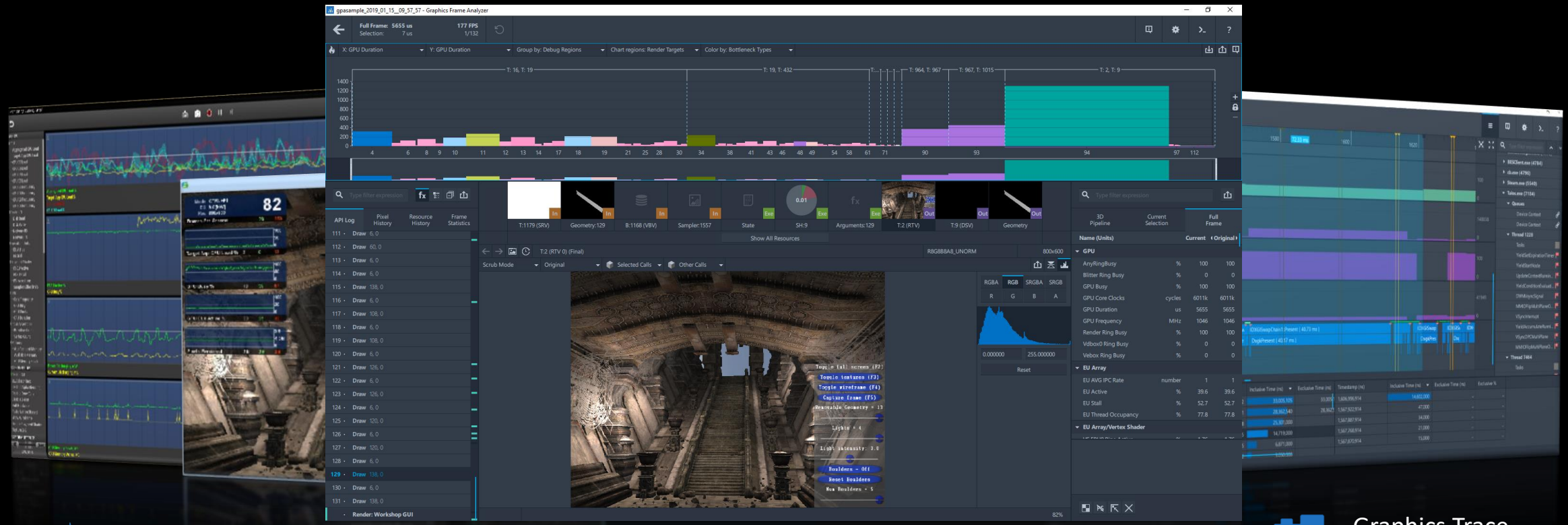
Does your game's performance spark joy? Profiling with Intel® GPA

Carlos Dominguez (Intel®)
Stanislav Volkov (Intel®)

Agenda

- Introduction:
  - Intel® GPA
  - Basic profiling workflow
- Profiling Workflows
  - Cross-tool Performance Profiling
  - Multiframe Analysis
  - Automated Performance Reporting
- Summary

Intel® Graphics Performance Analyzers
(Intel® GPA)



System Analyzer

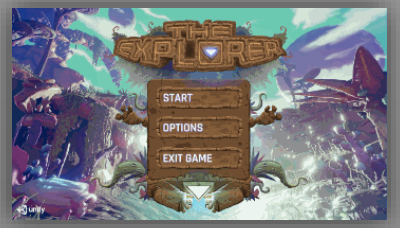
Graphics Frame Analyzer

Graphics Trace Analyzer

Intel® GPA Framework

Basic profiling workflow

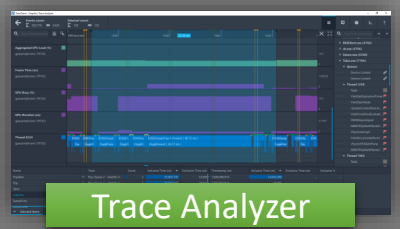
Identify Scene



+

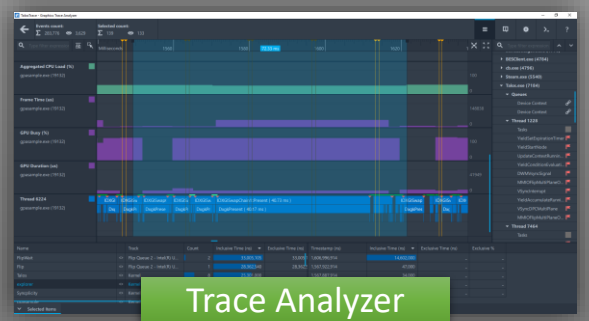
CPU vs GPU bound?

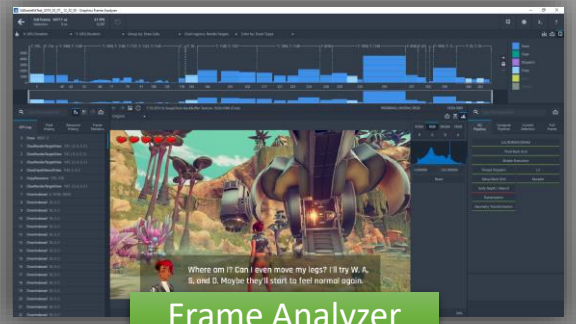
+



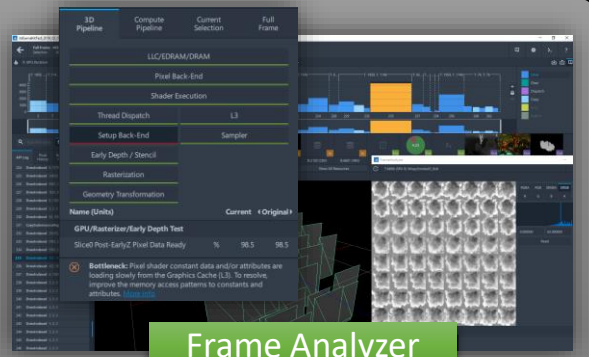
CPU bound

Identify Hotspots via instrumentation





Frame Analysis



Identify Hotspots

GPU bound

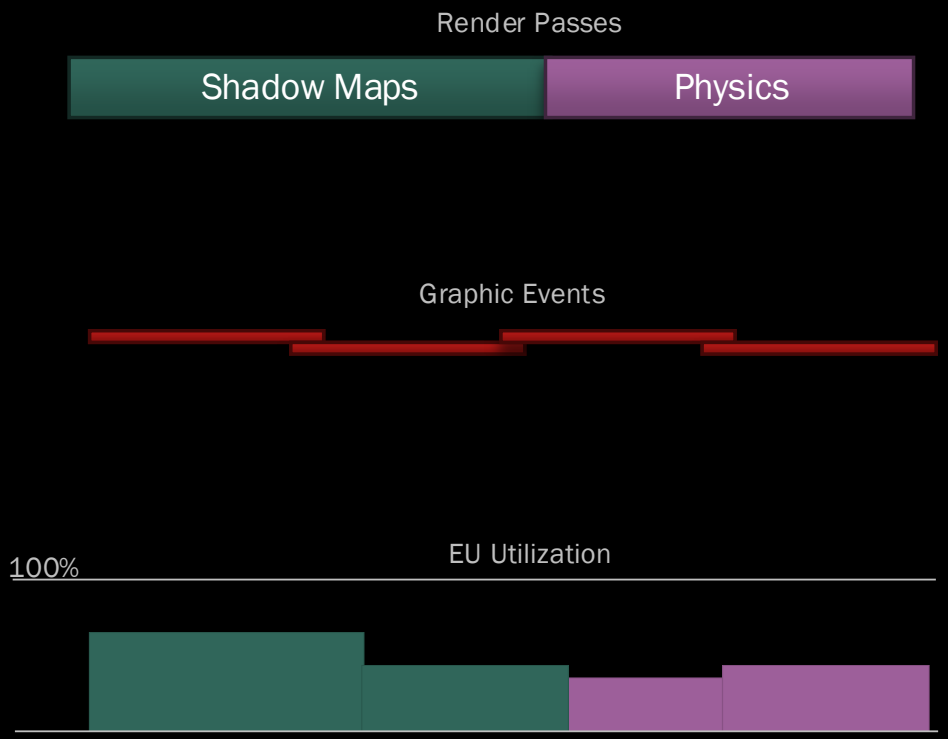
https://software.intel.com/en-us/videos/how-to-fix-performance-woes

Cross-tool Performance Analysis

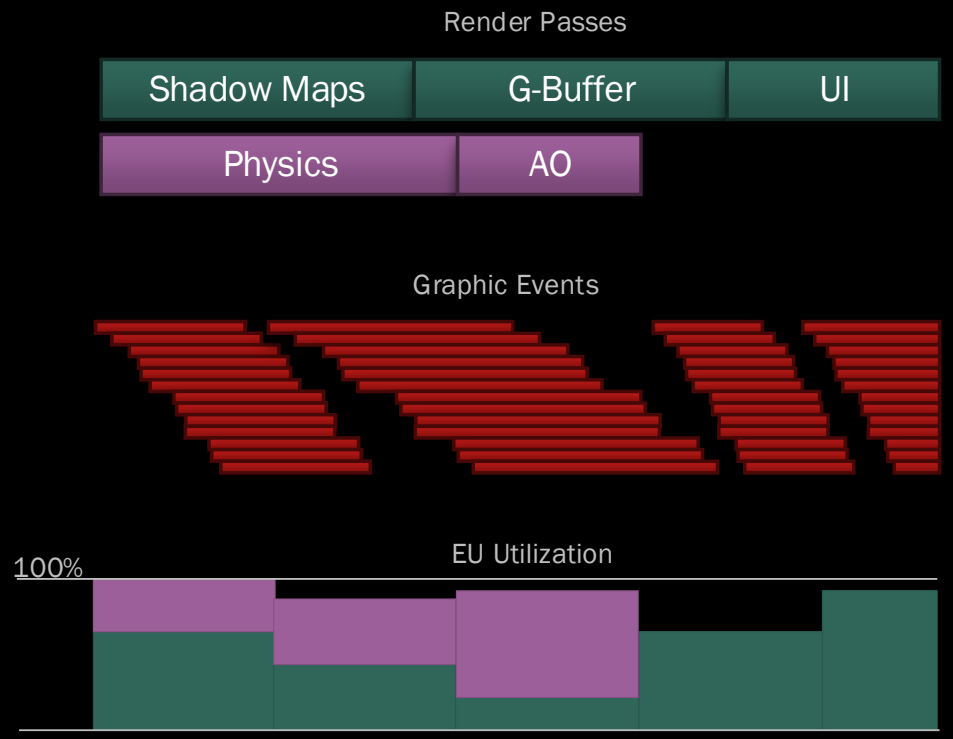
Using Graphics Frame and Trace Analyzer's to gain deeper insight

Modern graphics profiling challenges

Intel Gen9 Architecture



Intel Xe Architecture



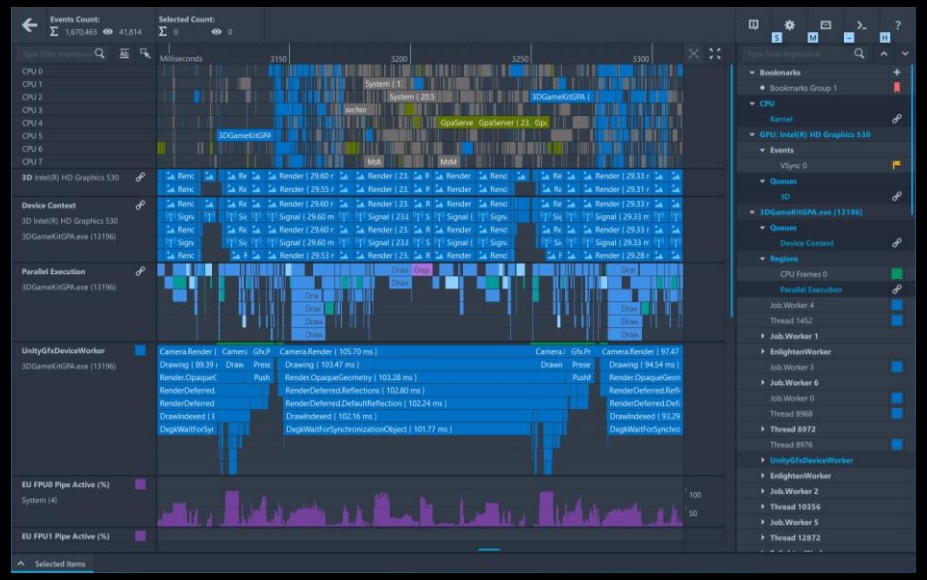
Performance profiling becomes more challenging as GPU evolves

Frame Analyzer & Trace Analyzer

Partners in performance analysis

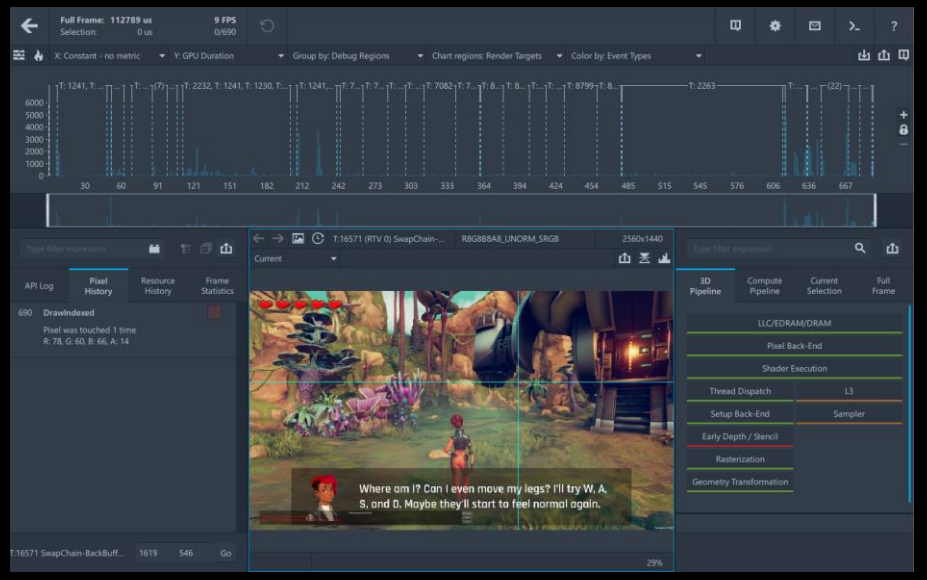
Trace Analyzer

Accurate application execution telemetry



Frame Analyzer

In-depth frame analysis & experimenting

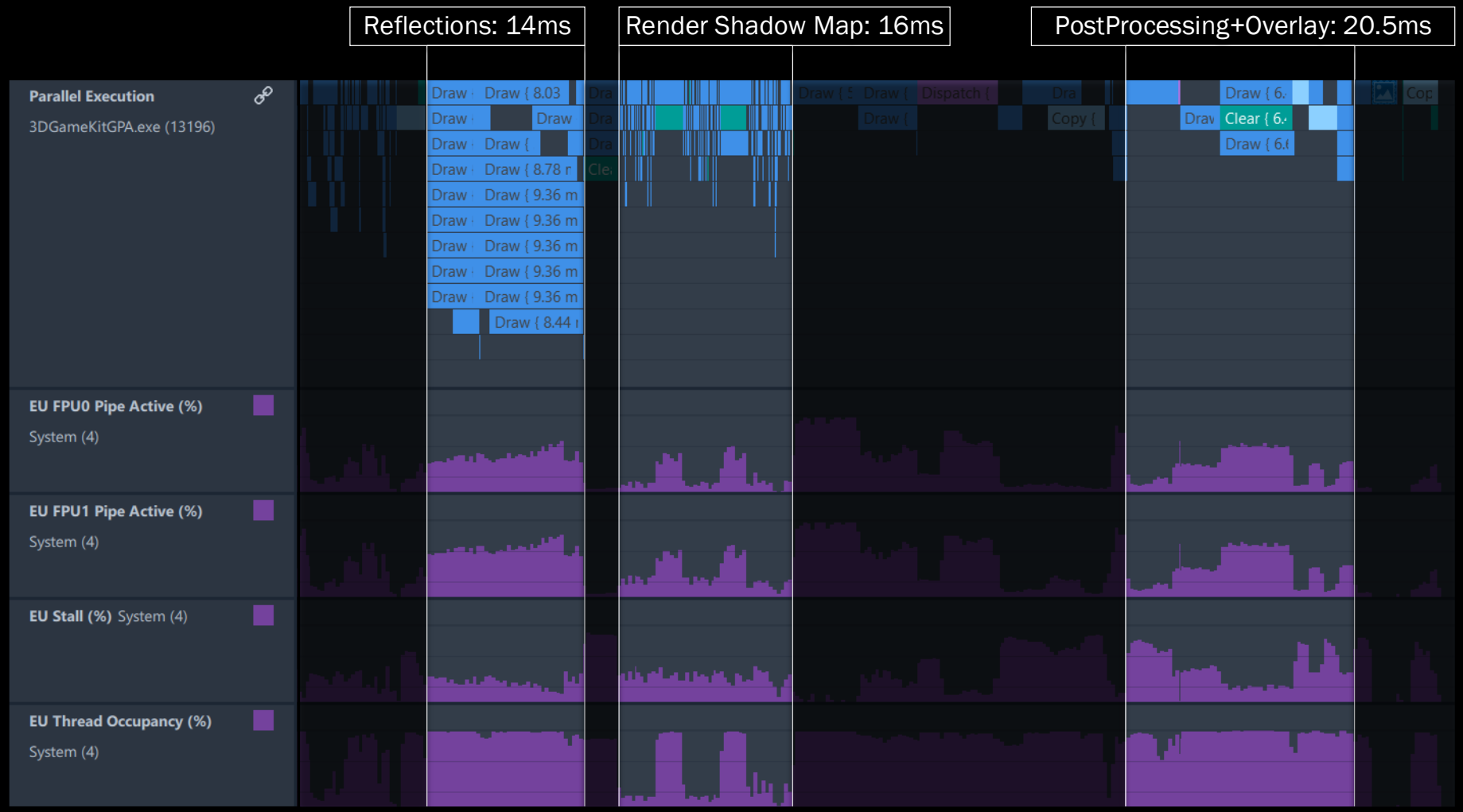


Use both tools to get a comprehensive insight on performance problems

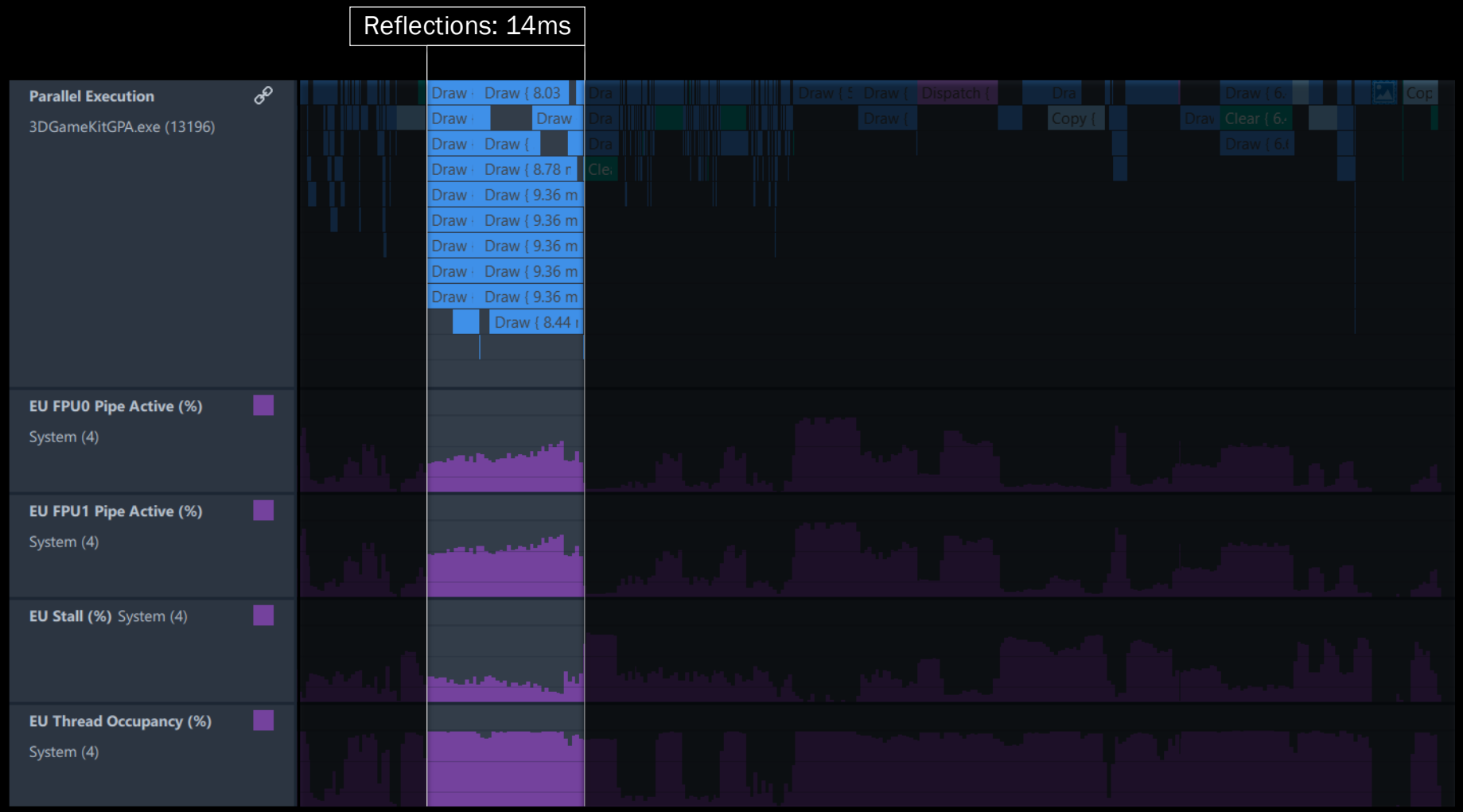
Hardware: KabyLake (Intel® HD 630)

Frame structure analysis

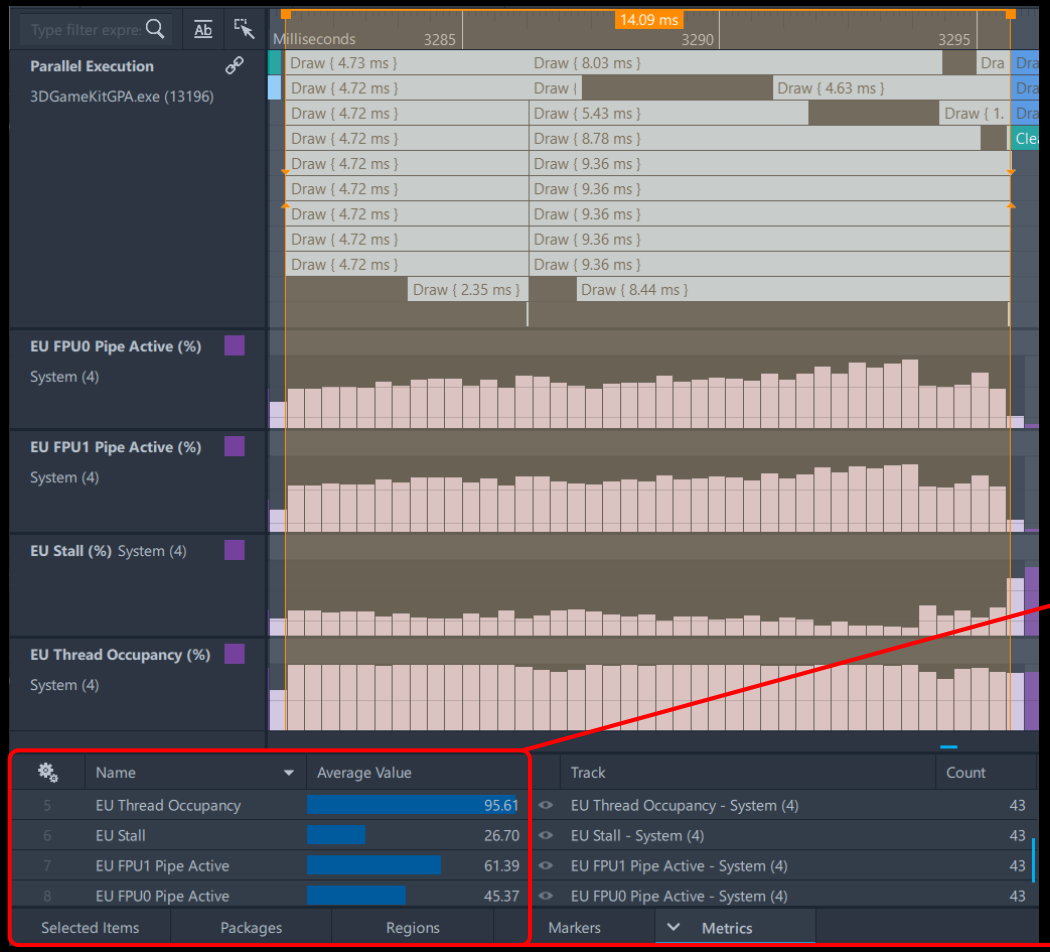
Unity 3D Game Kit



“Reflections” in Trace Analyzer



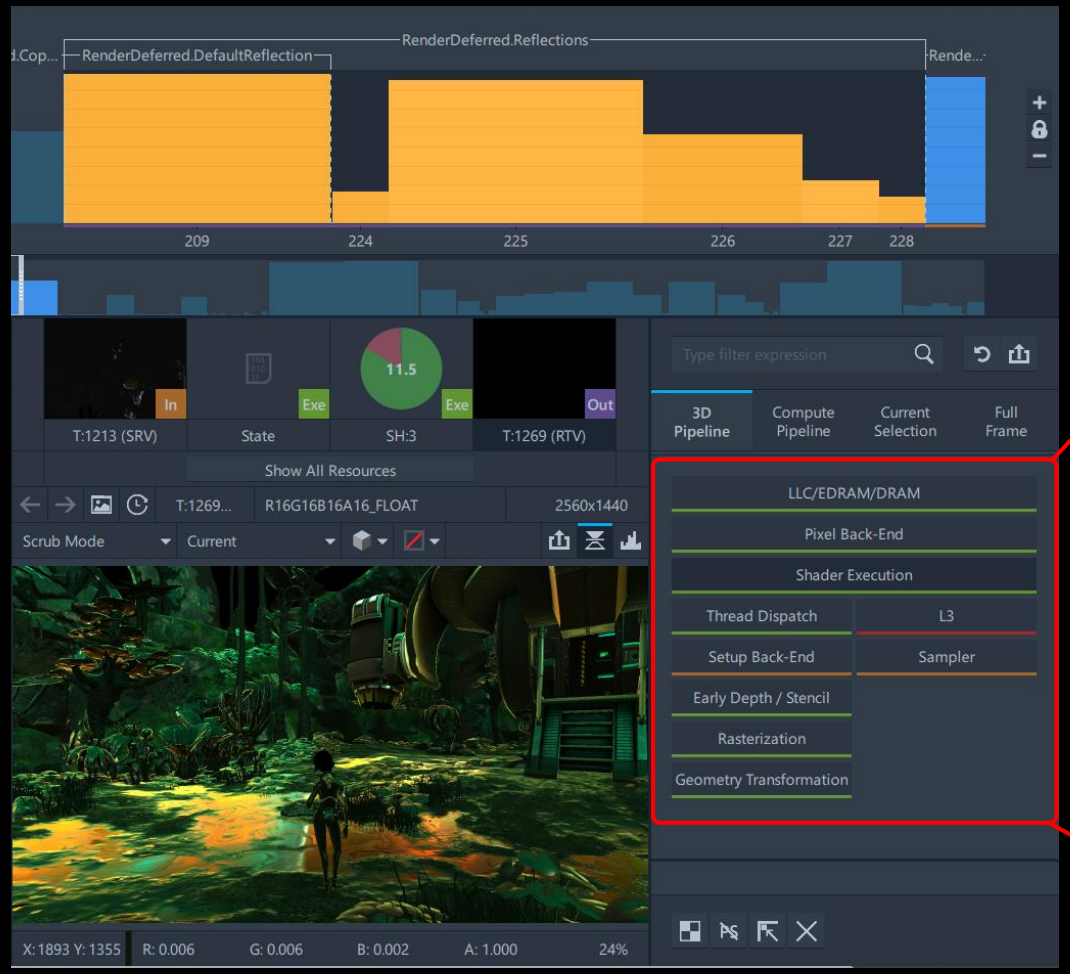
“Reflections” in Trace Analyzer



- Calls are executed in parallel
- Max EU thread saturation
- Calls potentially have memory bottleneck

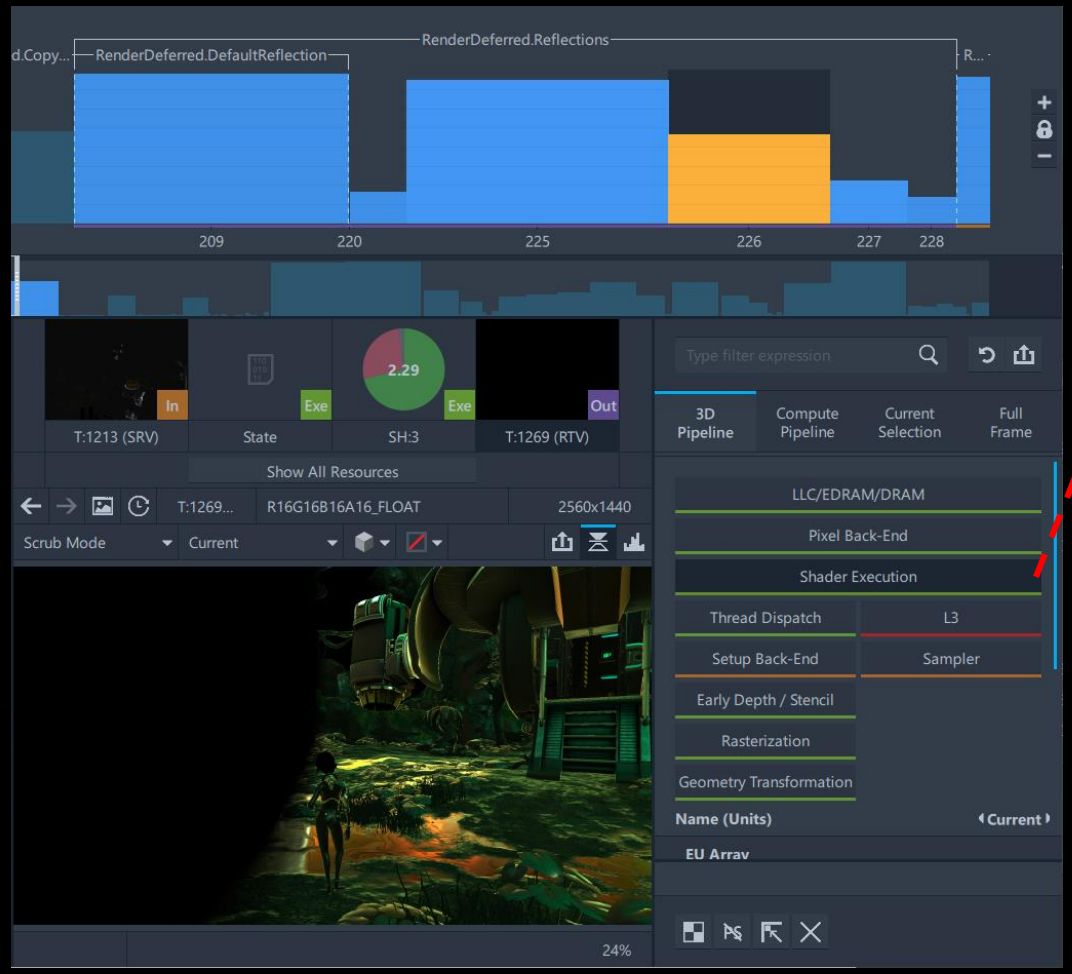
	Name	Average Value
5	EU Thread Occupancy	95.61
6	EU Stall	26.70
7	EU FPU1 Pipe Active	61.39
8	EU FPU0 Pipe Active	45.37

“Reflections” in Frame Analyzer



- ‘Metrics pane’ shows L3 bottleneck

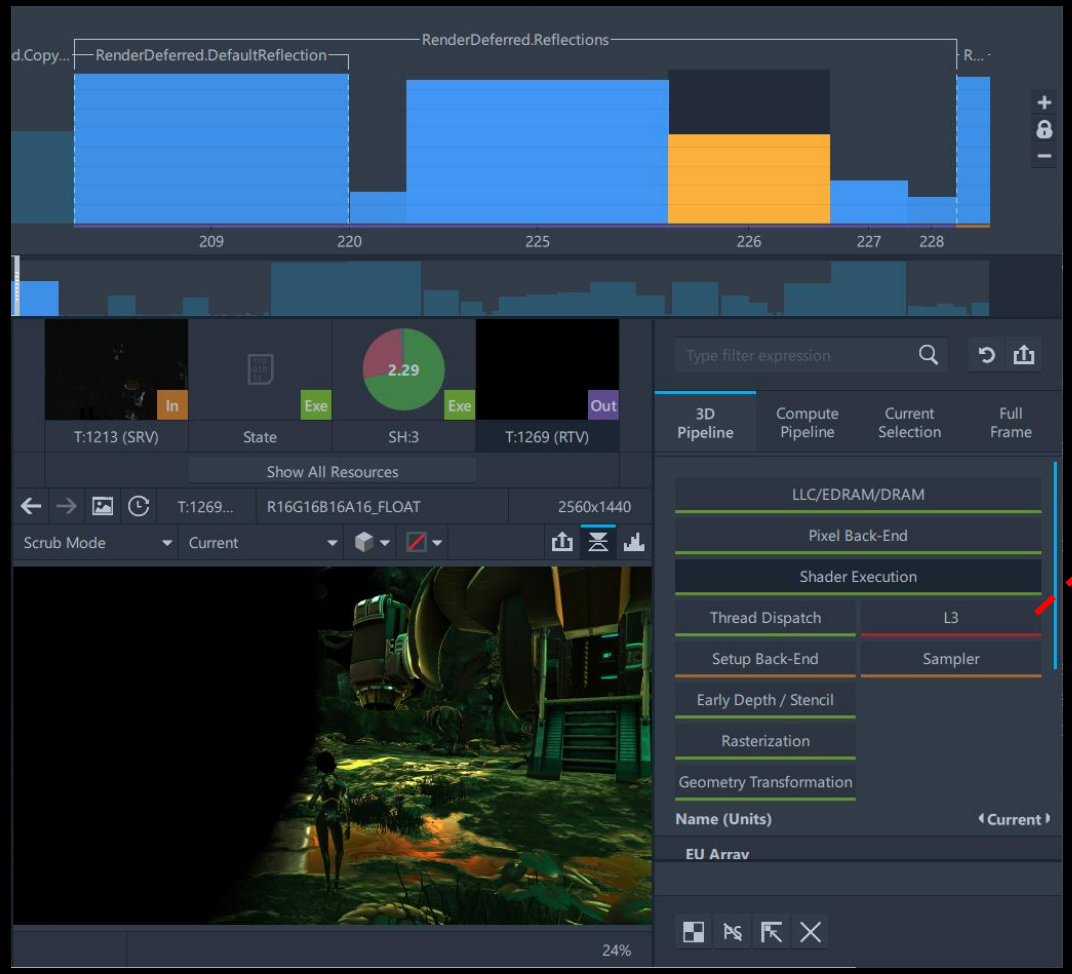
“Reflections” in Frame Analyzer



Each call has >90% EU Thread Occupancy

EU Active	%	70.3
EU Stall	%	28.3
EU Thread Occupancy	%	96.8

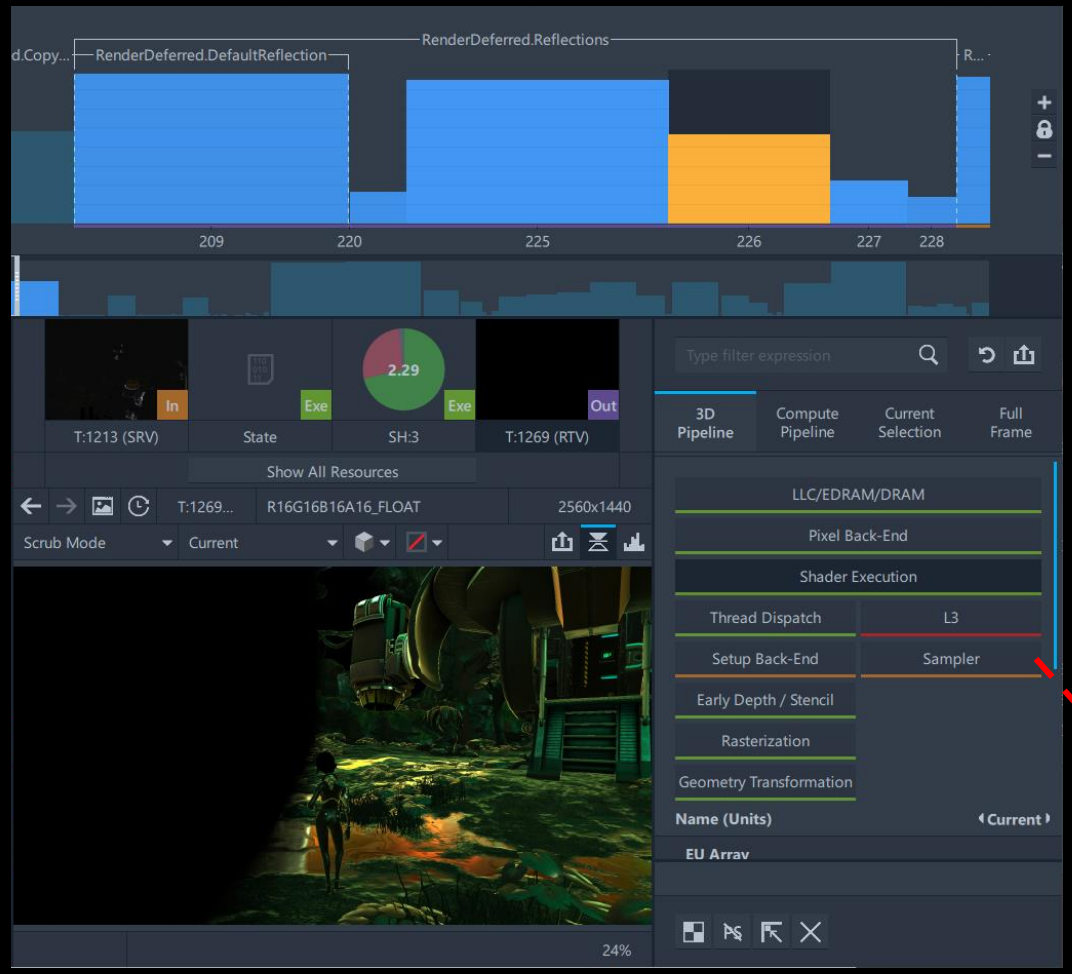
“Reflections” in Frame Analyzer



Each call Has L3 or DRAM bottleneck

Slice0 L3 Bank0 Active	%	98.6
Slice0 L3 Bank0 Stalled	%	34.6
Slice0 L3 Bank1 Active	%	98.6
Slice0 L3 Bank1 Stalled	%	33.7

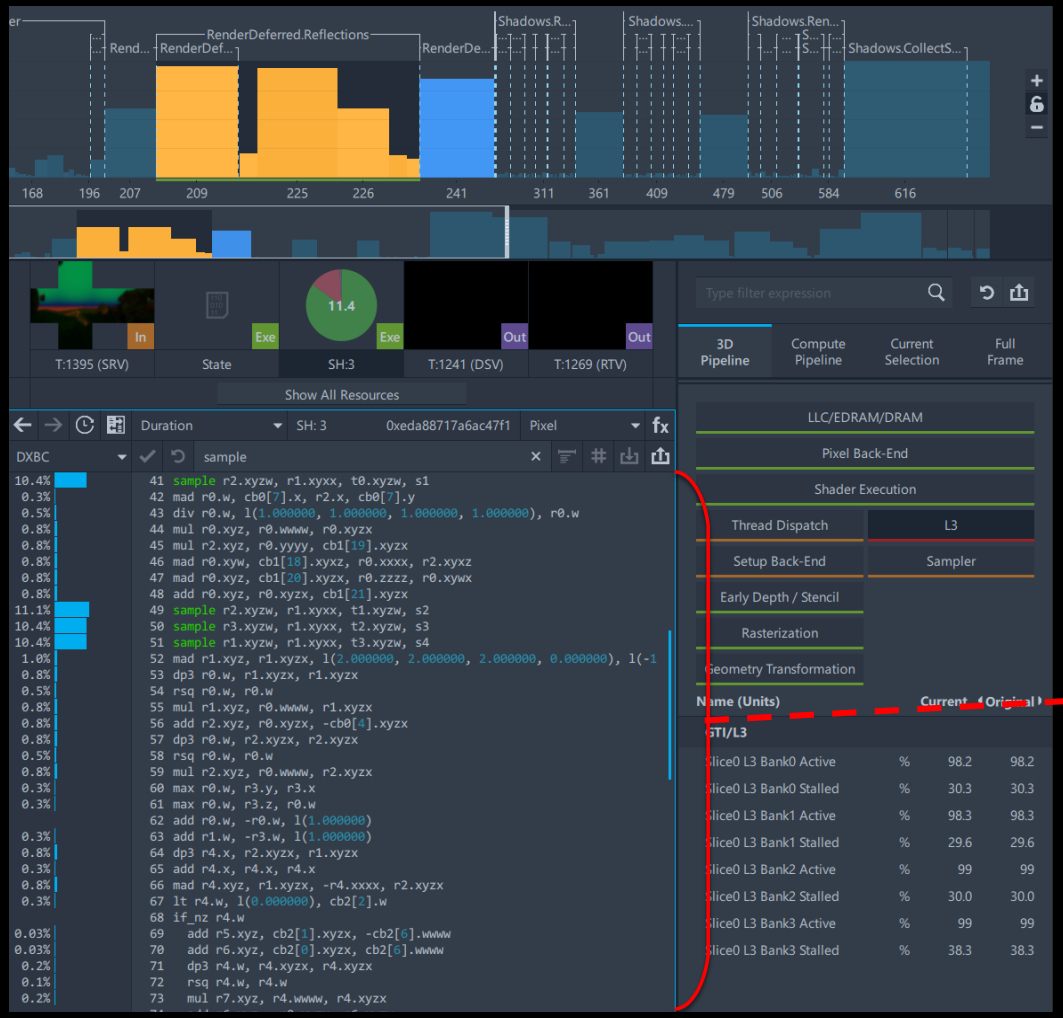
“Reflections” in Frame Analyzer



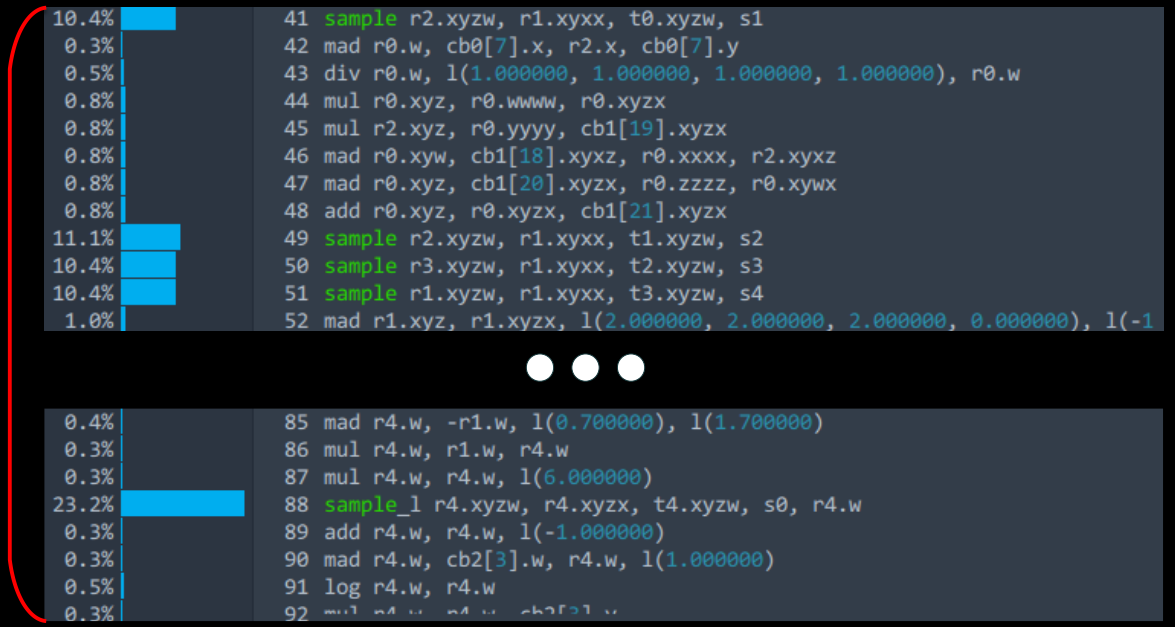
Each call has a sampler usage inefficiency

Slice0 Subslice0 Input Available	%	80.8
Slice0 Subslice0 Sampler Output Ready	%	41.0
Slice0 Subslice1 Input Available	%	81.2
Slice0 Subslice1 Sampler Output Ready	%	41.0

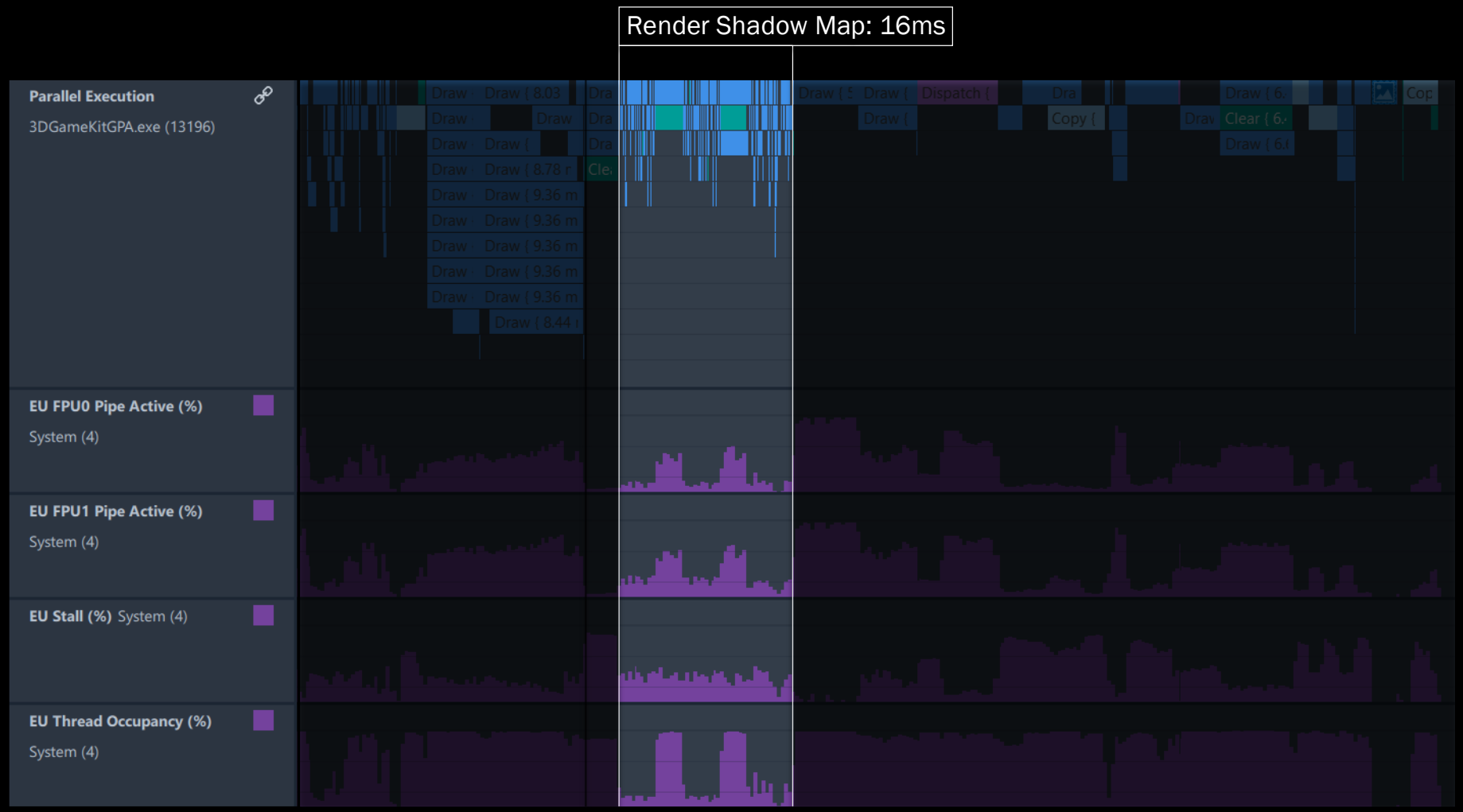
“Reflections” in Frame Analyzer



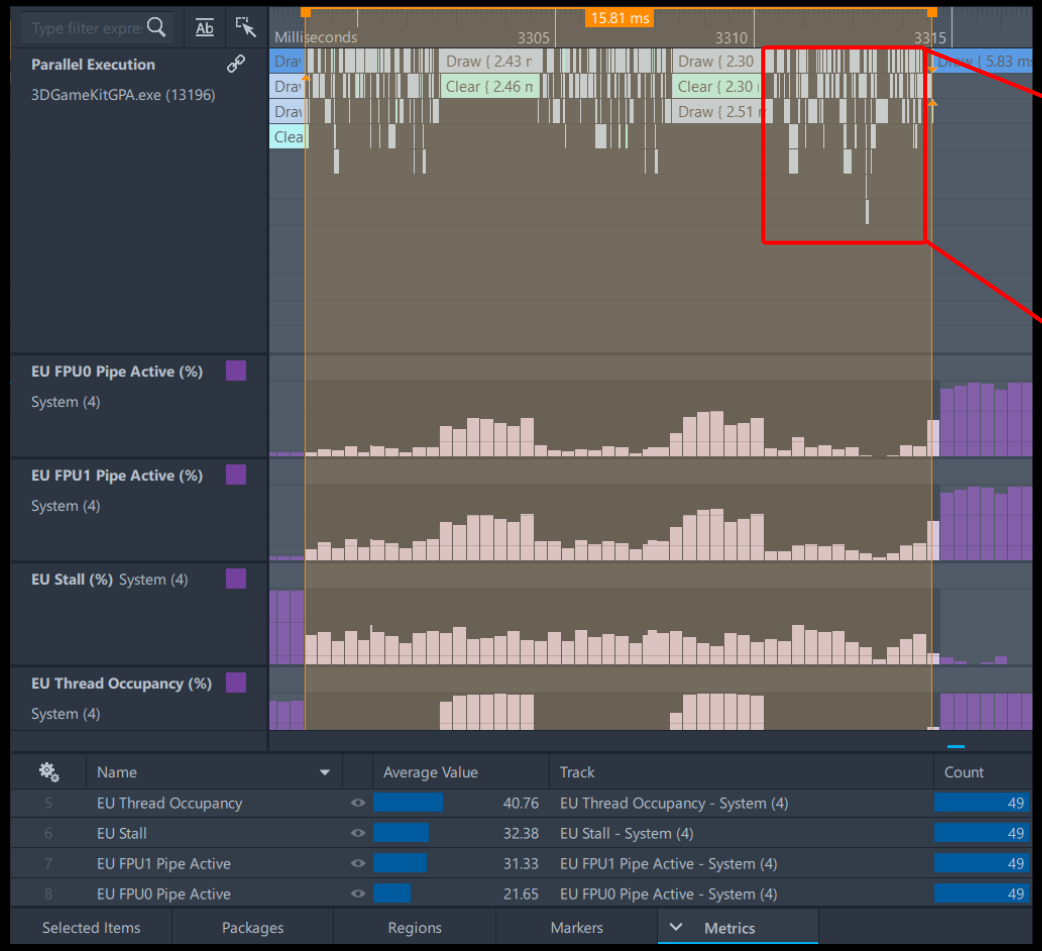
- Each call has the same shader
- ‘Shader Profiler’ detects that sampling from 5 textures takes 65% of time



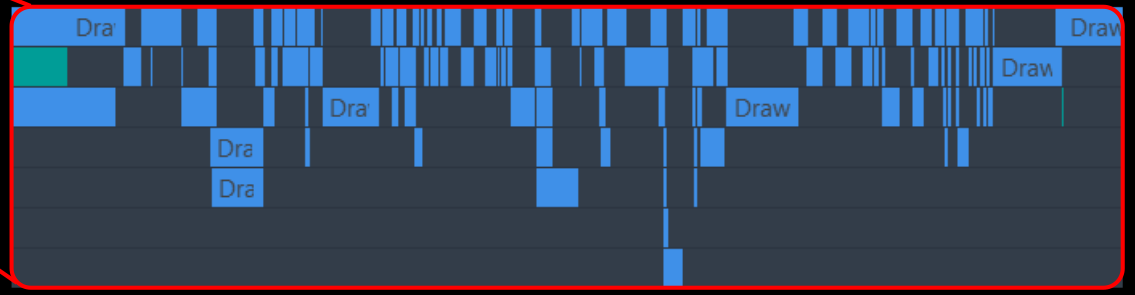
“Render Shadow Map“ in Trace Analyzer



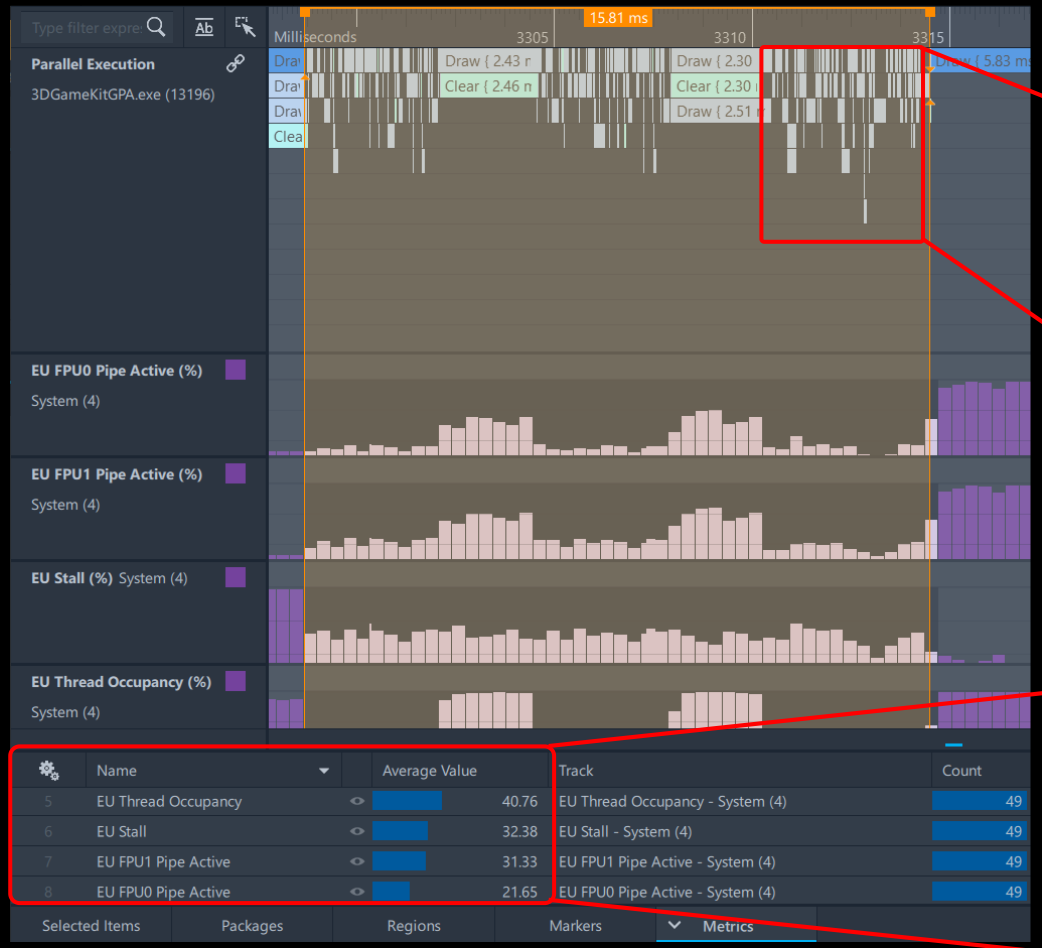
“Render Shadow Map“ in Trace Analyzer



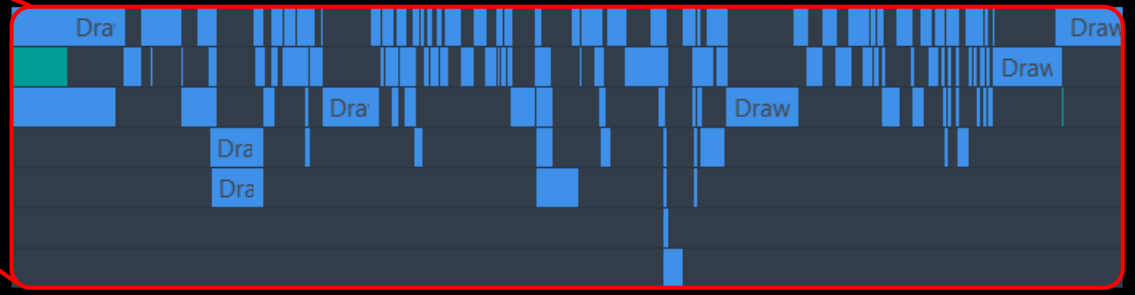
- There is some parallel execution, but it could be better



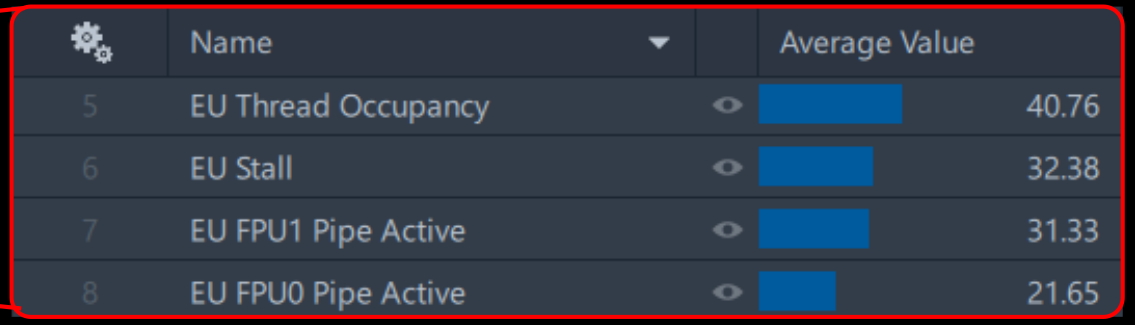
“Render Shadow Map“ in Trace Analyzer



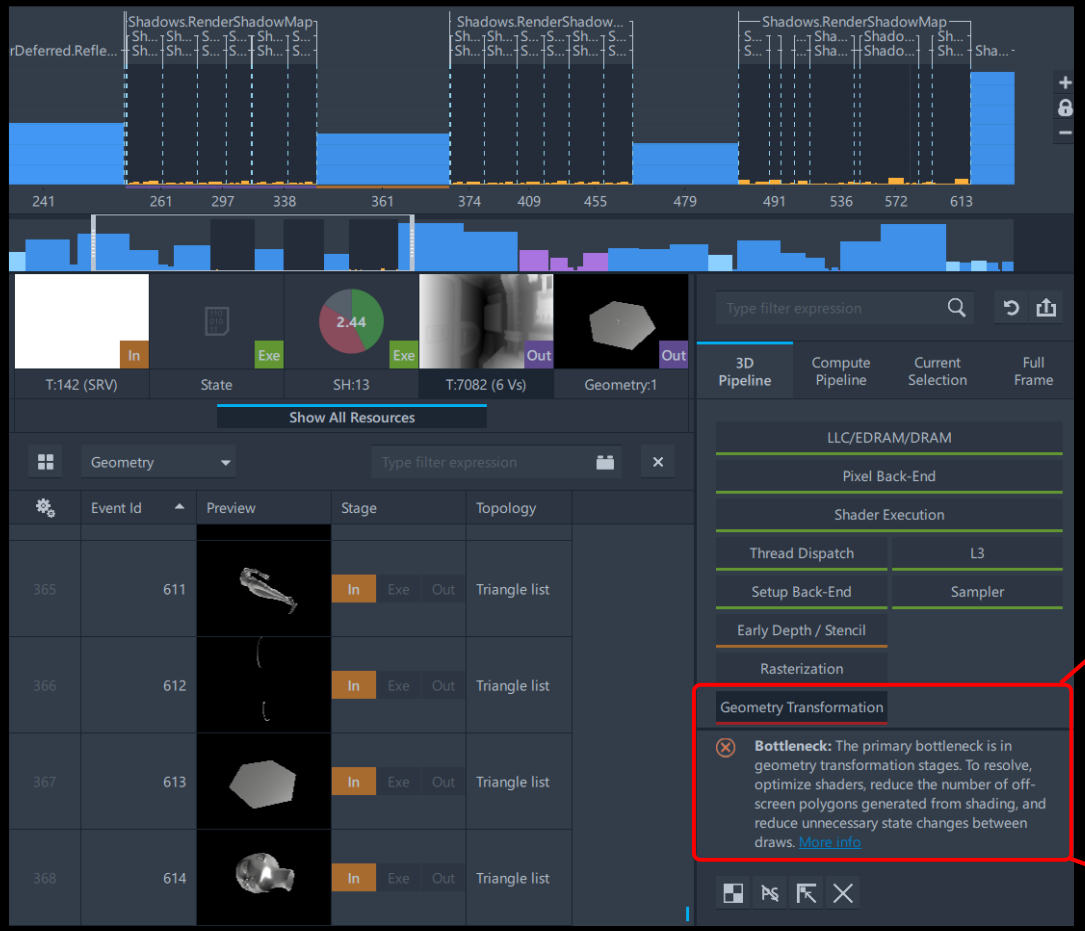
- There is some parallel execution, but it could be better



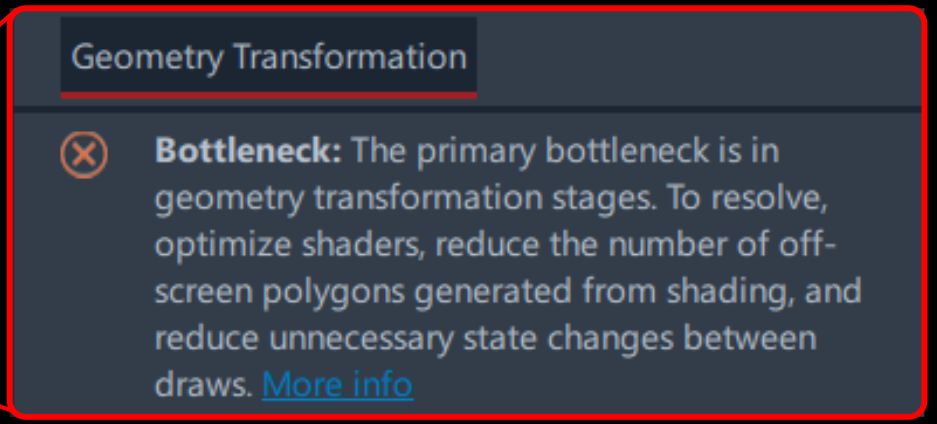
- Not enough threads being submitted to EUs



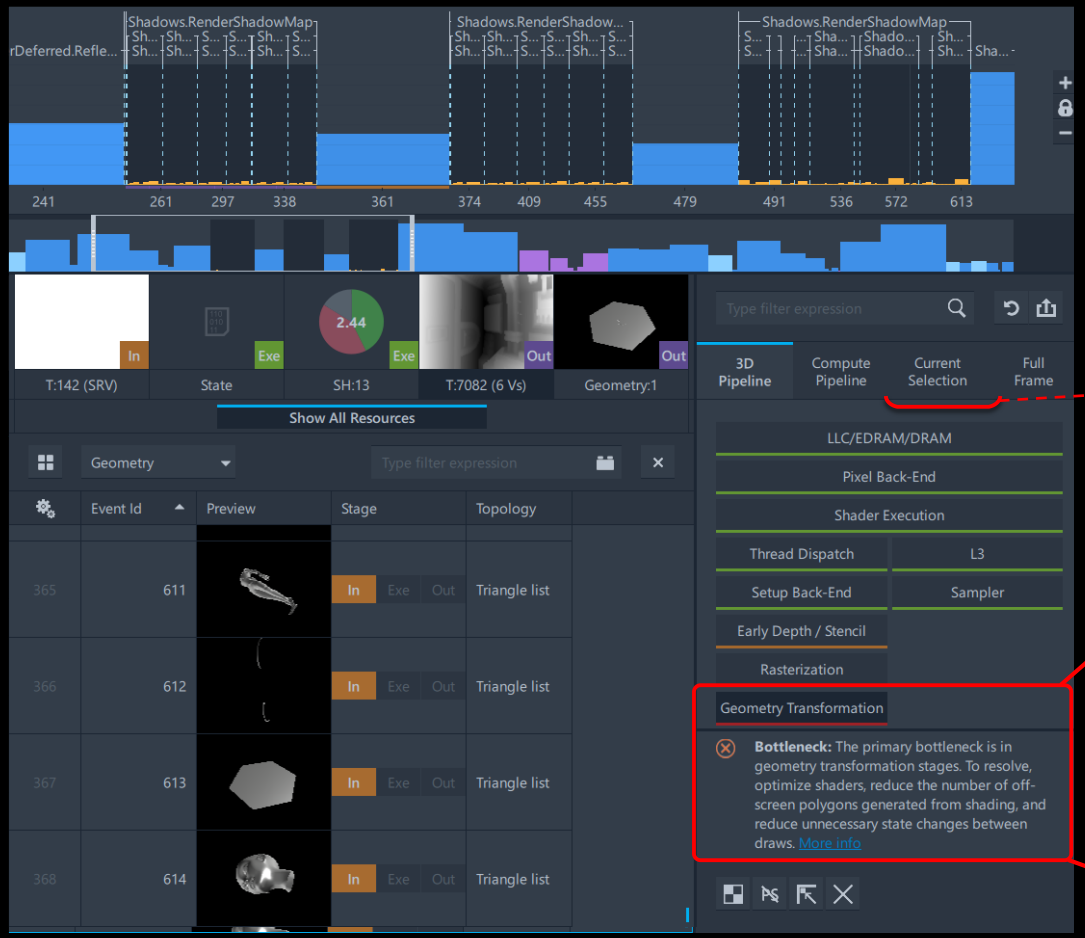
“Render Shadow Map“ in Frame Analyzer



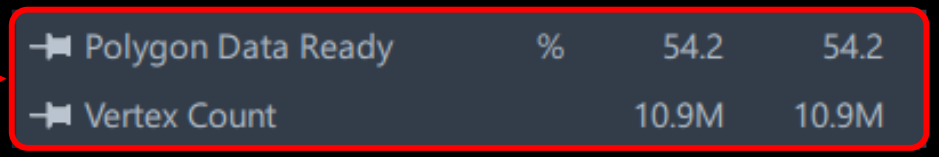
- Geometry bottleneck:
  - 368 meshes

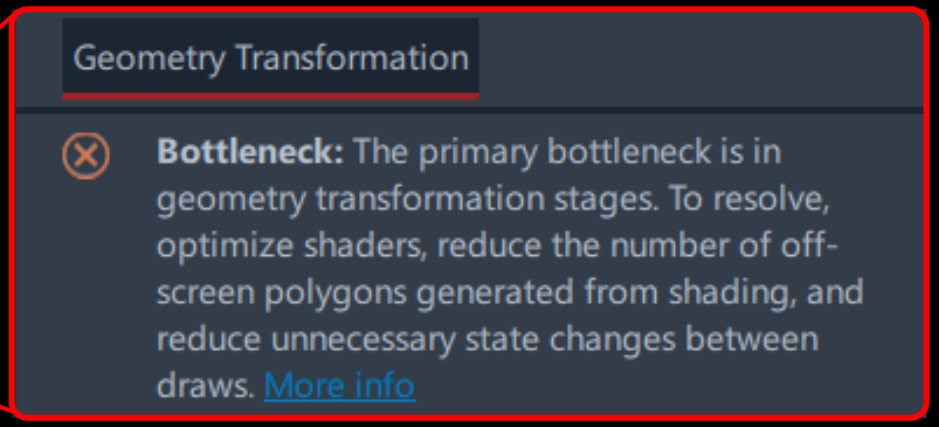


“Render Shadow Map“ in Frame Analyzer

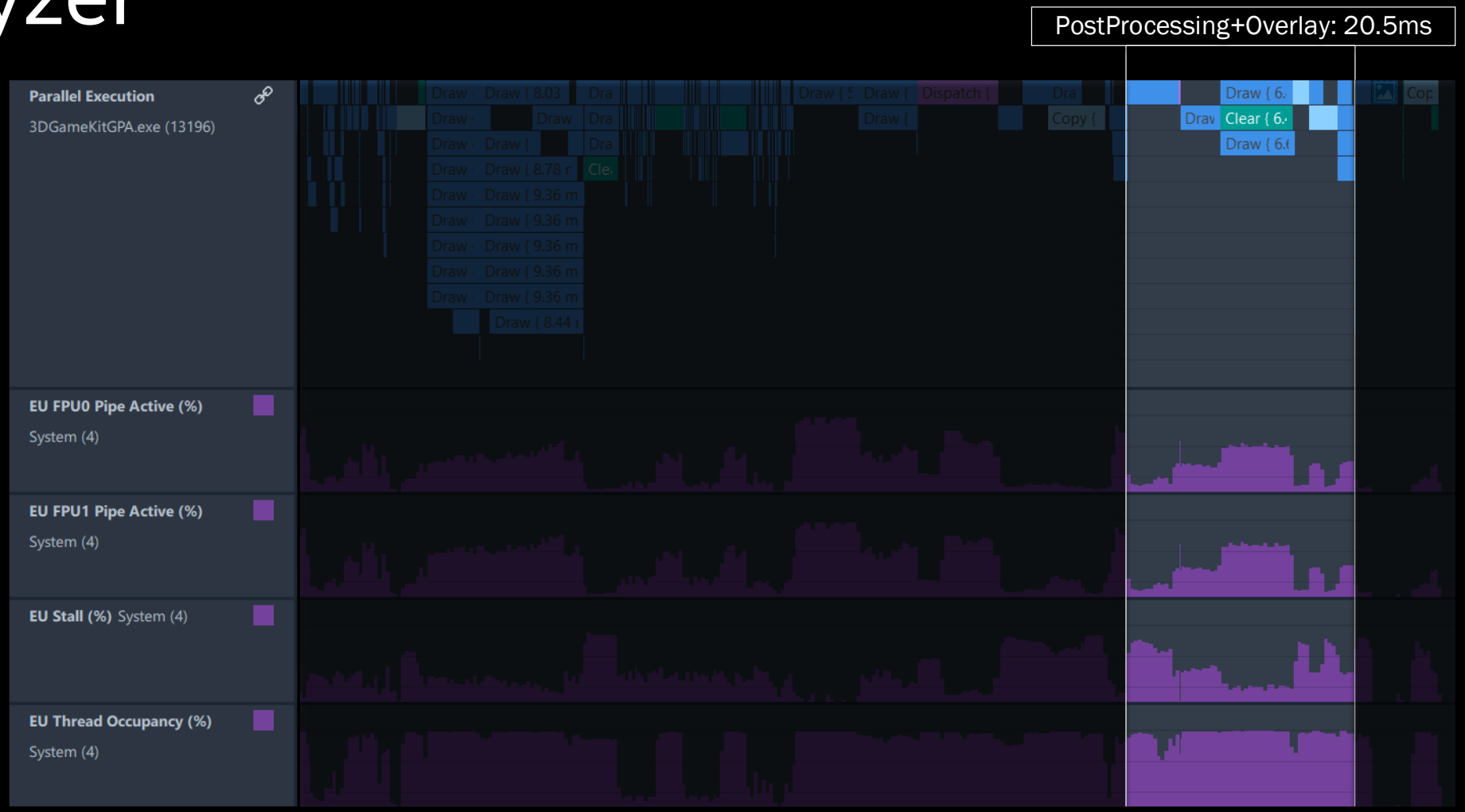


- Geometry bottleneck:
  - 368 meshes
  - 10.9M vertices (78% of frame)

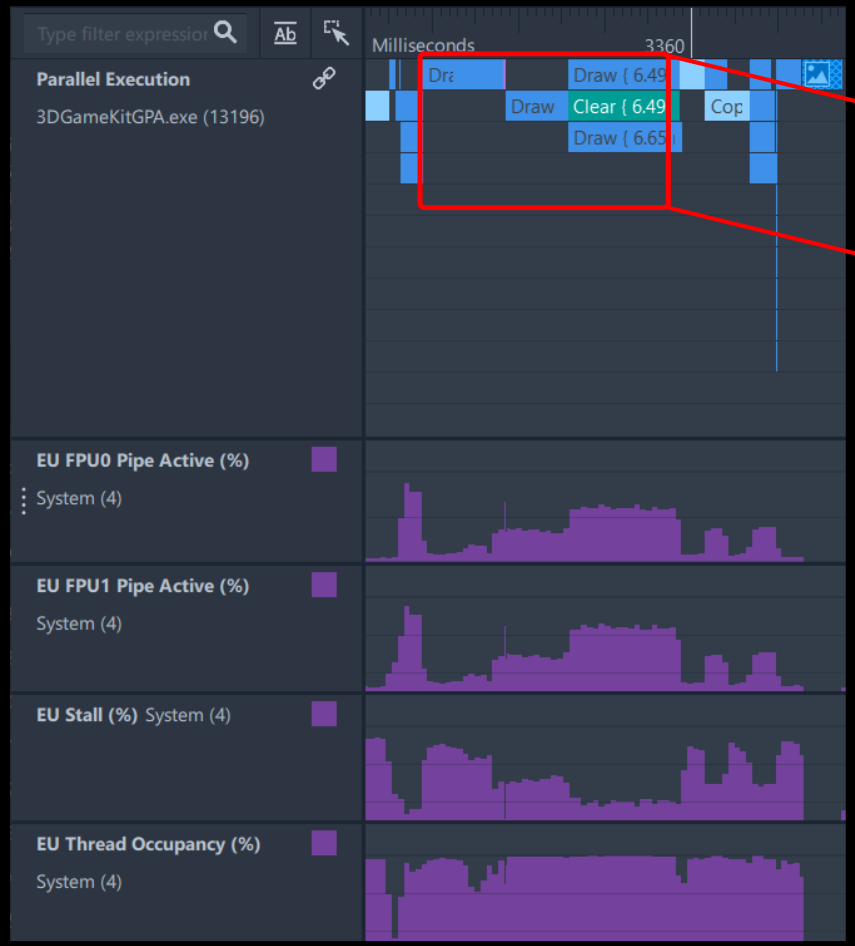




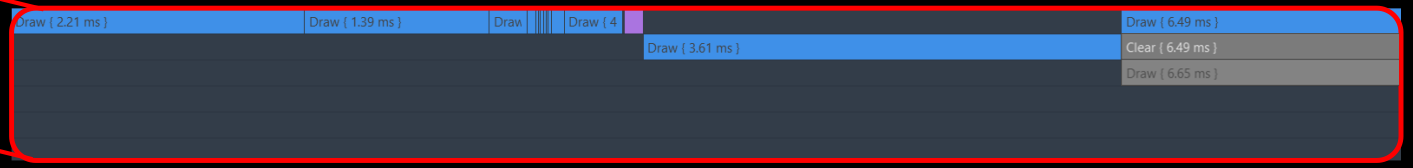
“Post-processing + Overlay” in Trace Analyzer



“Post-processing” in Trace Analyzer

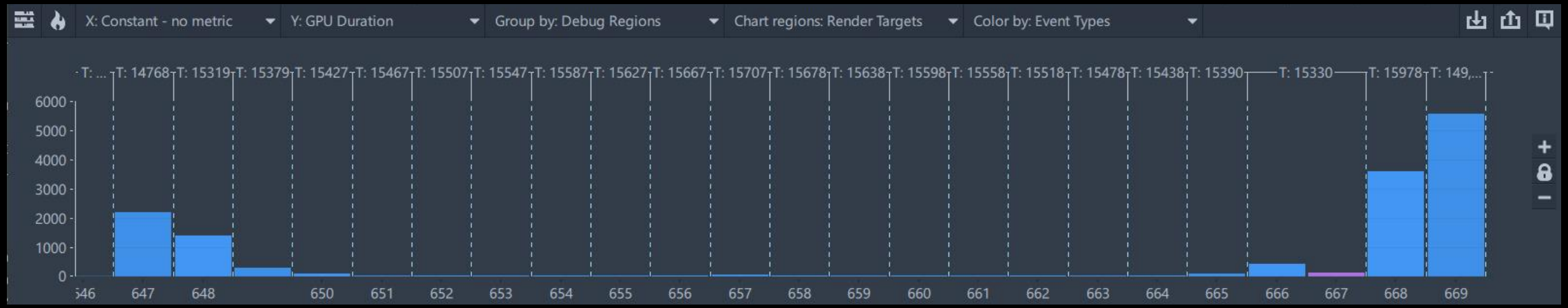


- 24 Calls, all are serialized!



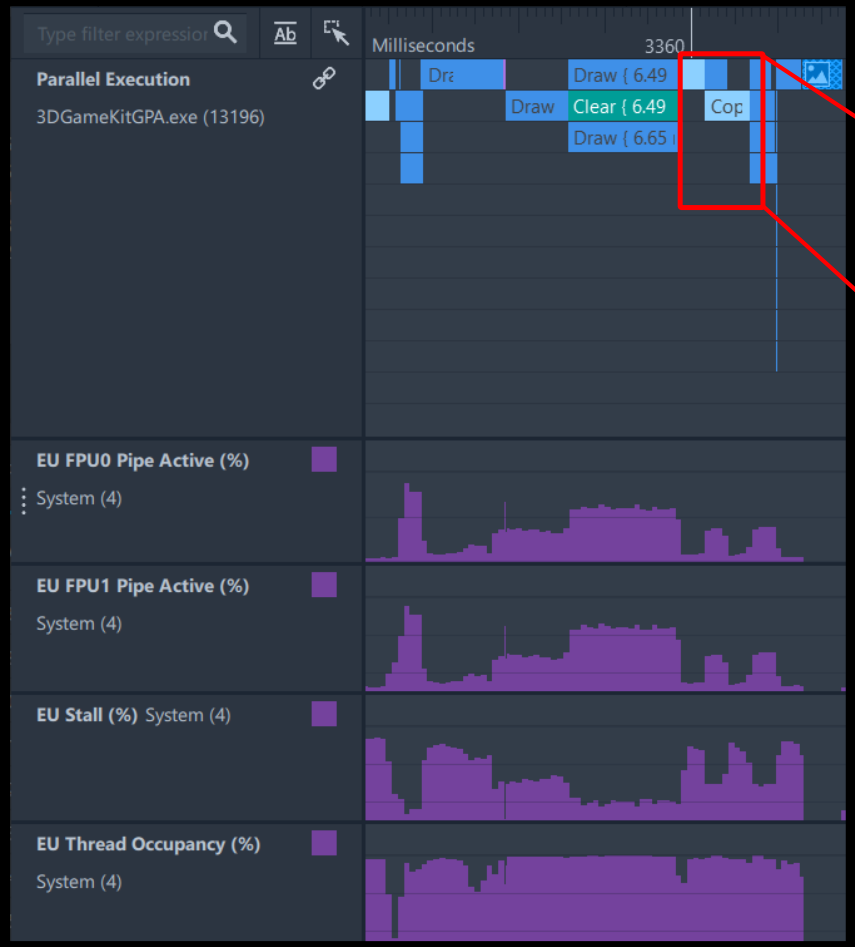
“Post-processing” in Frame Analyzer

- Render target changes after each draw and there are input/output dependencies between calls
- Dispatch/Draw transition

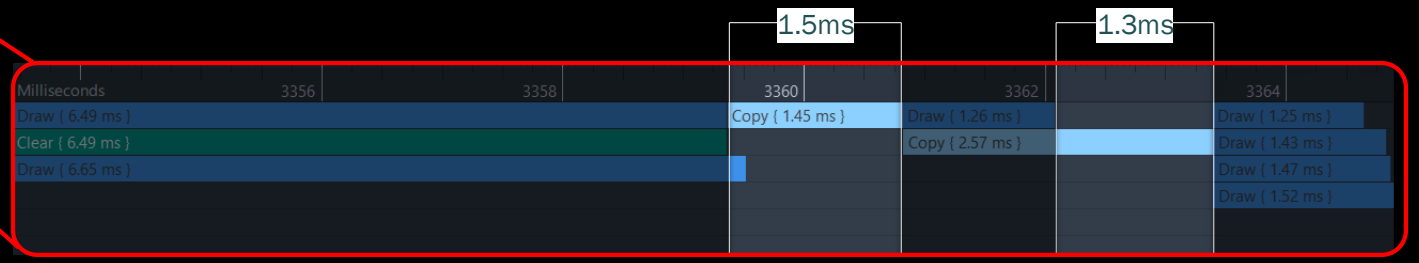


(Each dash line here is a Render Target change)

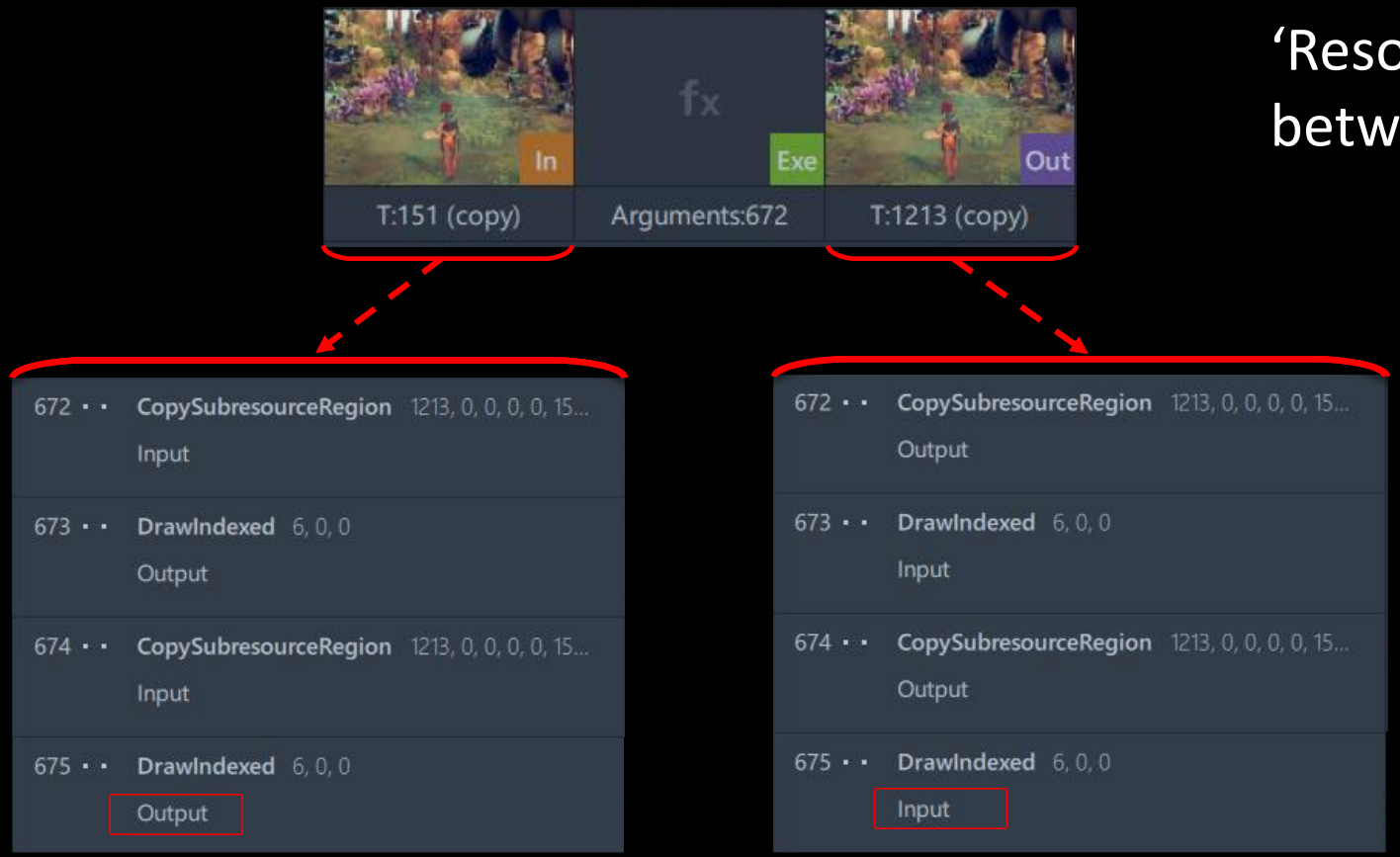
“Overlay” in Trace Analyzer



- Two Copy calls take 2.7ms!



“Overlay” in Frame Analyzer



‘Resource History’ shows dependencies between Copies and subsequent Draws!

Multiframe Analysis

Using Graphics Frame Analyzer to capture multiframe game streams

Multiframe Analysis

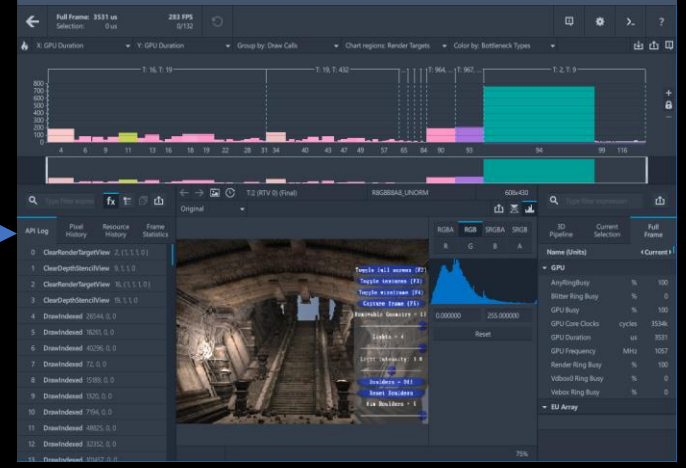
What is it?

Capture unbound stream of graphics API calls

Launch game

Capture a frame

Analyze Frame



Multiframe Analysis

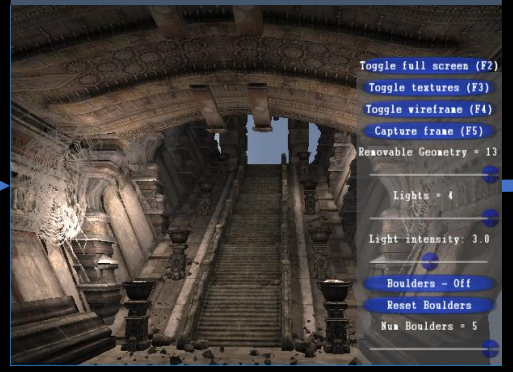
What is it?

Capture unbound stream of graphics API calls

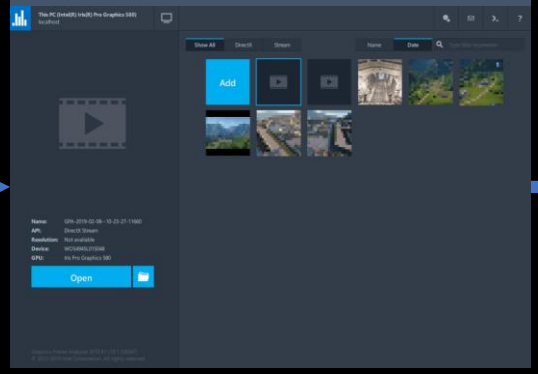
Launch game

Capture a frame

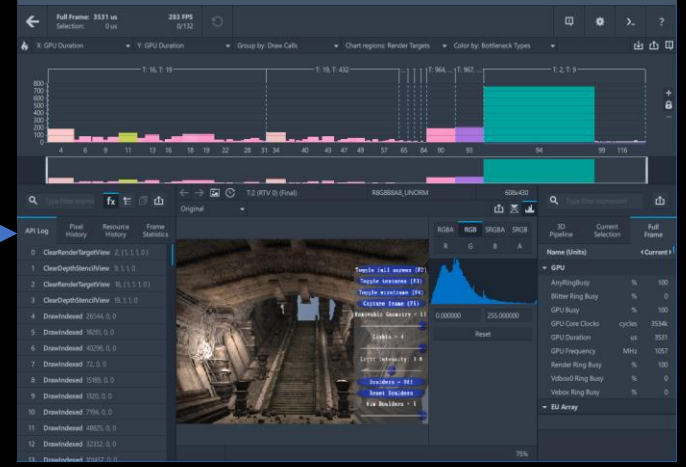
Stream captures **all frames**



Select Frame in Multi-frame View



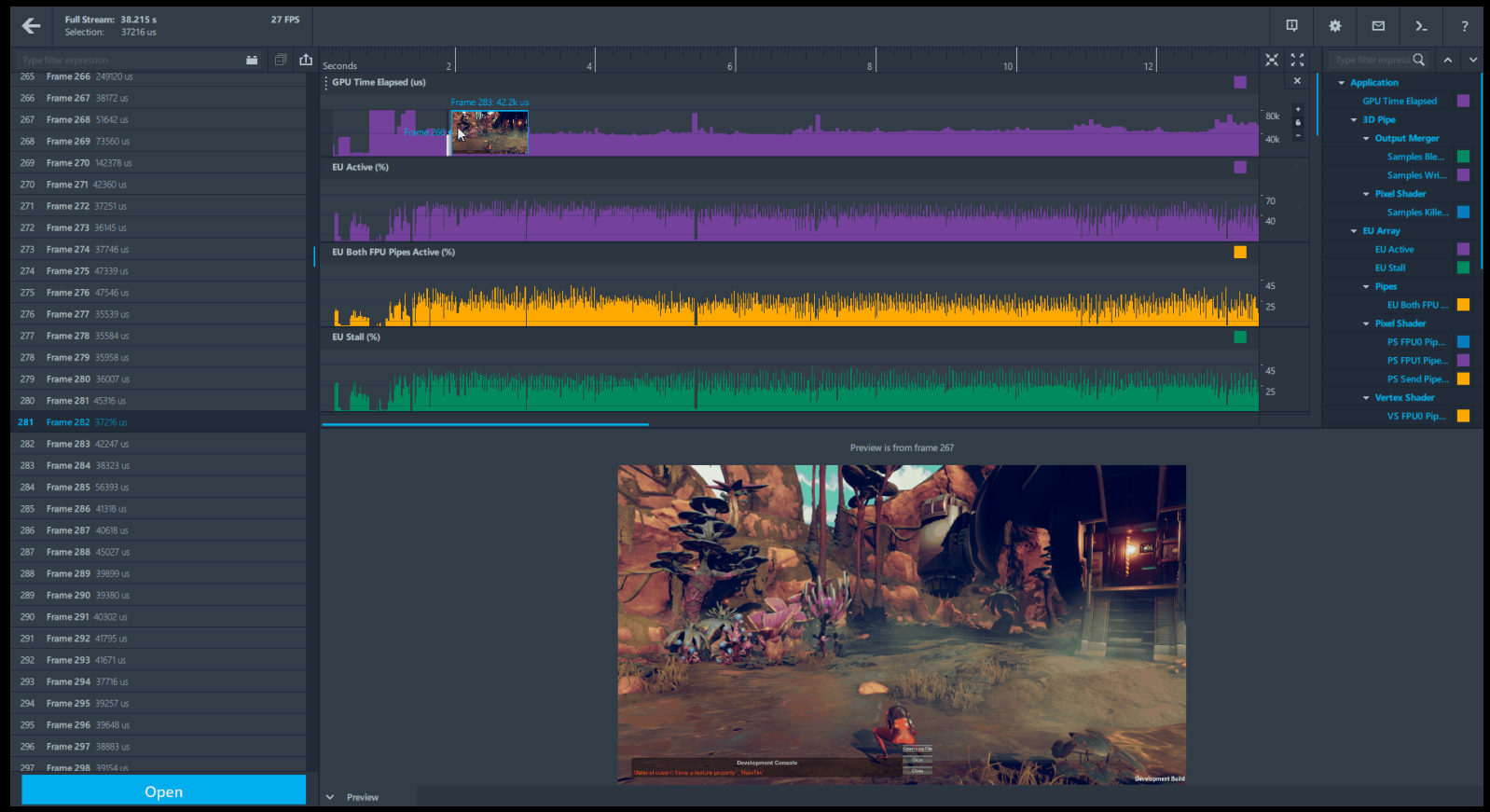
Analyze Frame



Multiframe Analysis

Why?

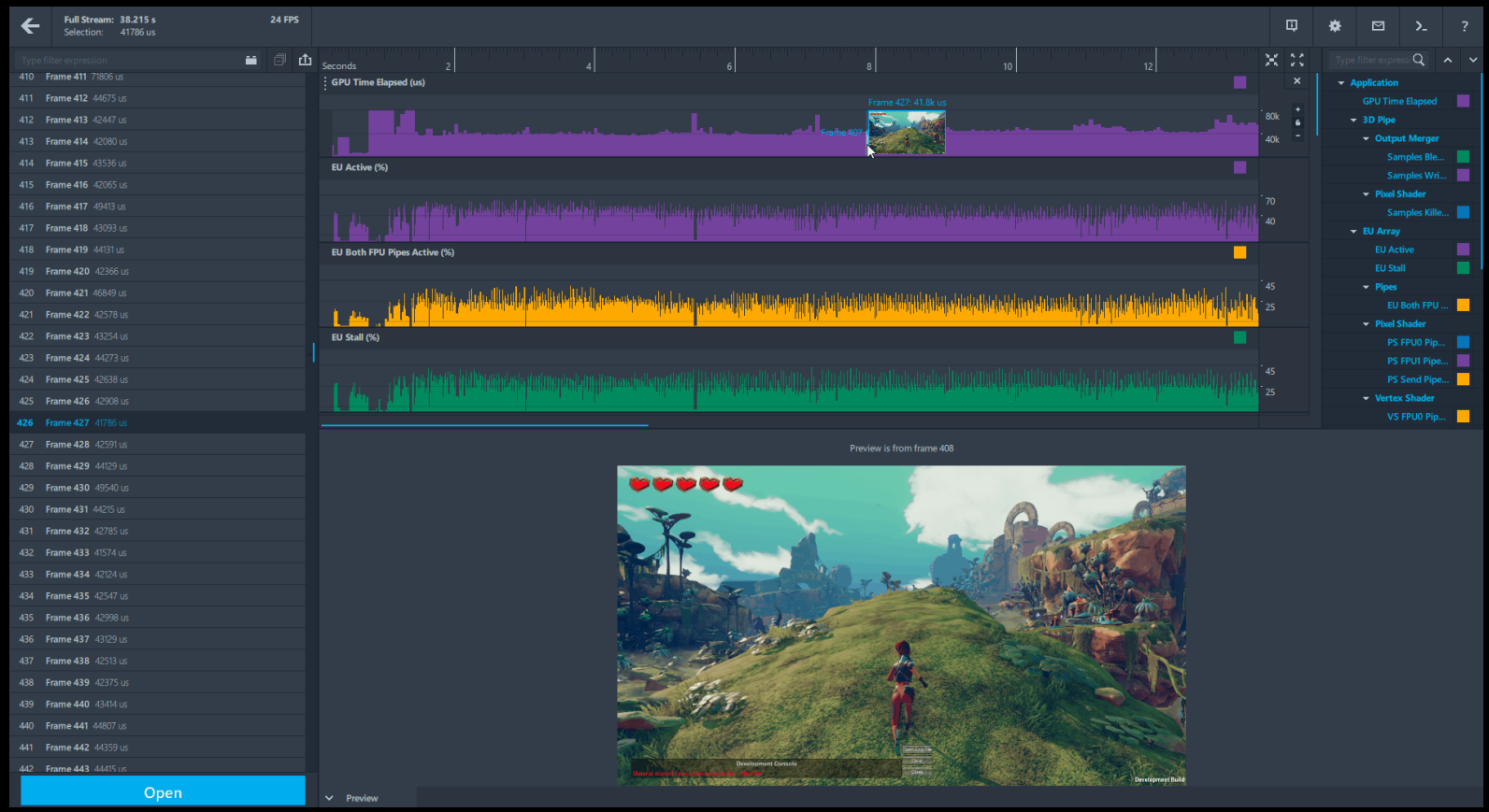
- Virtually impossible for single frame capture:
  - Intermittent glitches
  - ‘Random’ hitches
  - Multiframe algorithms



Multiframe Analysis

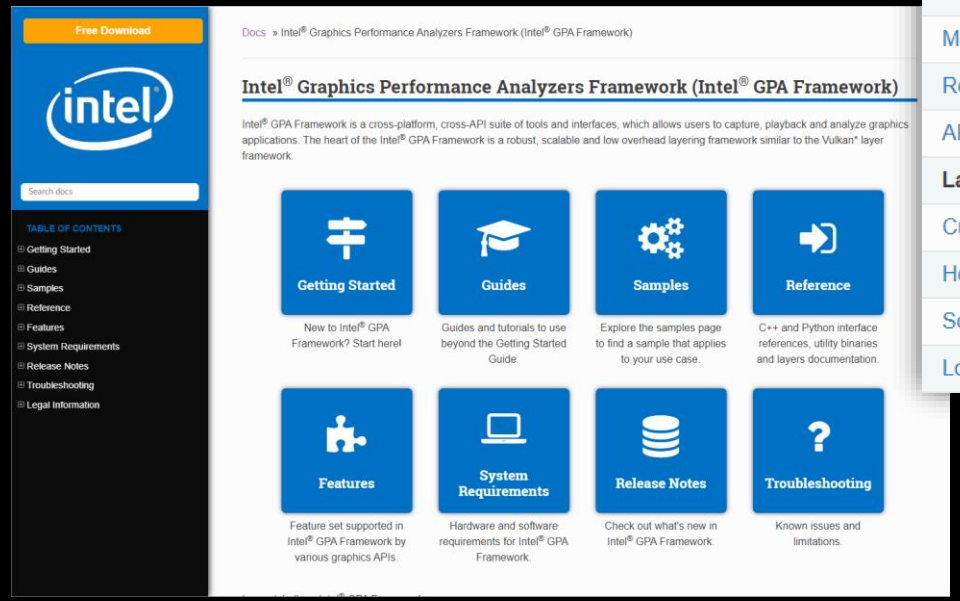
Why?

- Virtually impossible for single frame capture:
  - Intermittent glitches
  - ‘Random’ hitches
  - Multiframe algorithms



Intel® GPA Framework

- Intel® GPA's multiframe analysis backend
  - Standalone utilities to capture, playback and analyze multiframe streams
  - Cross-platform, cross-API layer mechanism
  - C++ and Python interfaces



	Vulkan*	DirectX 11*	DirectX 12*
Capture and playback			
Basic Capture and Playback	✓	✓	✓
Keyframing Capture	✓	✓	✓
Range Repeat	✓	✓	✓
Stream Analysis			
Metric Collection	✓		✓
Experiments	✓		
Metadata Extraction	✓		✓
Resource Extraction	✓		
API Call Inspection	✓		✓
Layers			
Custom (User) Layers	✓	✓	✓
Heads-Up Display(HUD) Layer	✓	✓	✓
Screenshot Layer	✓	✓	✓
Logging Layer	✓	✓	✓

Automated Performance Reporting

Using Intel® GPA Framework to generate performance reports

Performance profiling vs awareness

Performance Profiling

- Time consuming
- Requires domain knowledge
- Subset of engineers involved
- Profiling can be hard to do throughout dev cycle
- “Premature optimization is the root of all evil”

Performance profiling vs awareness

Performance Profiling

- Time consuming
- Requires domain knowledge
- Subset of engineers involved
- Profiling can be hard to do throughout dev cycle
- “Premature optimization is the root of all evil”

Performance Awareness

- Being conscious about performance at content creation
- Whole team effort
- Can alleviate profiling at end of dev cycle
- CI/CD workflow
- Not trivial and time consuming (capture, playback, analyze, create reports, etc.)

Performance profiling vs awareness

Performance Profiling

- Time consuming
- Requires domain knowledge
- Subset of engineers involved
- Profiling can be hard to do throughout dev cycle
- “Premature optimization is the root of all evil”

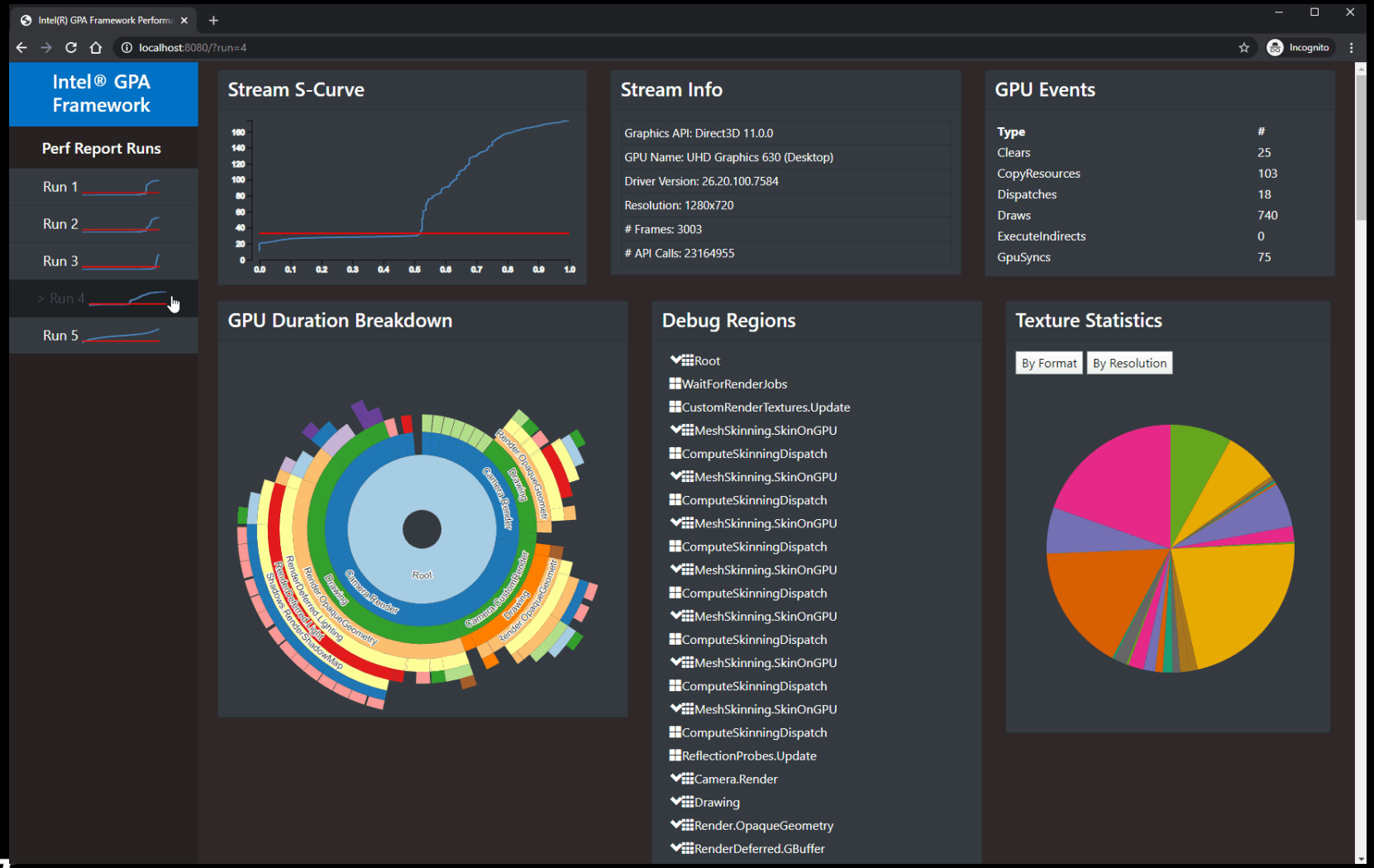
Performance Awareness

- Being conscious about performance at content creation
- Whole team effort
- Can alleviate profiling at end of dev cycle
- CI/CD workflow
- Not trivial and time consuming (capture, playback, analyze, create reports, etc.)

Intel® GPA Framework can help with this and more!

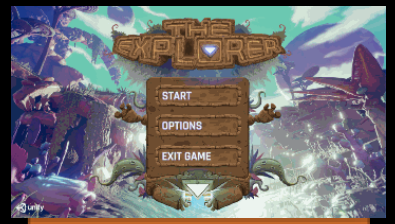
Automated Performance Reporting

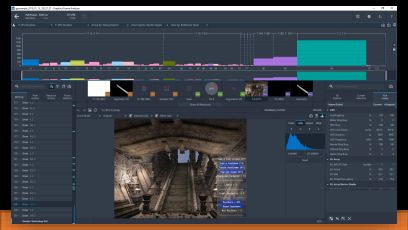
Sample available in Intel® DevMesh!

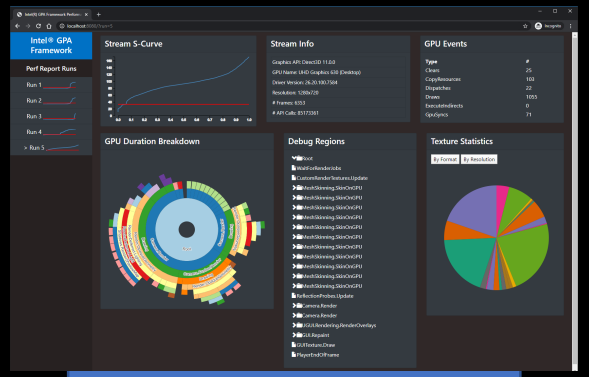


CI Workflow

10k feet view







Jenkins Nightly Build

Automated Playthrough

Automated Performance Profiling

Report Flagging Potential Regressions

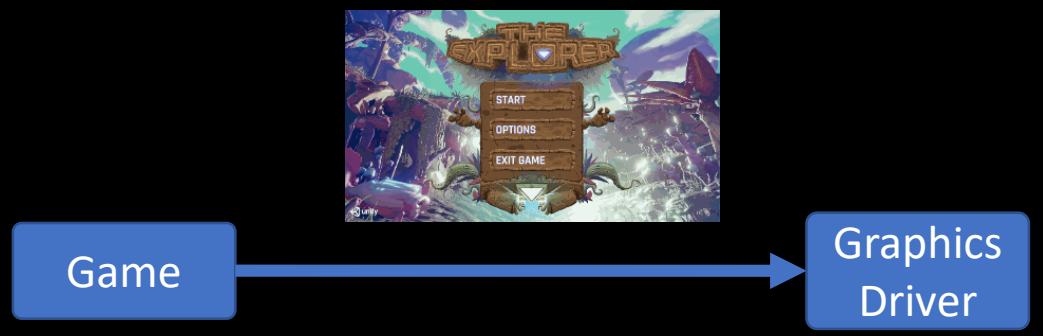
Someone forgot to create an LOD for their asset!

Intel® GPA Framework Stream

Further Analysis with GPA if needed

CI Workflow

Automated Playthrough

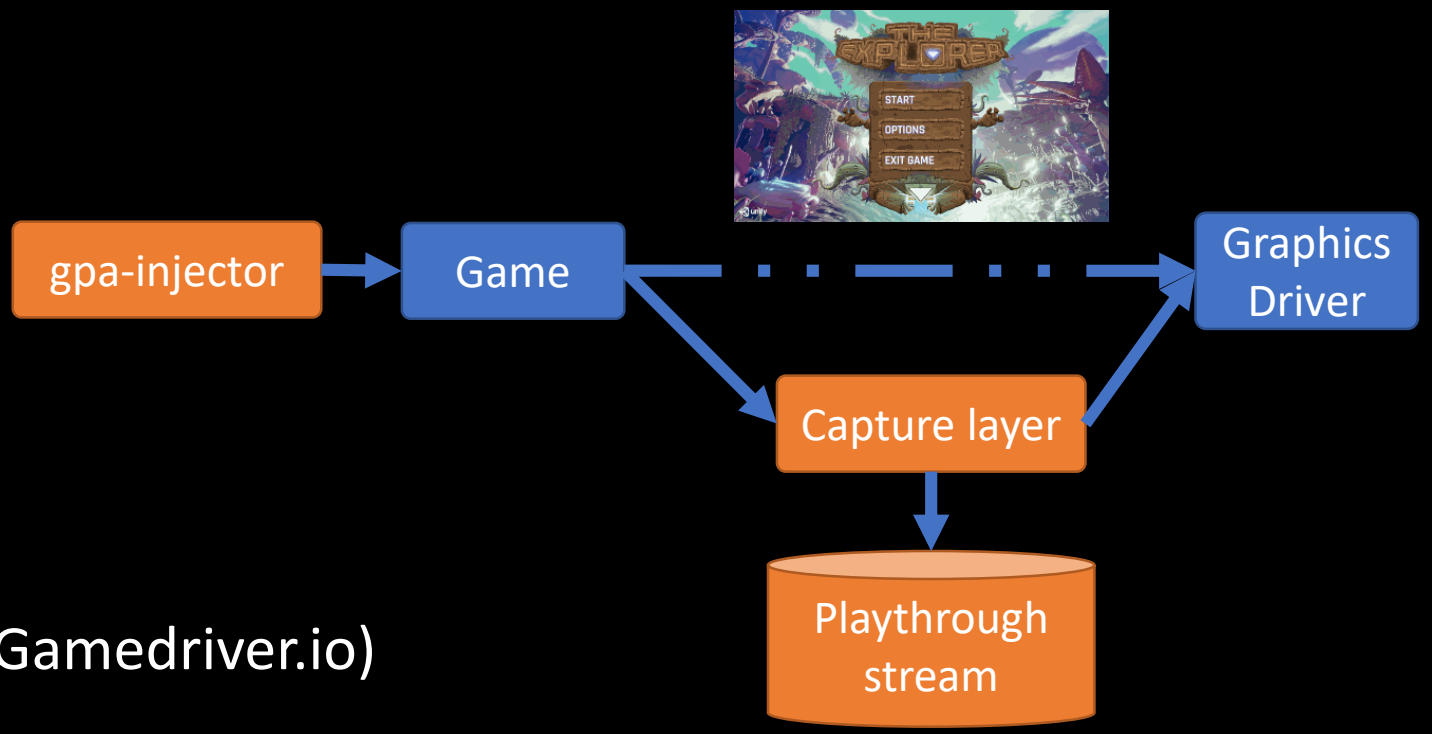


Reproducible scenario

- Benchmark
- Custom scene
- Gameplay unit test system (Gamedriver.io)

CI Workflow

Automated Playthrough



Reproducible scenario

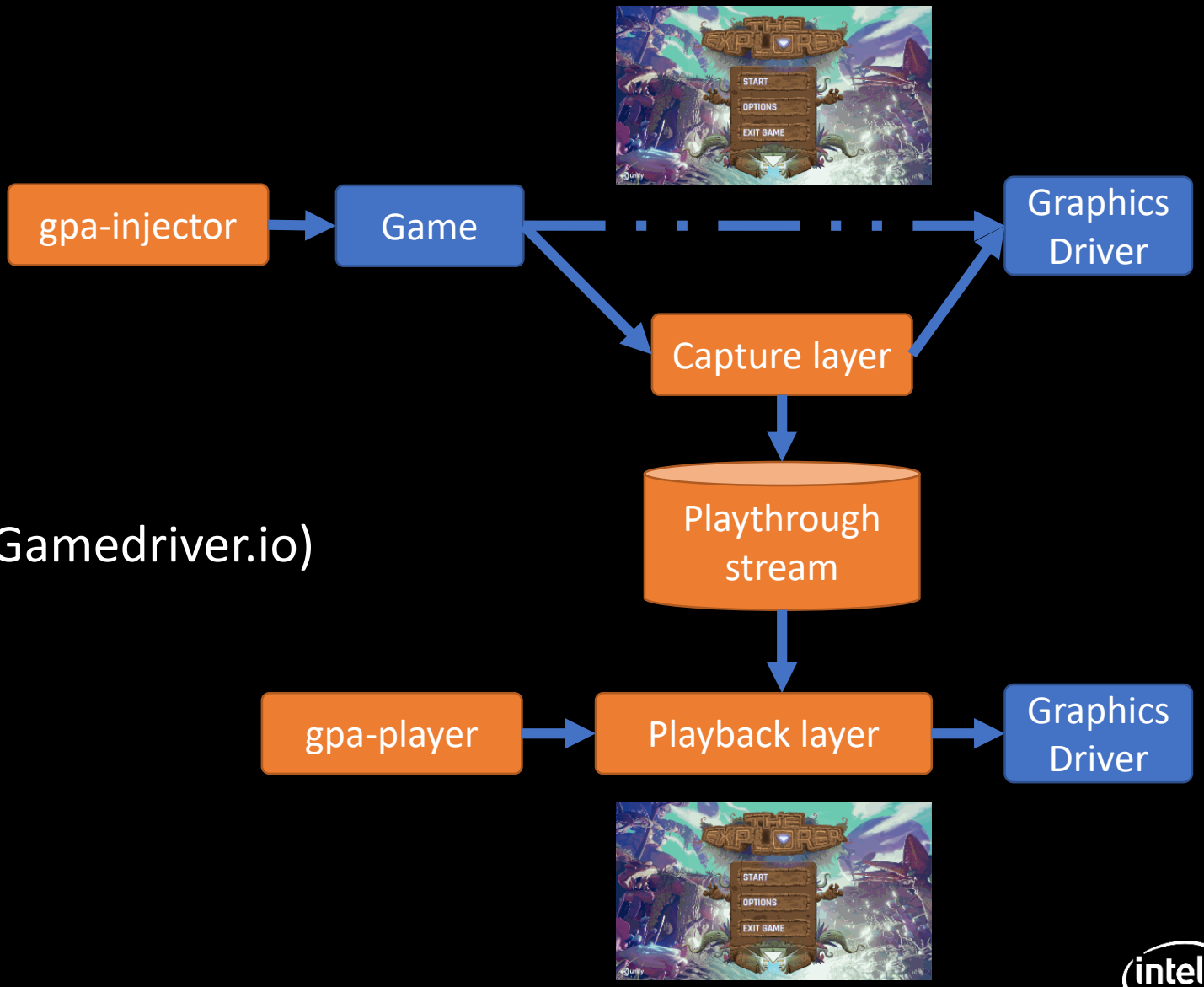
- Benchmark
- Custom scene
- Gameplay unit test system (Gamedriver.io)

CI Workflow

Automated Playthrough

Reproducible scenario

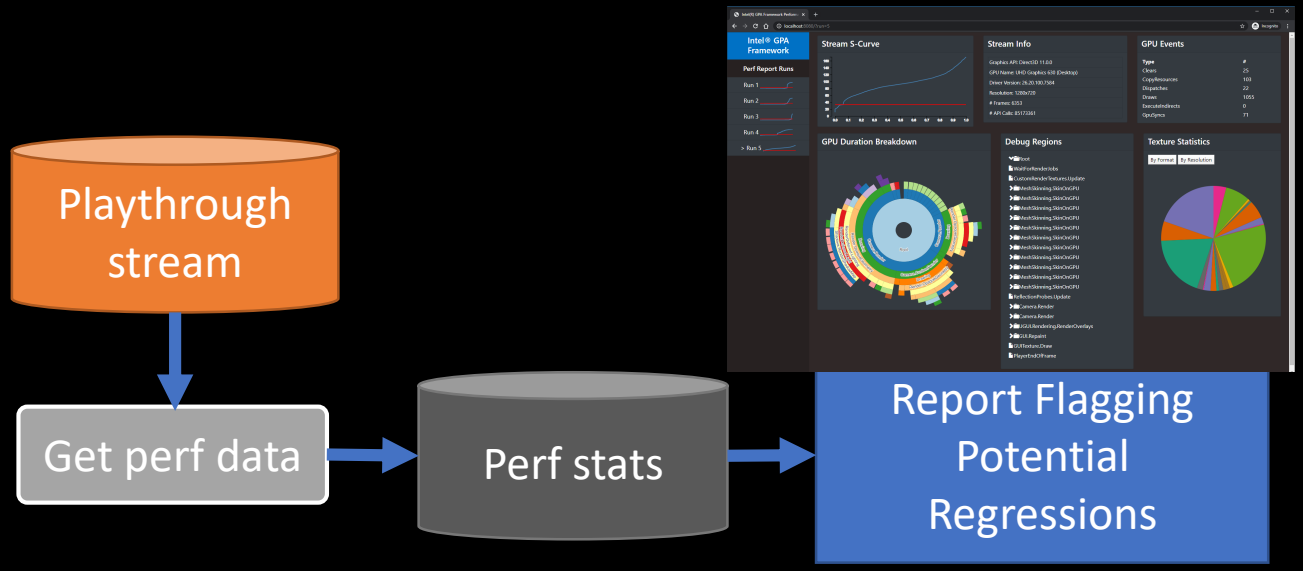
- Benchmark
- Custom scene
- Gameplay unit test system (Gamedriver.io)



CI Workflow

Automated Performance Analysis

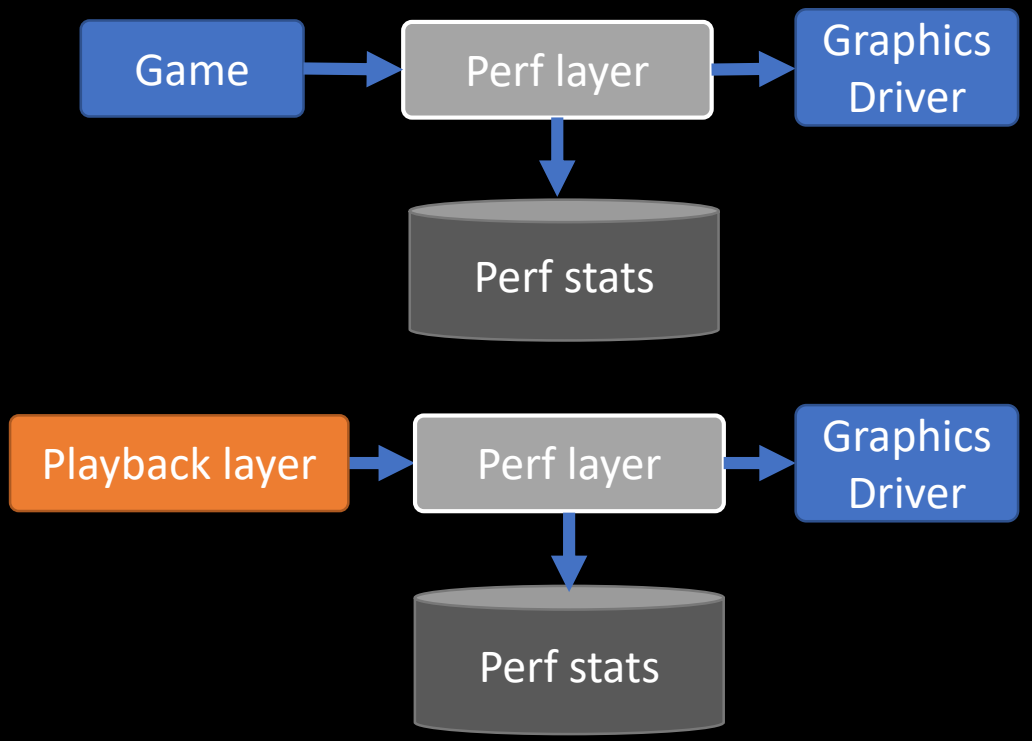
- **Extract performance information, how do we do this?**
  - Custom layer
  - C++ Interface
  - Python Interface



CI Workflow

Automated Performance Analysis

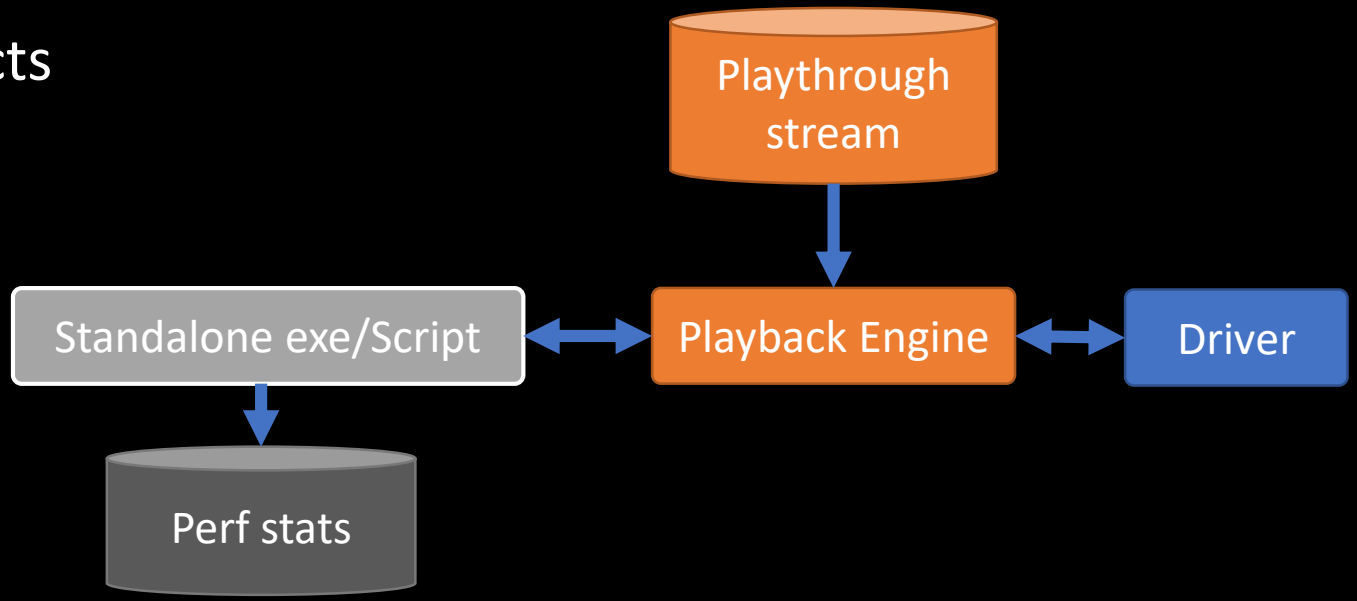
- **Custom layer**
  - ‘Manual’ approach
  - Intercept relevant calls
  - No ability to repeat ranges and experiments
  - Layer can be used in runtime and playback



CI Workflow

Automated Performance Analysis

- **C++/Python Interfaces**
  - Playback only
  - Finer playback control
  - Leverage Framework constructs



Intel® GPA Framework
Python Interface

- Basic sample: Print API log from 20% to 40% of stream

```
1   import GPA
2   import sys
3
4   stream_path = sys.argv[1]
5
6   stream = GPA.open_stream(stream_path)
7   range  = stream.get_range(0.2, 0.4)
8
9   for c in range.callables() :
10      print(c.name())
11
```

Intel® GPA Framework
C++ Interface

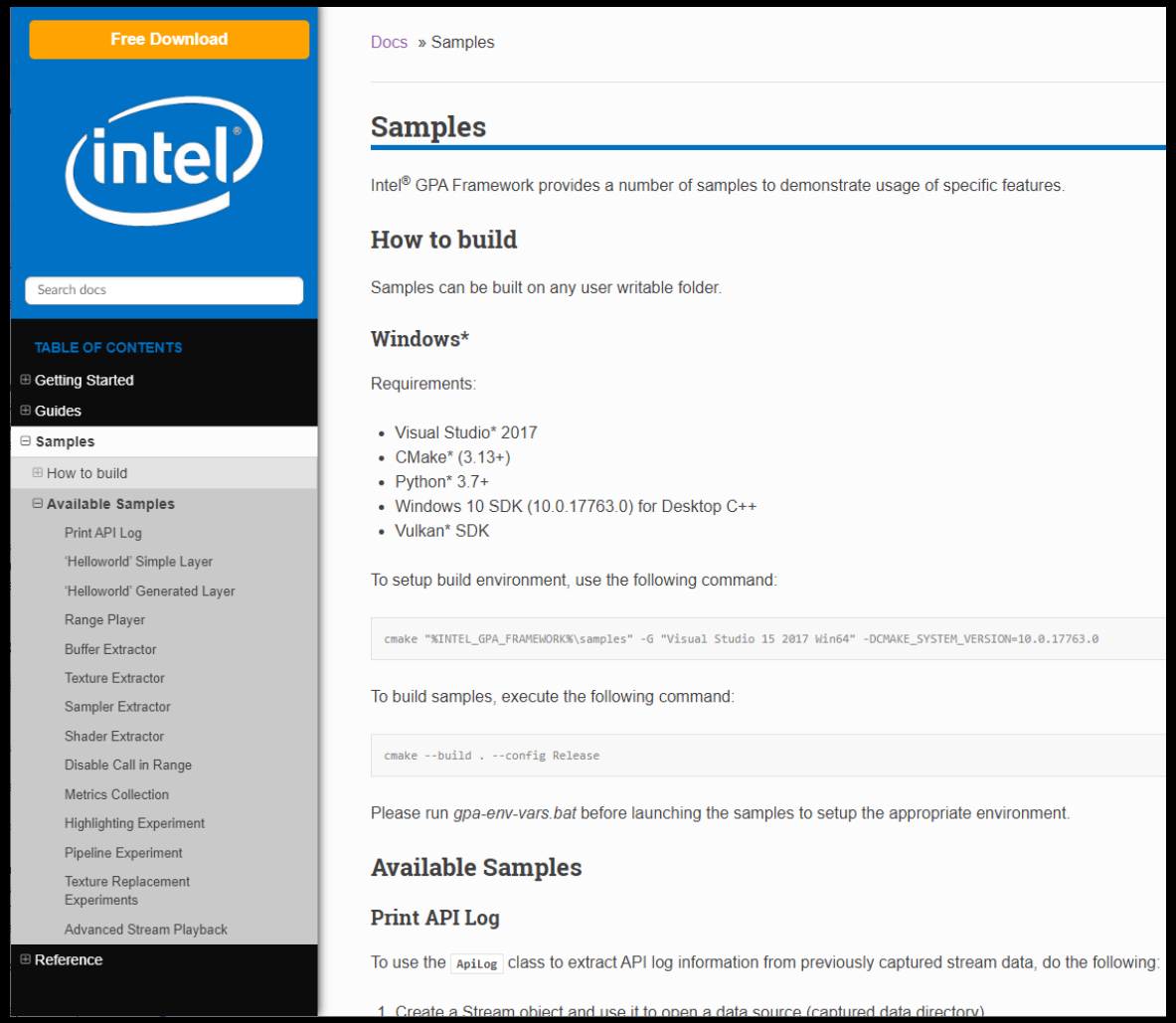
- Basic sample: Print API log from frame **100**

```
1   Stream stream;
2   stream.OpenDataSource([path_to_stream]));
3
4   ApiLog apiLog(&stream);
5   CallableCache cache = apiLog.Range(100);
6
7   cache.EnumerateCallables([](Callable* callable, uint64_t callableIndex) {
8       std::cout << callable->Name() << std::endl;
9   }
```

Intel® GPA Framework

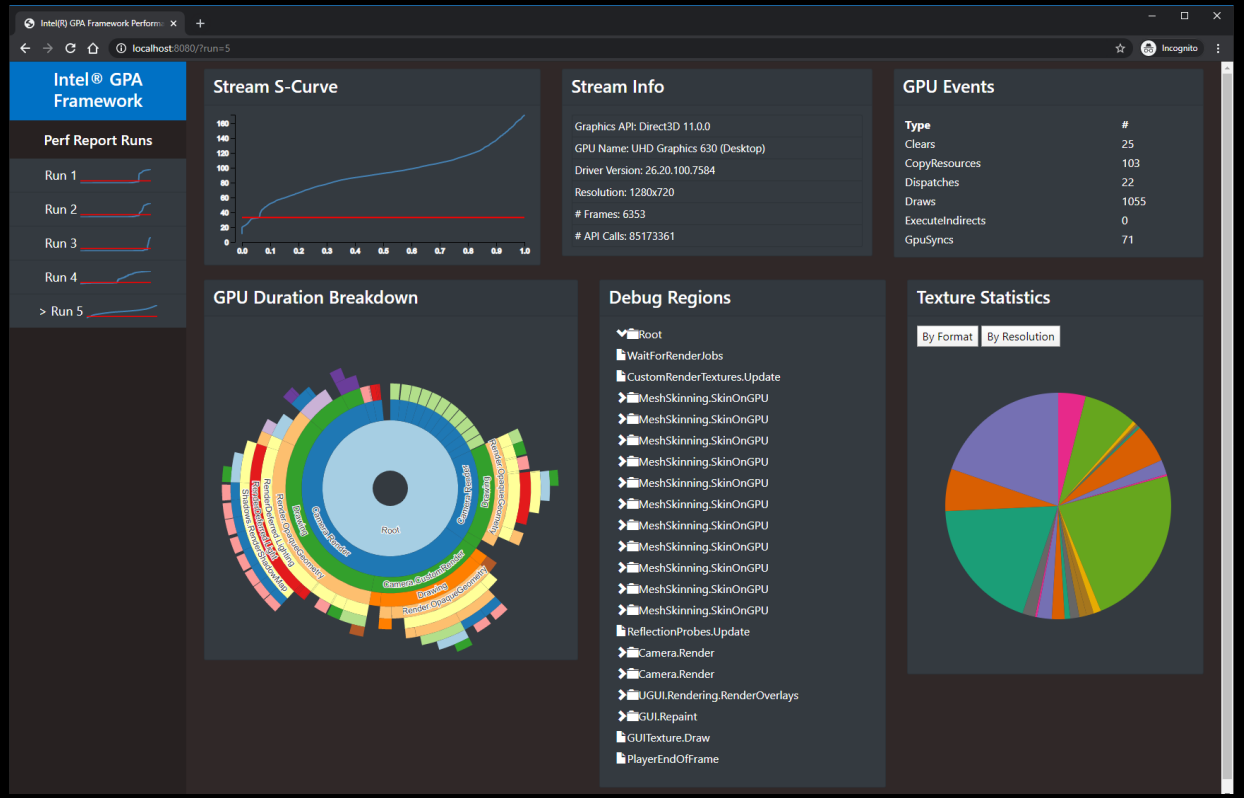
You don’t need to start from scratch!

- Print API Log
- ‘Helloworld’ Simple Layer
- ‘Helloworld’ Generated Layer
- Range Player
- Buffer/Texture/Shader Extractor
- Metadata Extractor
- Disable Call in Range
- Metrics Collection
- Highlighting Experiment
- Pipeline Experiment
- Texture Replacement Experiments
- Advanced Stream Playback



Automated Performance Analysis

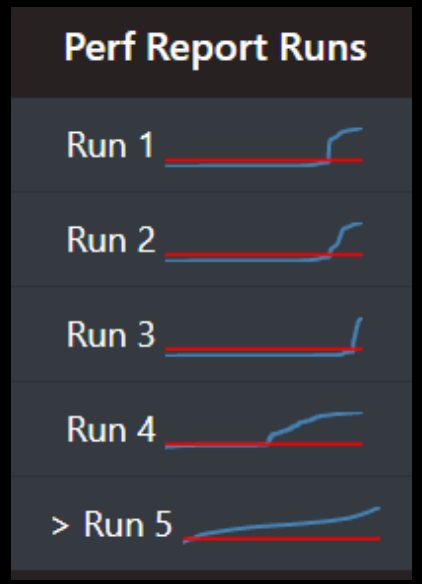
- Print API Log
- ‘Helloworld’ Simple Layer
- ‘Helloworld’ Generated Layer
- Range Player
- Buffer/Texture/Shader Extractor
- Metadata Extractor
- Disable Call in Range
- Metrics Collection
- Highlighting Experiment
- Pipeline Experiment
- Texture Replacement Experiments
- Advanced Stream Playback

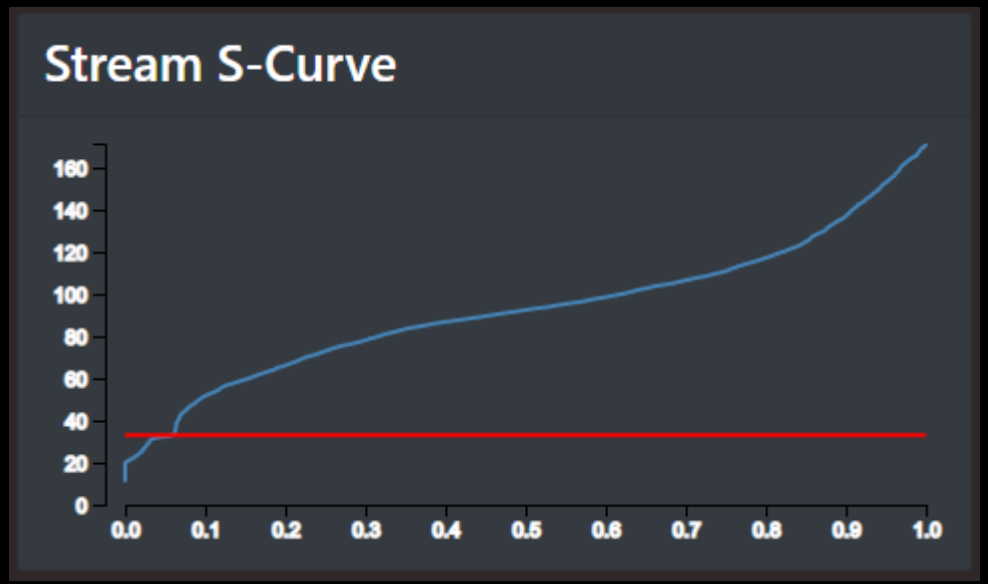


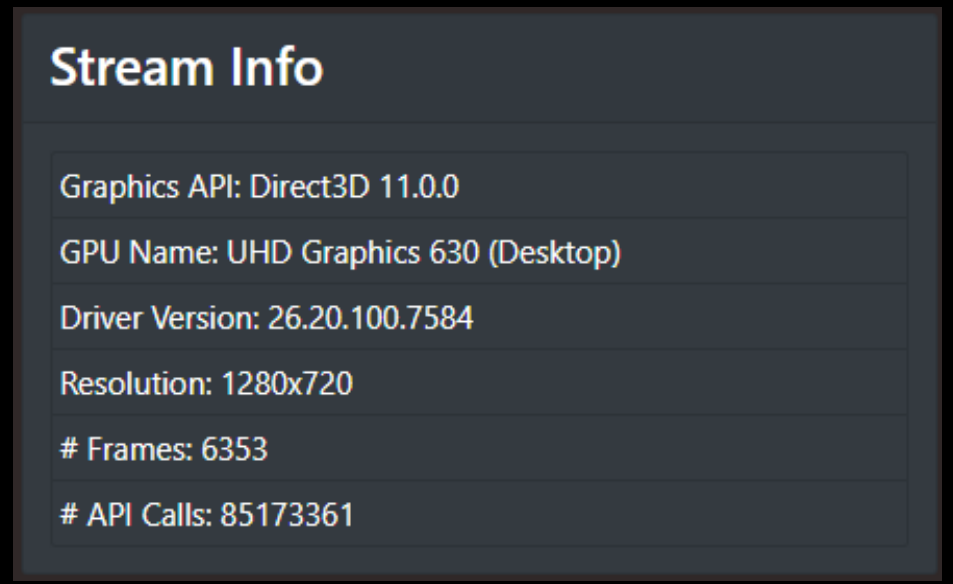
Automated Performance Analysis

Using built-in utilities

gpa-stream-analyzer provides capture time frame times of stream
gpa-stream-info reports information from stream header





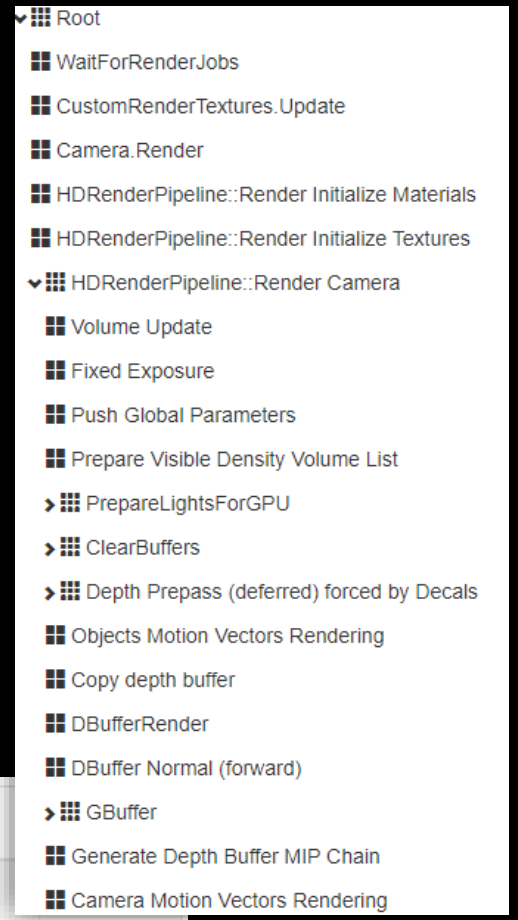


Automated Performance Analysis

Using sample: Print API Log

- Query ‘callable types’ to construct debug regions hierarchy tree
- Generate GPU event statistics like number of draws, clears, etc.

```
1   DebugRegion tree;
2   cache.EnumerateCallables([](Callable* callable, uint64_t callableIndex)
3   {
4       if (callable->GetCategory() & CallableType::BeginUserRegion)
5       {
6           // New child, add to DebugRegion tree
7       }
8
9       // Handle EndUserRegion
10      // Collect stats on other categories
11      …
12
13      std::cout << callable->Name() << std::endl;
14  }
```

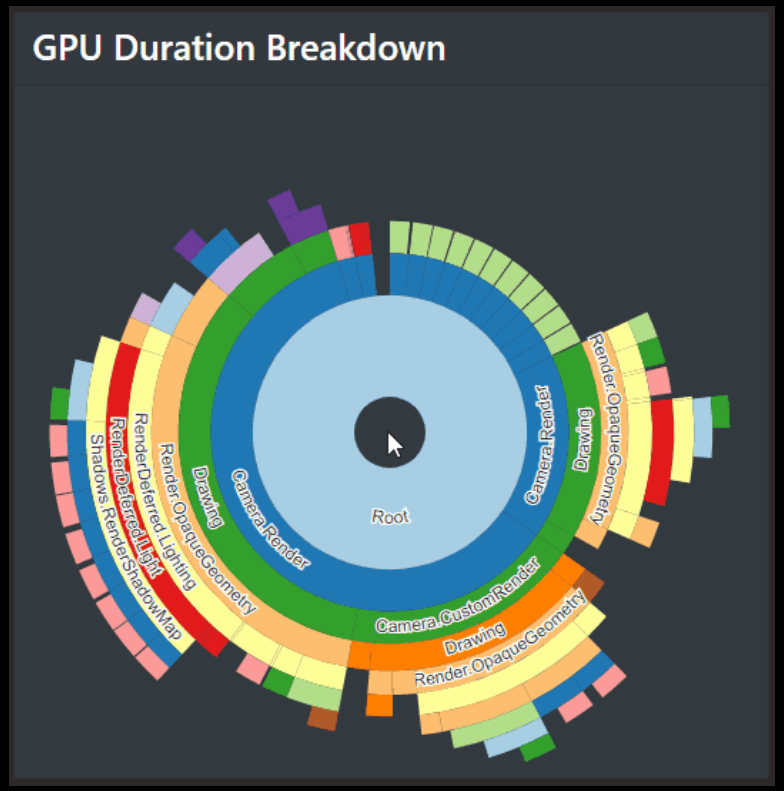


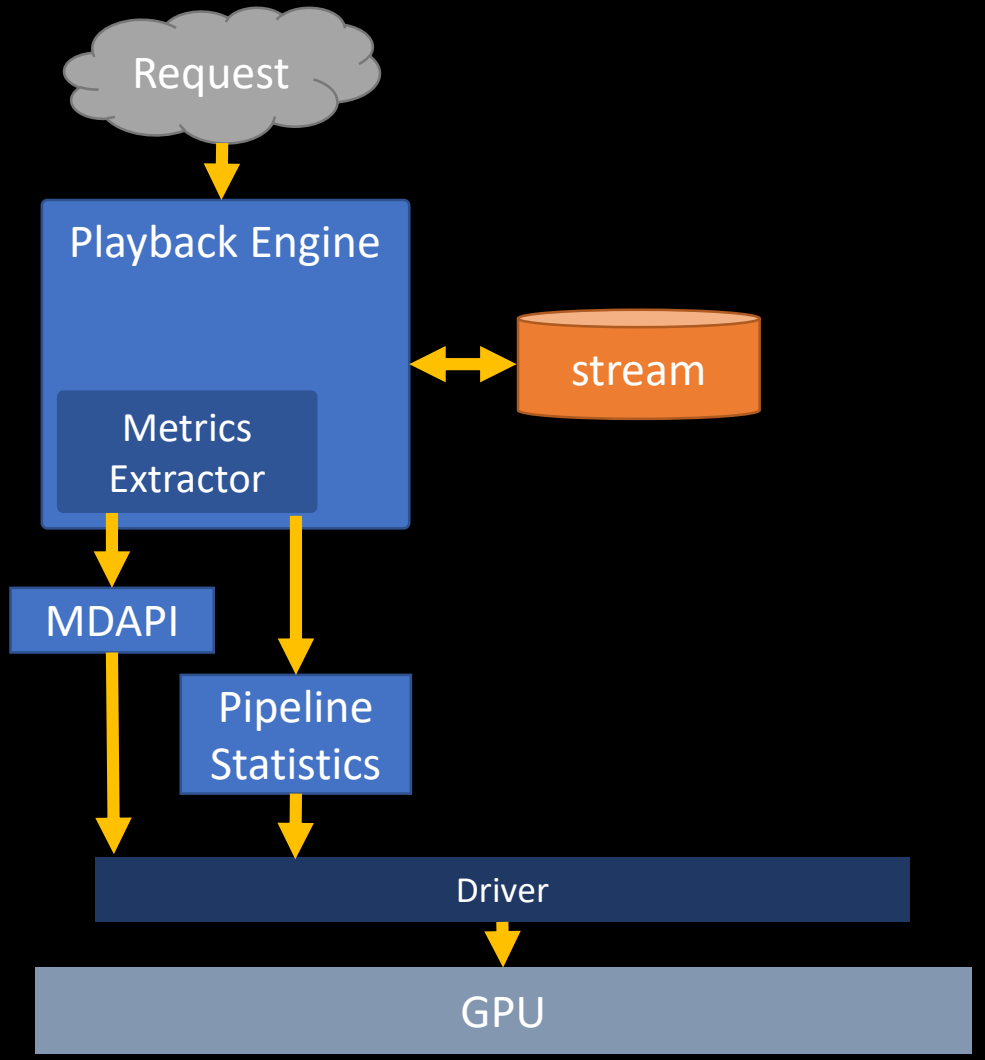
Statistic	
Clear	
CopyResource	8
Dispatch	51
Draw	292
ExecuteIndirect	29
GpuSync	207

Automated Performance Analysis

Using sample: Metrics Collection

- Query GPU duration per debug region
- Store other metrics for later analysis

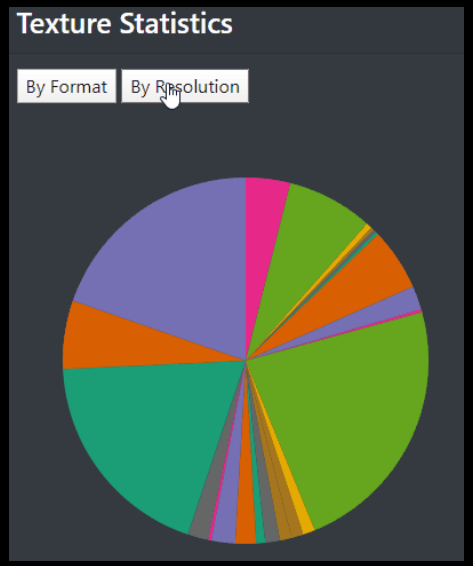


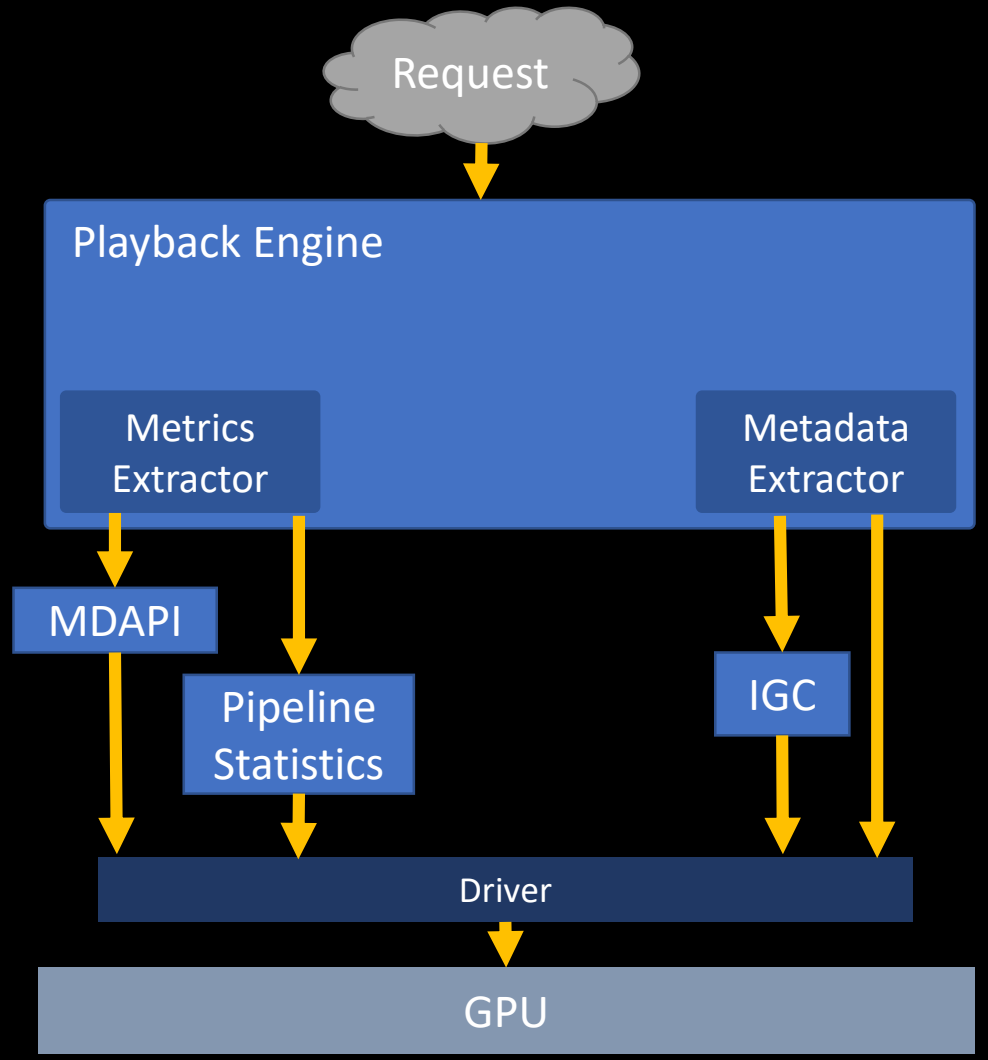


Automated Performance Analysis

Using sample: Metadata Extractor

- Query frames for used Textures/PSOs, etc.
- Generate statistical data and flag problematic usage (big textures, uncompressed formats, etc.)

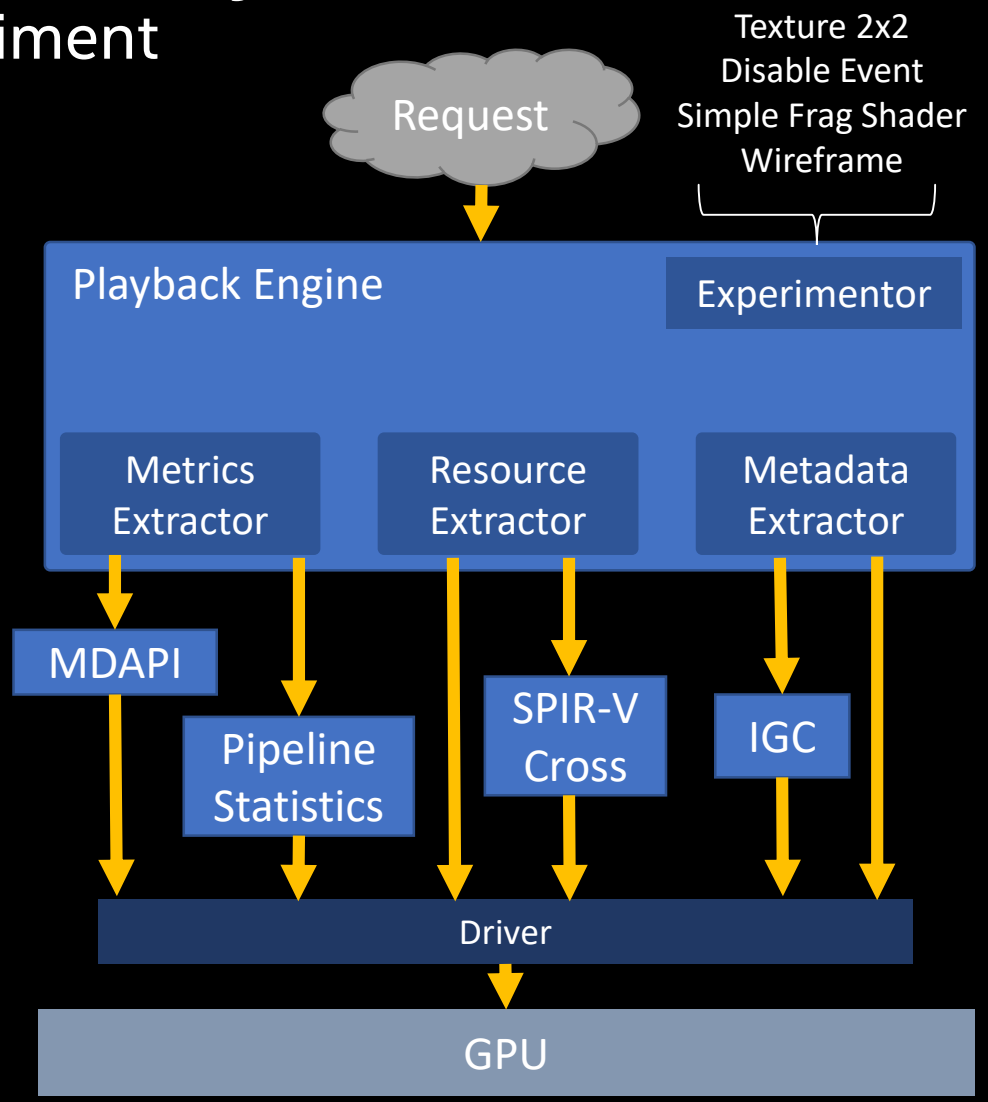




Automated Performance Analysis

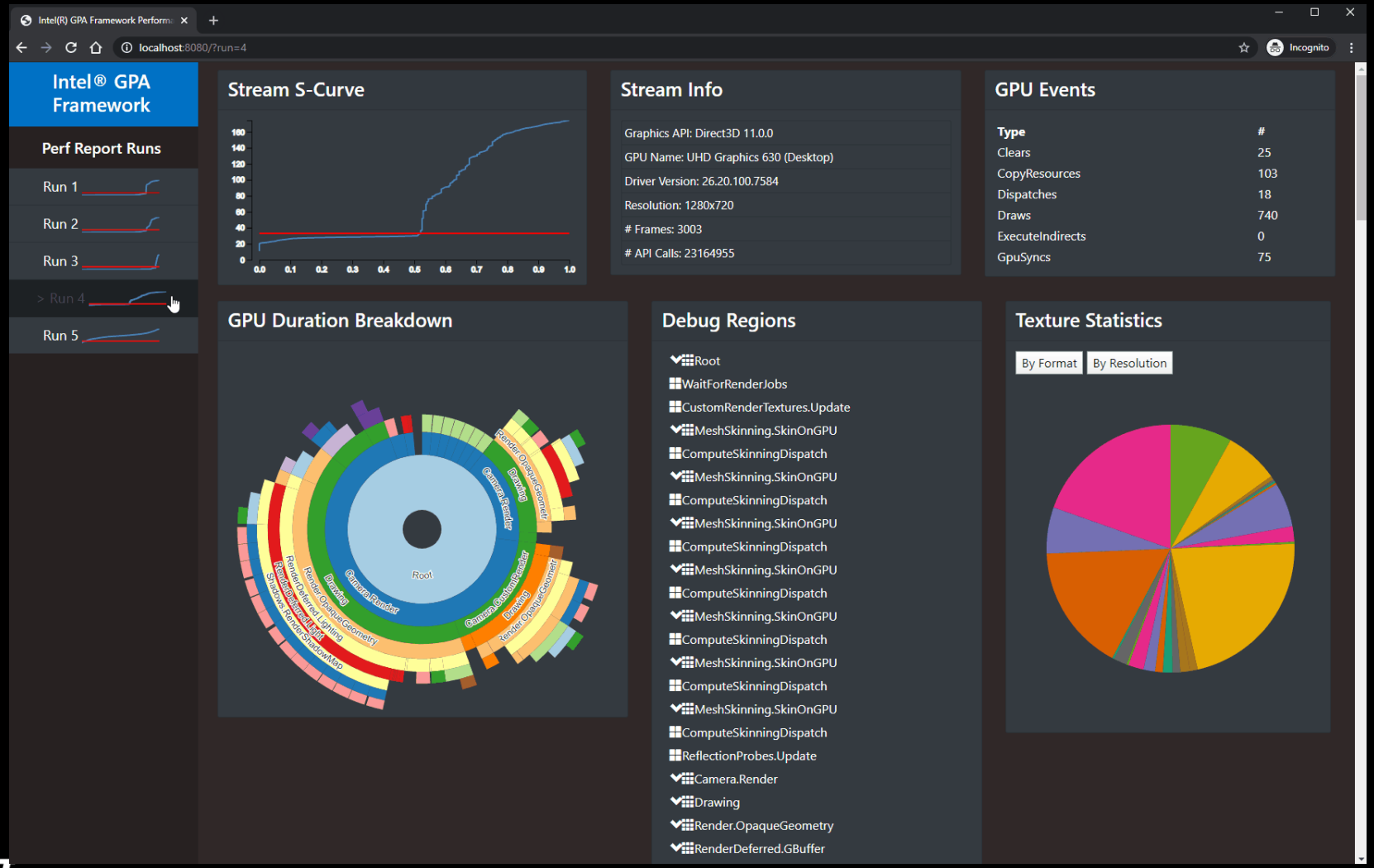
Using sample: Texture replacement experiment

- Detect problematic textures (large, uncompressed, etc.)
- Apply Texture 2x2 experiment
- Replay and gather metrics
- Report performance difference



Automated Performance Reporting

Sample available in Intel® DevMesh!

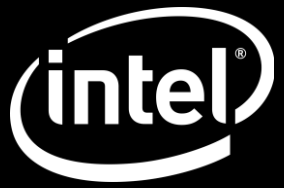


Summary

- Performance awareness is everyone's responsibility
- Intel® GPA is more than just Frame Analyzer
- Synergy of tools can help you gain more insight on your game's performance
- Multiframe Analysis opens up new possibilities for optimization
- CI/CD profiling usage of Intel® GPA Framework just the tip of the iceberg
- Twitter: @carlosadc

What's next?

- Grab Intel® GPA for free
  https://software.intel.com/en-us/gpa
- Intel® GPA Documentation:
  https://software.intel.com/en-us/gpa/documentation/view-all
- Developer and Optimization Guide for Intel® Processor Graphics Gen11 API:
  https://software.intel.com/en-us/articles/developer-and-optimization-guide-for-intel-processor-graphics-gen11-api
- Intel® Graphics Performance Analyzers Cookbook:
  https://software.intel.com/en-us/gpa-cookbook



Questions?