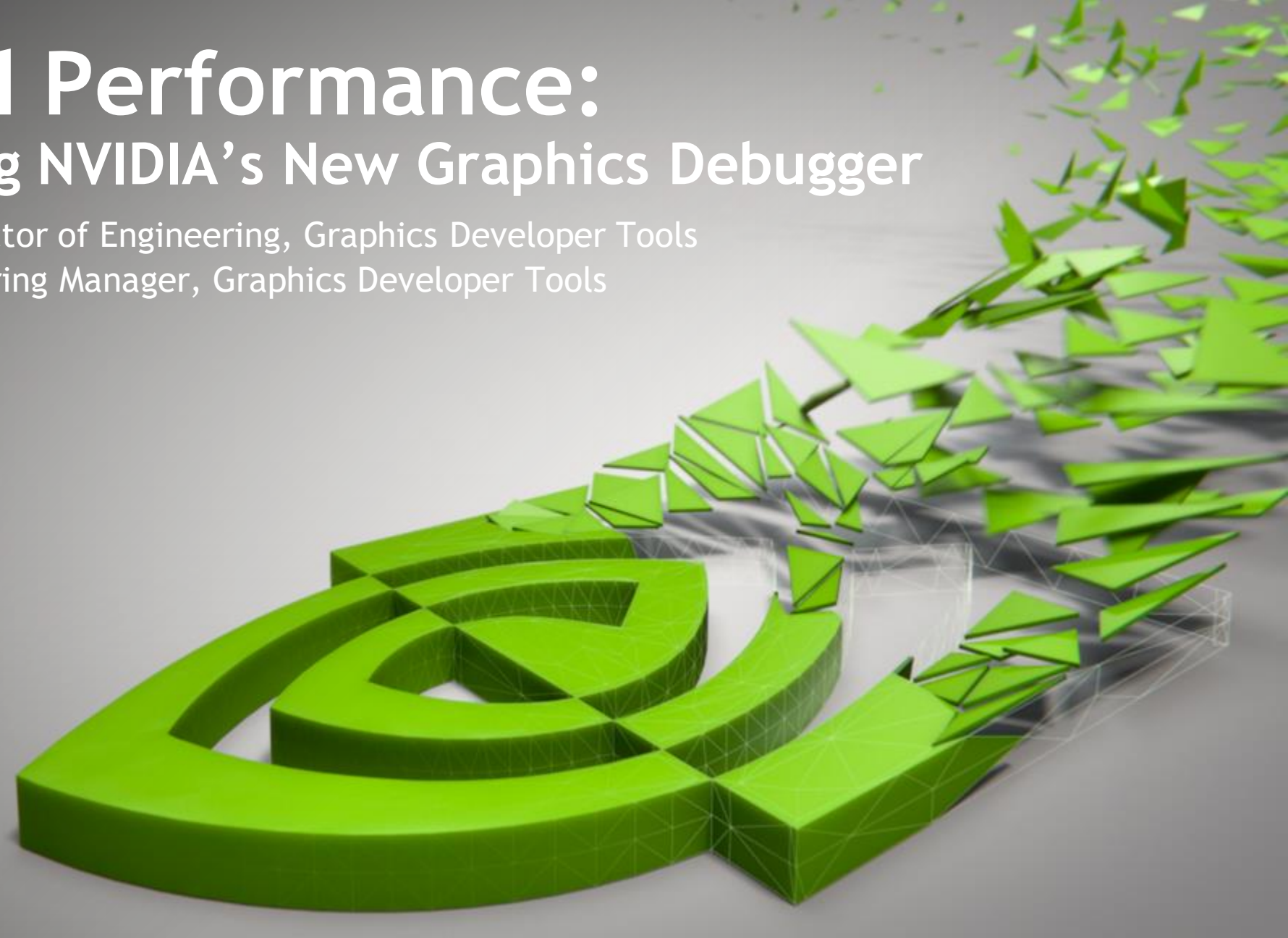
# Beyond Performance:
## Introducing NVIDIA's New Graphics Debugger

Aurelio Reis, Director of Engineering, Graphics Developer Tools
Jeff Kiel, Engineering Manager, Graphics Developer Tools

March 22

NVIDIA

Booth #223 - South Hall
www.nvidia.com/GDC



# Agenda

- Overview
- Demo
- Key Takeaways
- Q&A

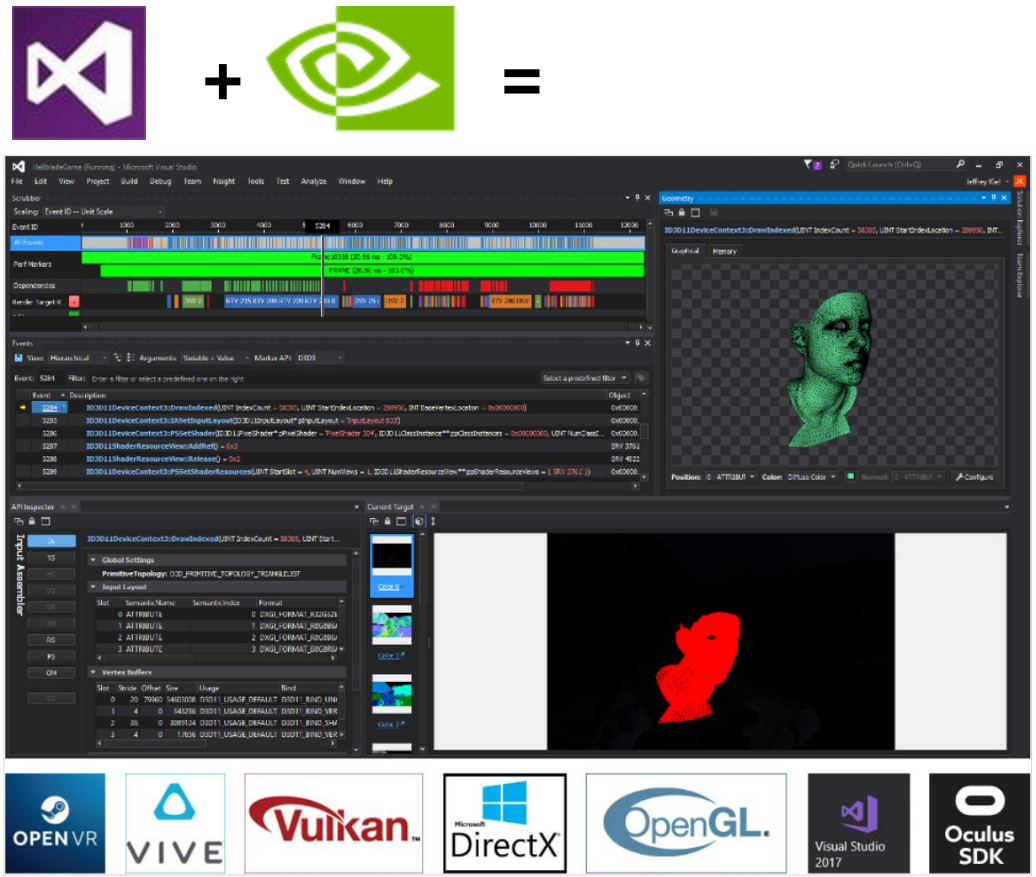
# Overview

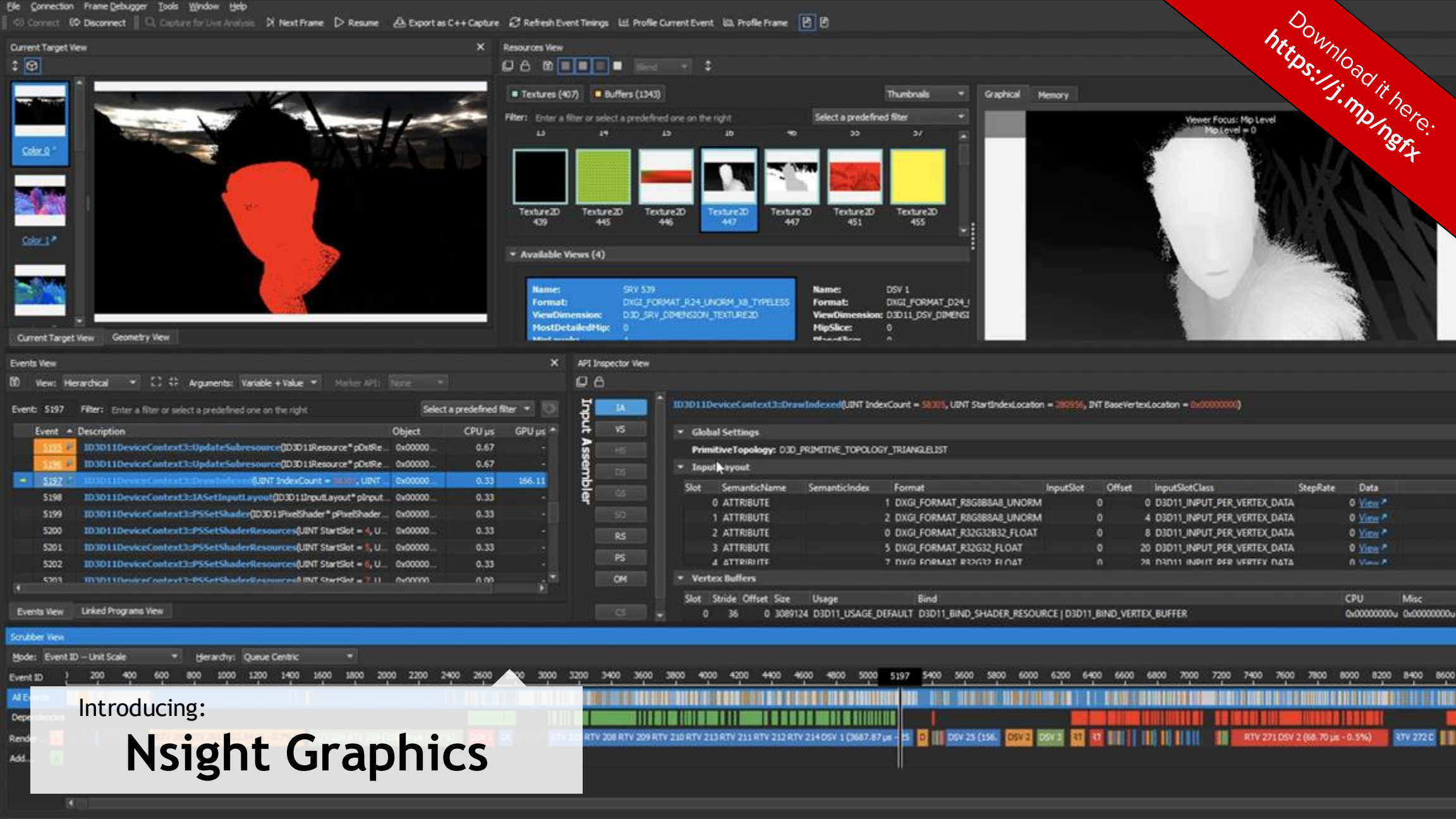
# The Story So Far...

- Nsight: Visual Studio Edition
  - Fully integrated into IDE
  - Graphics
  - Compute (CUDA)
  - 9+ years of learnings

- Ideal workflow for developers, but…
  - Reliant on VS ecosystem
  - Dependencies on compute for release

How can we improve?




Download it here:
https://j.mp/ngfx
Introducing:
Nsight Graphics

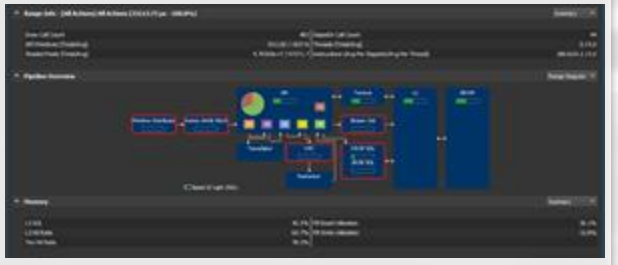
# Key Pillars

## Debugging

C++ Serialized Captures
Event Timeline (Scrubber)
Shader Editing
Pixel History
Resources Viewer
Geometry Viewer
API Inspection

## Profiling

Range Profiler
HW Perf Counters
Shader Stats
Event Timings
Counter Selector
API Stats
GPU Trace



## Ray Tracing (RTX)

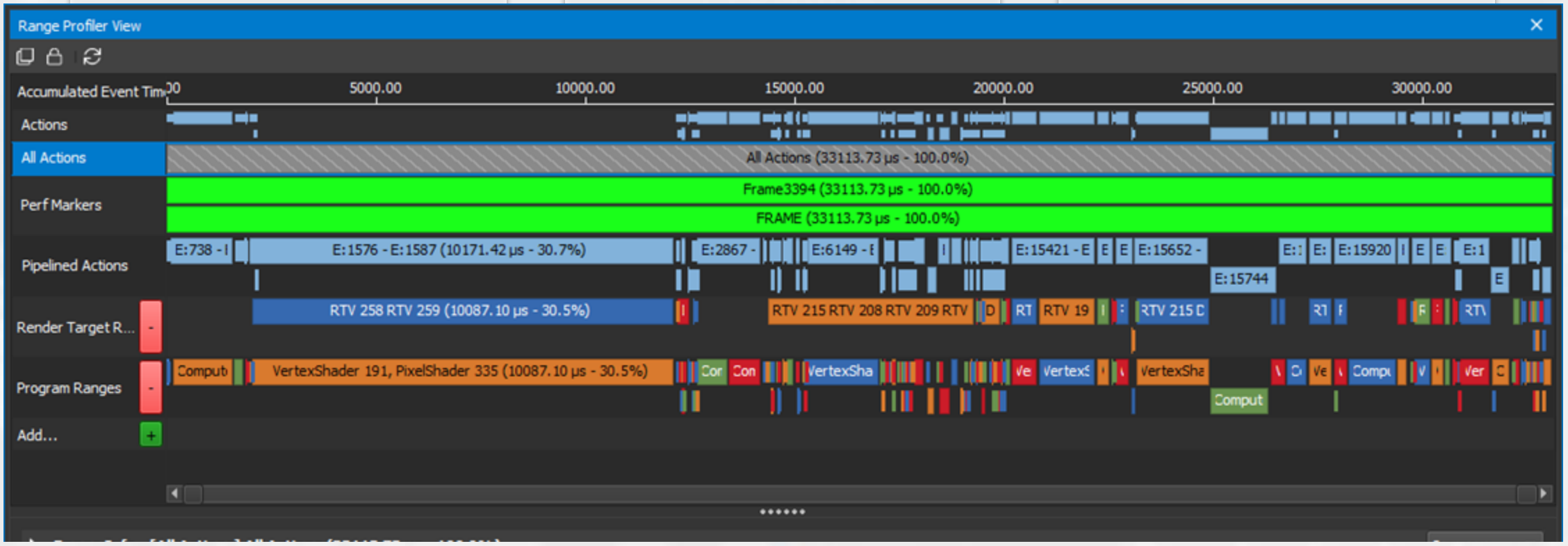
DXR Capture/Replay
DXR Frame Debugger
Shader Bind Table View

# Key Pillars

**Debugging**

**Profiling**

**Ray Tracing (RTX)**



# Key Pillars

**Debugging**

**Profiling**

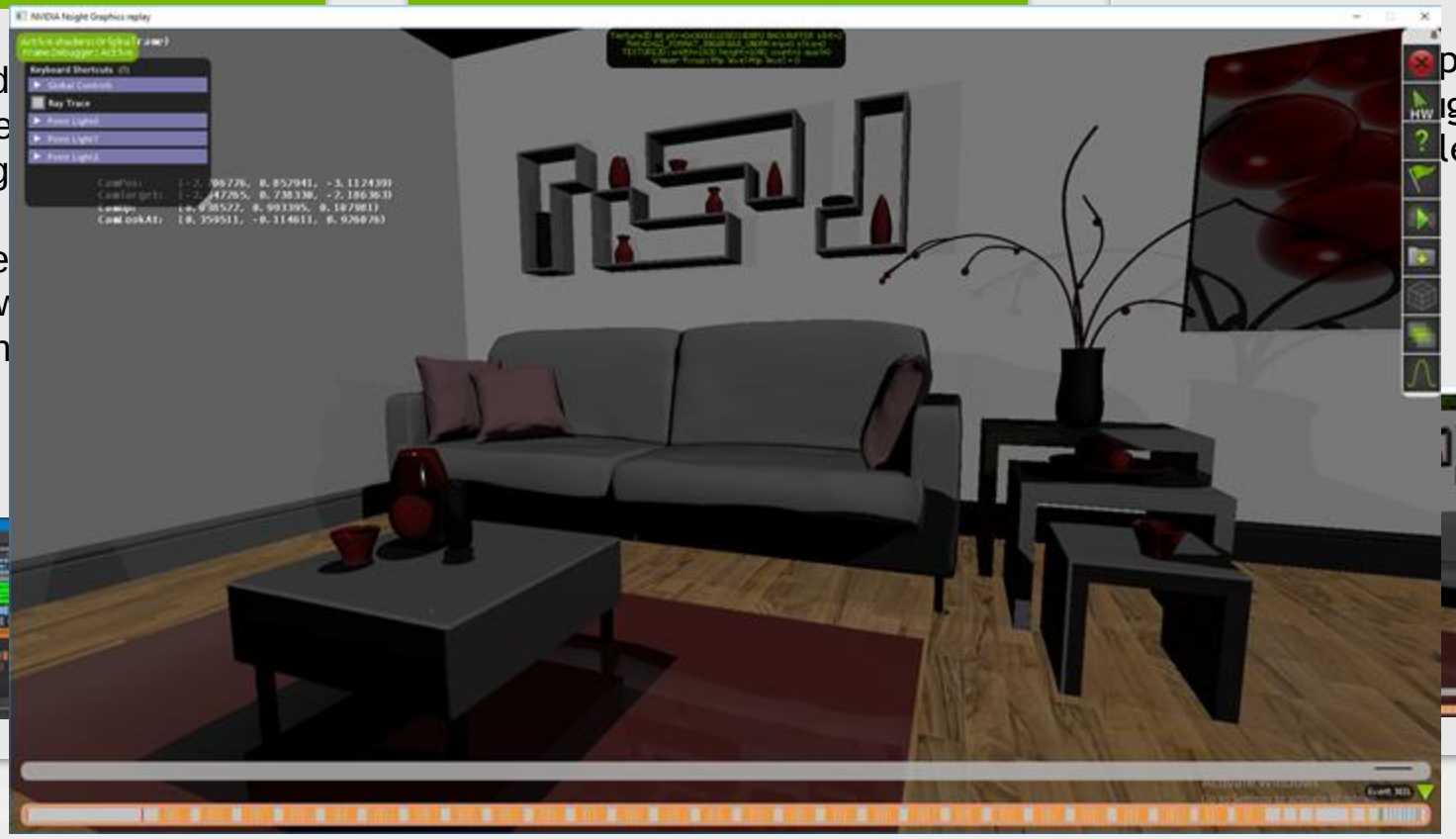
**Ray Tracing (RTX)**

# Key Pillars

## Debugging

C++ Serialized
Event Timeline
Shader Editing
Pixel History
Resources Vie
Geometry View
API Inspection

## Profiling



## Ray Tracing (RTX)

play
igger
le View

# Developer Choice

## Nsight: VSE

+ Visual Studio integration
+ Compute tools
+ Tracing tools

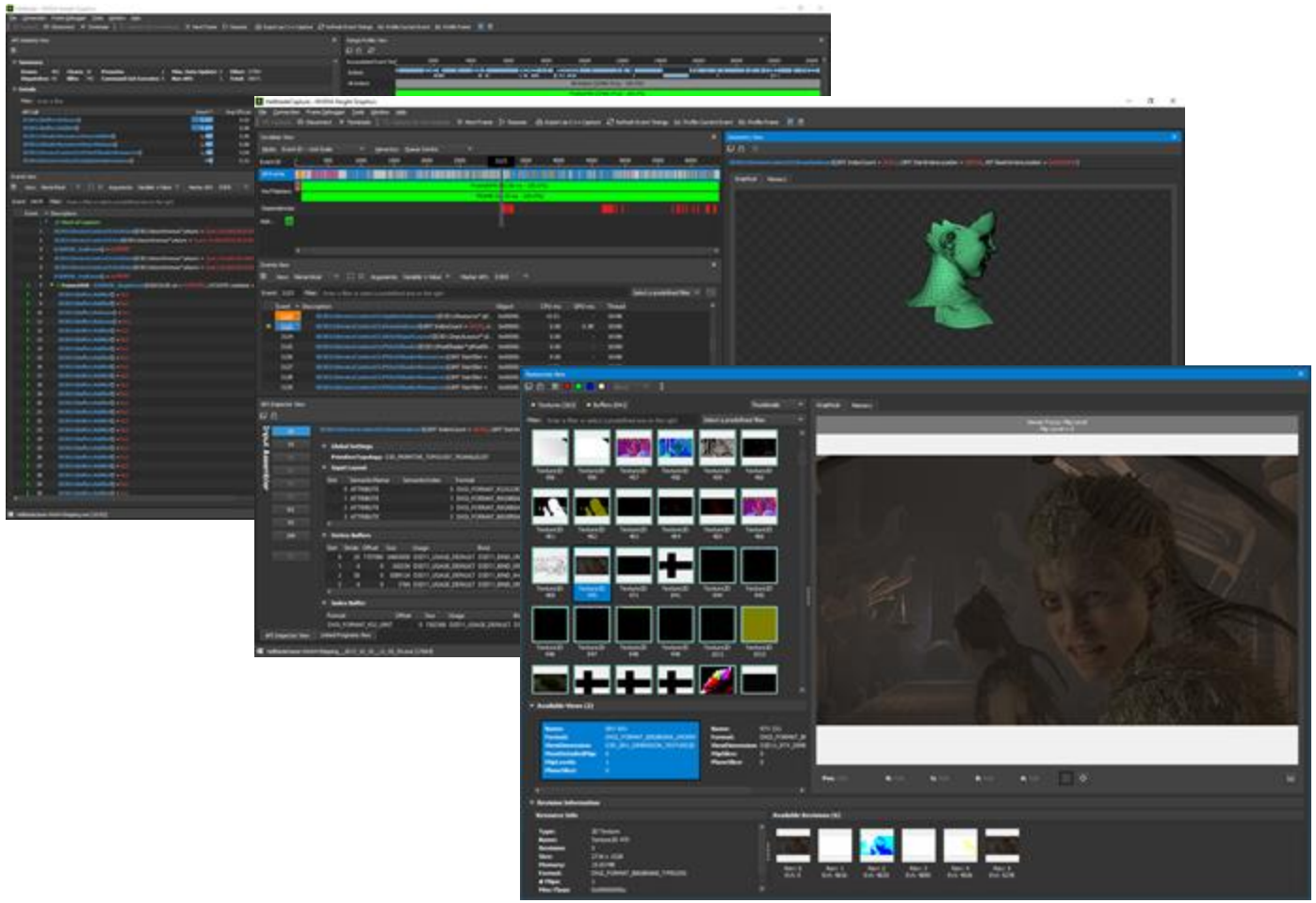
## Nsight Graphics

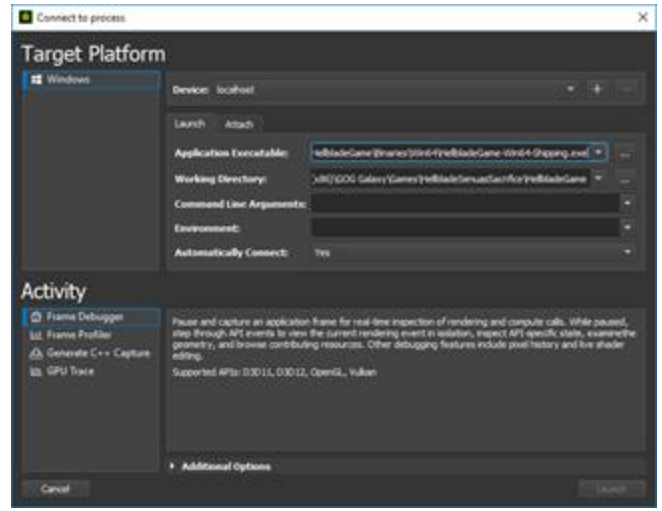
+ More frequent releases
+ Activities system
+ Advanced profiling

# Demo

1. Nsight Graphics Tour
2. DXR Debugging
3. Optimizing Async Compute

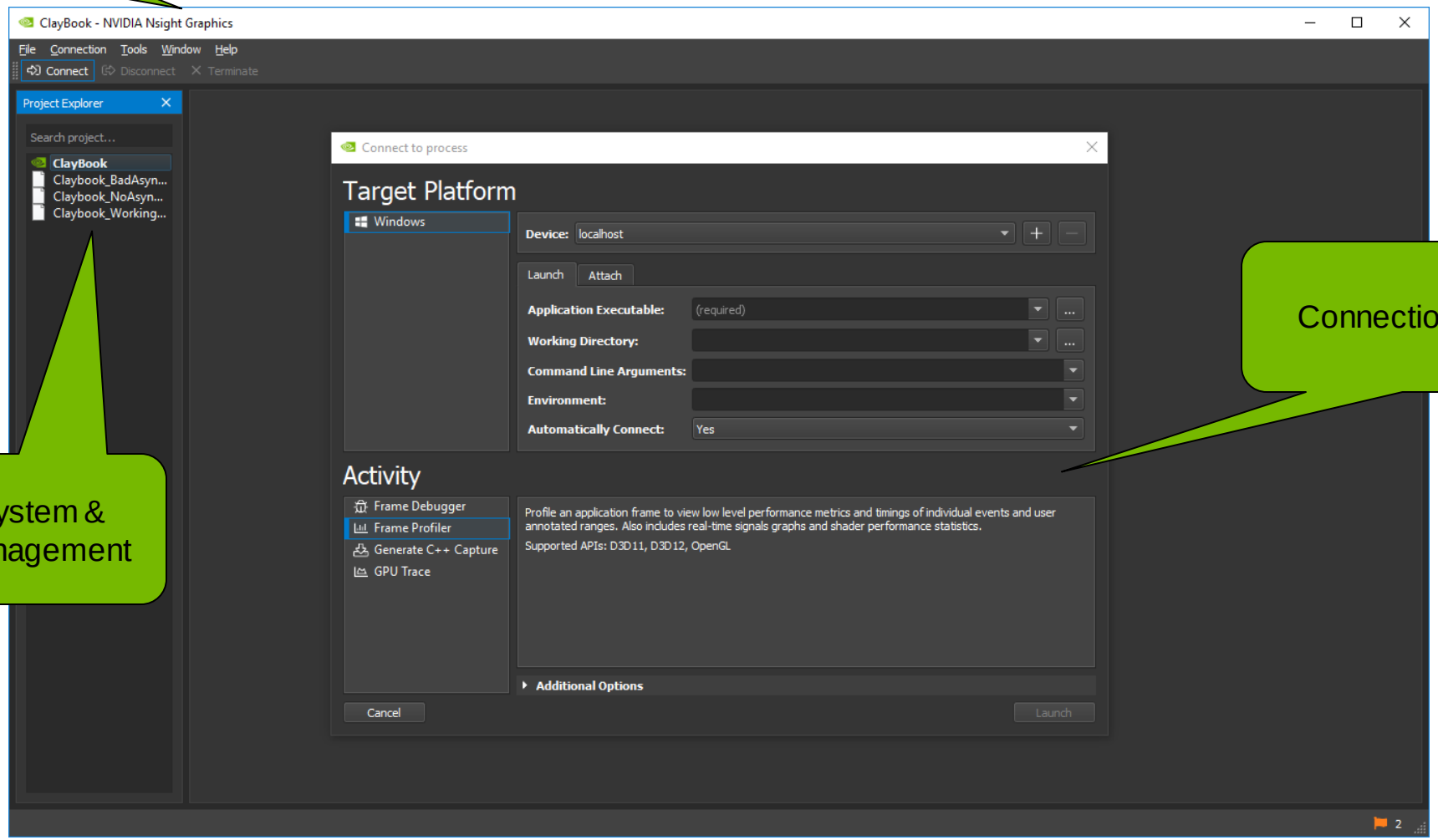
# Demo 1 - Nsight Graphics Tour






# Nsight Graphics Tour

New, stand alone shell!



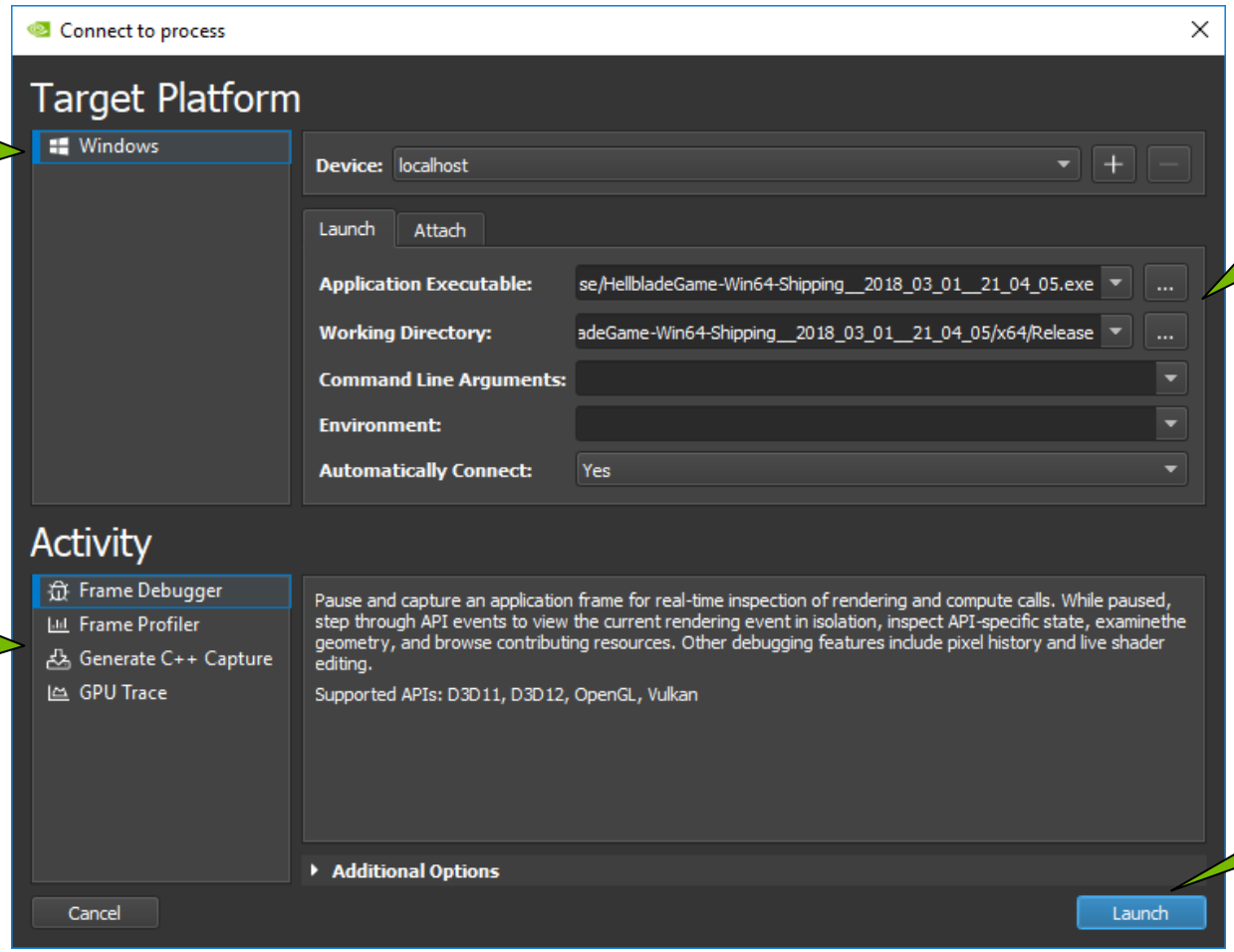

Project system & artifact management

Connection settings

# Nsight Graphics Tour

Select Platform….

…select Activity….

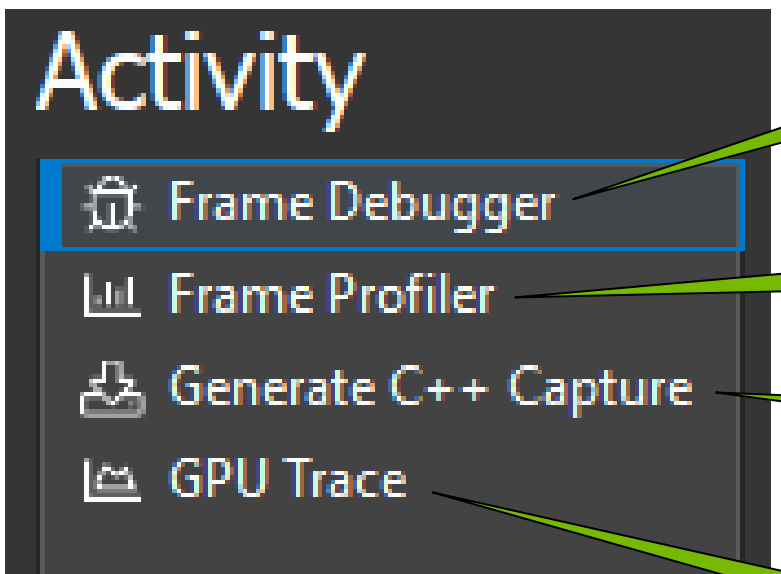



…add application parameters…

…Launch!

# Nsight Graphics Tour

## Targeted UI Layout Per Activity




Deep dive into frame rendering and API analysis

Automated throughput analysis

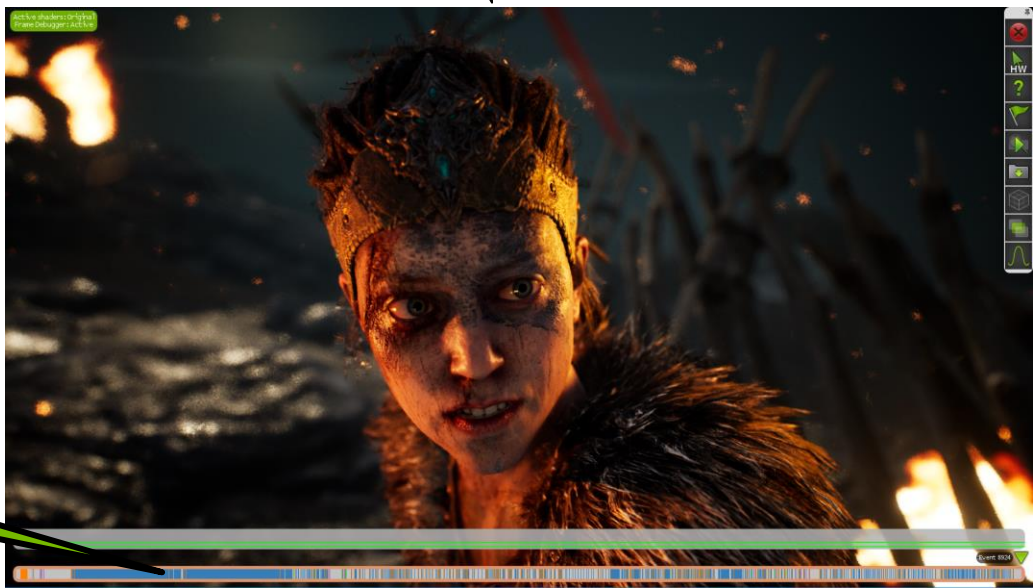
Create C++ Capture for standalone exe repros

Analysis of shader unit utilization…coming soon!

# Nsight Graphics Tour

Real time performance metrics

Live capture: return to your application at will

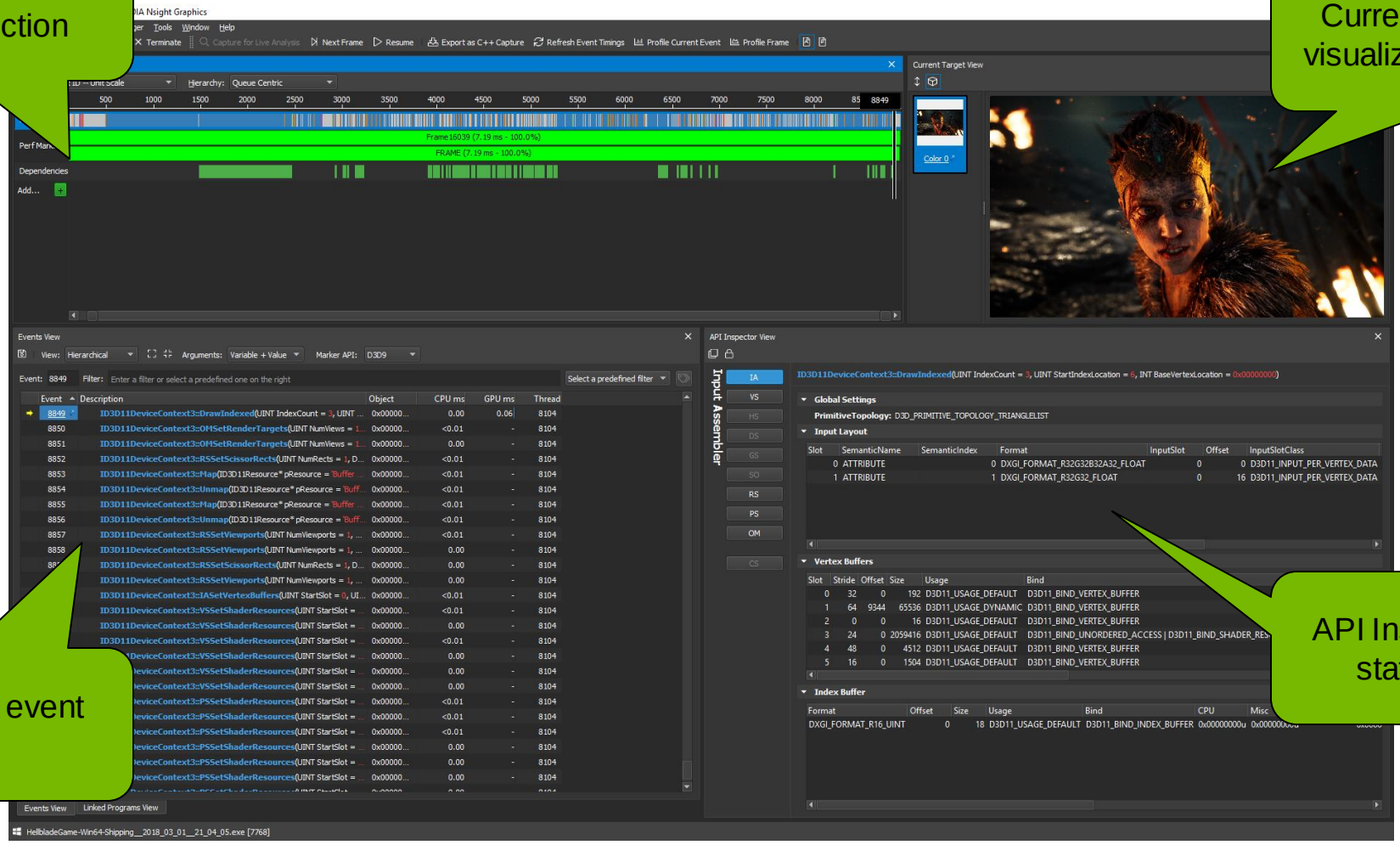


Scrub through the scene, better understand GPU rendering

# Nsight Graphics Tour

Scrubber: break down frame construction

Current Target View: visualize render targets



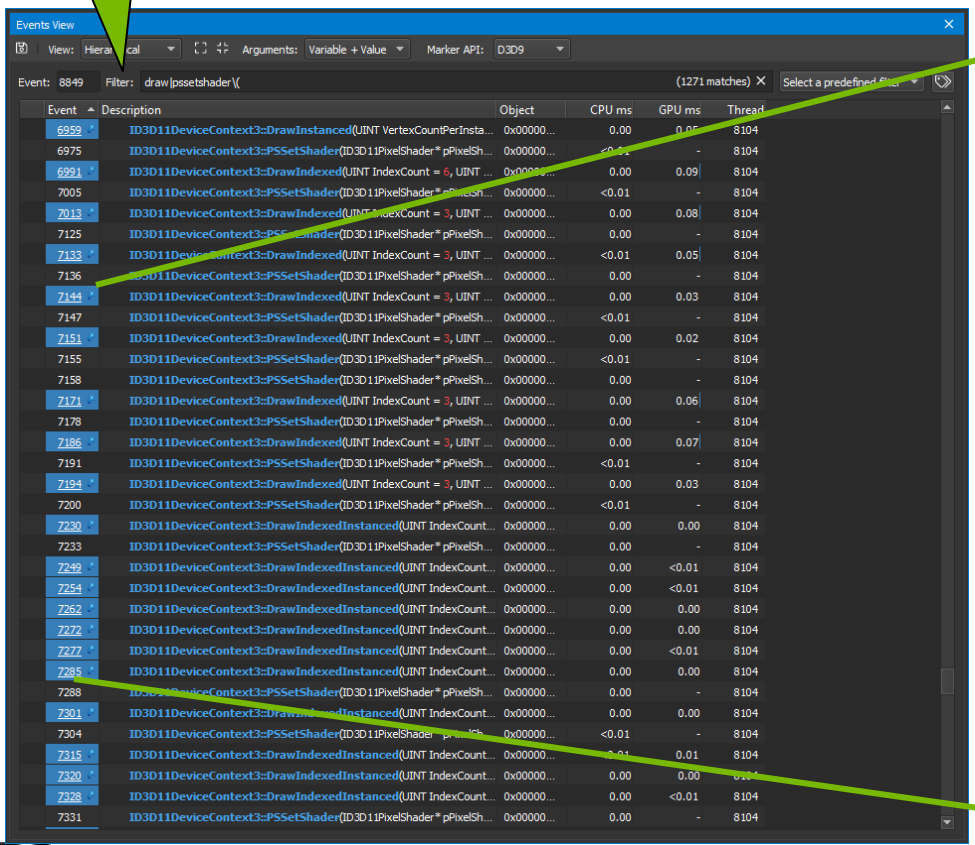

Event List: API event trace

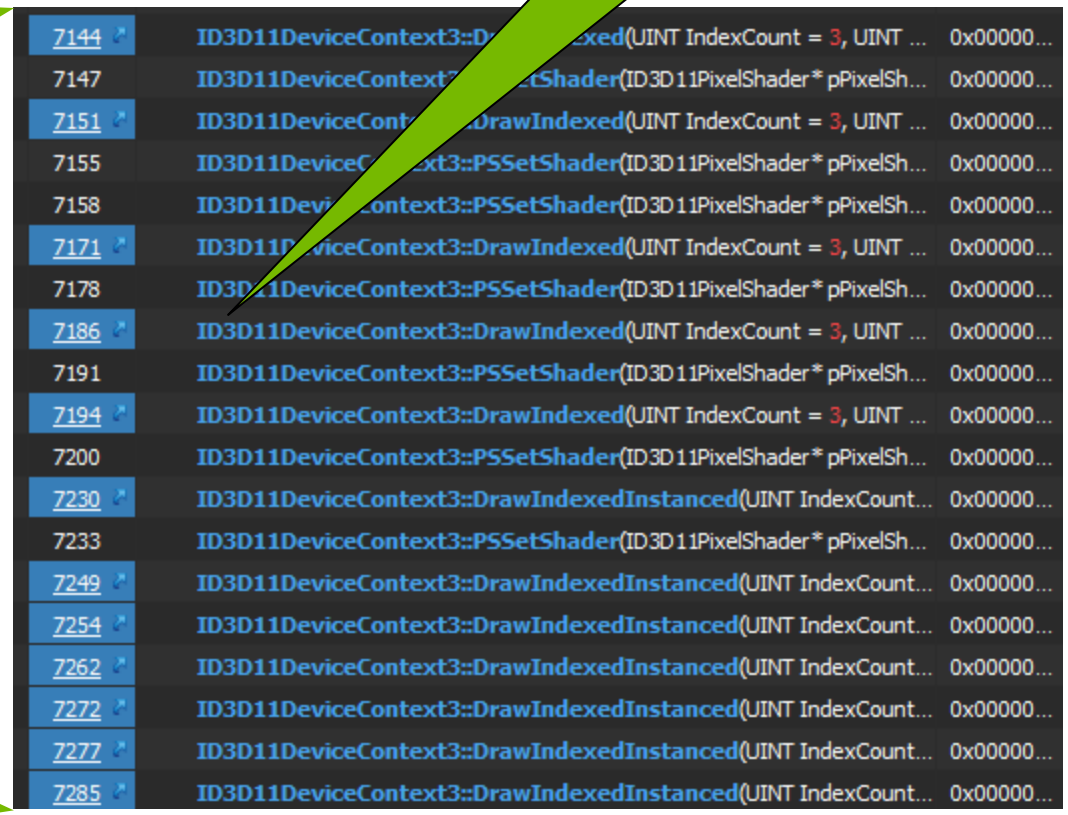
API Inspector: full API state inspection

# Nsight Graphics Tour

Events View: filter to find problematic API usage

Frequent shader changes can impact performance

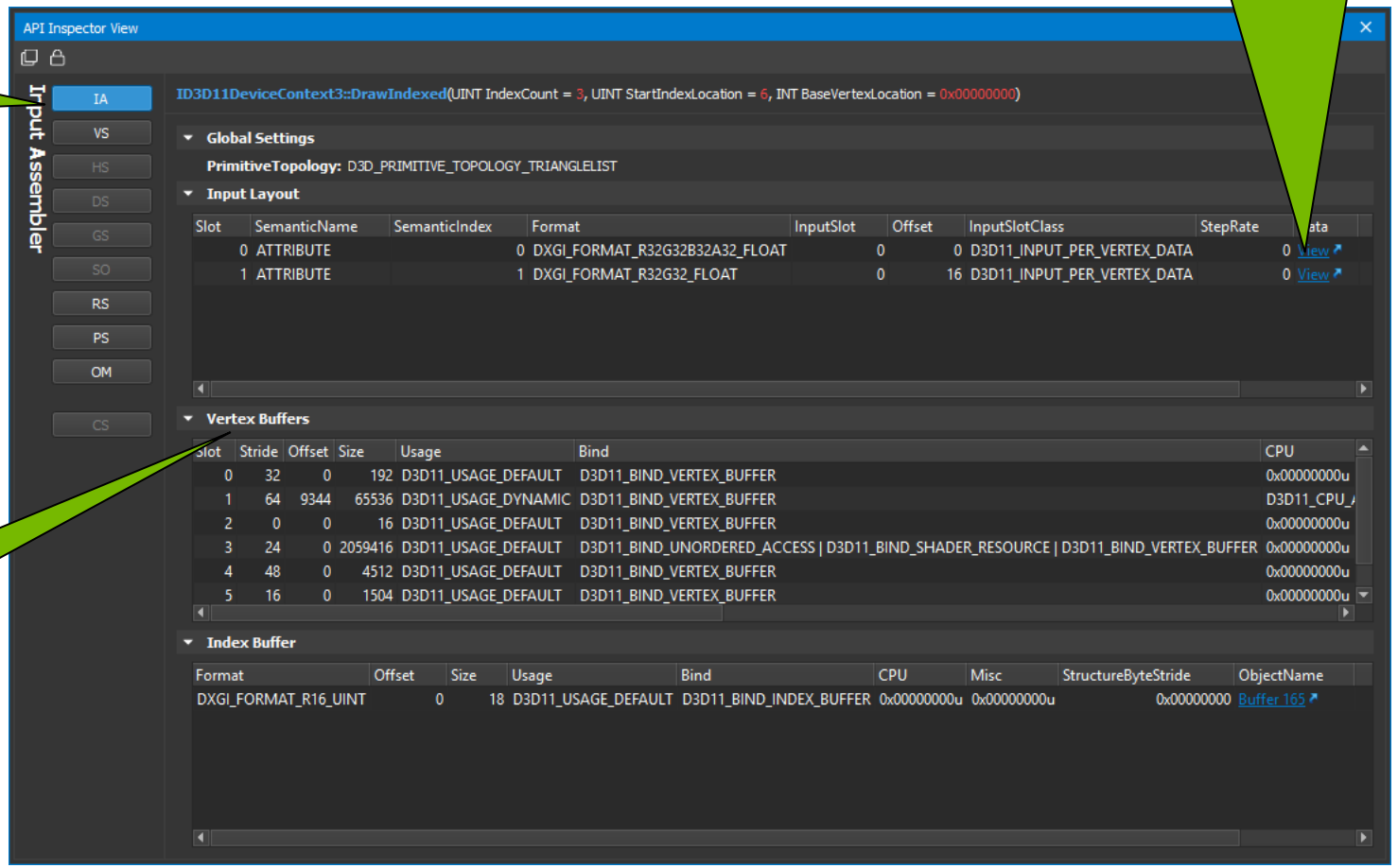


# Nsight Graphics Tour

Hyper-linked UI for easy navigation

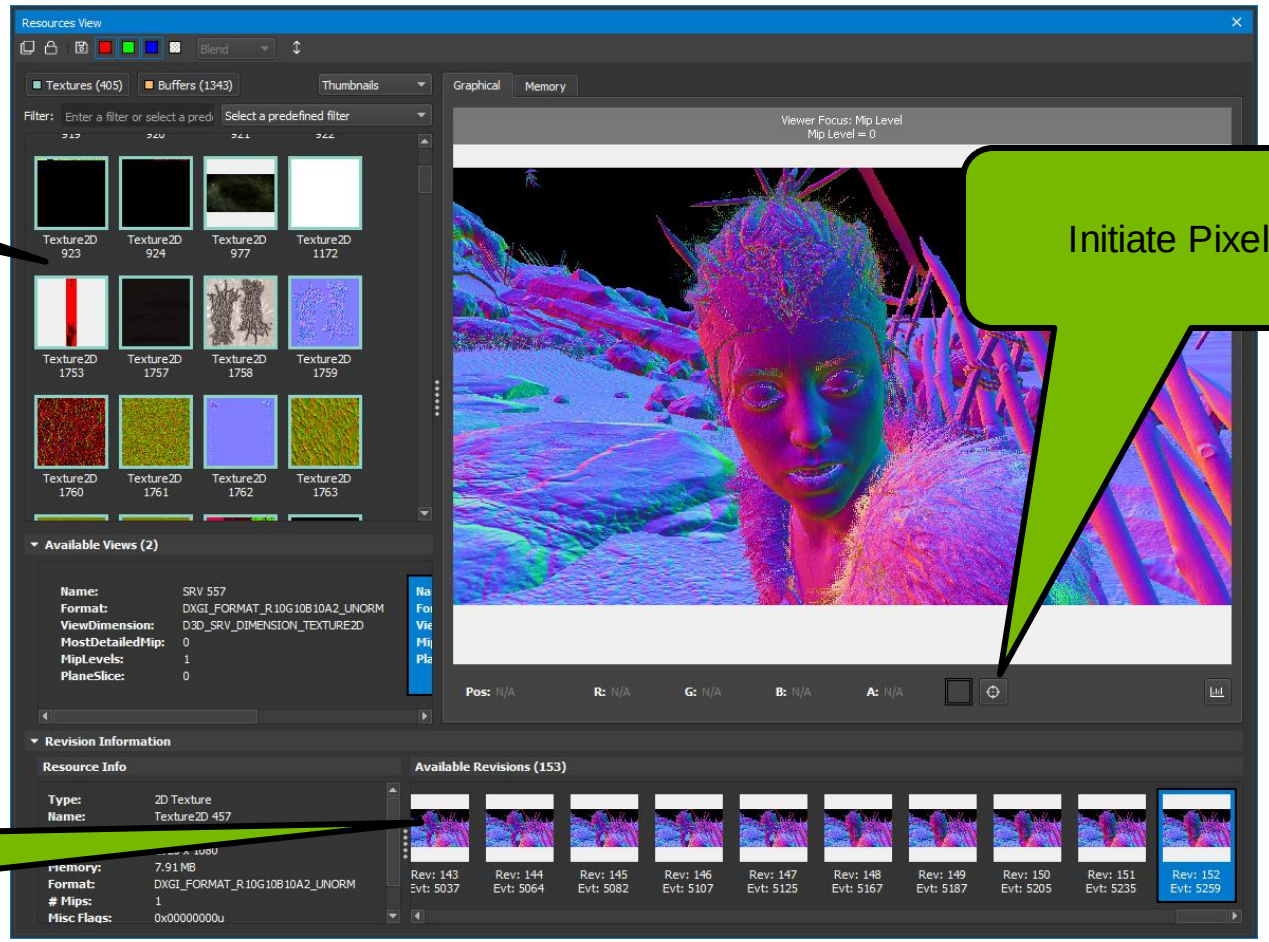
Pipeline Navigator: state categorized by GPU location




State laid out for easy inspection

# Nsight Graphics Tour

Inspect all resource types, views, etc.




Initiate Pixel History!

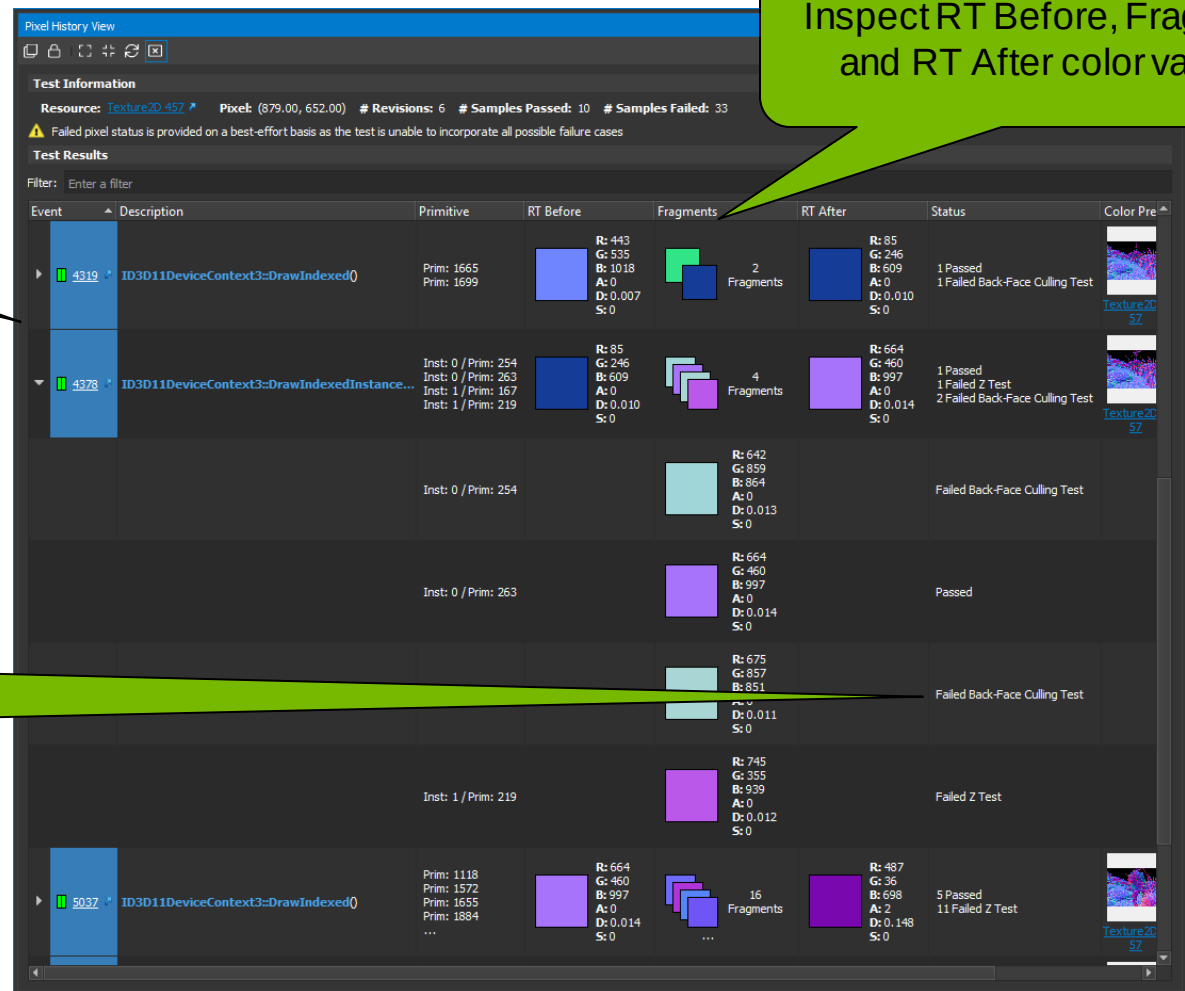
Scrub through buffer revisions

# Nsight Graphics Tour

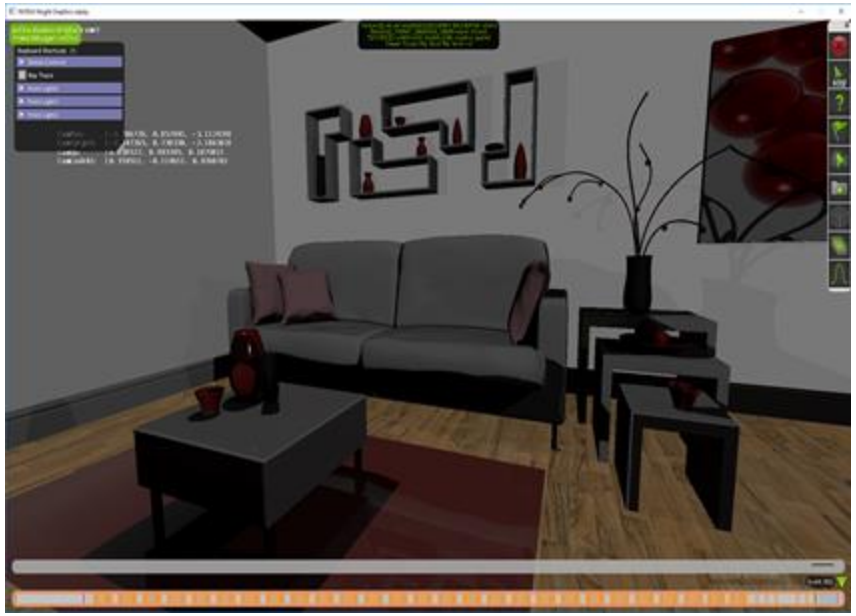
Show all fragments that potentially touched a pixel location

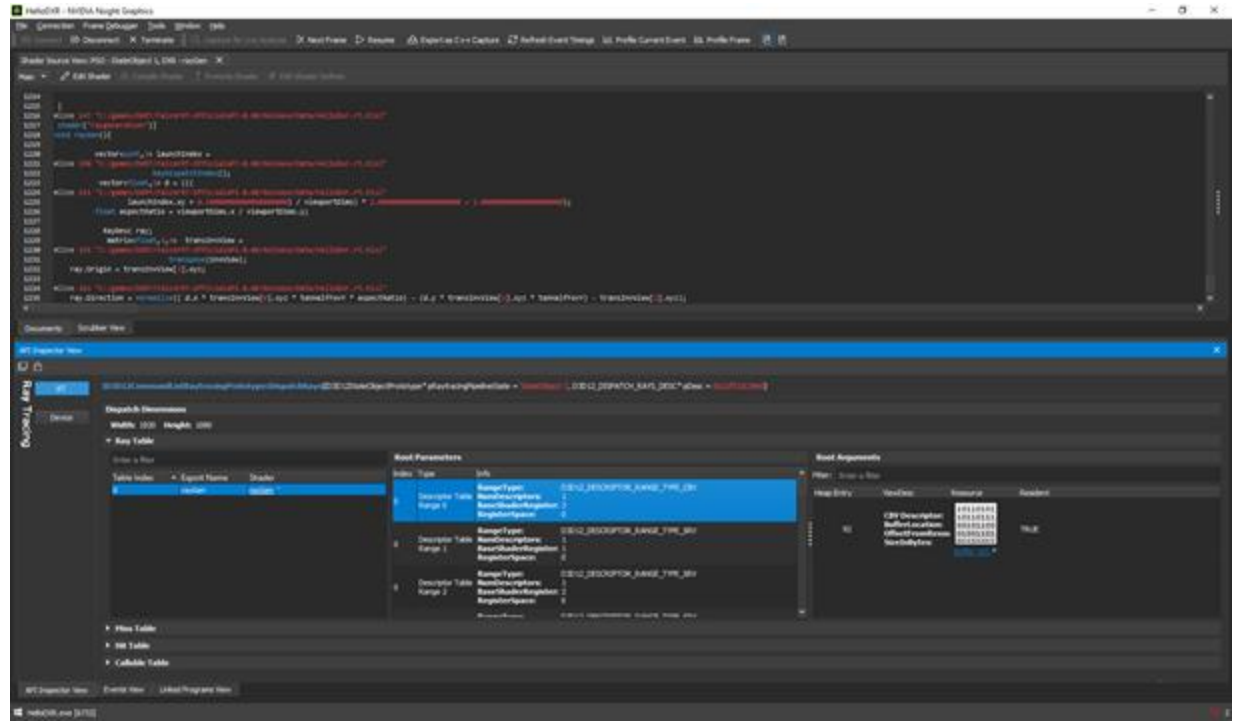
Inspect RT Before, Fragment, and RT After color values

Understand why fragments DON'T land in the frame buffer, like back facing or Z-culled



# DXR Debugging

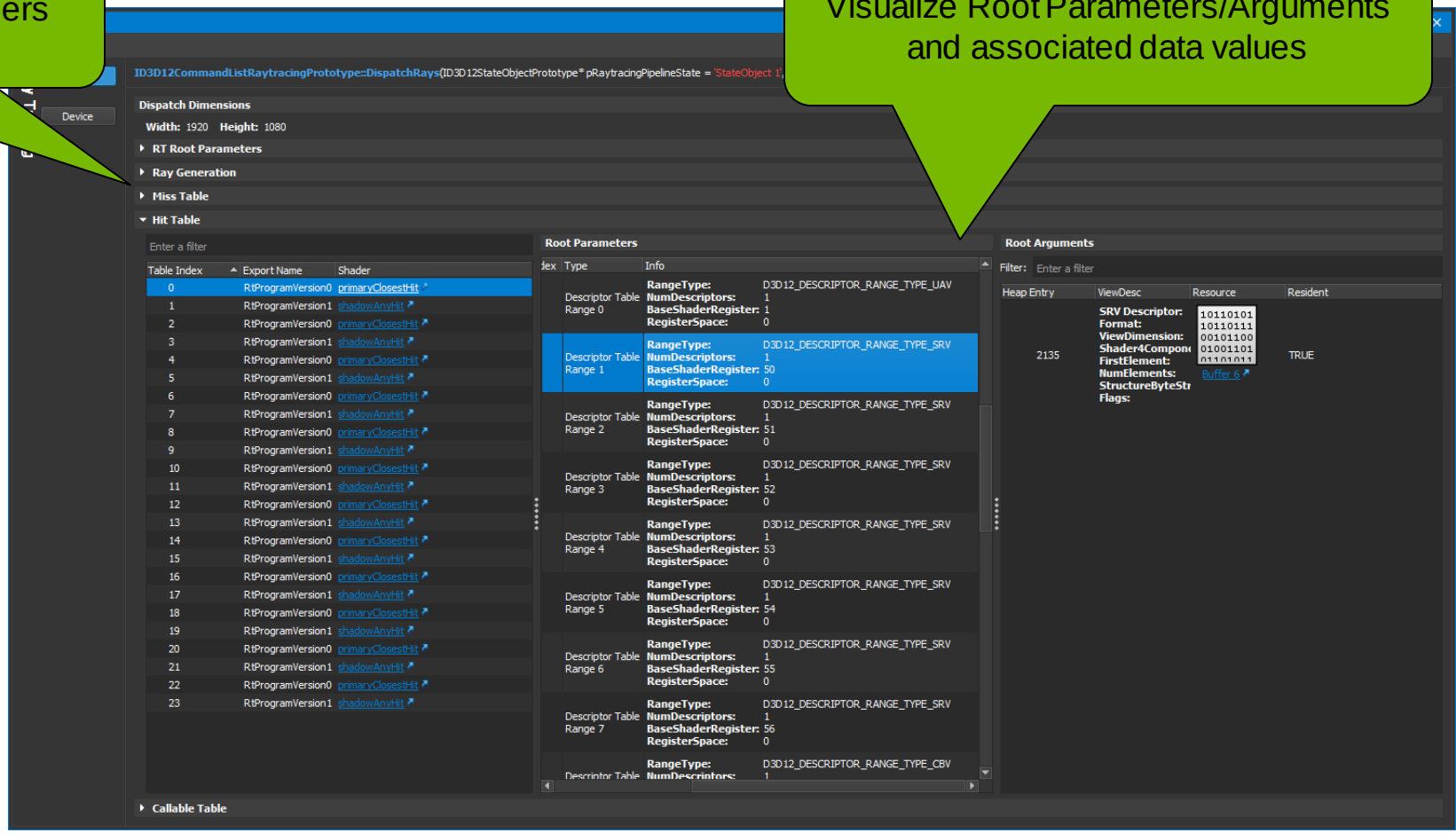




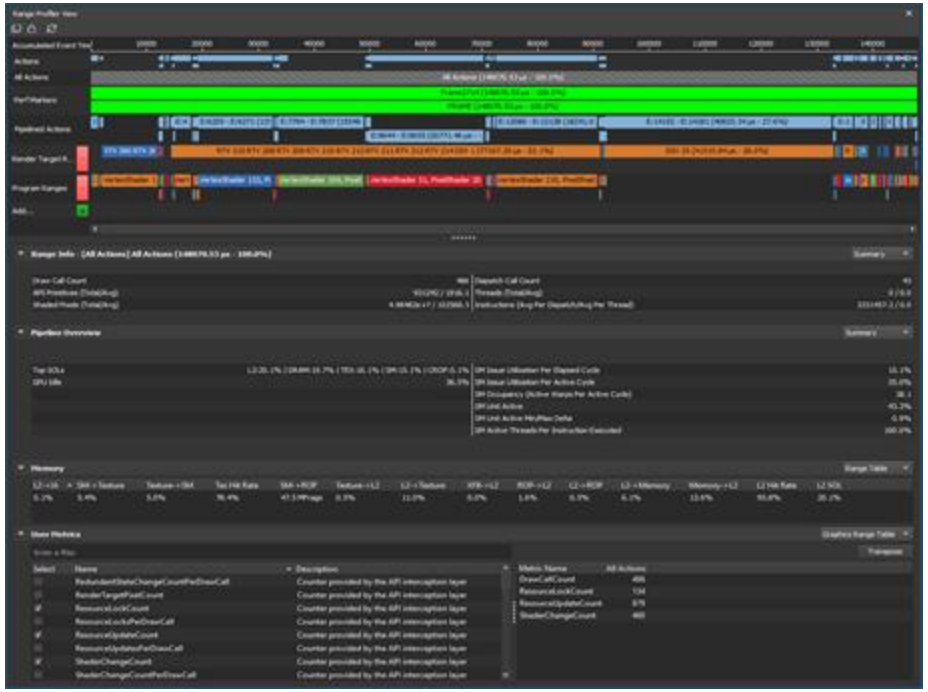
# DXR Debugging

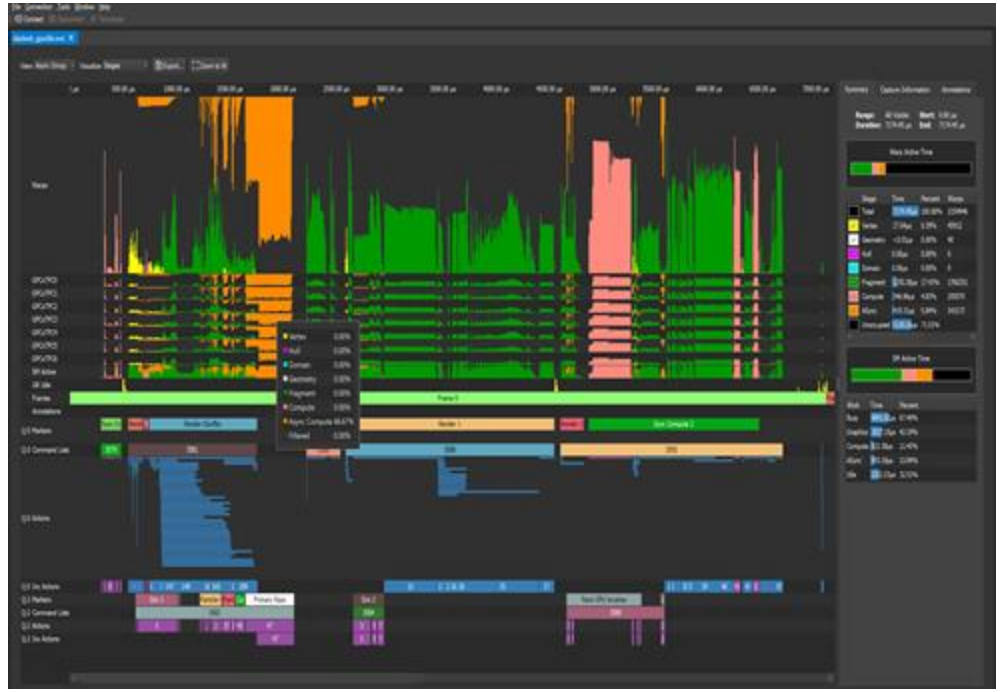
Inspect bound Ray/Miss/Hit Shaders

Visualize Root Parameters/Arguments and associated data values



# Demo 3 - Profiling



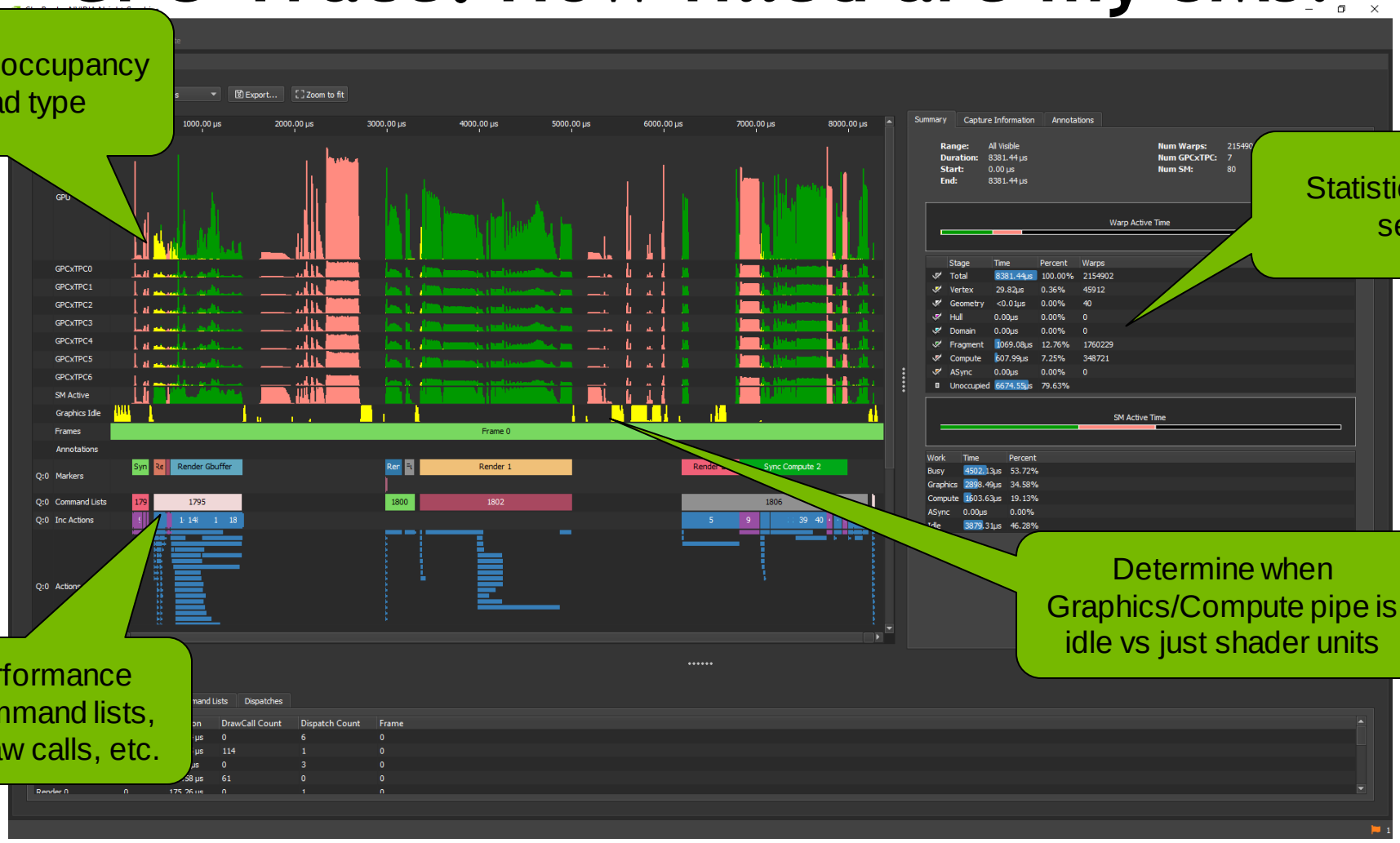


# “The Peak-Performance Analysis Method for Optimizing Any GPU Workload”

Louis Bavoil https://devblogs.nvidia.com/

# GPU Trace: How filled are my SMs?

Visualize warp occupancy by workload type

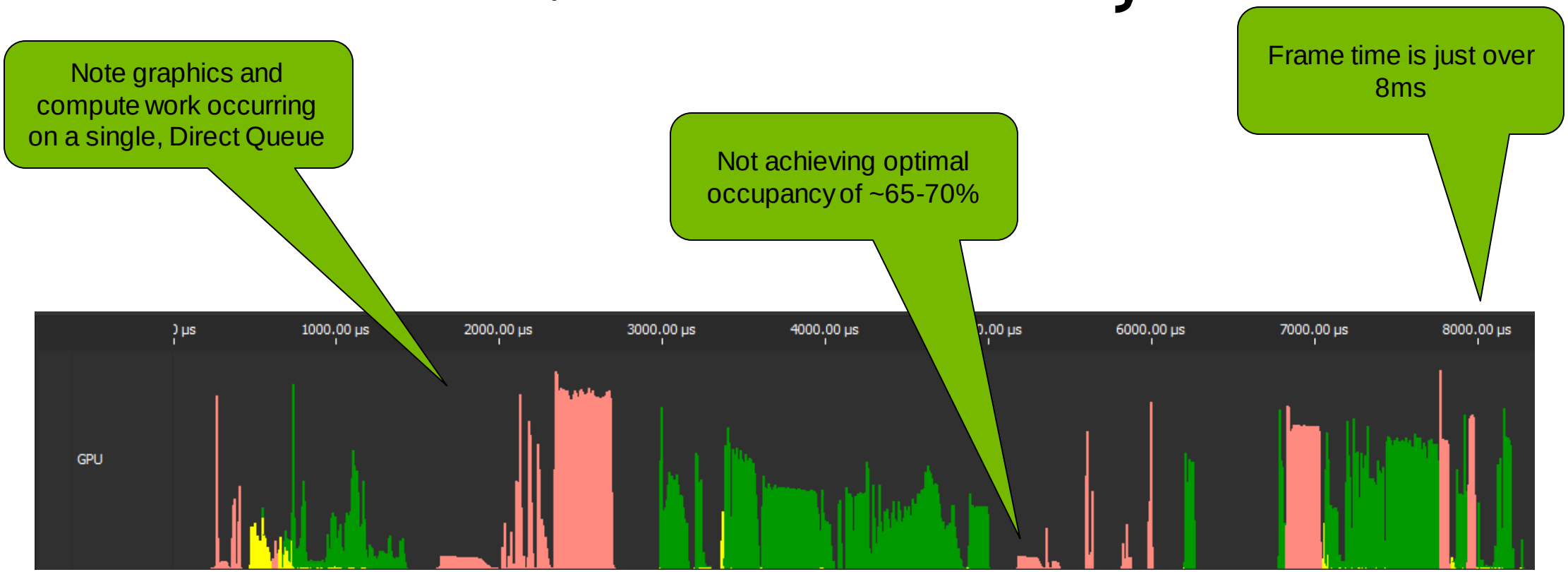



Statistics for current selection

Determine when Graphics/Compute pipe is idle vs just shader units

Display performance markers, command lists, individual draw calls, etc.

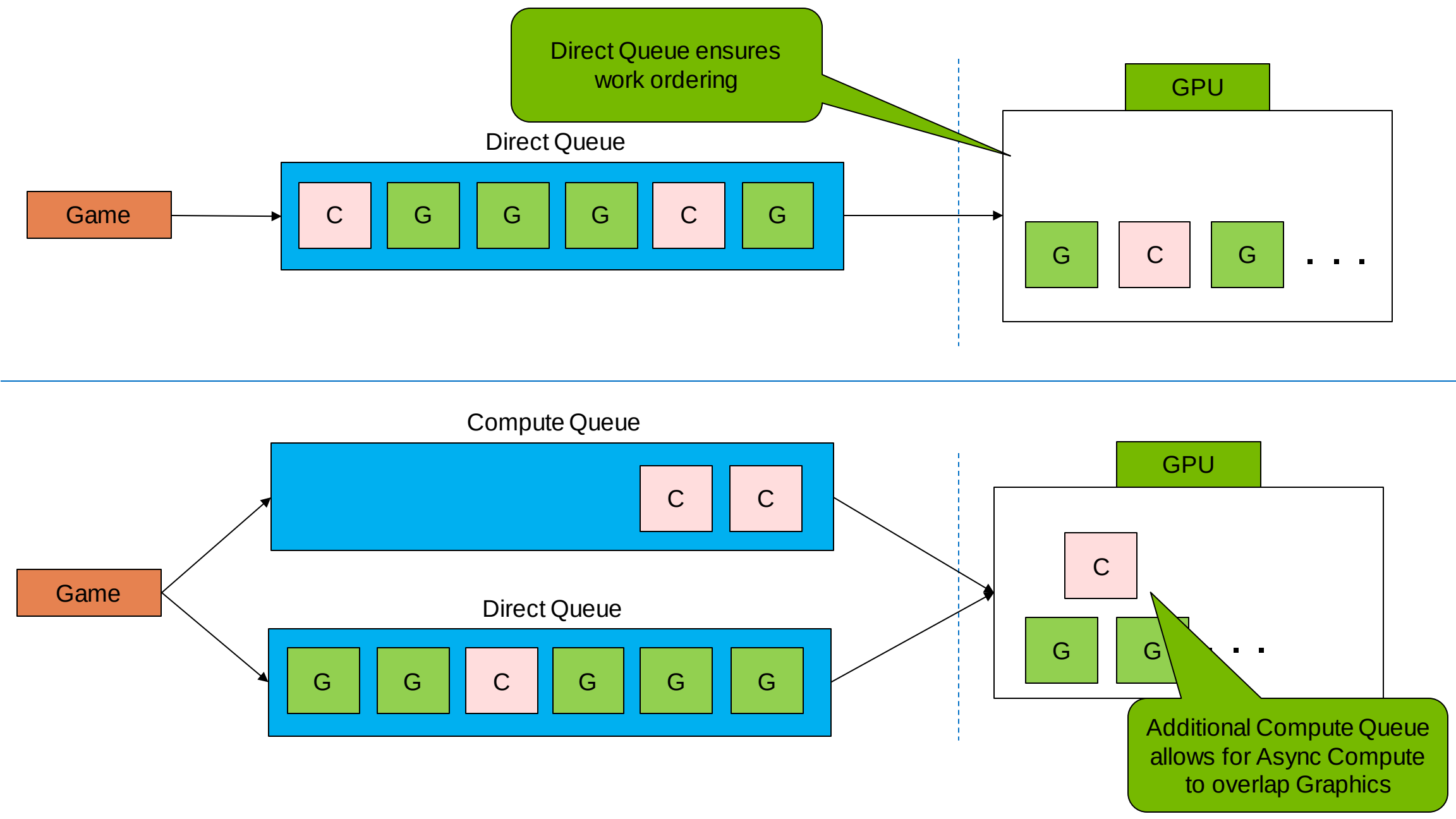
# GPU Trace: How filled are my SMs?



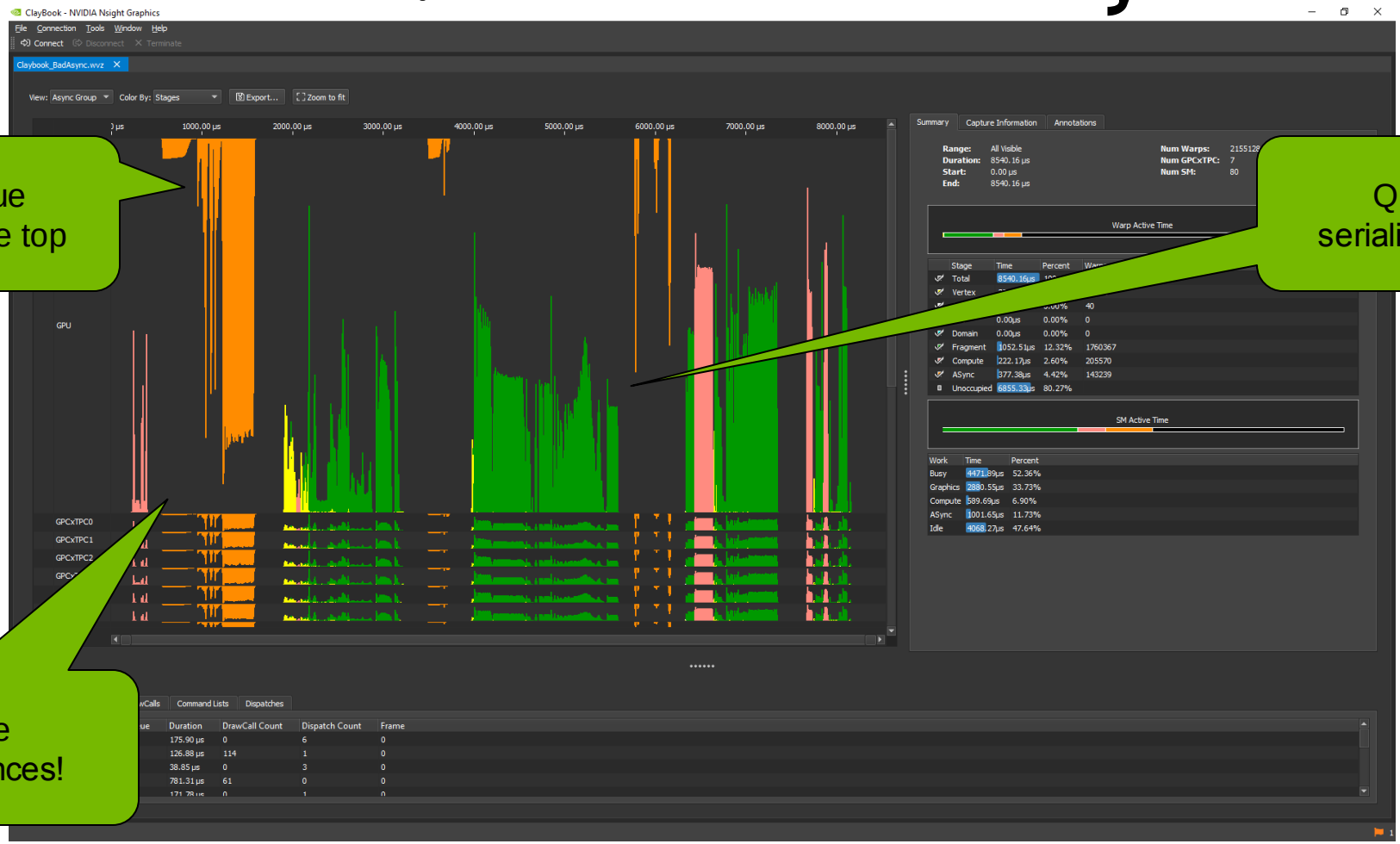

# Solution? Use a Compute Queue

# Construct Compute Queue

```
#include <d3d12.h>

//////////////////////////////////////////////////////////////////////////////
//
// Example of creating a compute command queue, needed to get any async compute
//
//////////////////////////////////////////////////////////////////////////////
D3D12_COMMAND_QUEUE_DESC computeQueueDesc =
{ D3D12_COMMAND_LIST_TYPE_COMPUTE, // Type
  0, // Priority
  D3D12_COMMAND_QUEUE_FLAG_NONE, // Flags
  0x0  // NodeMask
};

// Create a compute command queue
hResult = pD3DDevice->CreateCommandQueue(&computeQueueDesc, __uuidof(ID3D12CommandQueue), (void**)&pD3DComputeCommandQueue);

// Create a compute command allocator
hResult = pD3DDevice>CreateCommandAllocator(D3D12_COMMAND_LIST_TYPE_COMPUTE, __uuidof(ID3D12CommandAllocator),
(void**)&pComputeCommandAllocator);

// Create a compute command list
hResult = pD3DDevice>CreateCommandList(0x1, D3D12_COMMAND_LIST_TYPE_COMPUTE, pComputeCommandAllocator, NULL,
__uuidof(ID3D12GraphicsCommandList), (void**)&pComputeCommandList);
```


Direct Queue ensures work ordering
GPU
Direct Queue
Game
C
G
G
G
C
G
G
C
G
. . .
Compute Queue
C
C
Game
C
Direct Queue
G
G
C
G
G
G
G
G
. . .
Additional Compute Queue allows for Async Compute to overlap Graphics

# GPU Trace: How filled are my SMs?

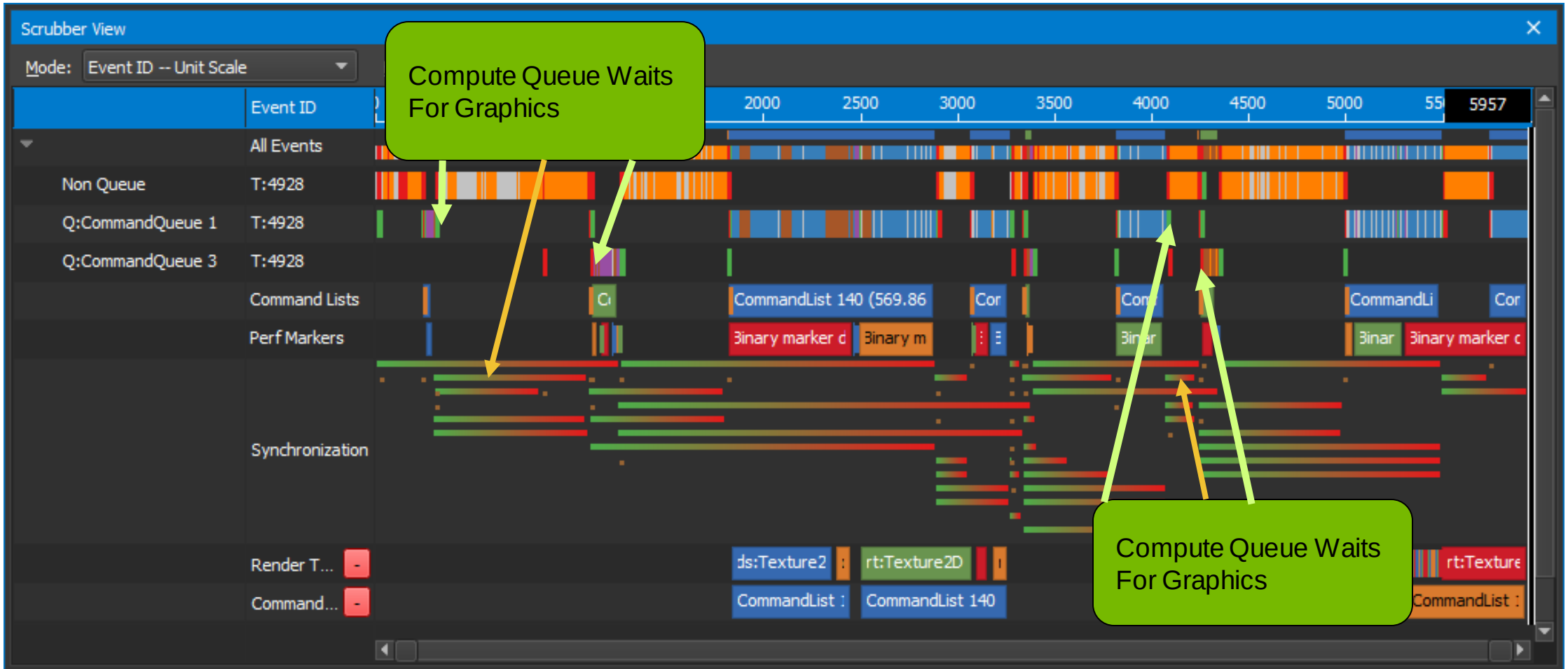



Compute Queue visualizes from the top

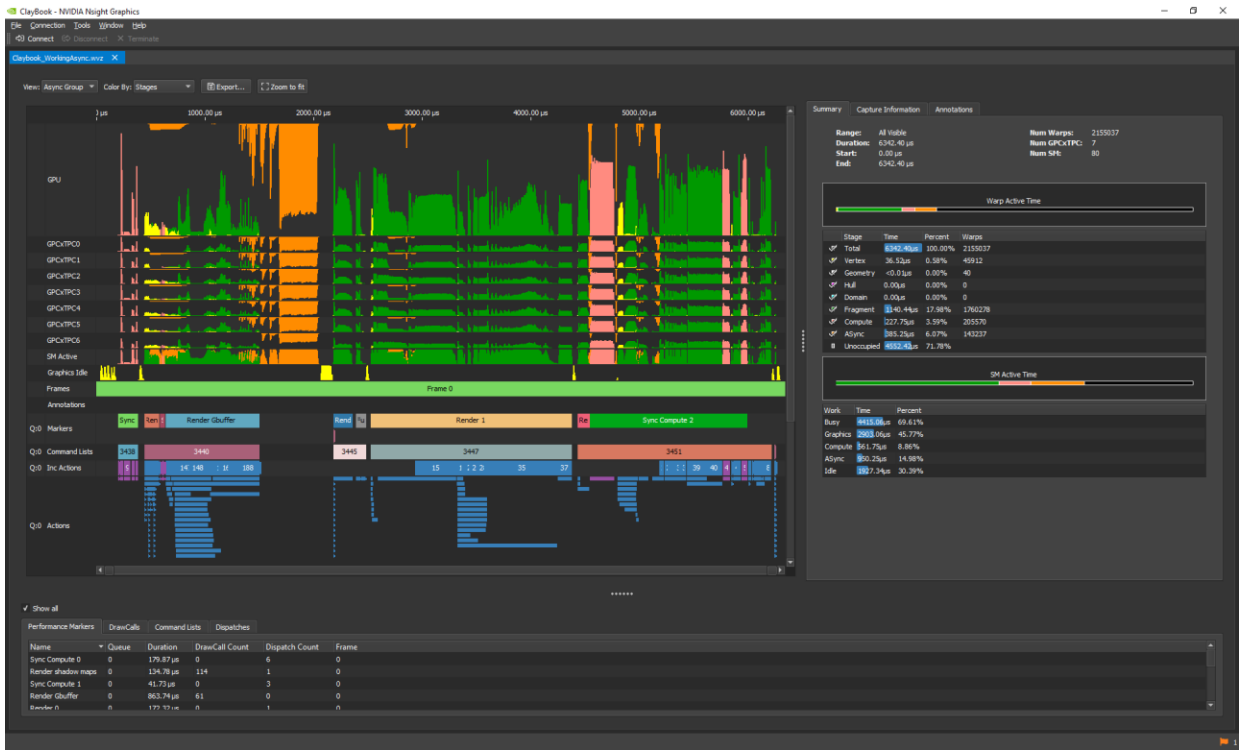
Queues are serialized…fences!

Queues are serialized…fences!

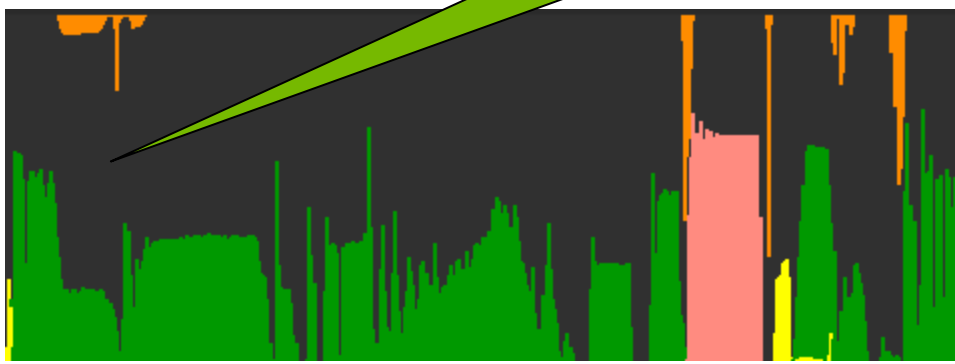
# No (Real) Async: Debug with Nsight Graphics

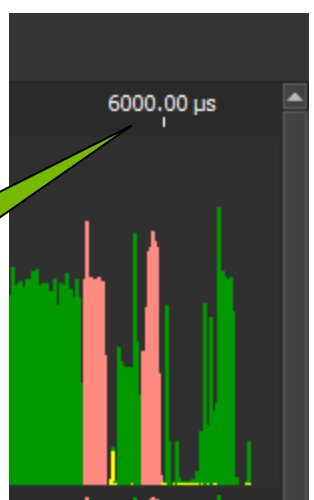


# GPU Trace: Much Better!



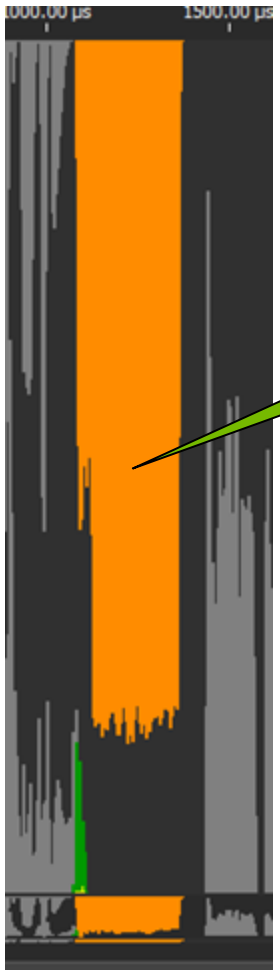

Graphics & Compute now overlap…better shader unit utilization!





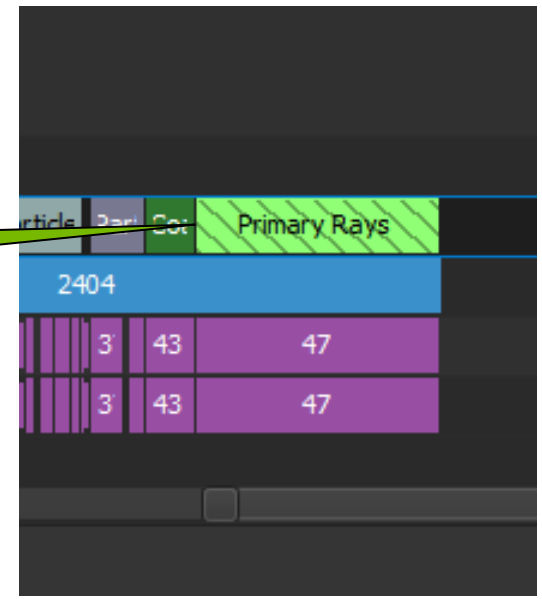

Frame time now just over 6ms…~2ms saved!

# But wait…why is compute not full?

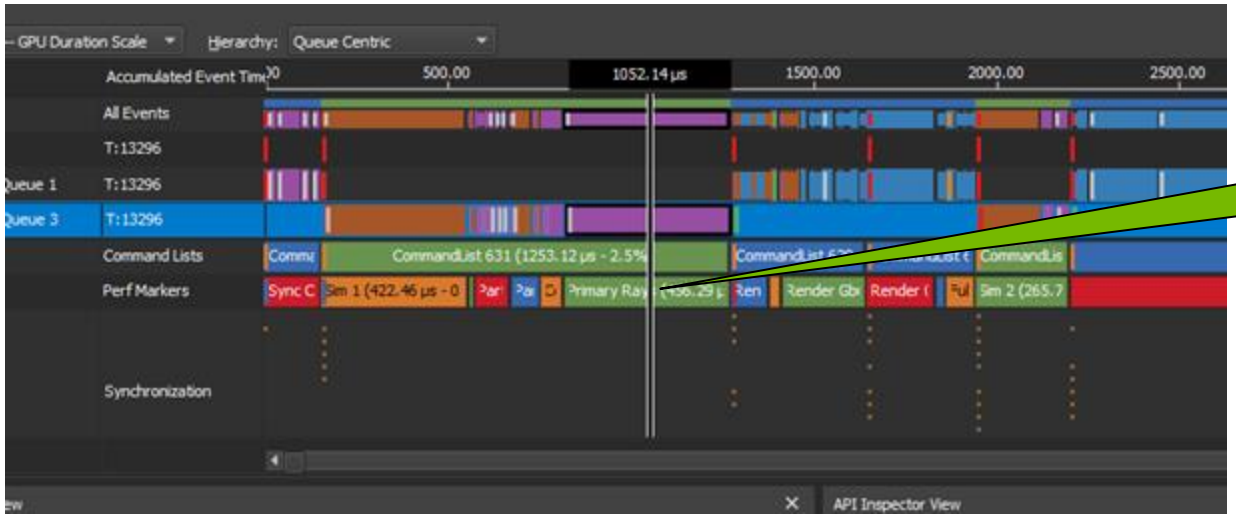


Compute work doesn't "fill" the GPU…why?

Comes from "Primary Rays" Performance Marker Region

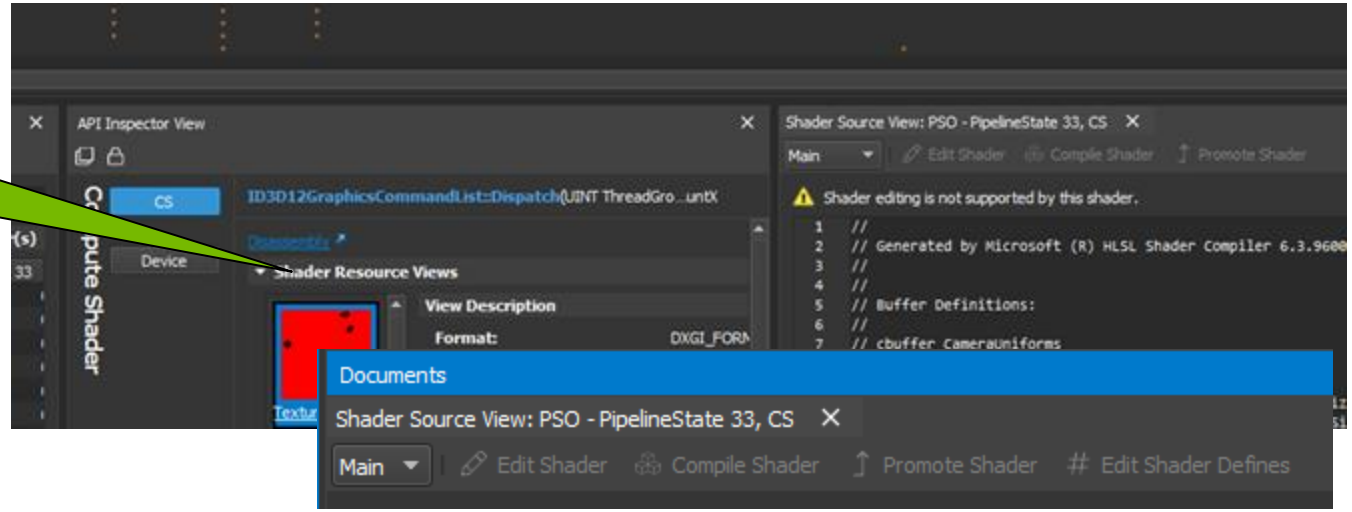


# GPU Not Full: Debug with Nsight Graphics…




Find “Primary Rays” section in Scrubber

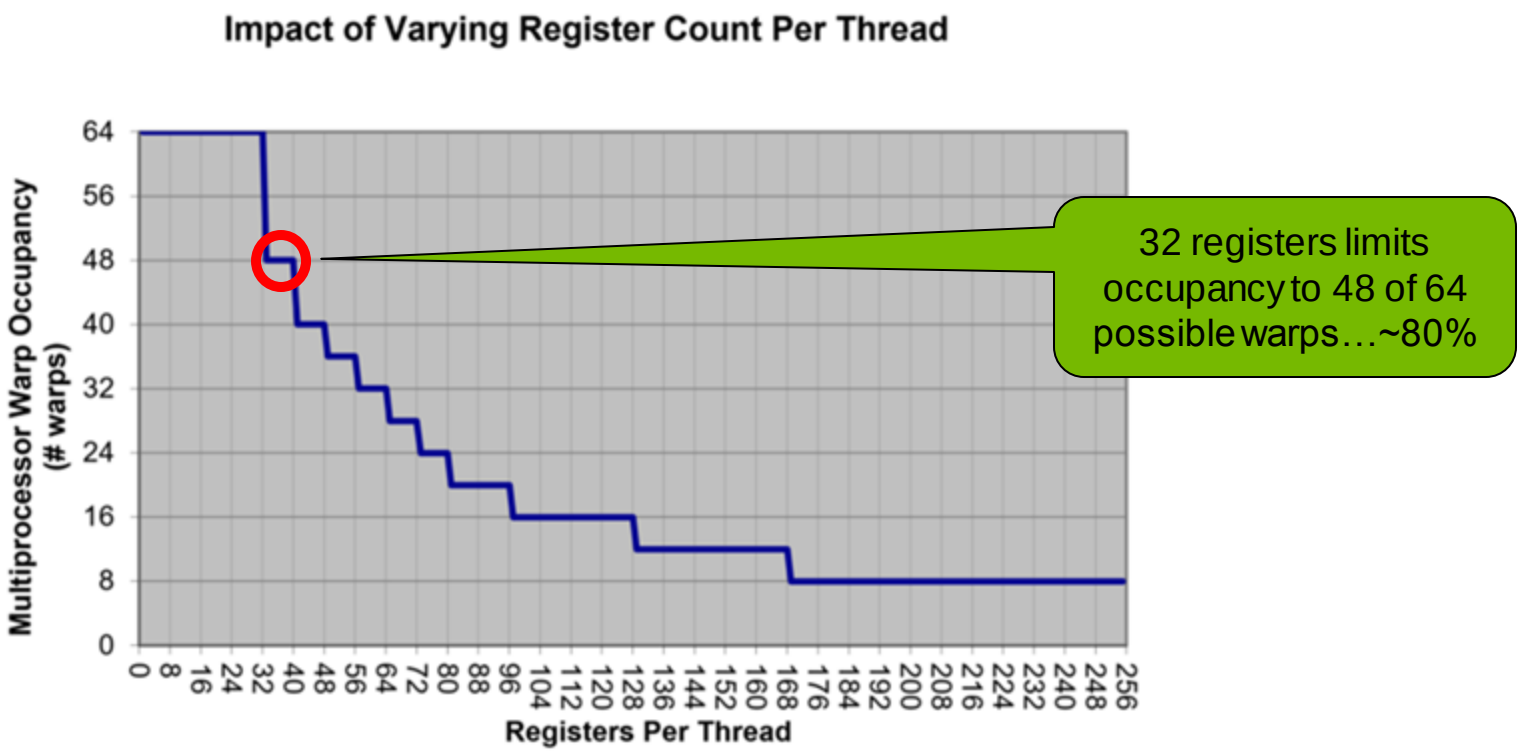
Identify shader in Scrubber/Source View



# GPU Not Full: Debug with Nsight Graphics…

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PipelineState 31 | PSO | Source + µCode | None | 0 | 0 | -- | -- | 32 |
| PipelineState 32 | PSO | Source + µCode | None | 0 | 0 | -- | -- | 24 |
| PipelineState 33 | PSO | Source + µCode | None | 0 | 0 | -- | -- | 32 |
| PipelineState 34 | PSO | Source + µCode | None | 0 | 0 | -- | -- | 40 |

Shader uses 32 registers




32 registers limits occupancy to 48 of 64 possible warps…~80%

# GPU Throughput: SOL

SOL = Speed of Light

< ~10%, unit just not taxed…


Unit not achieving maximum throughput, analyze other metrics to determine why

# GPU Throughput: SOL

SOL = Speed of Light


Grey area: can try to make unit more efficient, or could try and reduce work

# GPU Throughput: SOL

SOL = Speed of Light


Unit is efficient but near max throughput, will likely need to shift work to other units improve performance

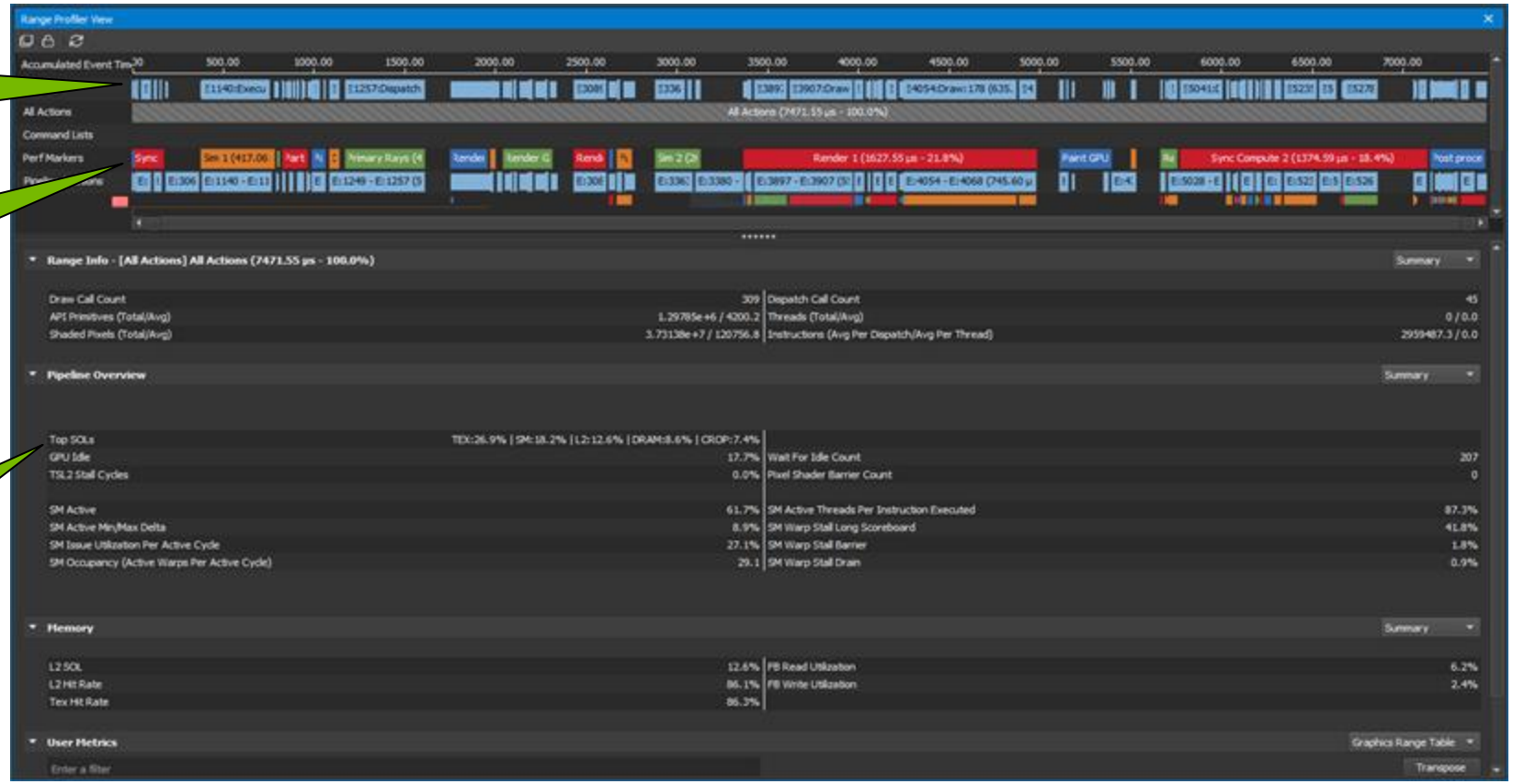
# Understanding GPU Performance

## Range Profiler

GPU time for every draw/dispatch

Elapsed time for performance marker ranges

Pipeline throughput/SOL values

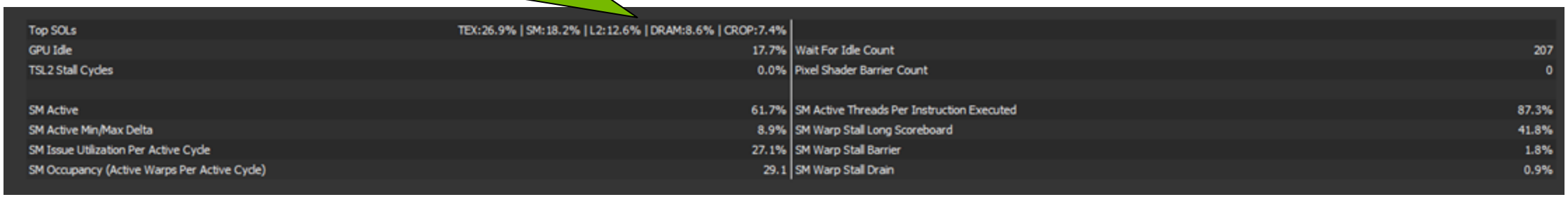


# Understanding GPU Performance

## Range Profiler

Overall frame is mainly texture limited, but better to dig into ranges that have similar workloads

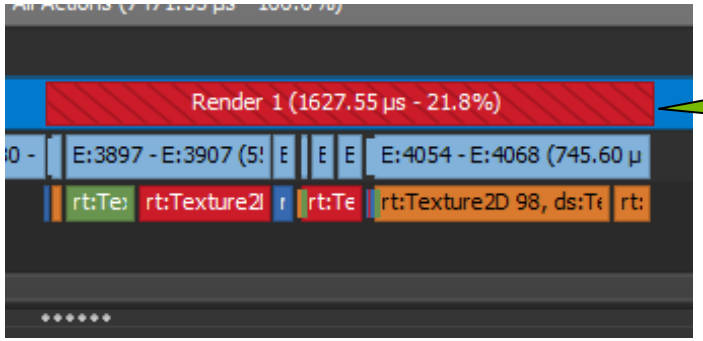
TEX: 26.9% | SM: 18.2% | L2: 12.6% | DRAM: 8.6% | CROP: 7.4%



| | | | |
|---|---|---|---|
| Top SOLs | TEX:26.9% \| SM:18.2% \| L2:12.6% \| DRAM:8.6% \| CROP:7.4% | | |
| GPU Idle | 17.7% | Wait For Idle Count | 207 |
| TSL2 Stall Cycles | 0.0% | Pixel Shader Barrier Count | 0 |
| SM Active | 61.7% | SM Active Threads Per Instruction Executed | 87.3% |
| SM Active Min/Max Delta | 8.9% | SM Warp Stall Long Scoreboard | 41.8% |
| SM Issue Utilization Per Active Cycle | 27.1% | SM Warp Stall Barrier | 1.8% |
| SM Occupancy (Active Warps Per Active Cycle) | 29.1 | SM Warp Stall Drain | 0.9% |

# Understanding GPU Performance

## Range Profiler



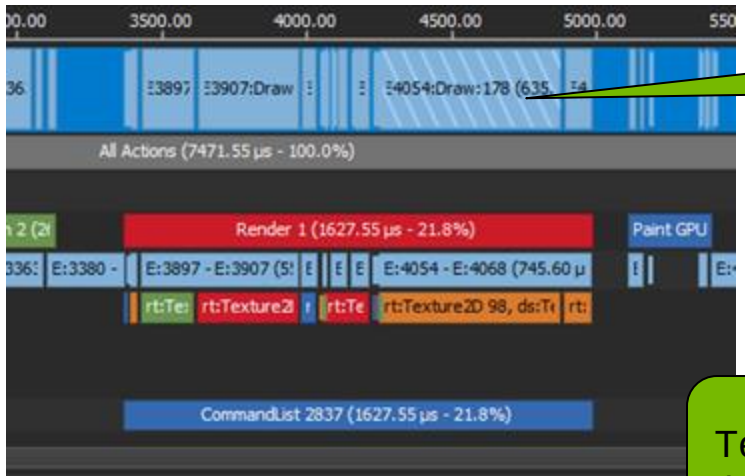

Select largest/most expensive marker range in scene…

This range is mainly texture and shader unit limited…but dig a little deeper…

TEX: 45.2% | SM: 29.5% | L2: 23.1% | DRAM: 8.4% | CROP: 6.0%

| | | | |
|---|---|---|---|
| Top SOLs | TEX:45.2% \| SM:29.5% \| L2:23.1% \| DRAM:8.4% \| CROP:6.0% | | |
| GPU Idle | 11.7% | Wait For Idle Count | 9 |
| TSL2 Stall Cycles | 0.0% | Pixel Shader Barrier Count | 0 |
| SM Active | 74.9% | SM Active Threads Per Instruction Executed | 89.4% |
| SM Active Min/Max Delta | 11.8% | SM Warp Stall Long Scoreboard | 33.4% |
| SM Issue Utilization Per Active Cycle | 33.4% | SM Warp Stall Barrier | 0.0% |
| SM Occupancy (Active Warps Per Active Cycle) | 25.4 | SM Warp Stall Drain | 0.0% |

# Understanding GPU Performance




G-Buffer resolve…

Texture & Shader throughput low…

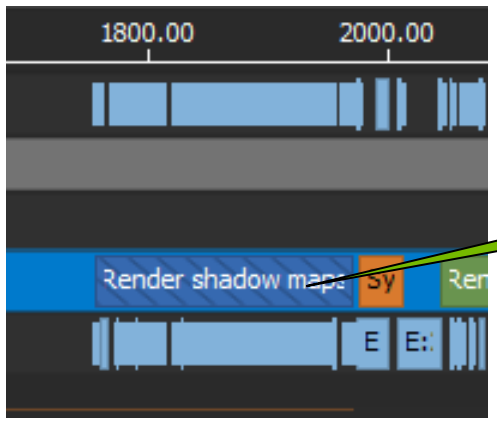
SM Active Threads Per Instruction Executed = 73.5%
SM Warp Stall Long Scoreboard = 51.6%

Shader unit waiting for texture fetches to return

| | | | |
|---|---|---|---|
| Top SOLs | TEX:47.6% \| SM:25.1% \| L2:2.4% \| CROP:2.2% \| DRAM:2.0% | | |
| GPU Idle | 2.8% | Wait For Idle Count | 5 |
| TSL2 Stall Cycles | 0.0% | Pixel Shader Barrier Count | 0 |
| SM Active | 76.5% | SM Active Threads Per Instruction Executed | 73.5% |
| SM Active Min/Max Delta | 24.1% | SM Warp Stall Long Scoreboard | 51.6% |
| SM Issue Utilization Per Active Cycle | 32.8% | SM Warp Stall Barrier | 0.0% |
| SM Occupancy (Active Warps Per Active Cycle) | 22.2 | SM Warp Stall Drain | 0.0% |

SM Active Min/Max Delta @ 24.1% => shader divergence

# Understanding GPU Performance




Shadow Map Rendering,
typically vertex and z limited

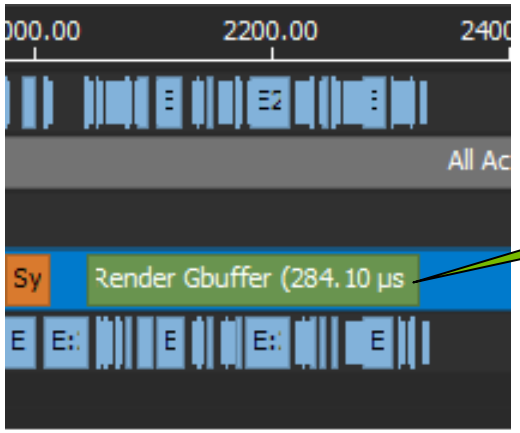
Top SOLs – typical signature for vertex limited work

PD: Primitive Distributor (vertex work)
VPC: Viewpore Cull/Clip (vertex work)
ZROP: Z “blending” (Z buffering)

PD: 61.8% | VPC: 28.9% | ZROP:14.5% | L2:7.8% | DRAM: 6.1%

| | | | |
|---|---|---|---|
| Top SOLs | PD:61.8% \| VPC:28.9% \| ZROP:14.5% \| L2:7.8% \| DRAM:6.1% | | |
| GPU Idle | 10.6% | Wait For Idle Count | 8 |
| TSL2 Stall Cycles | 0.0% | Pixel Shader Barrier Count | 0 |
| SM Active | 69.0% | SM Active Threads Per Instruction Executed | 87.4% |
| SM Active Min/Max Delta | 4.5% | SM Warp Stall Long Scoreboard | 20.7% |
| SM Issue Utilization Per Active Cycle | 6.4% | SM Warp Stall Barrier | 0.0% |
| SM Occupancy (Active Warps Per Active Cycle) | 6.8 | SM Warp Stall Drain | 0.4% |

# Understanding GPU Performance




Render Gbuffer typically color blending texture, and shader limited

Top SOLs

CROP/ZROP: Blending unit (blending the 6 “fat” render targets & depth)
TEX: Texture unit (reading in diffuse textures, etc.)
SM: Shader unit (calculating Gbuffer values)

CROP: 43.4% | ZROP: 27.7% | TEX: 24.0% | SM: 19.0% | DRAM: 15.1%

| | | | |
|---|---|---|---|
| Top SOLs | CROP:43.4% \| ZROP:27.7% \| TEX:24.0% \| SM:19.0% \| DRAM:15.1% | | |
| GPU Idle | 4.0% | Wait For Idle Count | 5 |
| TSL2 Stall Cycles | 0.0% | Pixel Shader Barrier Count | 0 |
| SM Active | 64.9% | SM Active Threads Per Instruction Executed | 92.3% |
| SM Active Min/Max Delta | 11.0% | SM Warp Stall Long Scoreboard | 27.0% |
| SM Issue Utilization Per Active Cycle | 23.4% | SM Warp Stall Barrier | 0.0% |
| SM Occupancy (Active Warps Per Active Cycle) | 11.9 | SM Warp Stall Drain | 0.2% |

# Key Takeaways

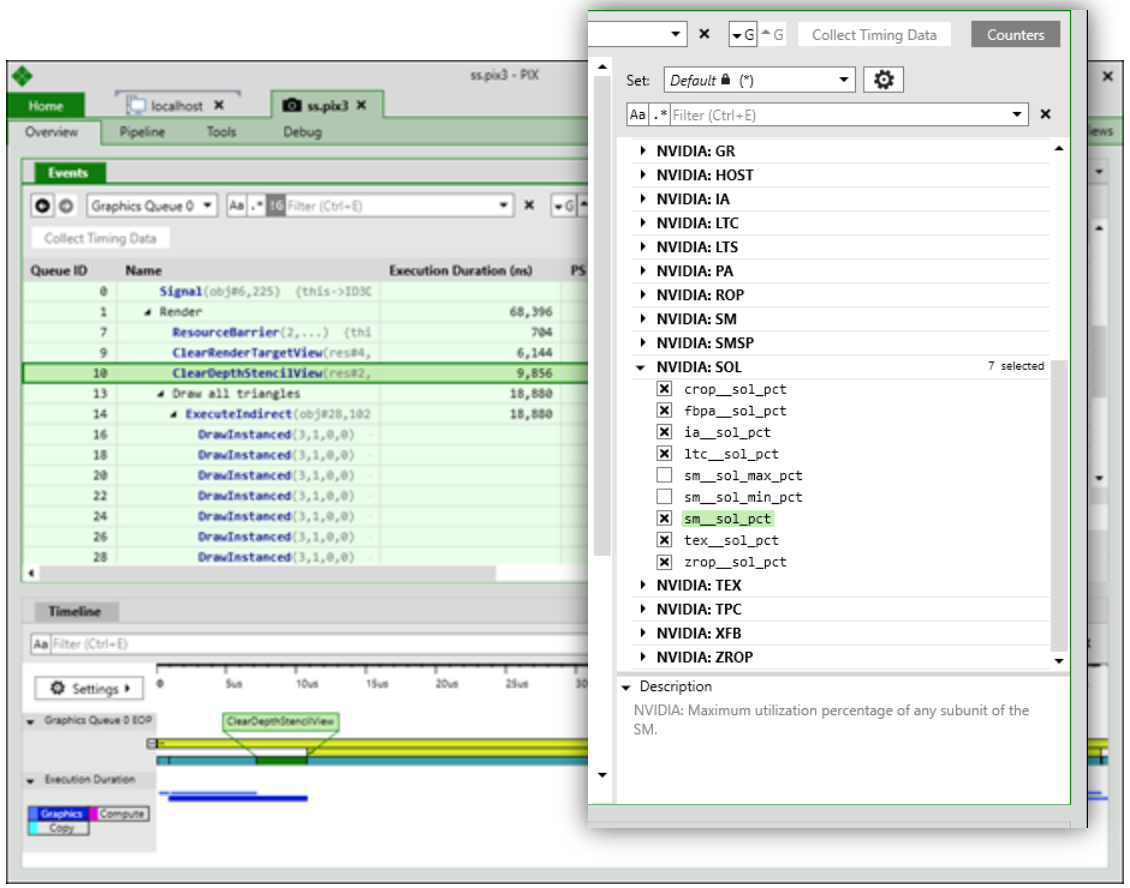
# Summary

- **Nsight Graphics** is *the* next-gen graphics debugger
  - DXR Debugging
  - Range Profiler
  - GPU Trace

- **NVIDIA DevTools** is committed to...
  - increasing release frequency
  - solving developer pain points
  - improving developer productivity

# Microsoft PIX for Windows

- Close collaboration to help make the best tools for our developers
  - Work history buffer i.e. timing activity
  - HW Performance Counters
  - GPU Occupancy Viewer
  - Performance warnings
  - DXR visualization



# The Future of Nsight Graphics

- H1 ‘18
  - Vulkan 1.1 support
  - Linux Support
  - Improvements to Range Profiler
  - GPU Trace 1.0
  - DXR improvements
- H2 ‘18
  - GPU Trace for Vulkan
  - Pixel History (DX12 & Vulkan)
  - DXR Profiling

# What's Next?

- **Visit our Booth** (#223) to check out *Nsight Graphics*, *DXR Debugging* and *GPU Trace*
- **Check out Louis Bavoil's talk** in 30 minutes to learn more about performance triaging with the *Range Profiler*
- **Watch our Videos:** http://j.mp/nvidia-devtools-videos

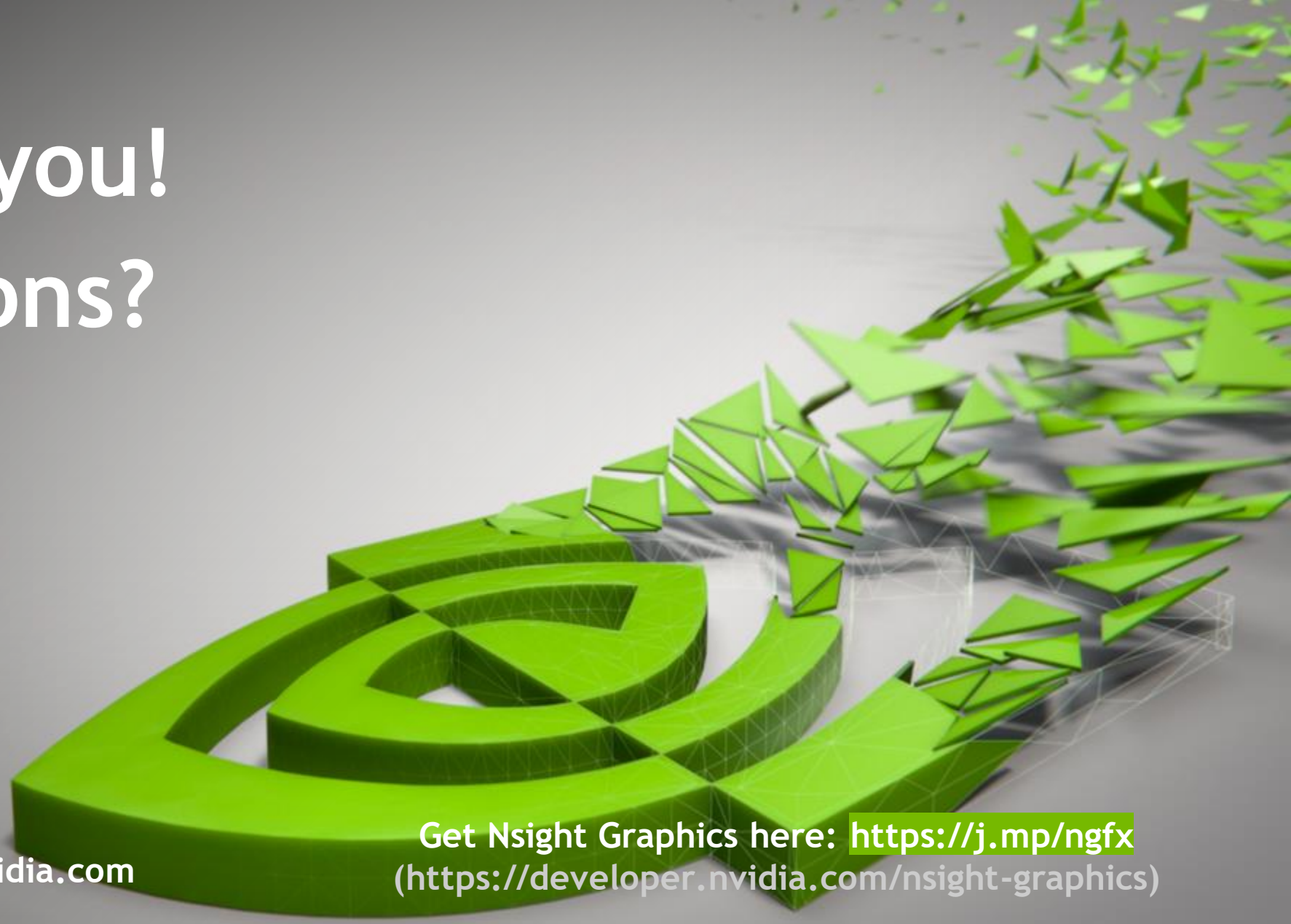
**Join our Early Access Program**
**and get**
**Nsight Graphics here:**
**https://j.mp/ngfx**
(https://developer.nvidia.com/nsight-graphics)

# We're Hiring!

**http://www.nvidia.com/object/careers.html**

# Thank you!
# Questions?


Booth #223 - South Hall
www.nvidia.com/GDC



Need to contact us?
**NsightGraphics@nvidia.com**

**Get Nsight Graphics here: https://j.mp/ngfx**
**(https://developer.nvidia.com/nsight-graphics)**

# NVIDIA Sessions

| Day | Time | Room | Speaker | Title |
|---|---|---|---|---|
| Thurs Mar 21 | 17:30-18:30 | 3022 | Louis Bavoil<br>Principal Engineer | Fixing the Hyperdrive - Maximizing Rendering Performance on NVIDIA GPUs |
| Fri Mar 23 | 10:00-11:00 | 3022 | Nuno Subtil<br>Senior Developer Technology Engineer | NVIDIA Vulkan Update |
| Fri Mar 23 | 11:30-12:00 | 3001, 3003 | Alex Dunn<br>Senior Developer Technology Engineer | Aftermath – Advances in GPU Crash Debugging |
| Fri Mar 23 | 12:15-13:15 | 3022 | Bryan Dudash<br>Senior Manager Developer Technology | Capture Amazing Content with NVIDIA Ansel Photo Mode and Highlights Video Capture Tool |
| Fri Mar 23 | 13:30-14:30 | 3022 | Evan Hart, Principal Engineer | Advances in the HDR Eco-System |
| Fri Mar 23 | 15:00-16:00 | 3022 | Cem Cebenoyan<br>Director of Engineering | Accelerating your VR Games with VRWorks |

Need to contact us?
**NsightGraphics@nvidia.com**

**Get Nsight Graphics here: https://j.mp/ngfx**
**(https://developer.nvidia.com/nsight-graphics)**