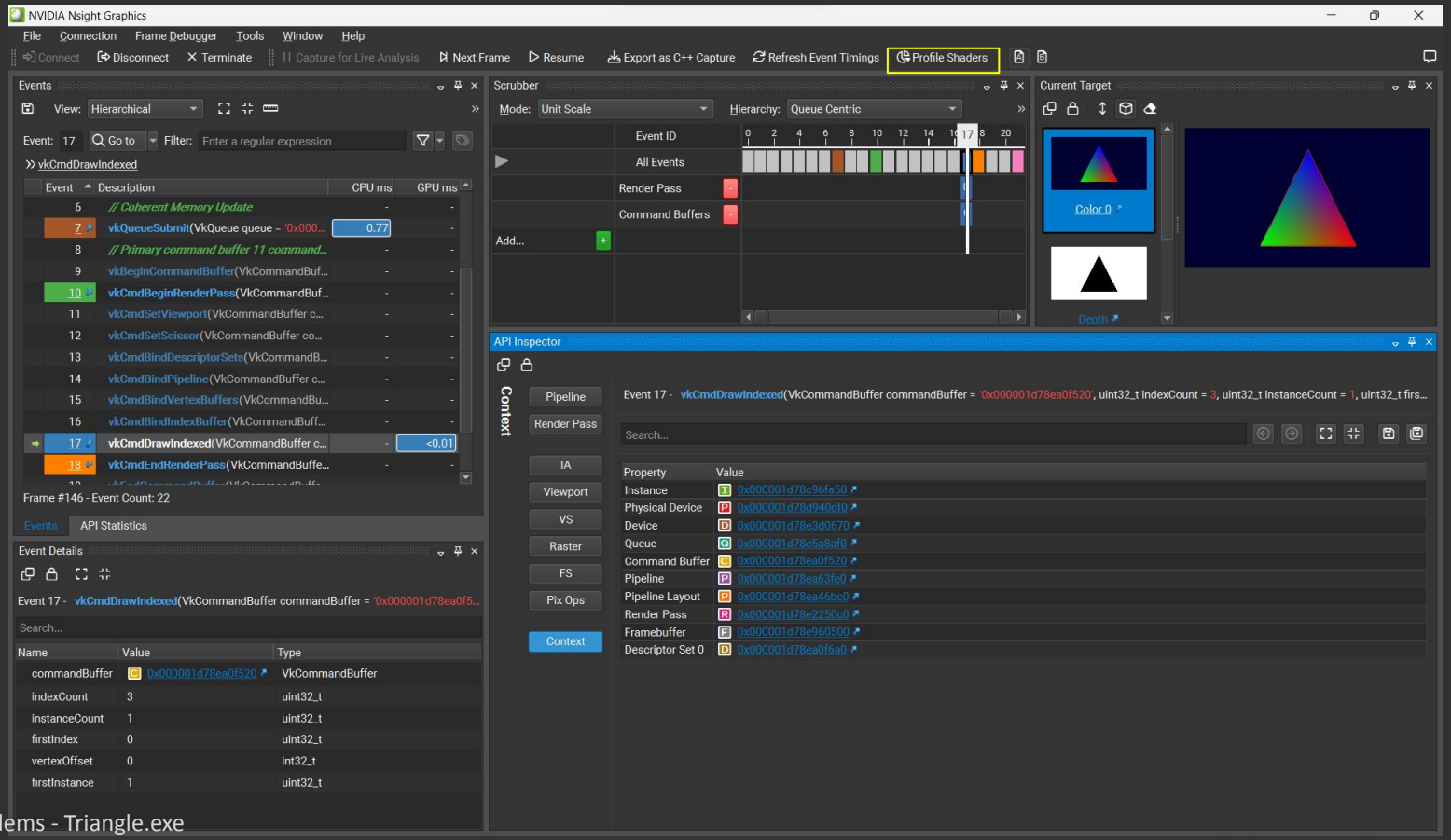


2. Frame Debugger - Acceleration Structure View



NSight Graphics > Help > Samples > vk_ray_tracing__advance_KHR

1. Frame Debugger - Shader Profiler

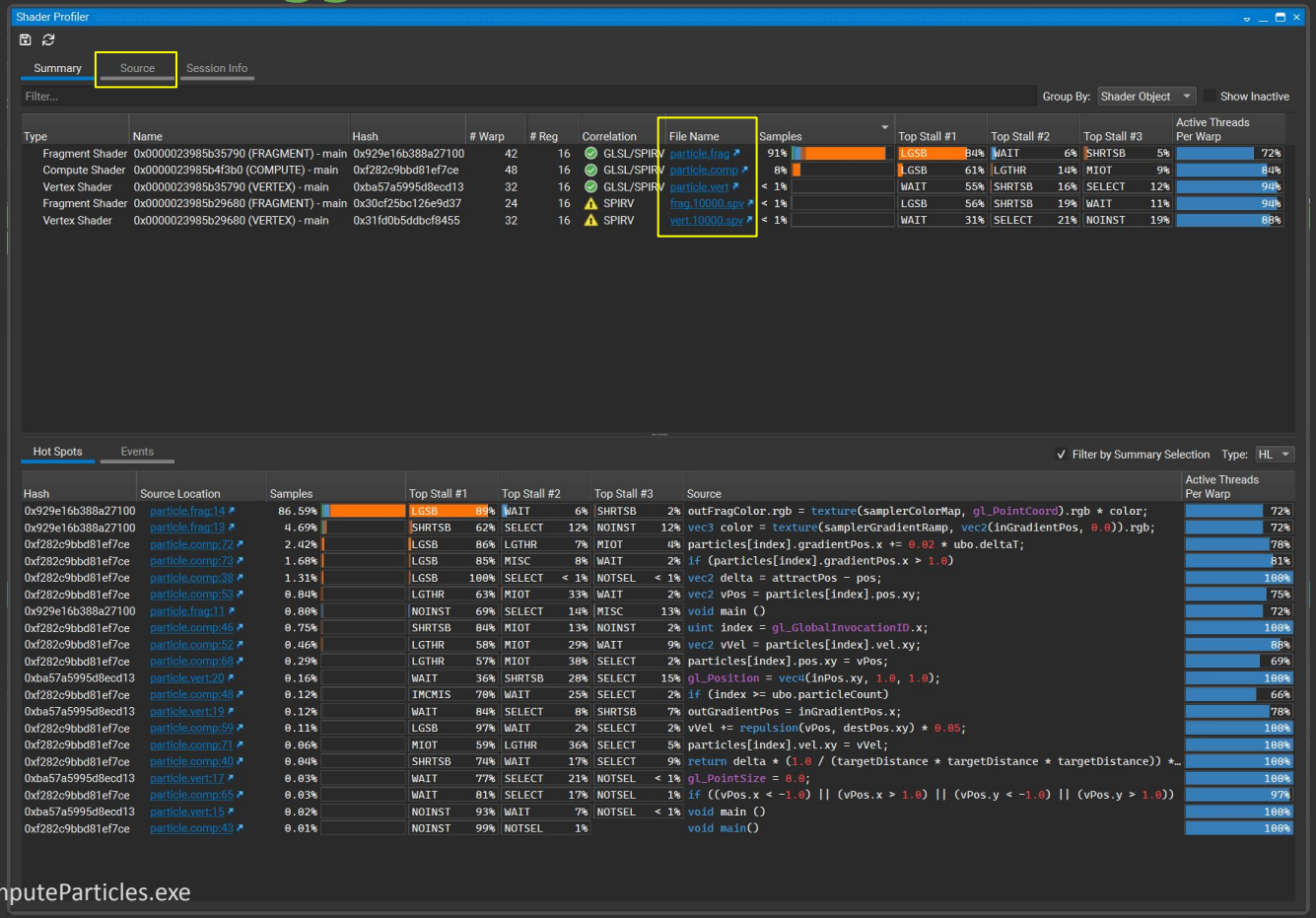


Sascha Willems - Triangle.exe

1. Frame Debugger - Shader Profiler

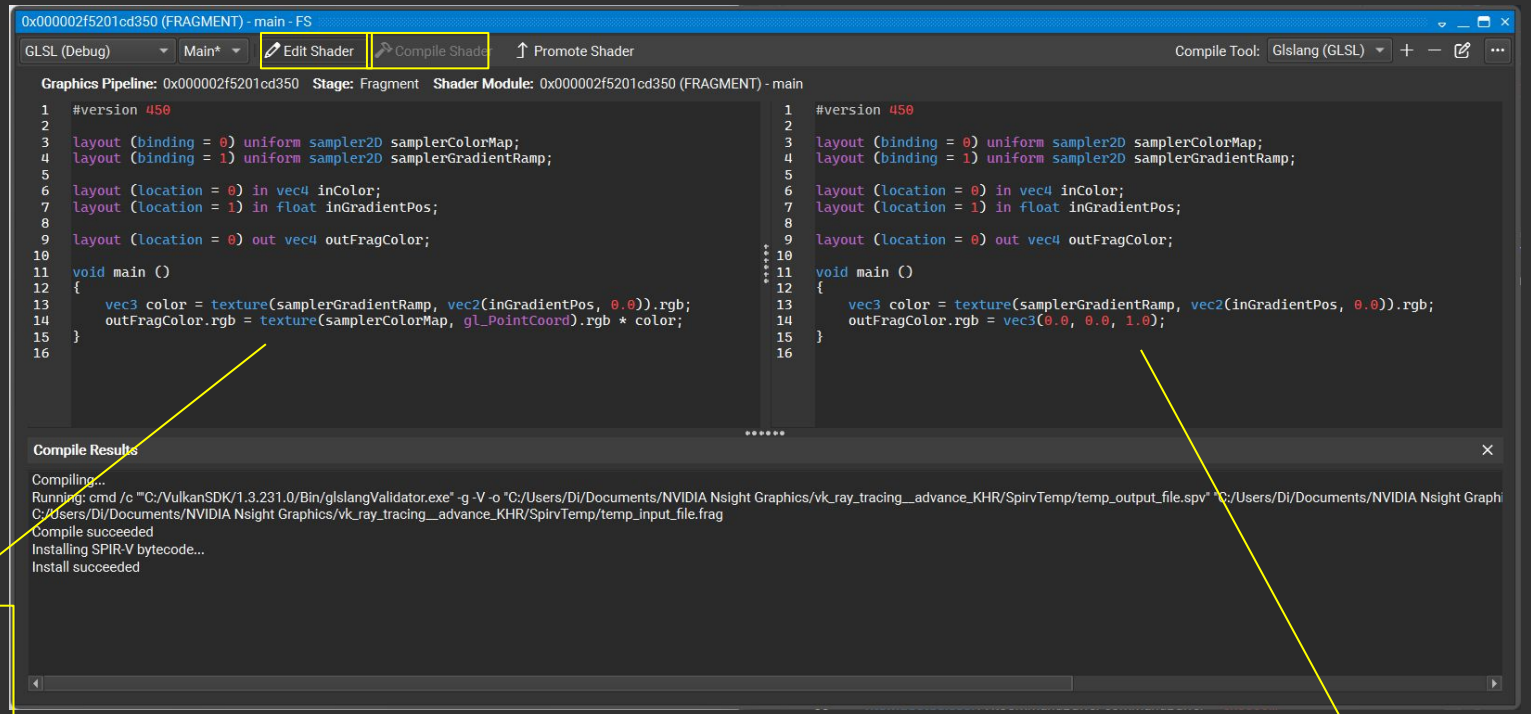
- Allows you to see how your shaders are performing
- Live compiling of shaders after editing!
- In order to see Shader correlation, you MUST compile your shaders with debug symbols: https://docs.nvidia.com/nsight-graphics/UserGuide/index.html#gcd_aftermath_source_shader_debug_info
- In Vulkan, your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

1. Frame Debugger - Shader Profiler

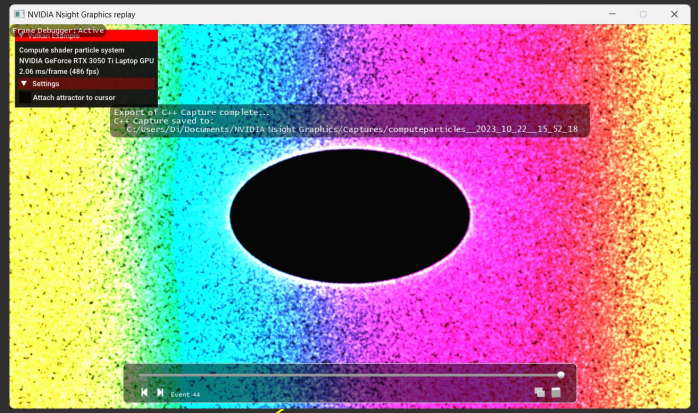


Sascha Willems - computeParticles.exe

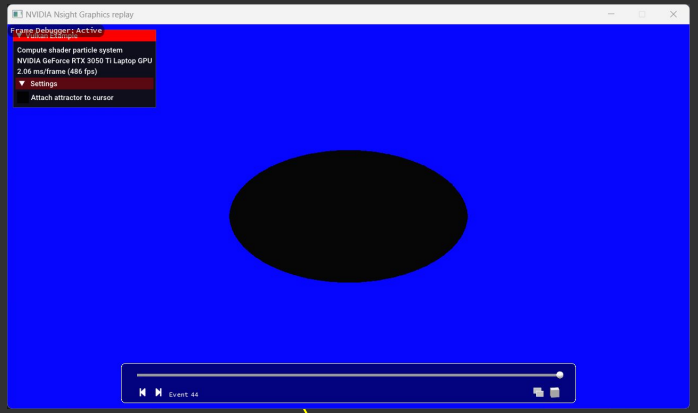
1. Frame Debugger - Shader Profiler



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.

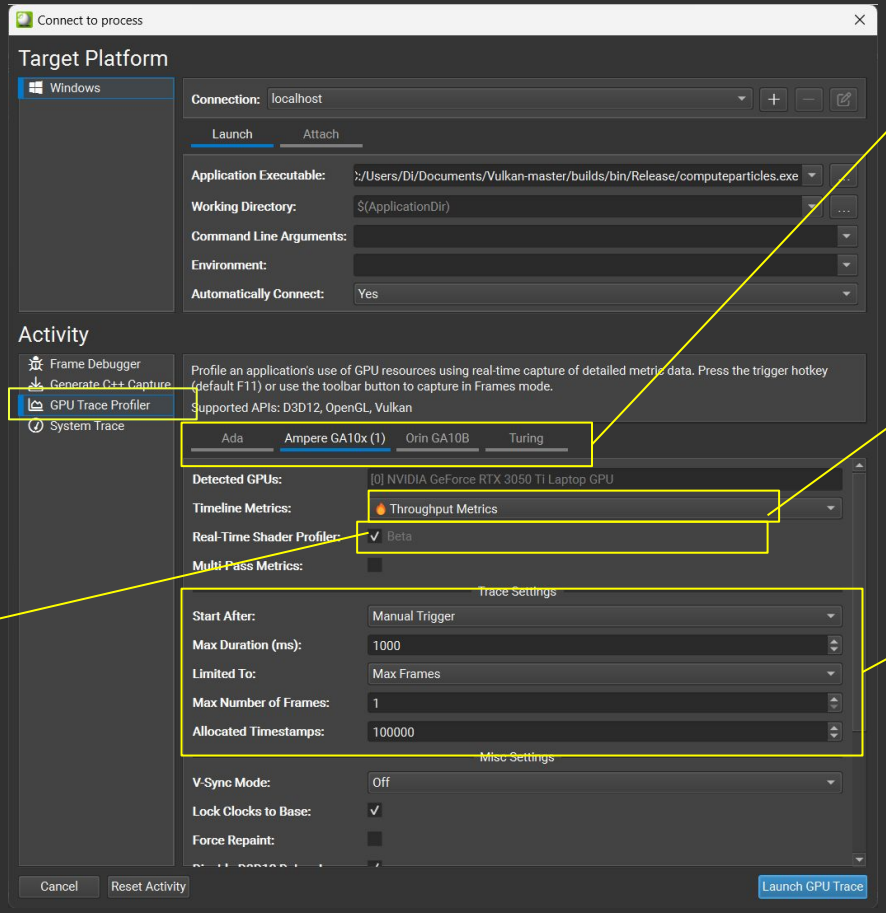
2. GPU Trace - Motivations

- 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.

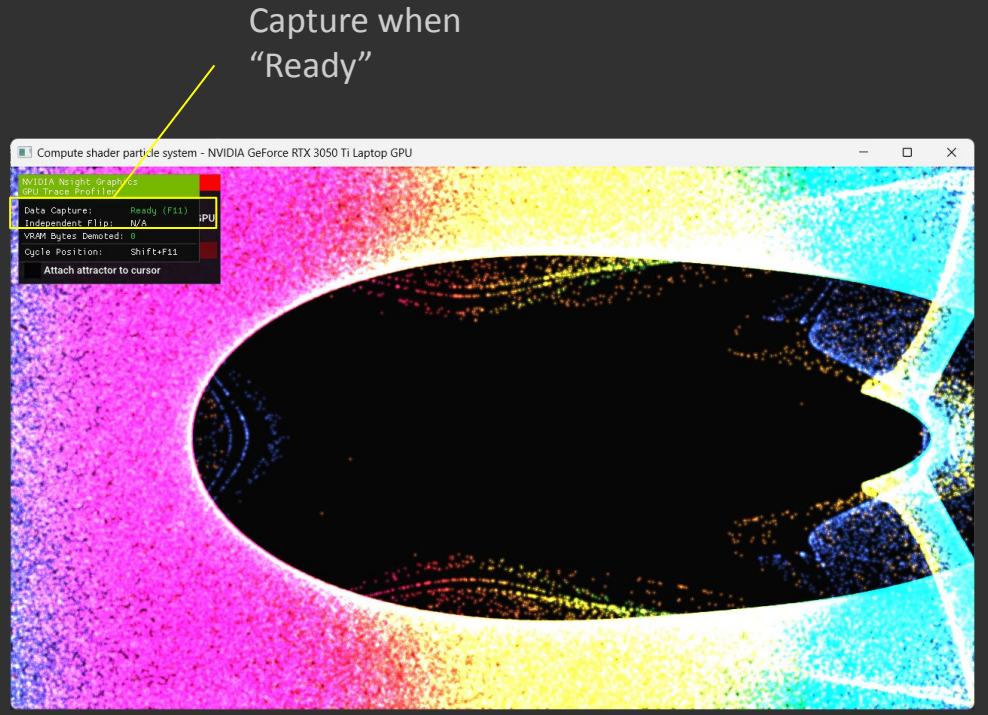
The type of metrics you want to measure (There are 2 options)

How do I want to trigger a capture?

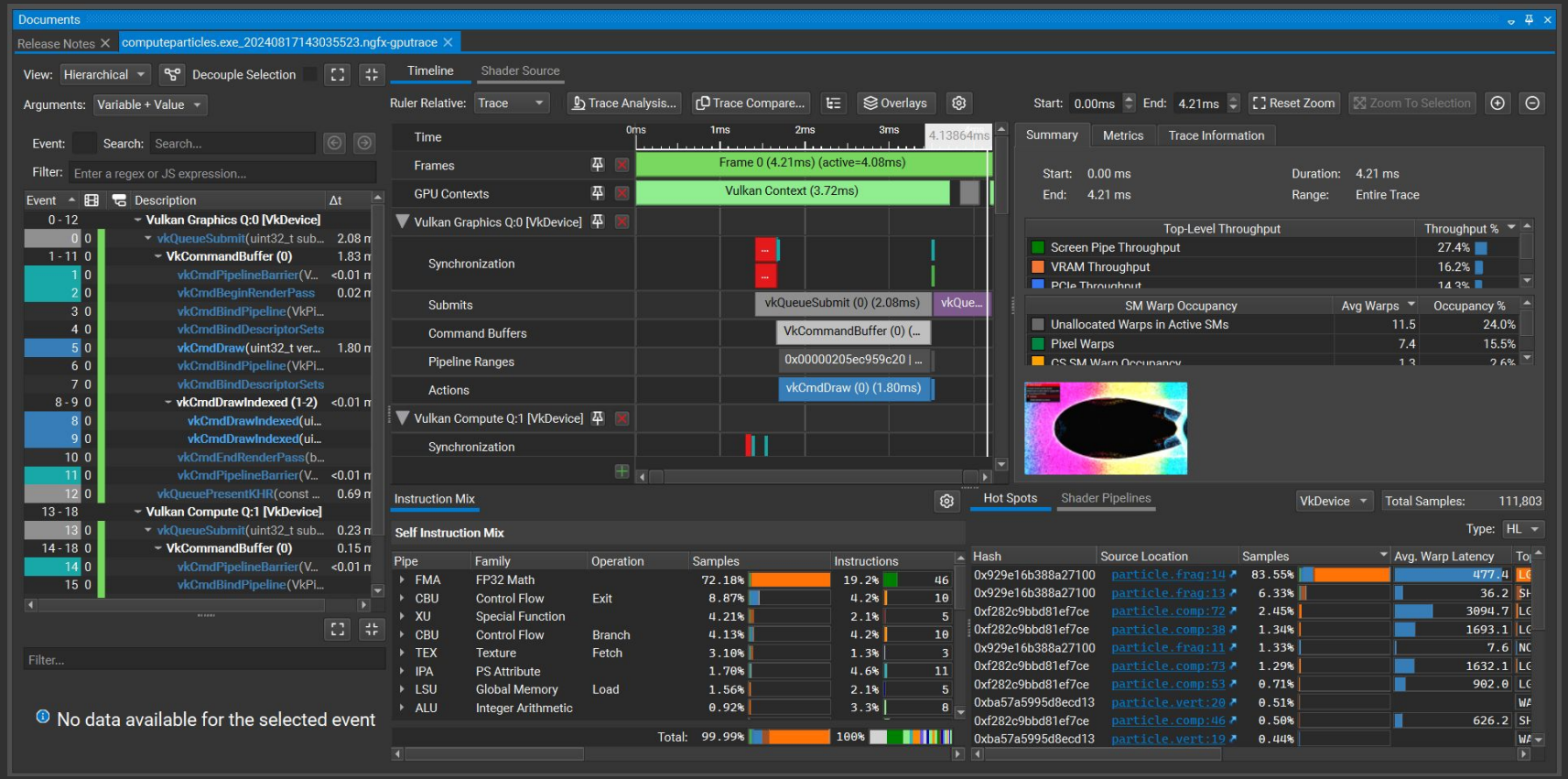
Turn off/on shader profiling information

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



Sascha Willems - computeParticles.exe

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

Useful in detecting Starvation.

GR = Graphics

Unit Throughput

Identify which HW Units are the performance limiters

SM Instruction Throughput

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

Goal: Get throughput to 80%

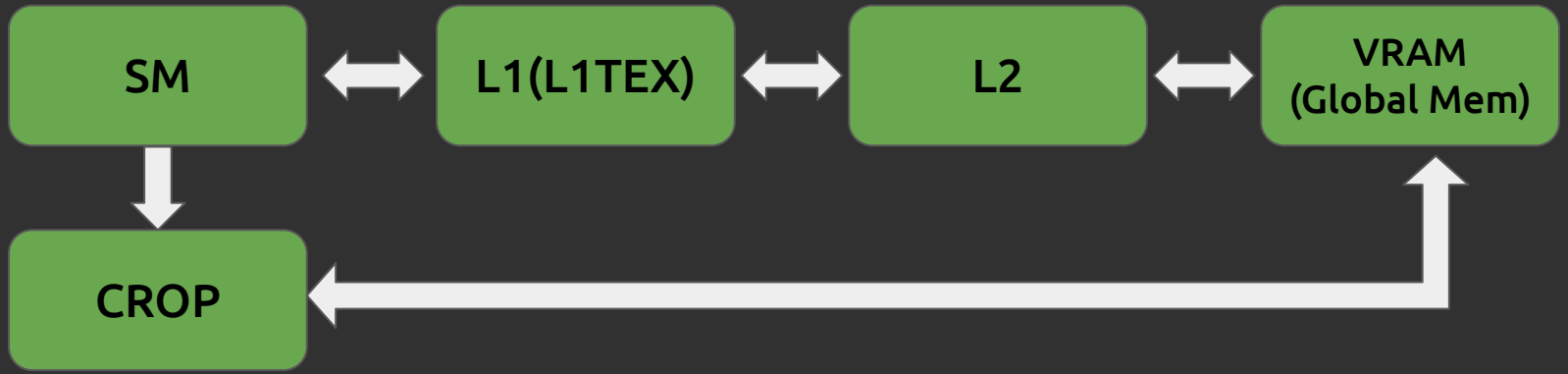
SM Warp Occupancy

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

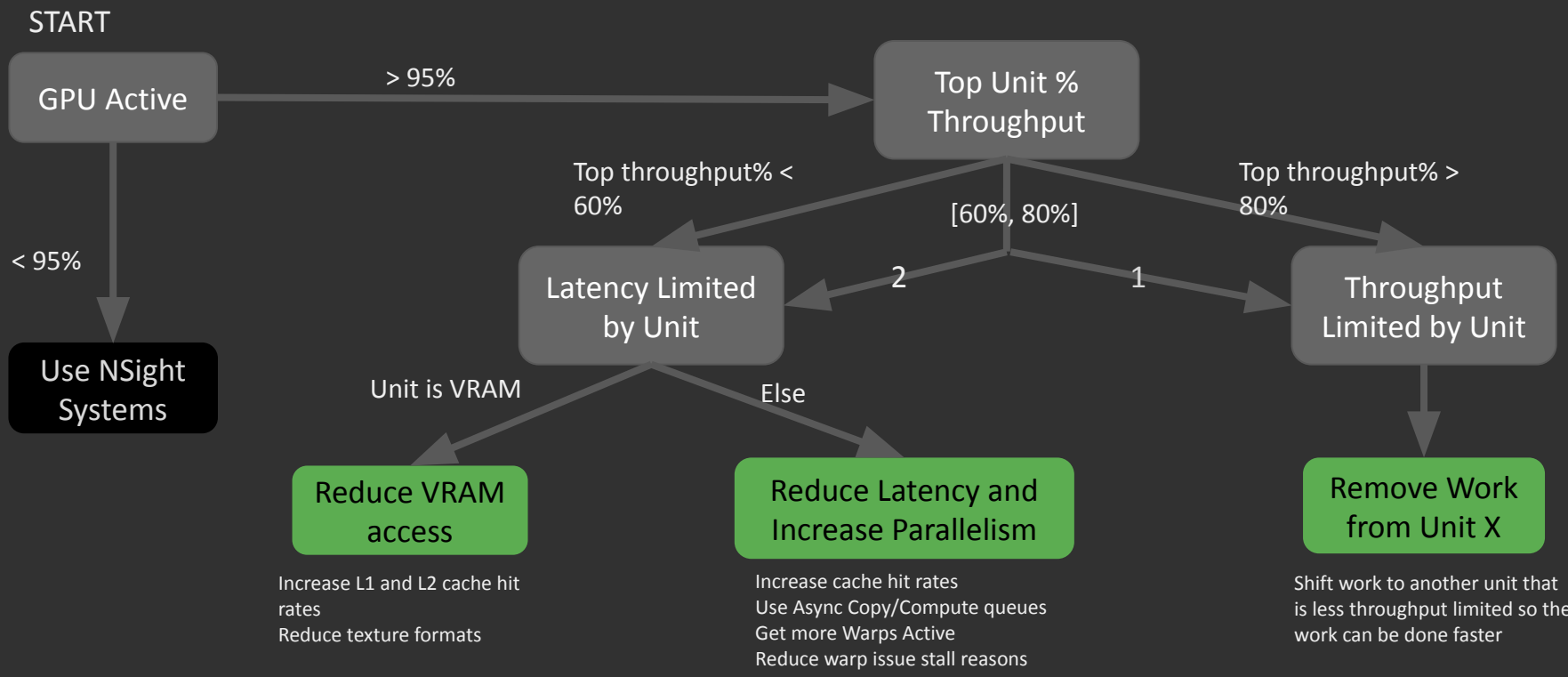
Unit Throughput % Metrics



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

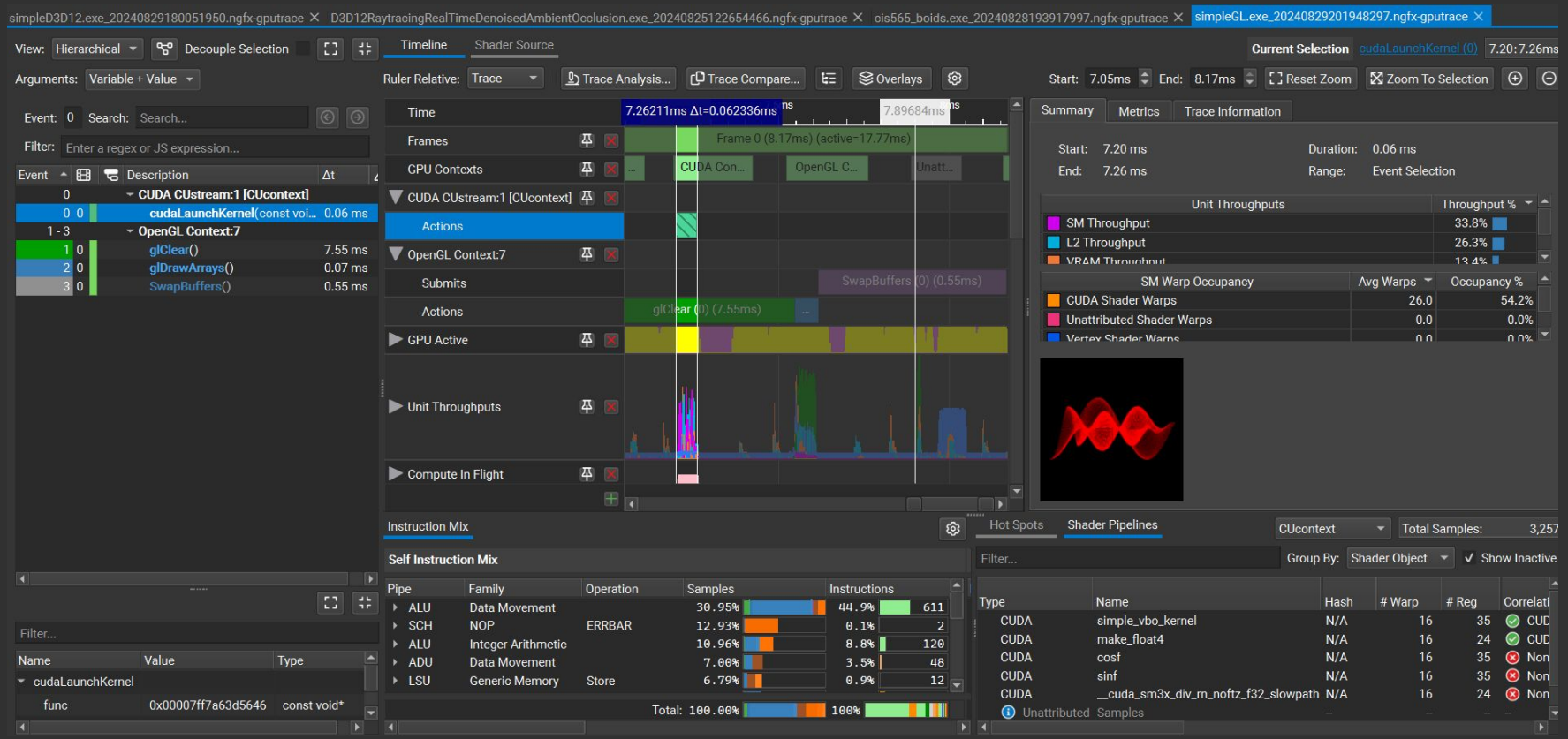
Credit: Louis Bavoil, GDC 2019

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



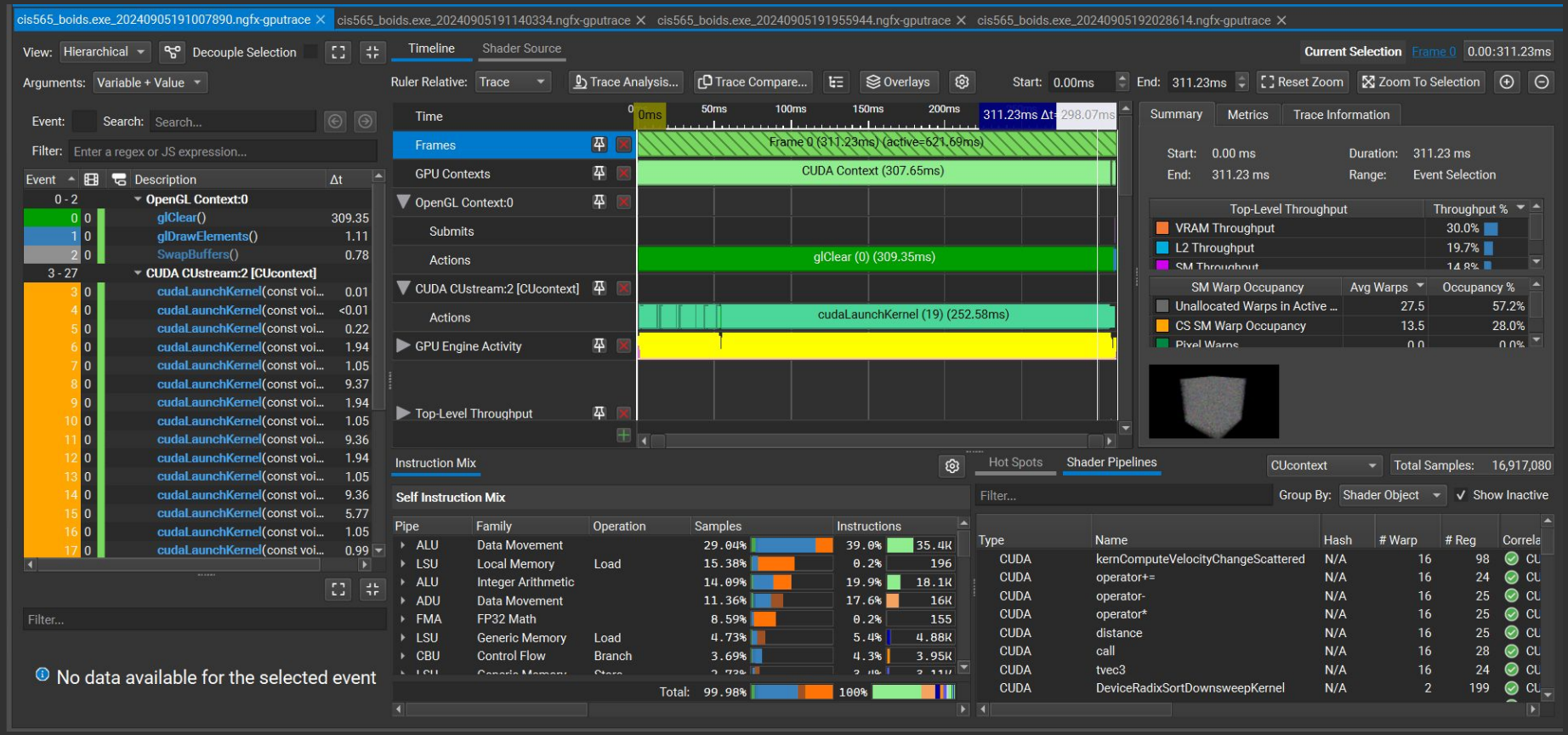
Credit: Louis Bavoil, GDC 2019

2. GPU Trace - CUDA Support



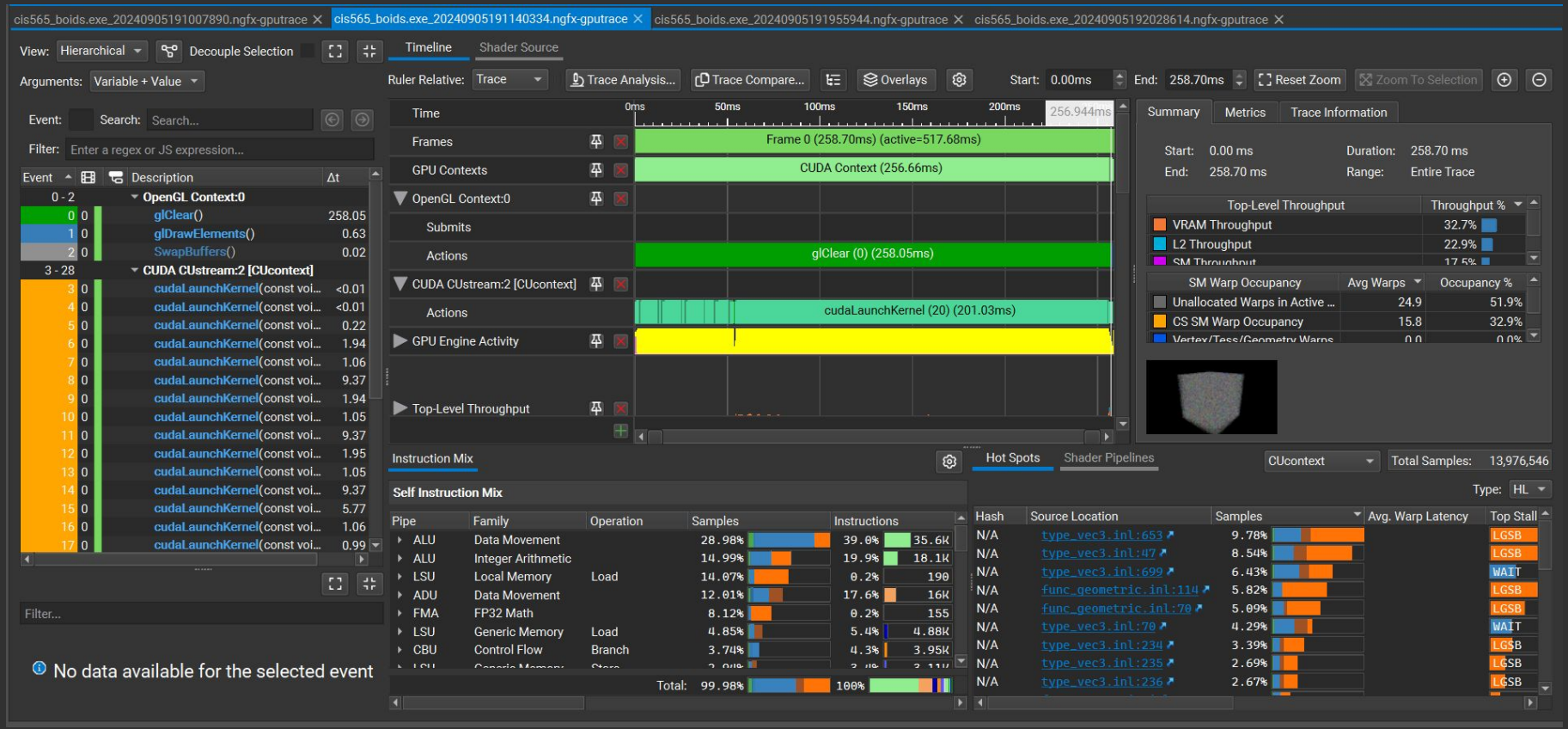
[Cuda-samples simpleGL.exe]

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



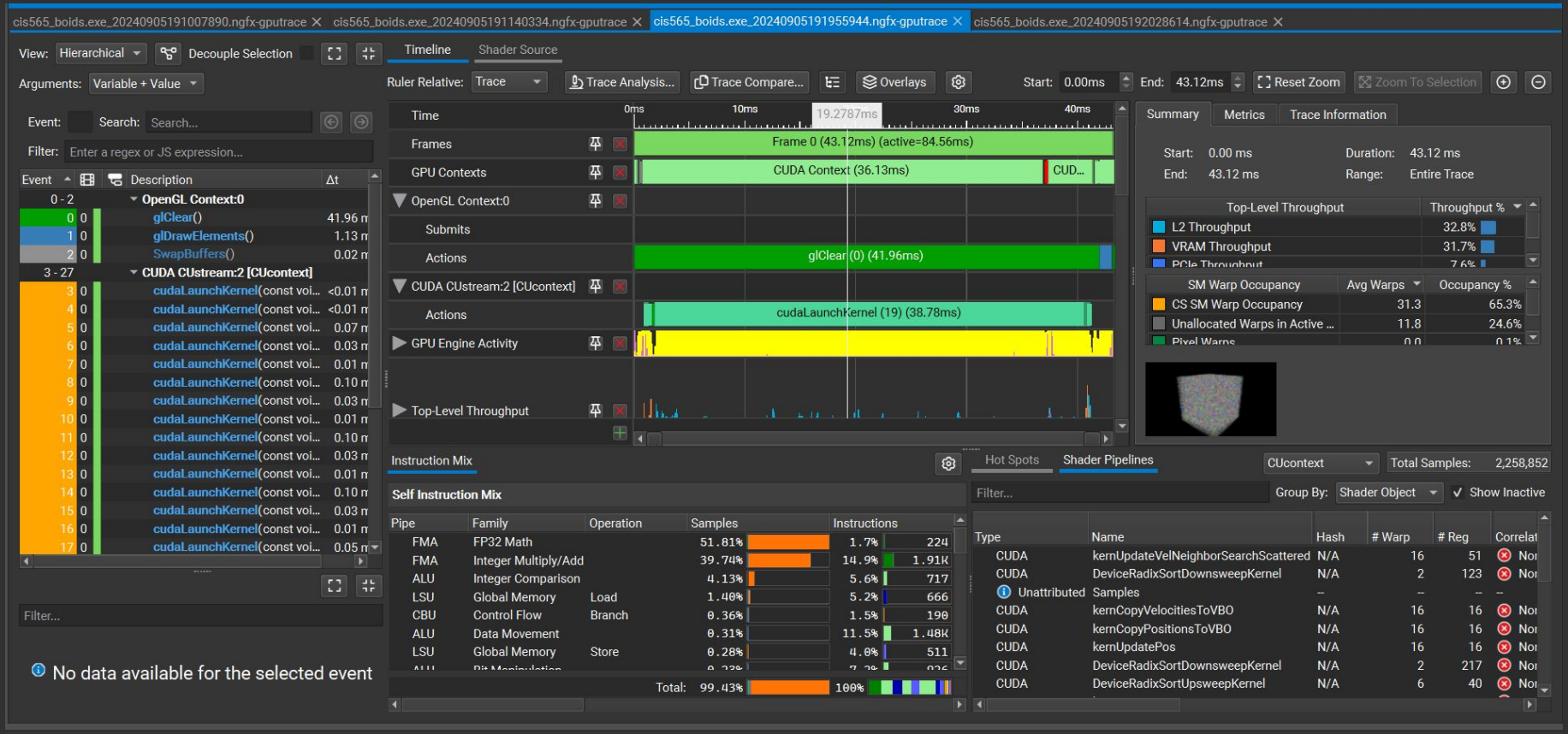
My boids project - Uniform Grid (500,000) - Debug Mode

2. GPU Trace - Flocking



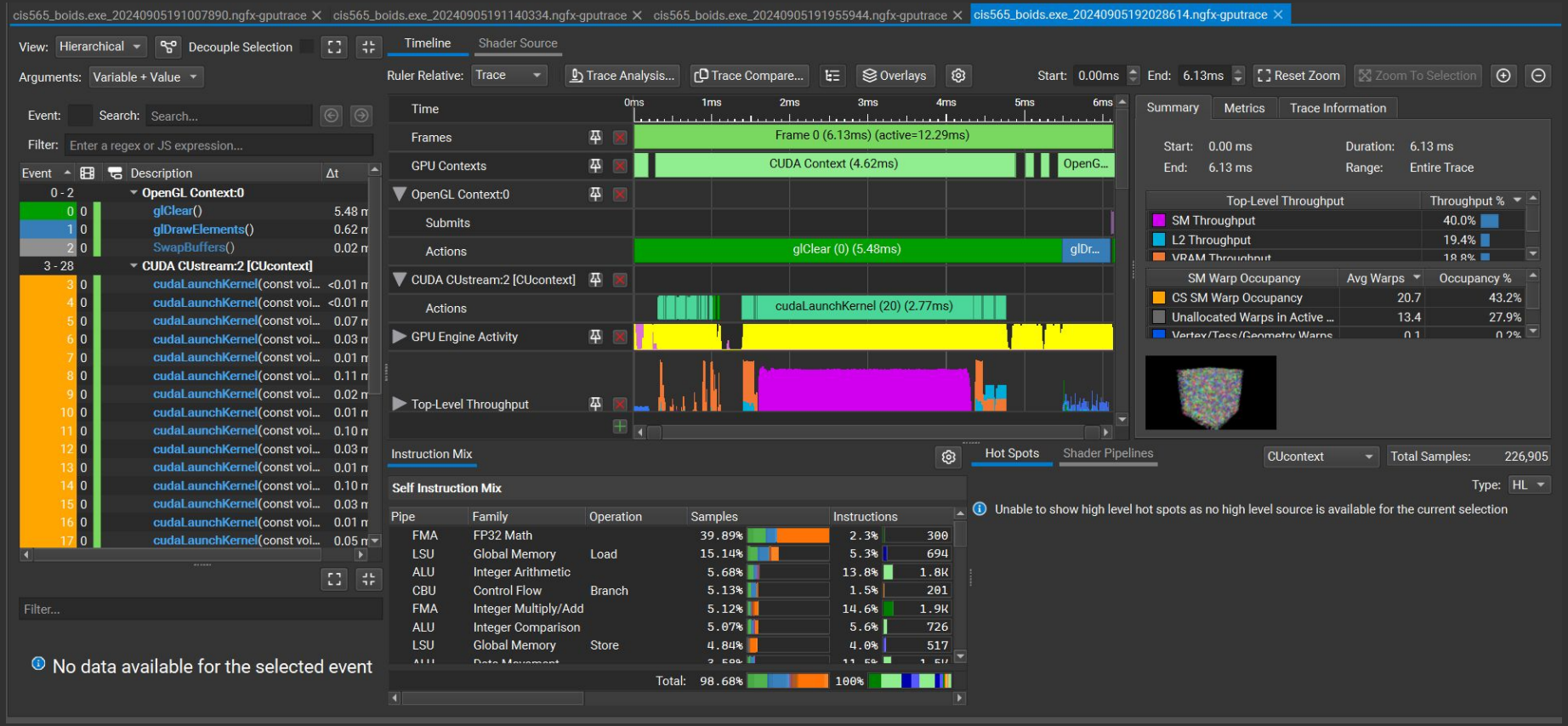
My boids project - Coherent Grid (500,000) - Debug Mode

2. GPU Trace - Flocking



My boids project - Uniform Grid (500,000) - Release Mode

2. GPU Trace - Flocking



My boids project - Coherent Grid (500,000) - Release Mode

2. GPU Trace - Flocking

Live Demo

My boids project - Coherent Grid (500,000) - Release Mode

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

Warps 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?

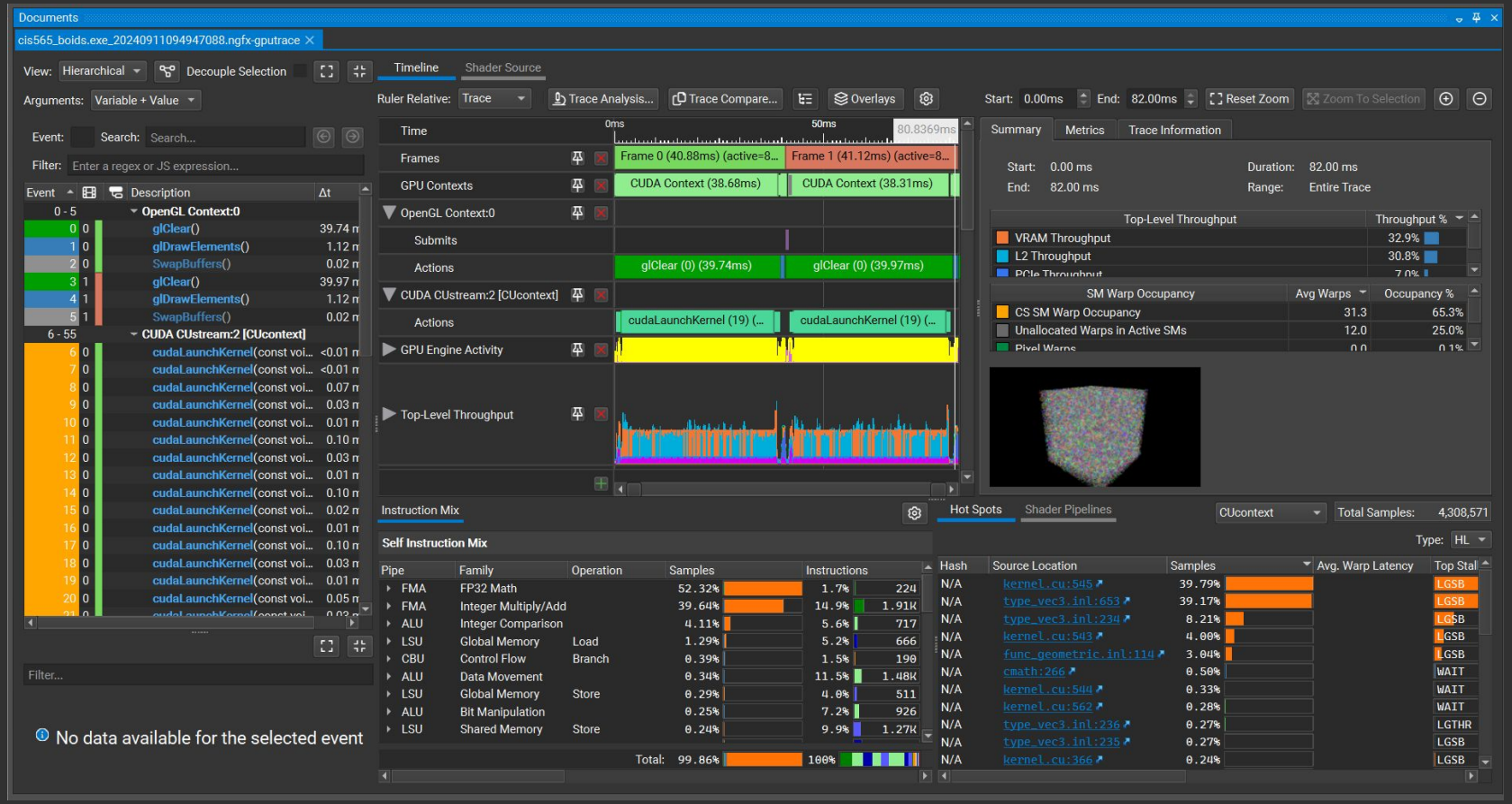
Cache Thrashing

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.

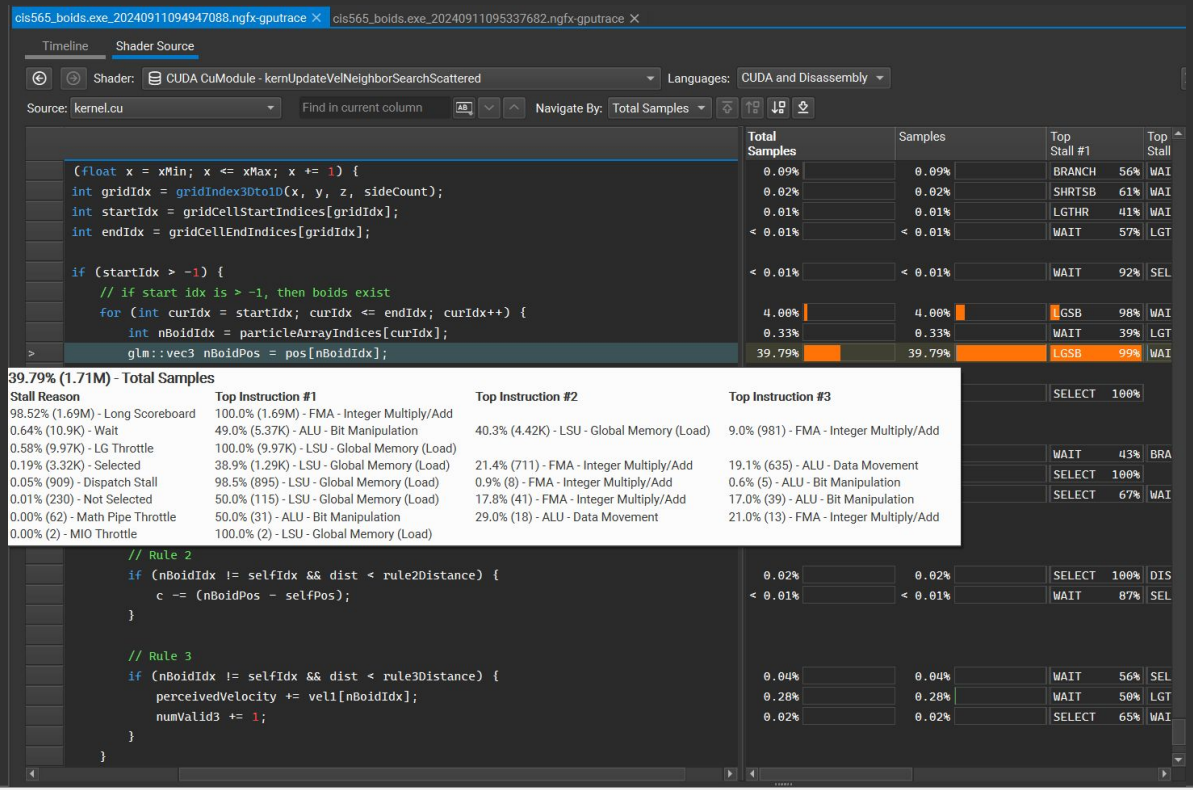
Memory Latency

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 implemented by using hw counters, stream buffers, prefetch tables, etc.)

2. GPU Trace - Flocking; Per-Line Analysis

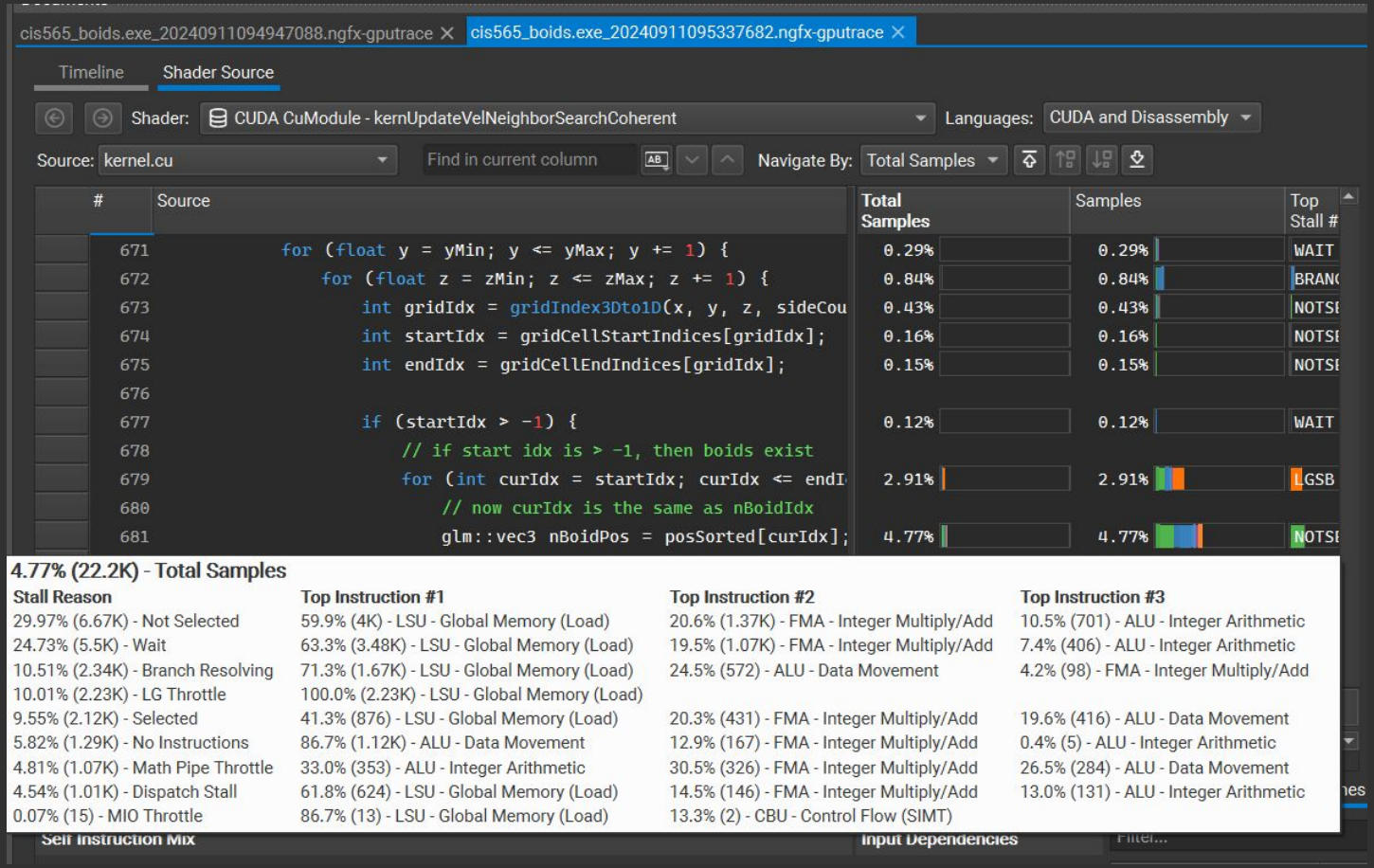


2. GPU Trace - Flocking; Per-Line Analysis

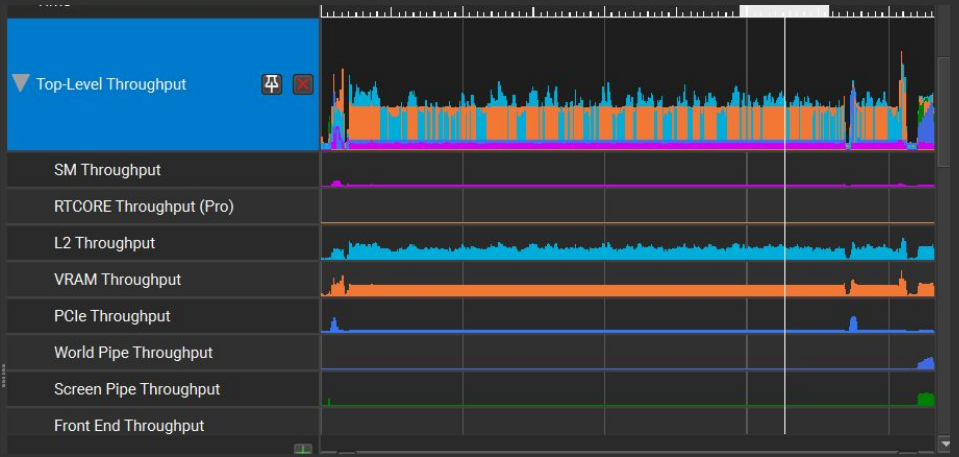


Rearranged POS and VEL, such that information from boids closer together will be physically to each other

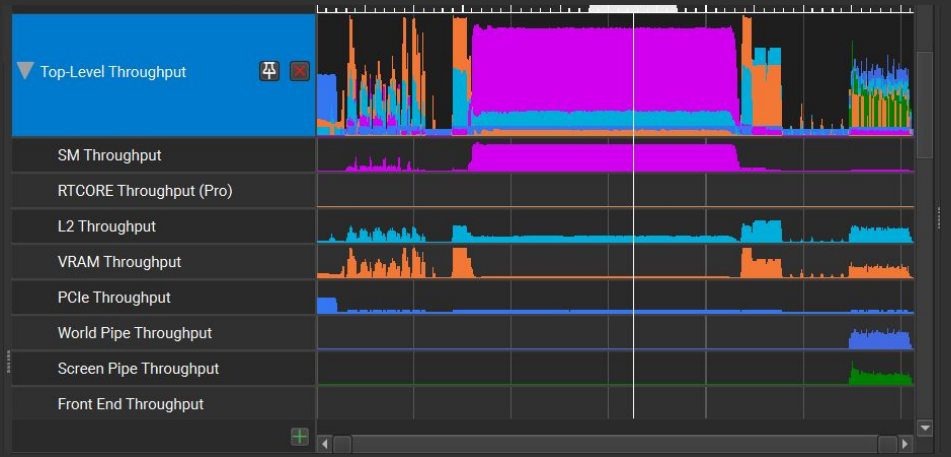
2. GPU Trace - Flocking; Per-Line Analysis



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
Warps Stalled - Long Scoreboard (warps)	33 (68.7%)	9.2 (19.2%)

2. GPU Trace - Case Study 1

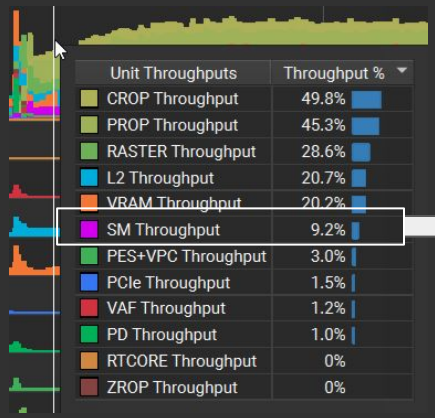
Example: SM Sub-Throughput Metrics on D3D12RealtimeRaytraceDenoisedAmbientOcclusion >> Denoise pass

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%



Unit Throughputs	Throughput %
CROP Throughput	49.8%
PROP Throughput	45.3%
RASTER Throughput	28.6%
L2 Throughput	20.7%
VRAM Throughput	20.2%
SM Throughput	9.2%
PES+VPC Throughput	3.0%
PCIe Throughput	1.5%
VAF Throughput	1.2%
PD Throughput	1.0%
RTCORE Throughput	0%
ZROP Throughput	0%

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%

Credit: Louis Bavoil, GDC 2019

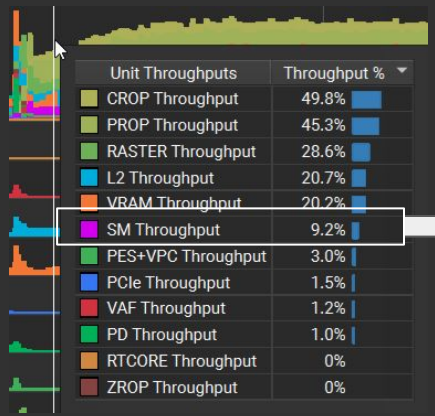
2. GPU Trace - Case Study 1

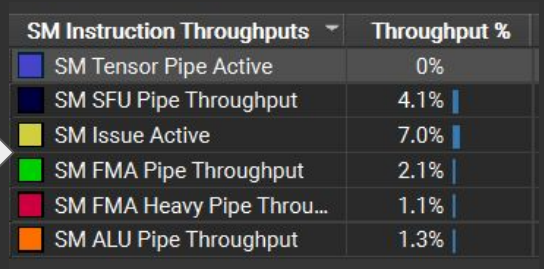
Example: SM Sub-Throughput Metrics on *D3D12RealtimeRaytraceDenoisedAmbientOcclusion* >> Denoise pass

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





Credit: Louis Bavoil, GDC 2019

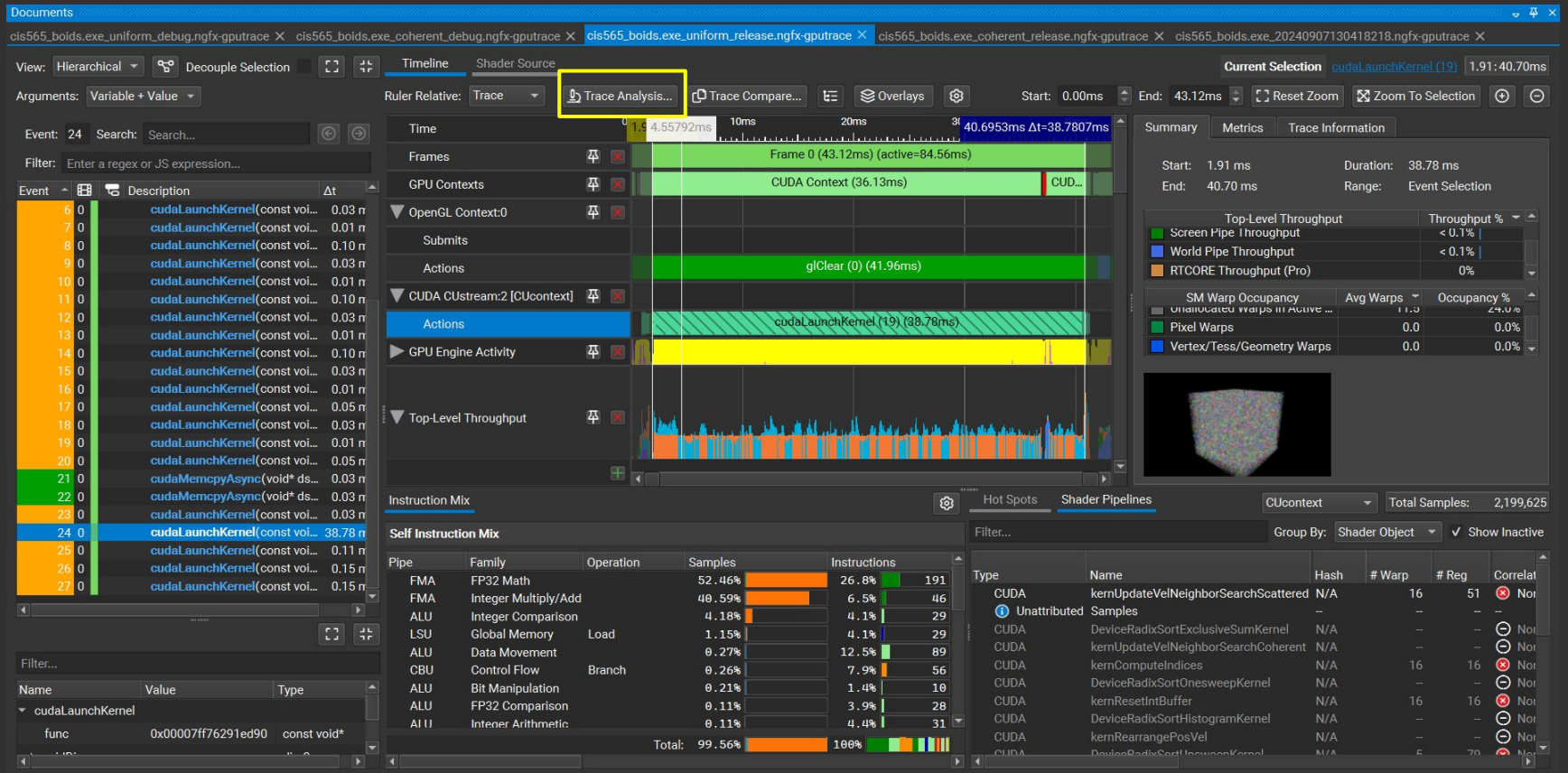
2. GPU Trace - Other Common Patterns

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

Credit: Louis Bavoil, GDC 2019

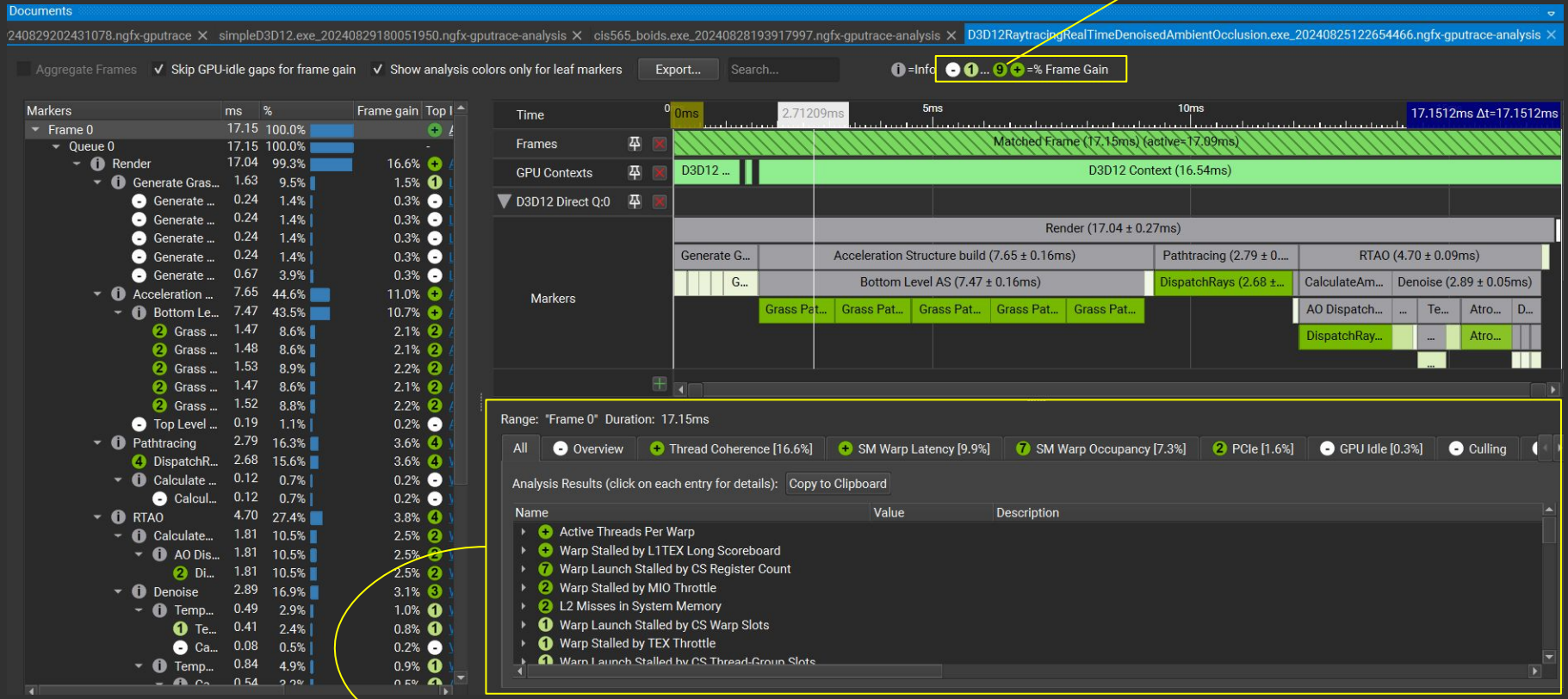
2. GPU Trace - Trace Analysis



Sascha Willems - computeParticles.exe

2. GPU Trace - Trace Analysis

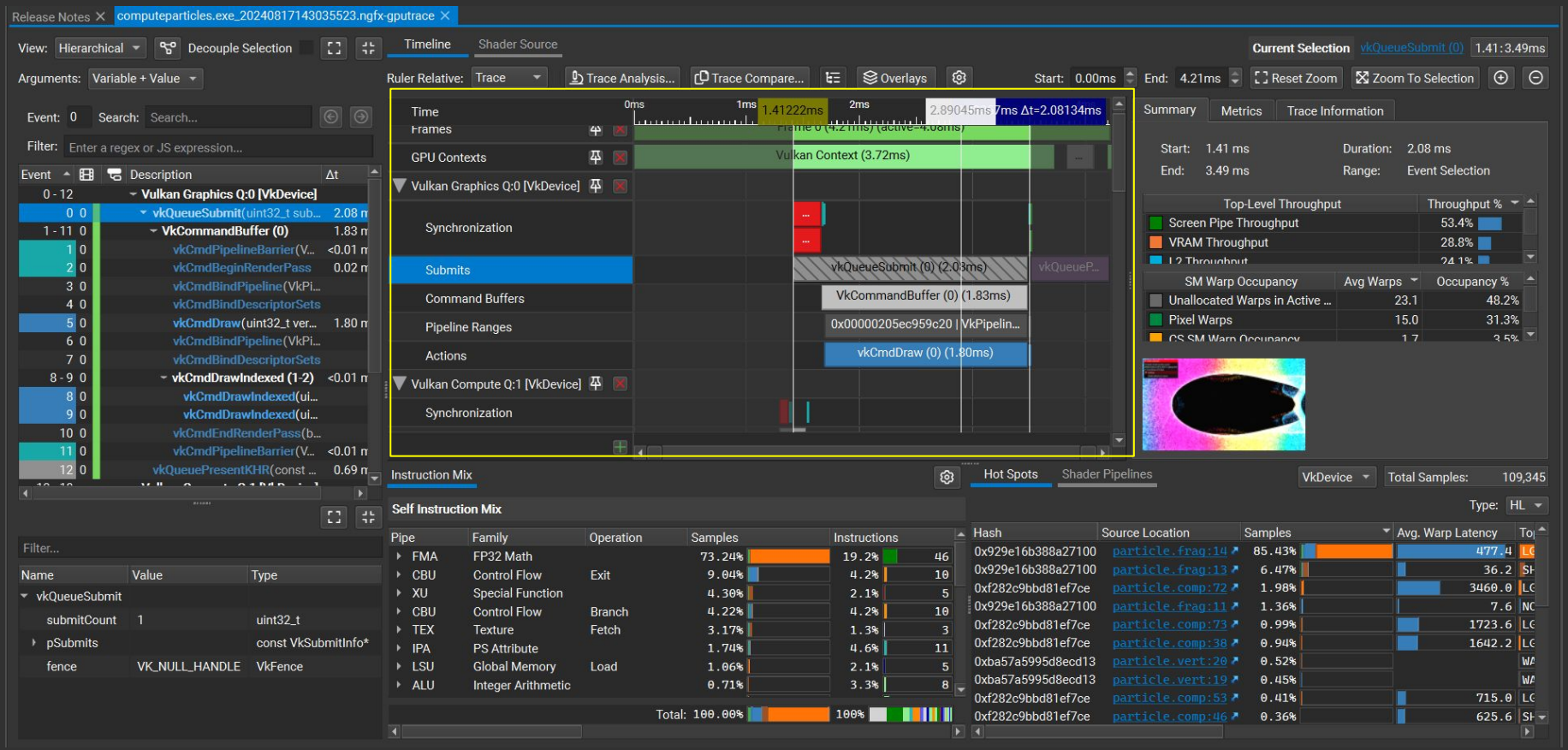
Projected Frame gain % if you apply the suggested changes



Potential problems, explanation of them, and solution suggestions

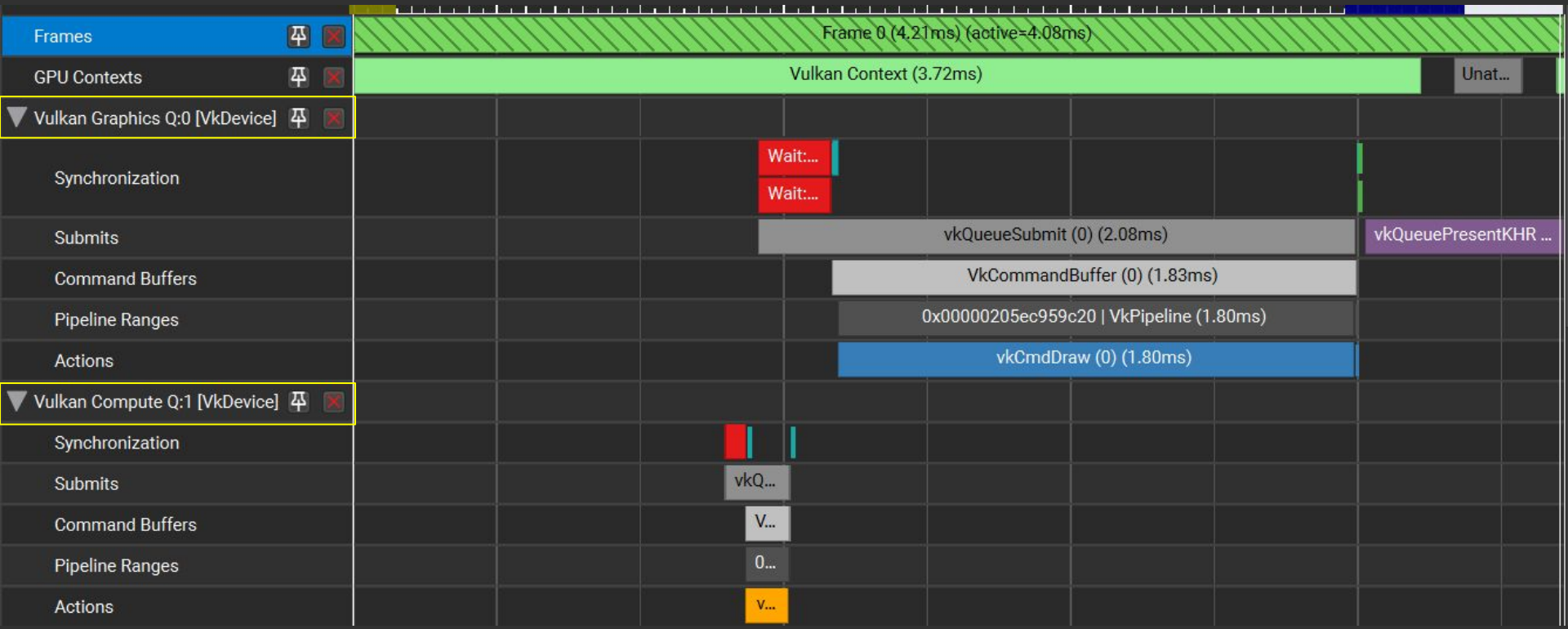
D3D12RealtimeRayTraceDenoiseAmbientOcclusion.exe

2. GPU Trace - Timeline



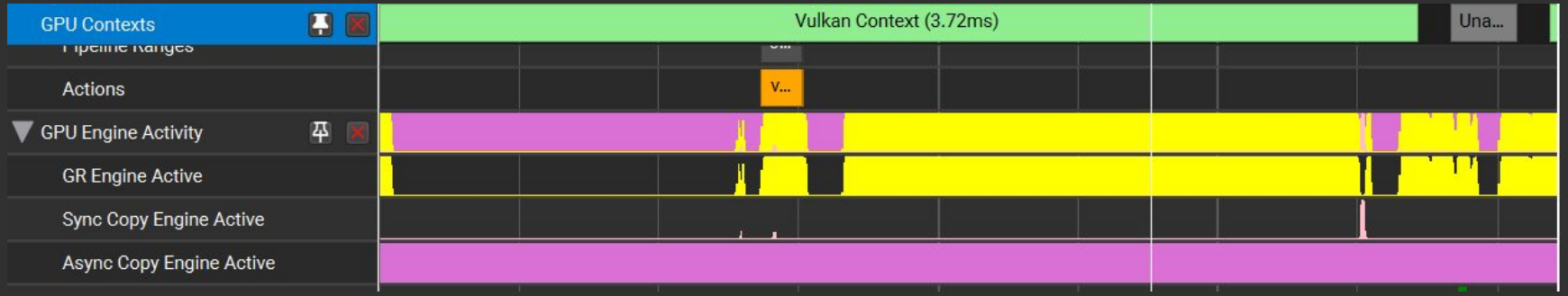
Sascha Willems - computeParticles.exe

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



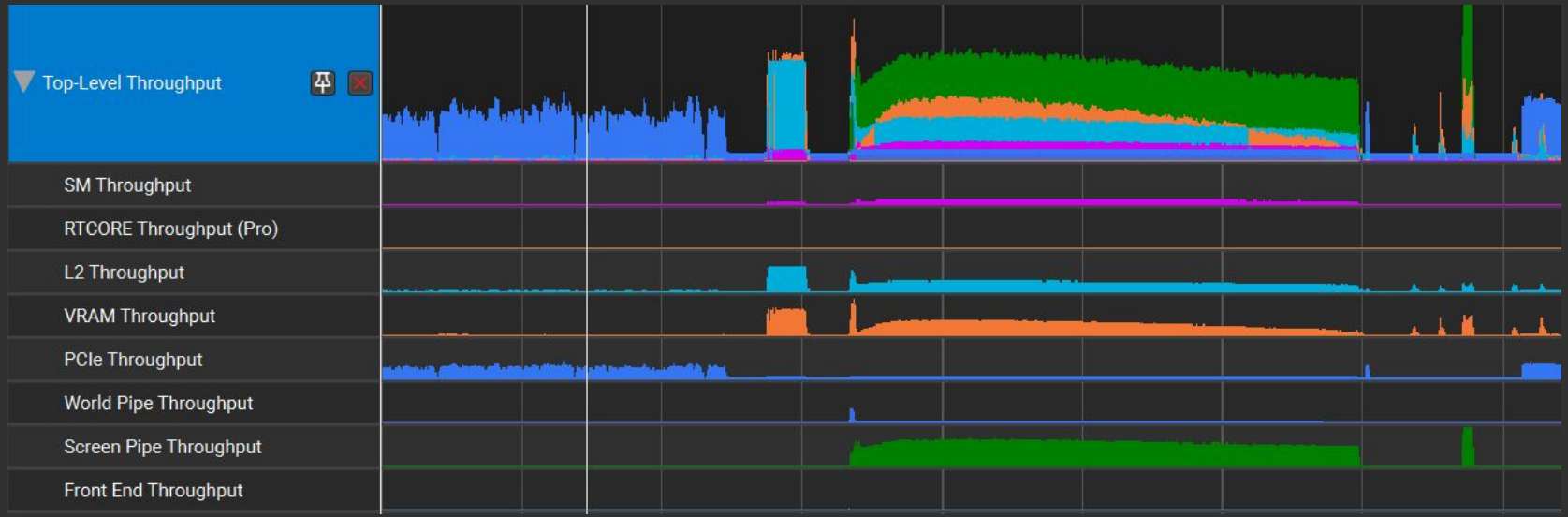
GPU Active - Duration during which any Graphics or Copy Engine were active

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

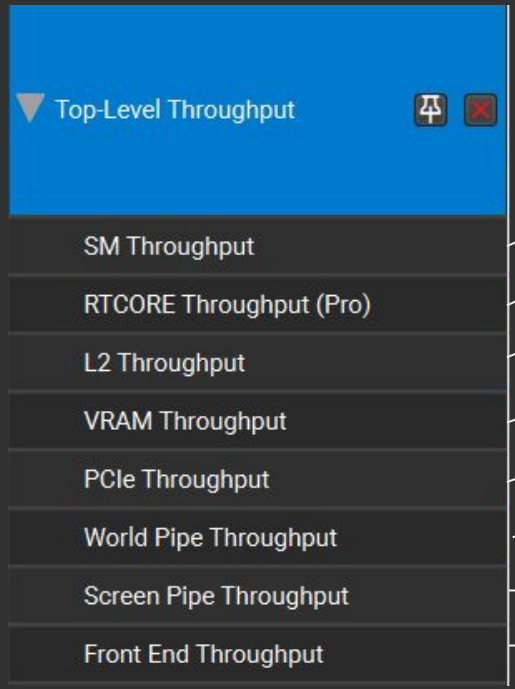
Sync Copy Engine - [CPU and GPU Memcpy] Blocking CPU until copy is done

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



SM (sum of throughput of its pipes)

RTCore Cycles

L2 Cache (Global Mem, shared across all units on the GPU)

VRAM Read/Write

Data Transfer over PCIe Bus

[Primitive Distributor, Vertex Attribute Fetch, Primitive Engine, and VPC Clip Cull Throughputs]

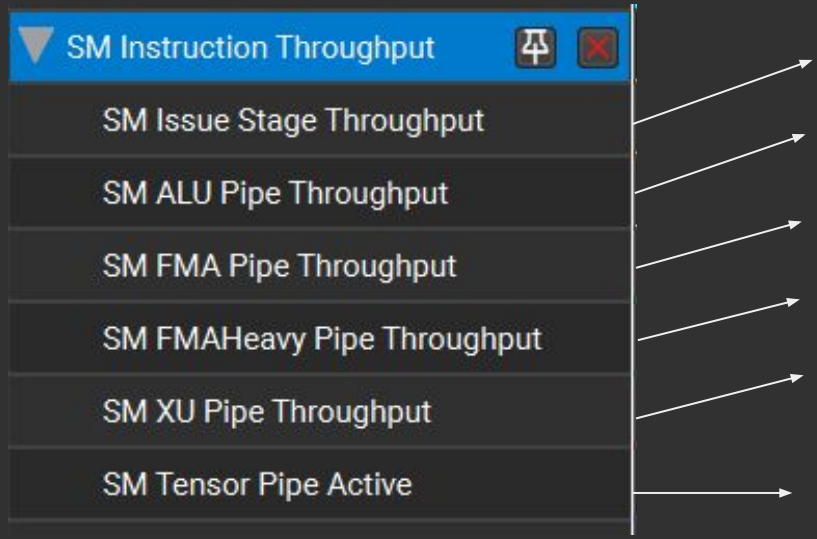
[RASTER, ZROP, CROP, PROP]

Input Assembler

2. GPU Trace - Timeline

2. GPU Trace - Timeline

Shader cores' performance:



Instruction Issuing

INT32 (excluding multiply and dot product)

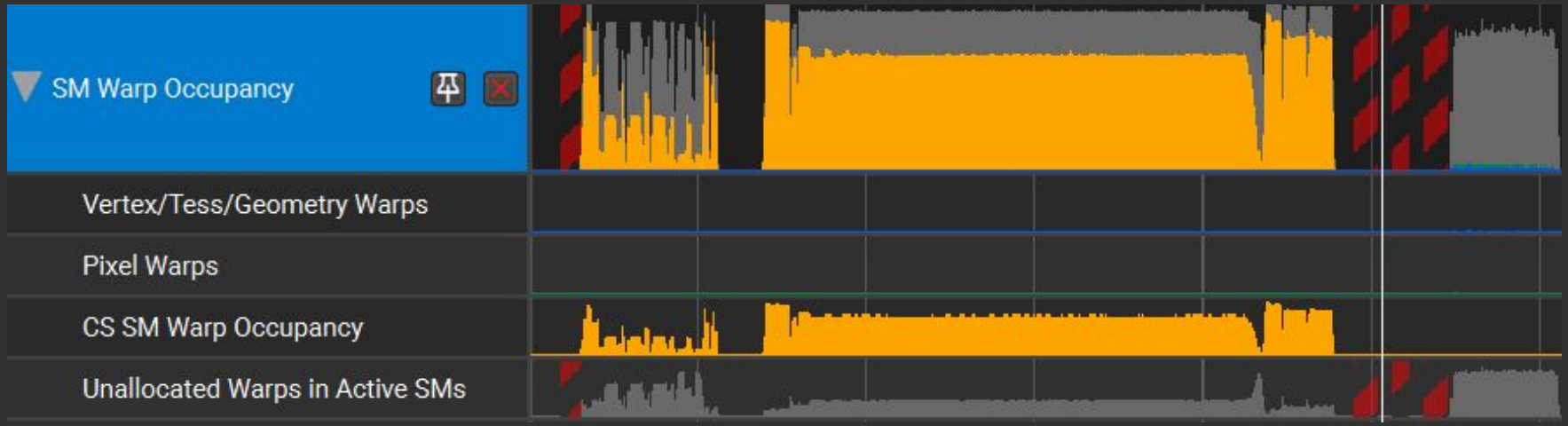
FP16 and FP32 (sum of FMAHeavy + FMALite)

Int multiplication & dot product

Special Functions: reciprocals, sqrt, transcendentals

Mixed-precision(data type) matrix multiplication (HLSL Wave Matrix, CUDA MMA)

2. GPU Trace - Timeline

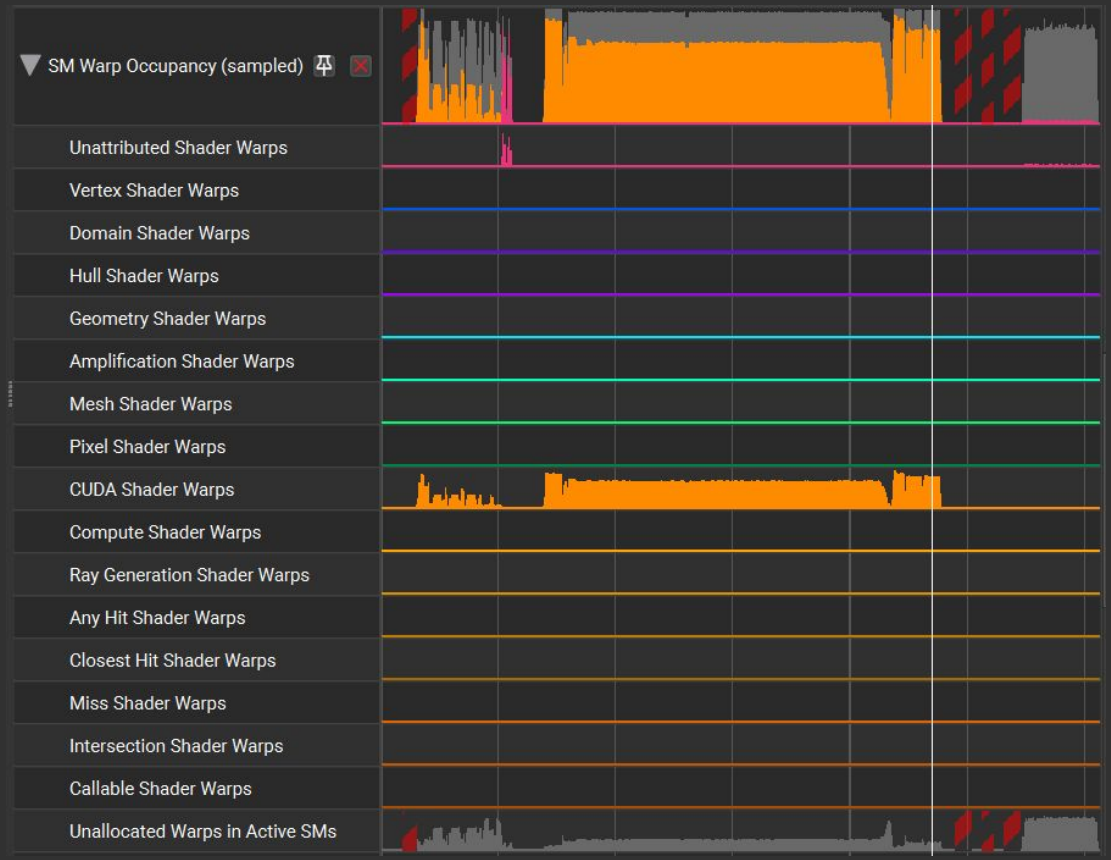


Active SM Unused Warp Slots:

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

Sascha Willems - computeParticles.exe >> Real-Time Shader Profiler turned OFF

2. GPU Trace - Timeline



My boids project

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

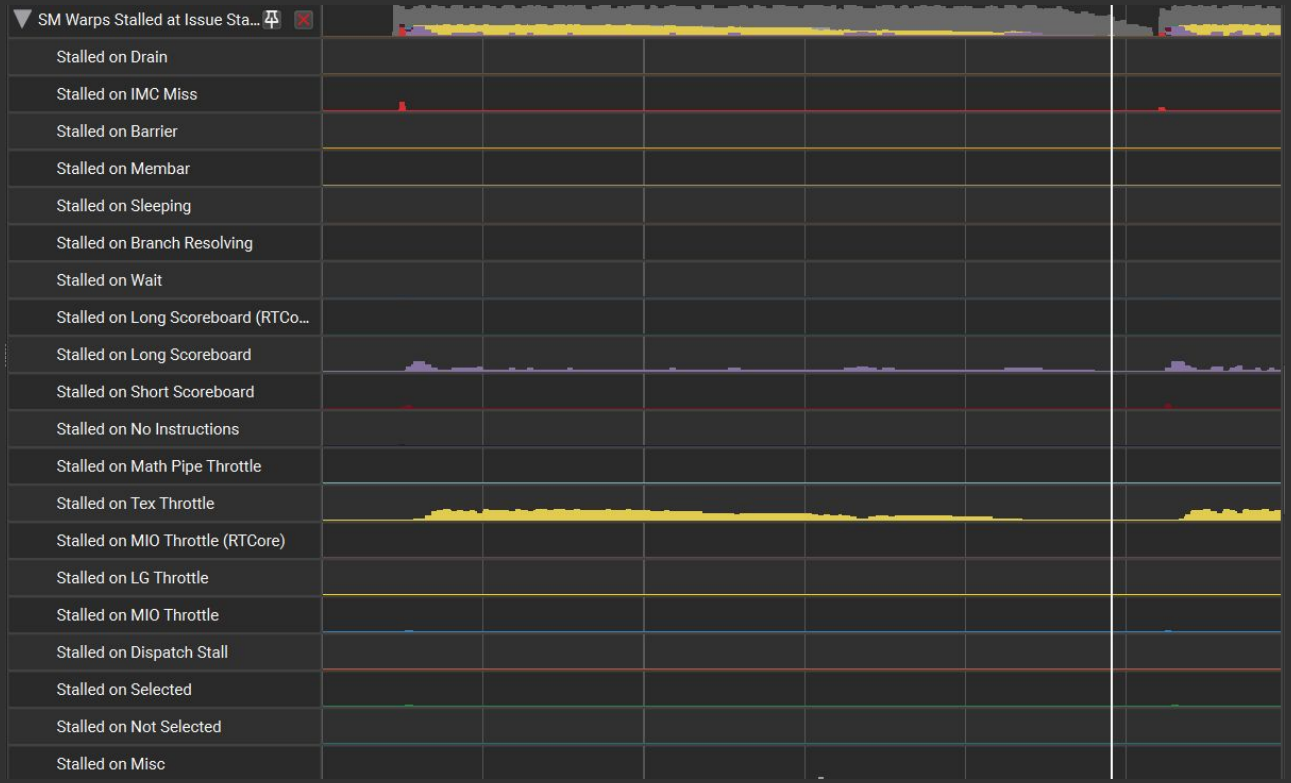
Warps Issue Stalled

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

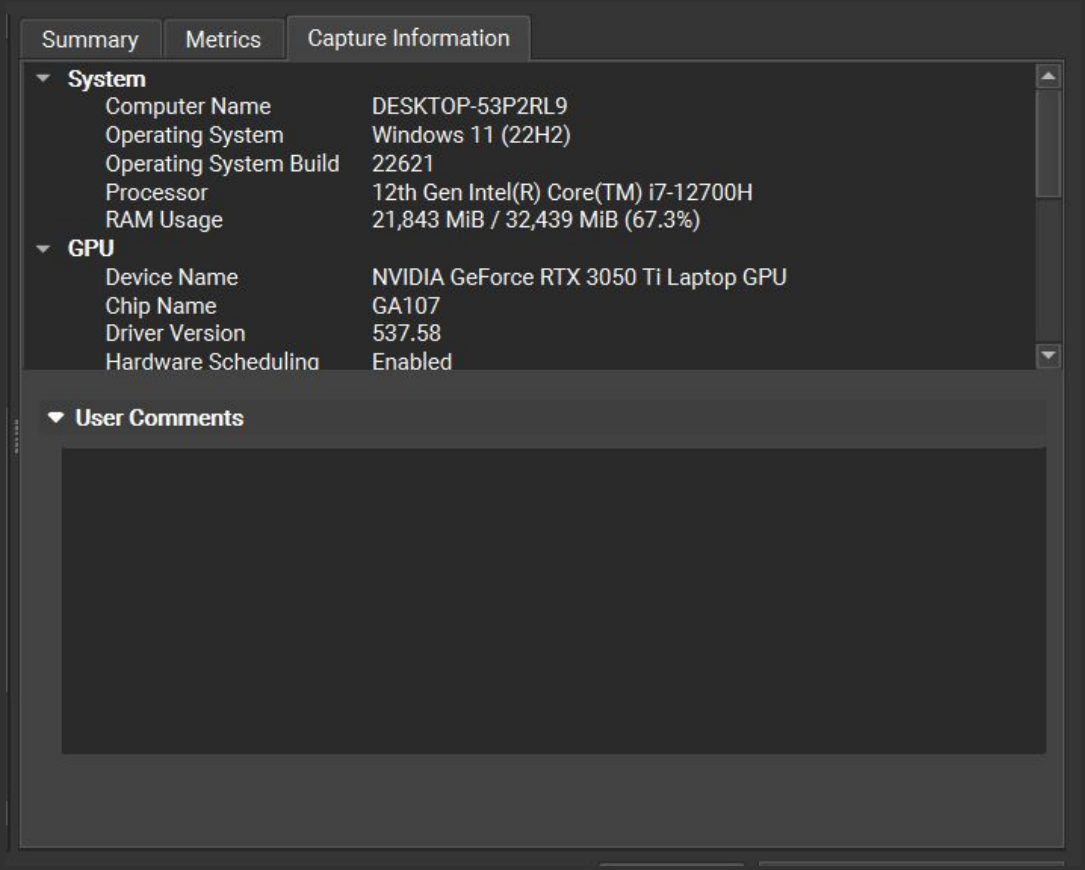
SM Warps Stalled at Issue Stage	Avg Warps	Occupancy %
Stalled on Barrier	0.0	0.0%
Stalled on Branch Resolving	0.2	0.3%
Stalled on Dispatch Stall	0.0	0.0%
Stalled on Drain	0.0	0.1%
Stalled on IMC Miss	0.0	0.0%
Stalled on LG Throttle	0.2	0.4%
Stalled on Long Scoreboard	6.5	13.5%
Stalled on Long Scoreboard (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

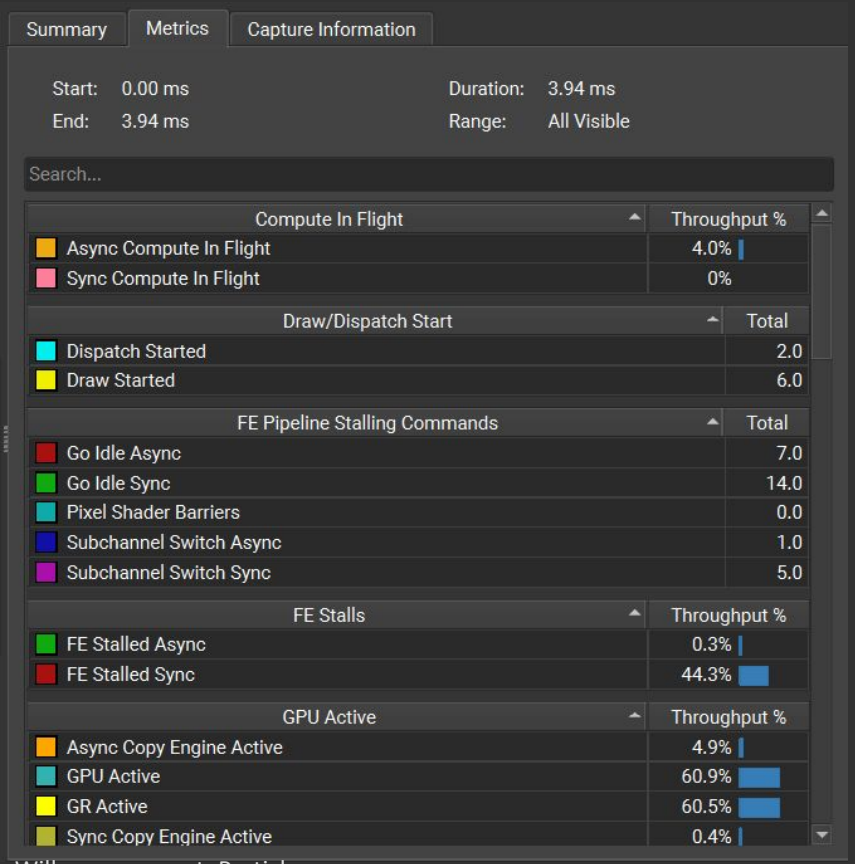


2. GPU Trace - Capture Information



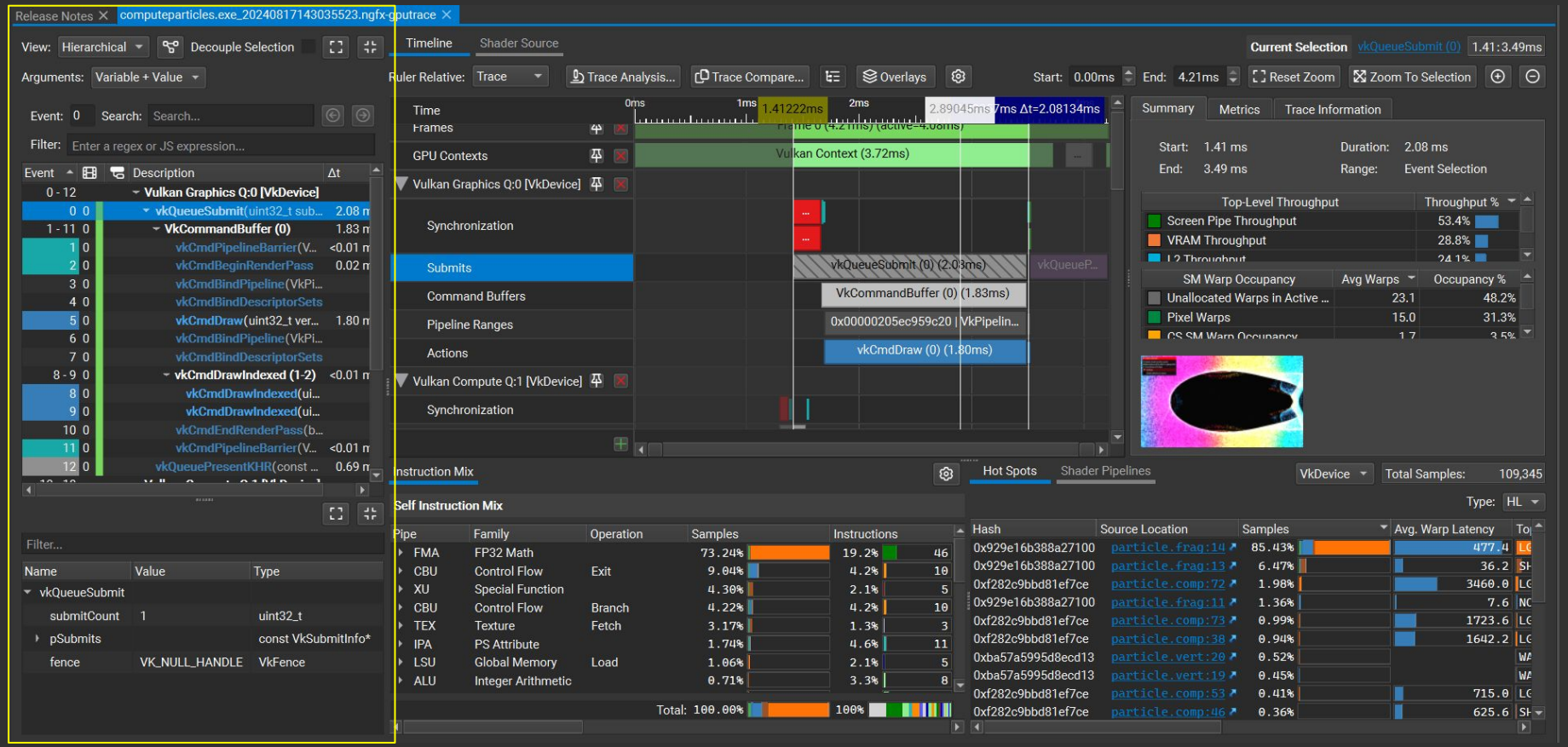
Sanity check for what the application is running against.

2. GPU Trace - Metrics Info



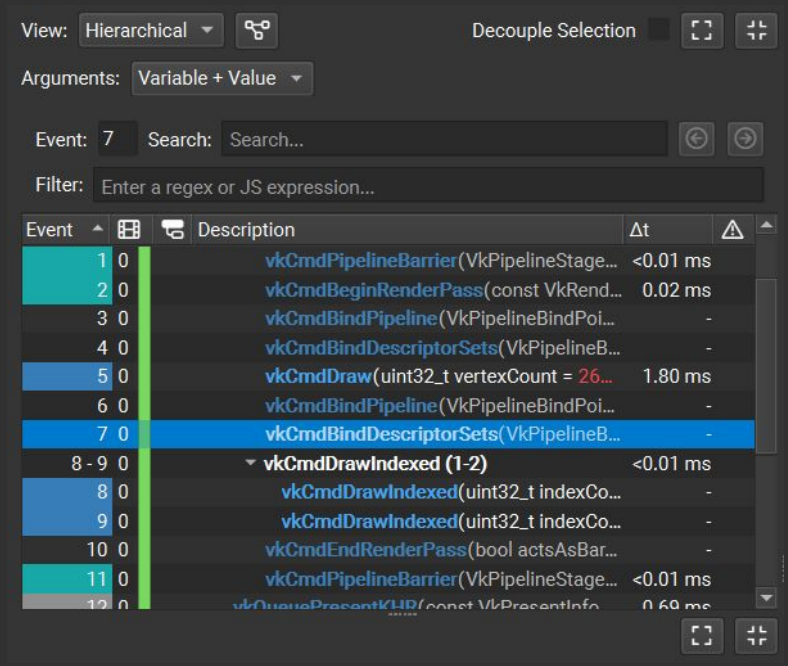
NOTE:
Your machine being plugged into power will affect the duration of your frame

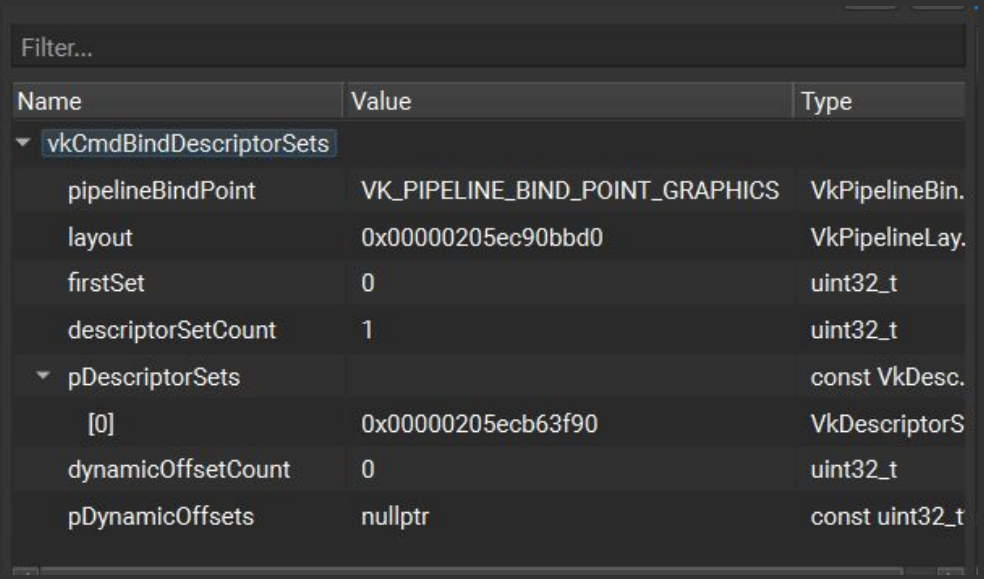
2. GPU Trace - API Event List



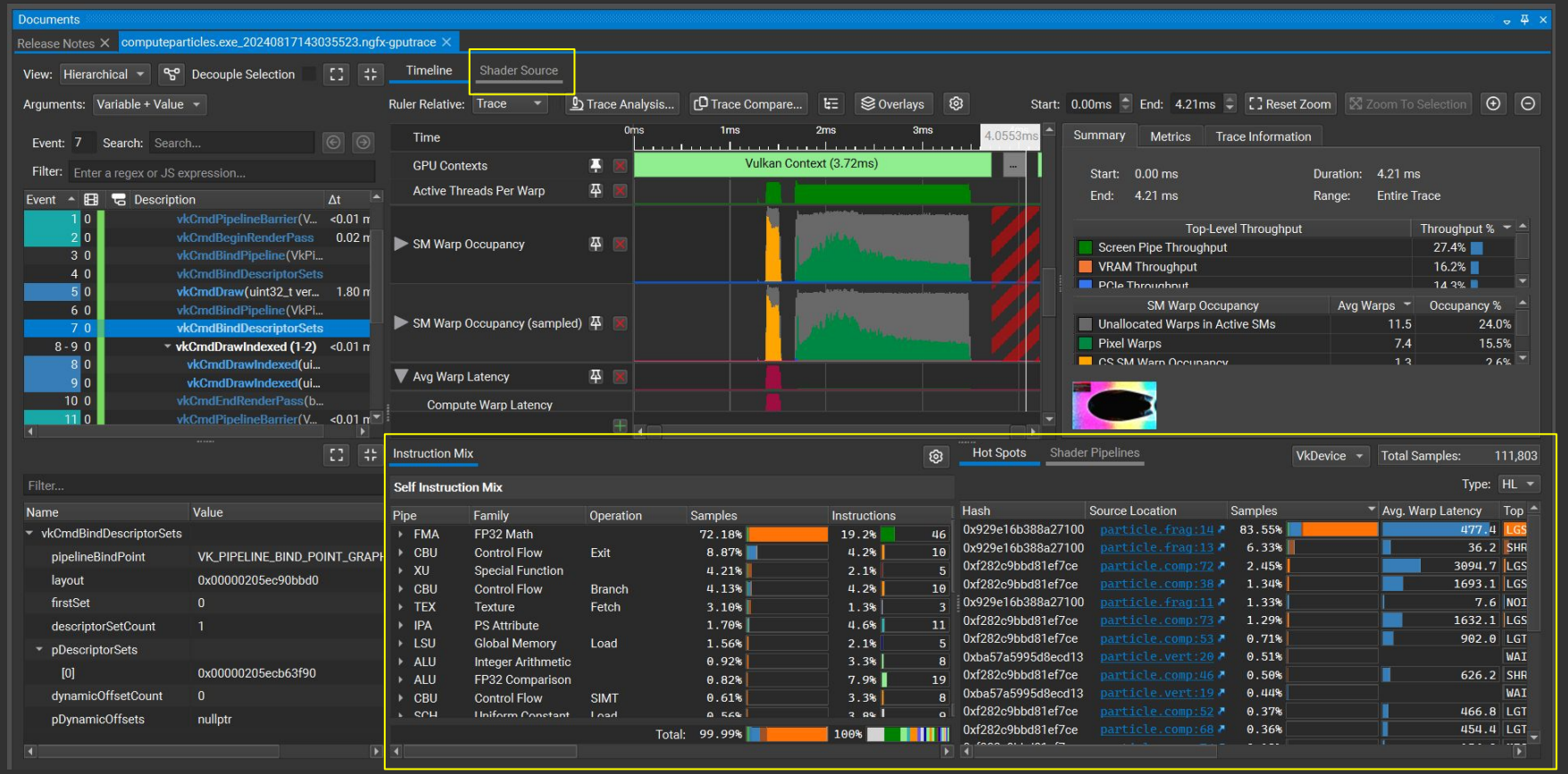
Sascha Willems - computeParticles.exe

2. GPU Trace - Items Navigator



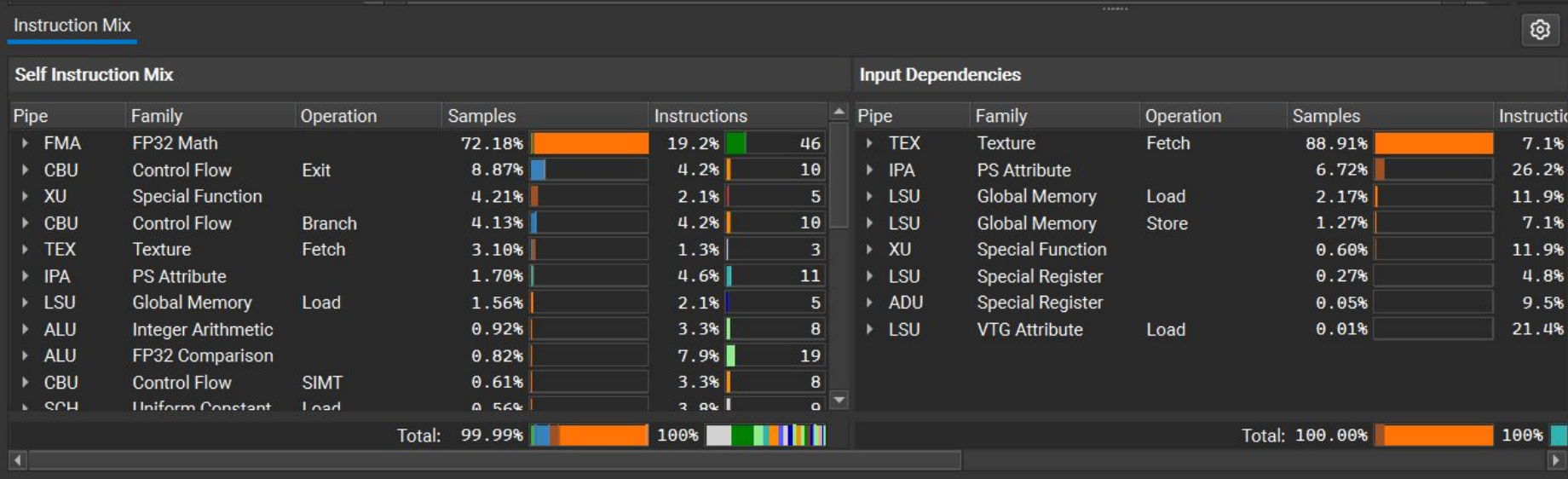


2. GPU Trace - Real-Time Shader Profiler



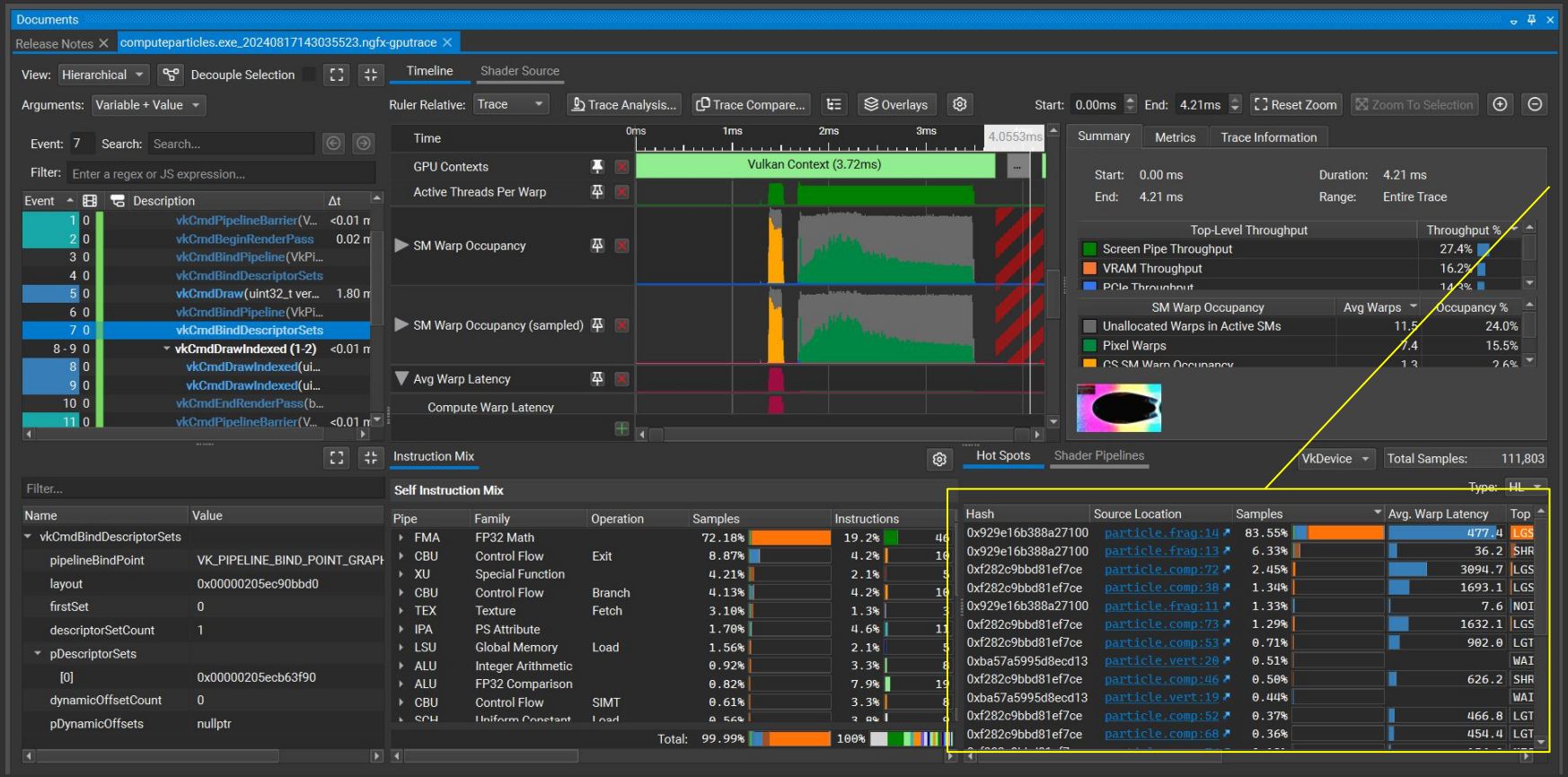
Sascha Willems - computeParticles.exe

2. GPU Trace - Instruction Mix

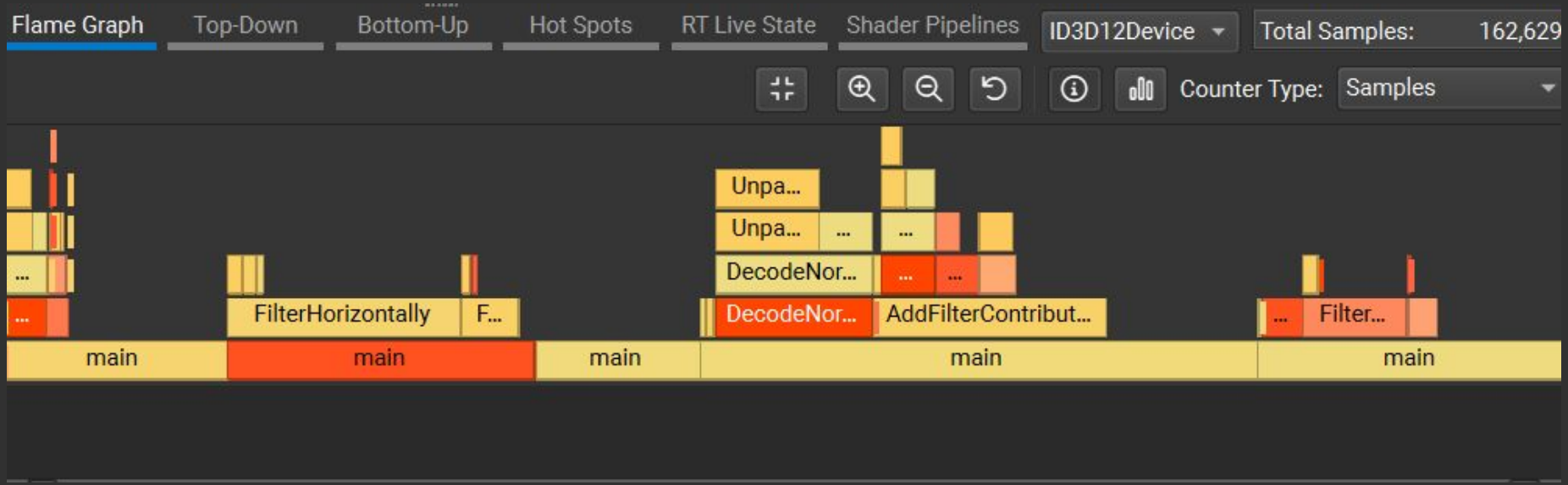


What types of instructions most commonly occur across your shaders?

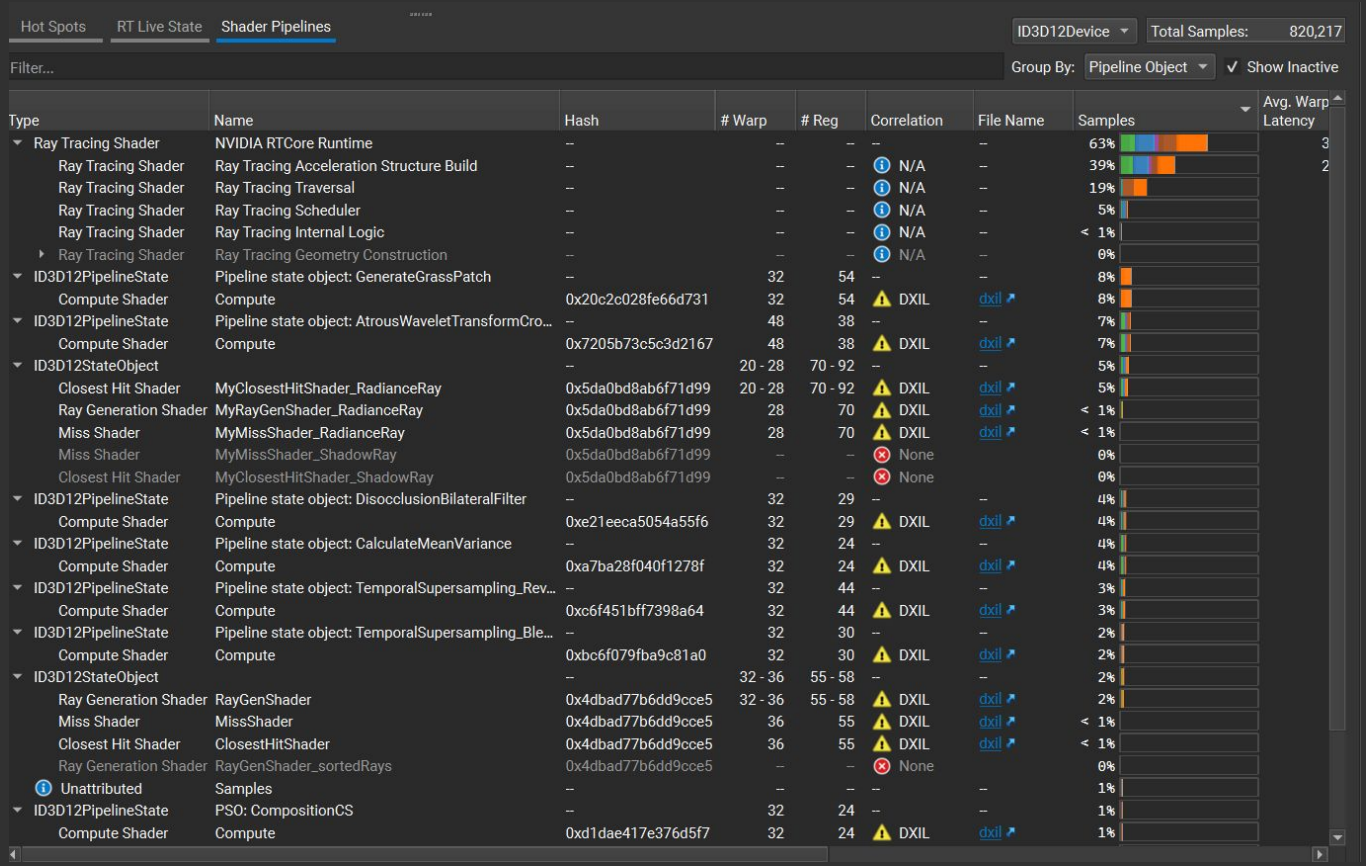
2. GPU Trace - Hot Spots



2. GPU Trace - Flame Graph

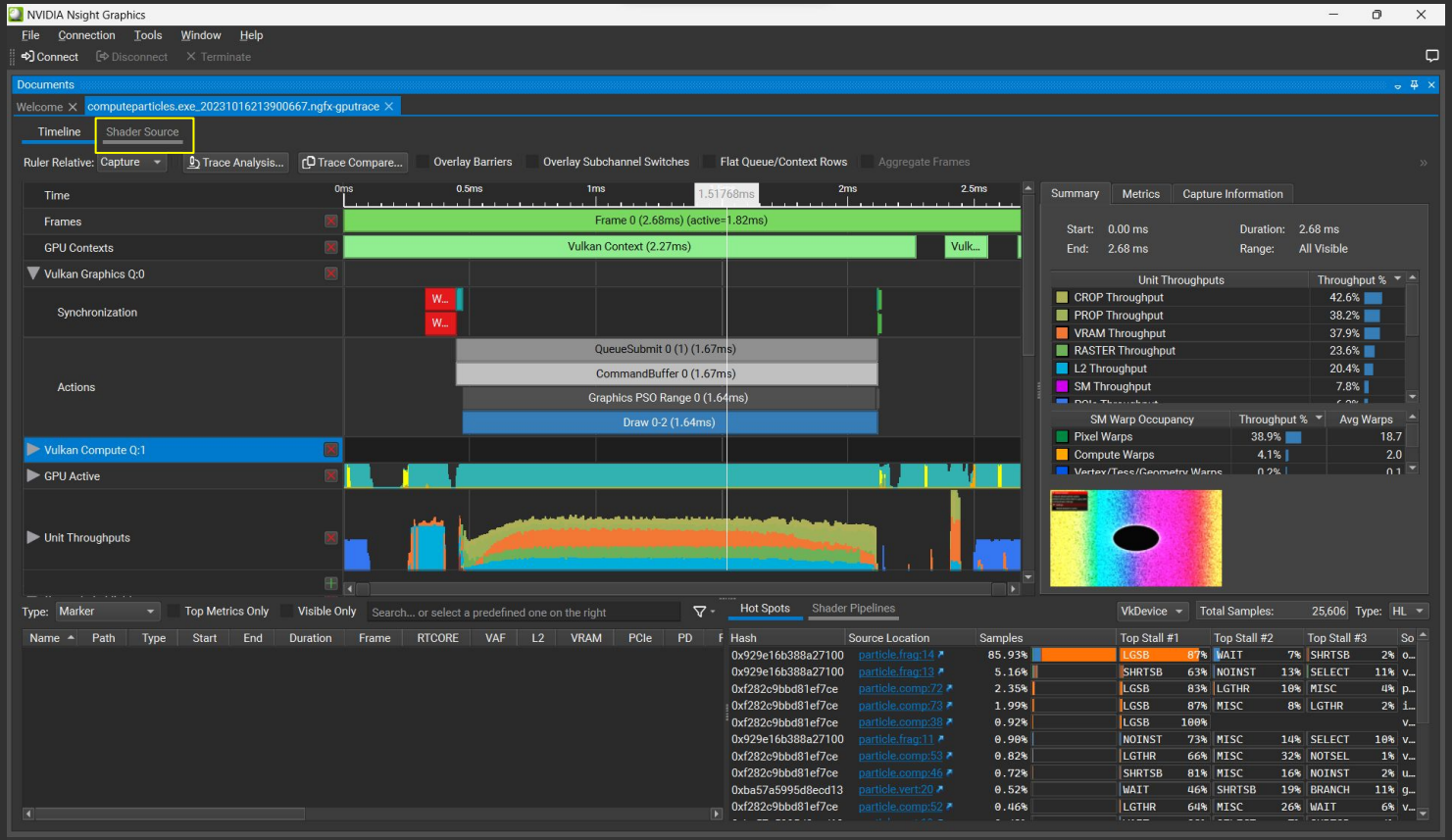


2. GPU Trace - Shader Pipelines/Shader Objects/Source



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	dxil	8%	
ID3D12PipelineState	Pipeline state object: AtrousWaveletTransformCro...	--	48	38	--	--	7%	
Compute Shader	Compute	0x7205b73c5c3d2167	48	38	DXIL	dxil	7%	
ID3D12StateObject		--	20 - 28	70 - 92	--	--	5%	
Closest Hit Shader	MyClosestHitShader_RadianceRay	0x5da0bd8ab6f71d99	20 - 28	70 - 92	DXIL	dxil	5%	
Ray Generation Shader	MyRayGenShader_RadianceRay	0x5da0bd8ab6f71d99	28	70	DXIL	dxil	< 1%	
Miss Shader	MyMissShader_RadianceRay	0x5da0bd8ab6f71d99	28	70	DXIL	dxil	< 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	0xe21eeca5054a55f6	32	29	DXIL	dxil	4%	
ID3D12PipelineState	Pipeline state object: CalculateMeanVariance	--	32	24	--	--	4%	
Compute Shader	Compute	0xa7ba28f040f1278f	32	24	DXIL	dxil	4%	
ID3D12PipelineState	Pipeline state object: TemporalSupersampling_Rev...	--	32	44	--	--	3%	
Compute Shader	Compute	0xc6f451bff7398a64	32	44	DXIL	dxil	3%	
ID3D12PipelineState	Pipeline state object: TemporalSupersampling_Ble...	--	32	30	--	--	2%	
Compute Shader	Compute	0xbc6f079fba9c81a0	32	30	DXIL	dxil	2%	
ID3D12StateObject		--	32 - 36	55 - 58	--	--	2%	
Ray Generation Shader	RayGenShader	0x4dbad77b6dd9cce5	32 - 36	55 - 58	DXIL	dxil	2%	
Miss Shader	MissShader	0x4dbad77b6dd9cce5	36	55	DXIL	dxil	< 1%	
Closest Hit Shader	ClosestHitShader	0x4dbad77b6dd9cce5	36	55	DXIL	dxil	< 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	dxil	1%	

1. GPU Trace - Shader Source View

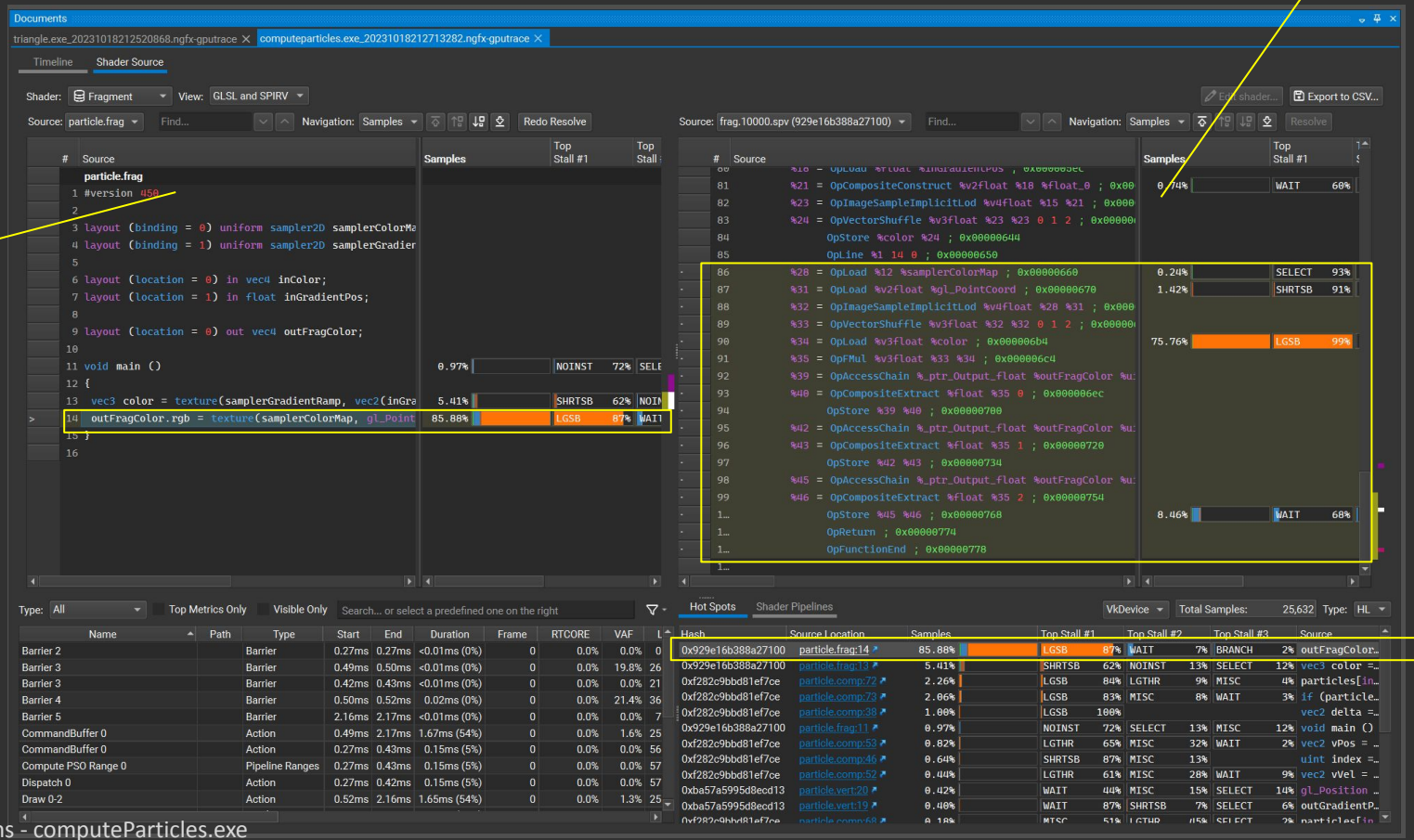


Sascha Willems - computeParticles.exe

1. GPU Trace - Shader Source View

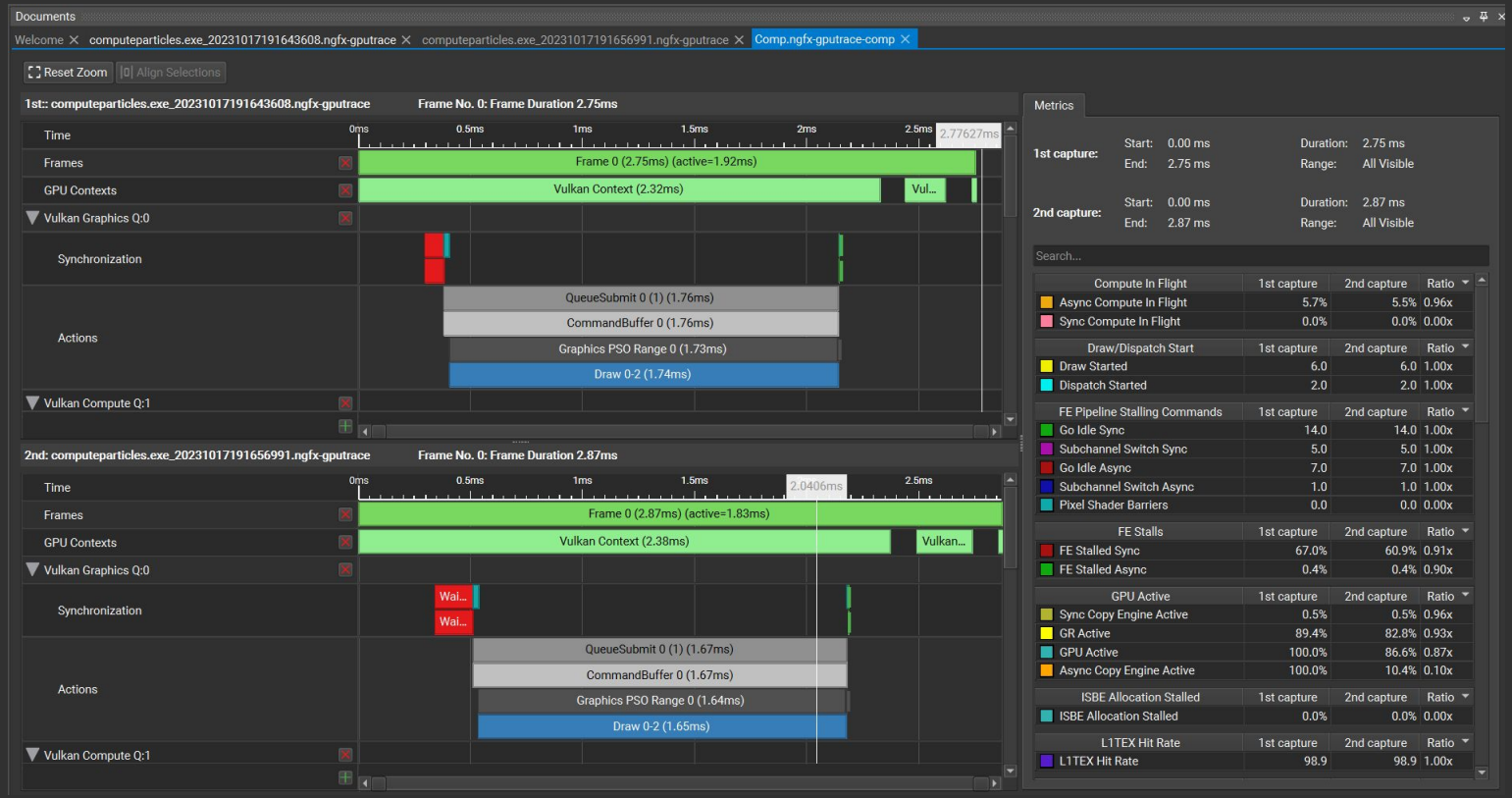
SPIR-V View

GLSL View



Sascha Willems - computeParticles.exe

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)

Credit: Louis Bavoil, GDC 2019

Career Advice (?)

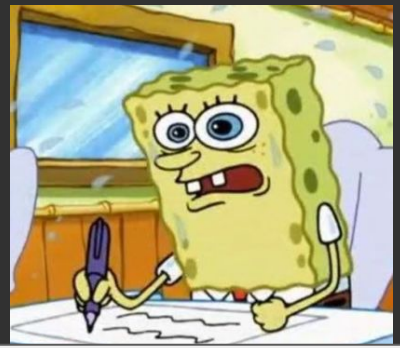
Build things from scratch

Spend time with your peers & random folks

Work smart and use tools!

Niche Knowledge

Set Realistic Expectations



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