Using Render Doc

UE and Oculus Quest 2

Rob Segal
Get Set Games

Who am I?

- I’m Rob!
- Co-Founder/Technical Director at Get Set Games
- Using Unreal for about 8 years
- Currently doing some exploration work in VR on Quest 2
- I write occasionally at **sarcasticcoder.com**

What are we talking about?

- RenderDoc. What is it?
- Non documented cases in setup and usage
- Example usage scenarios
  - Auto instancing
  - Distance Culling

Not covering...

- Unreal/RenderDoc integration
- General usage of RenderDoc
- CPU or GPU bound

Development Setup

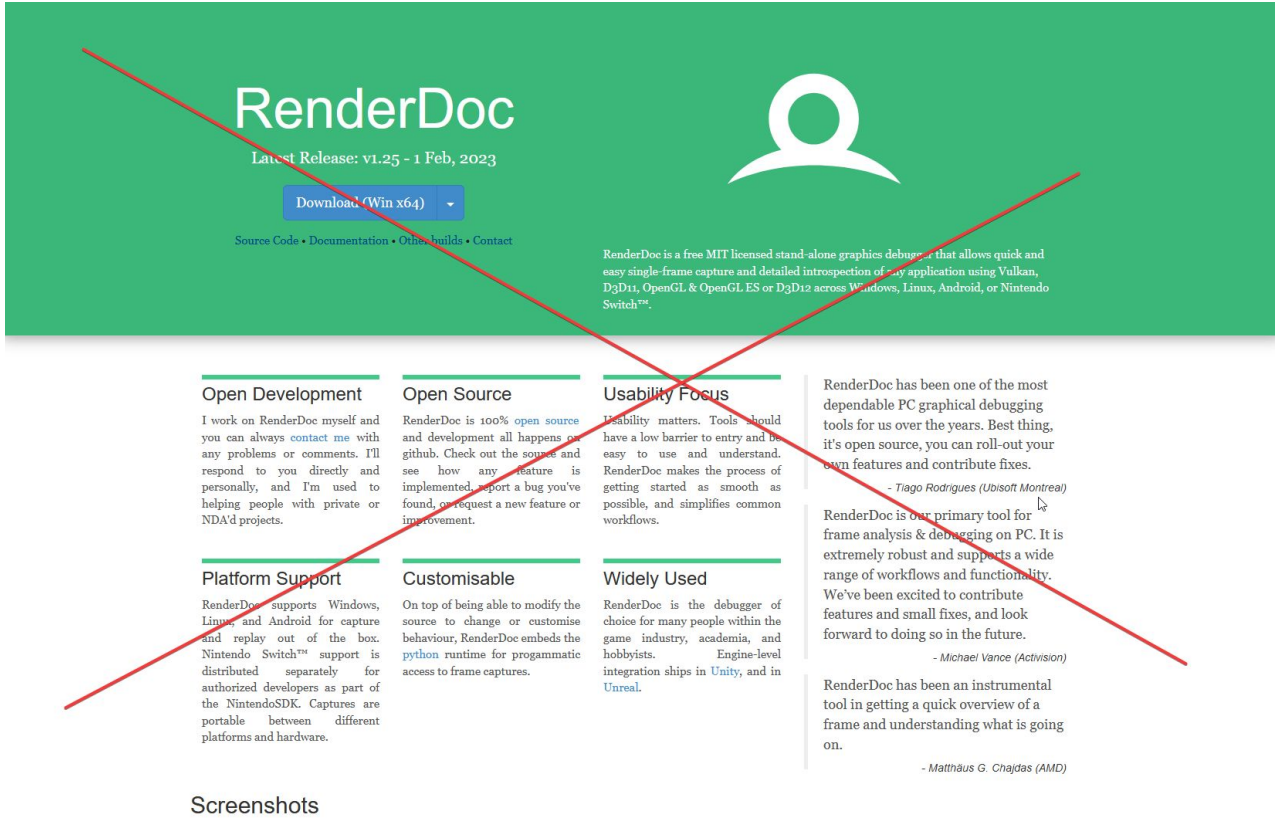
- UE 4.27.2
  - Github Oculus Fork
- Quest/Quest 2
- Render Doc
  - Meta Fork - v44.1 (forked from v1.22)

Development Setup

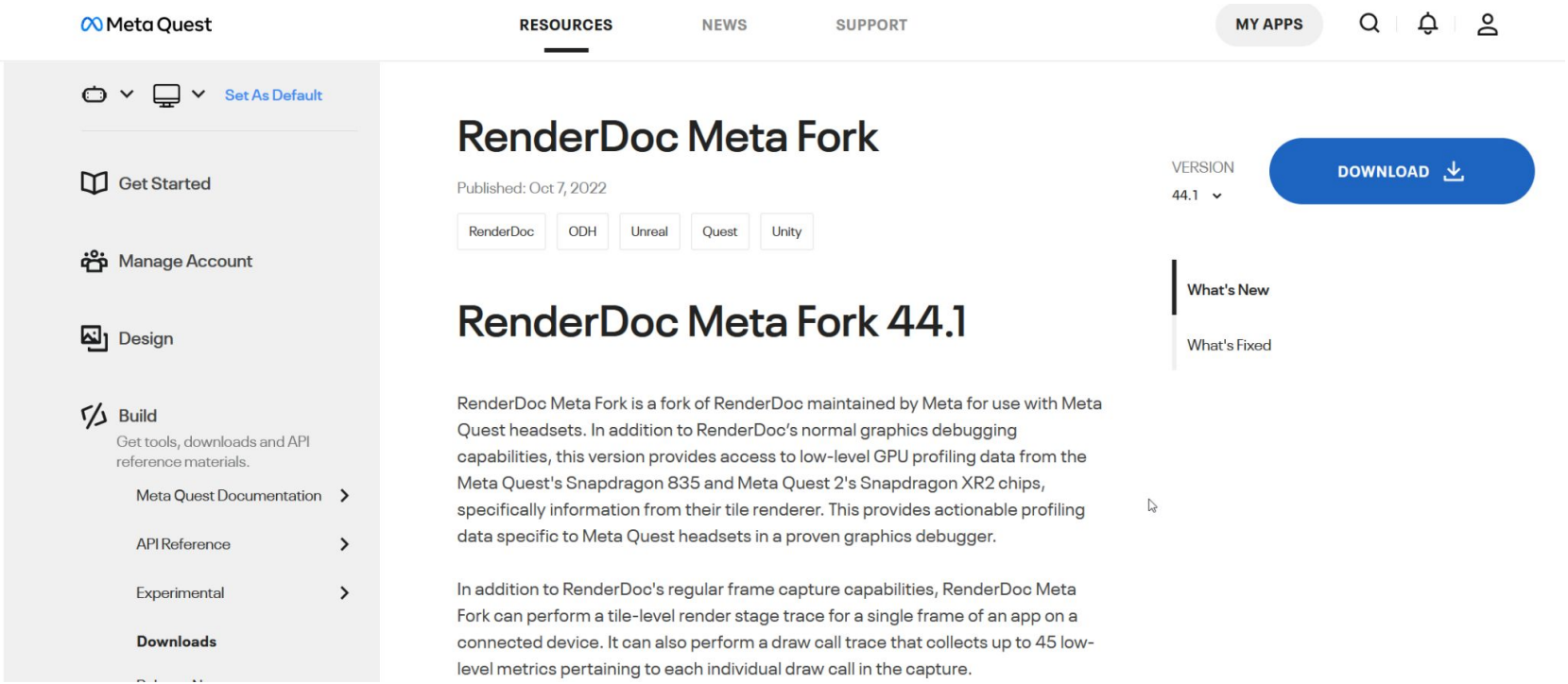
- Build engine from source
  - Distribute through UGS (Unreal Game Sync)
- Using Vulkan but equally applicable to OpenGL projects
- Development, Test and Shipping configuration built nightly

What is RenderDoc?

A graphics debugger that allows quick and easy single-frame capture and detailed introspection of any application



Render Doc... but not the one you want



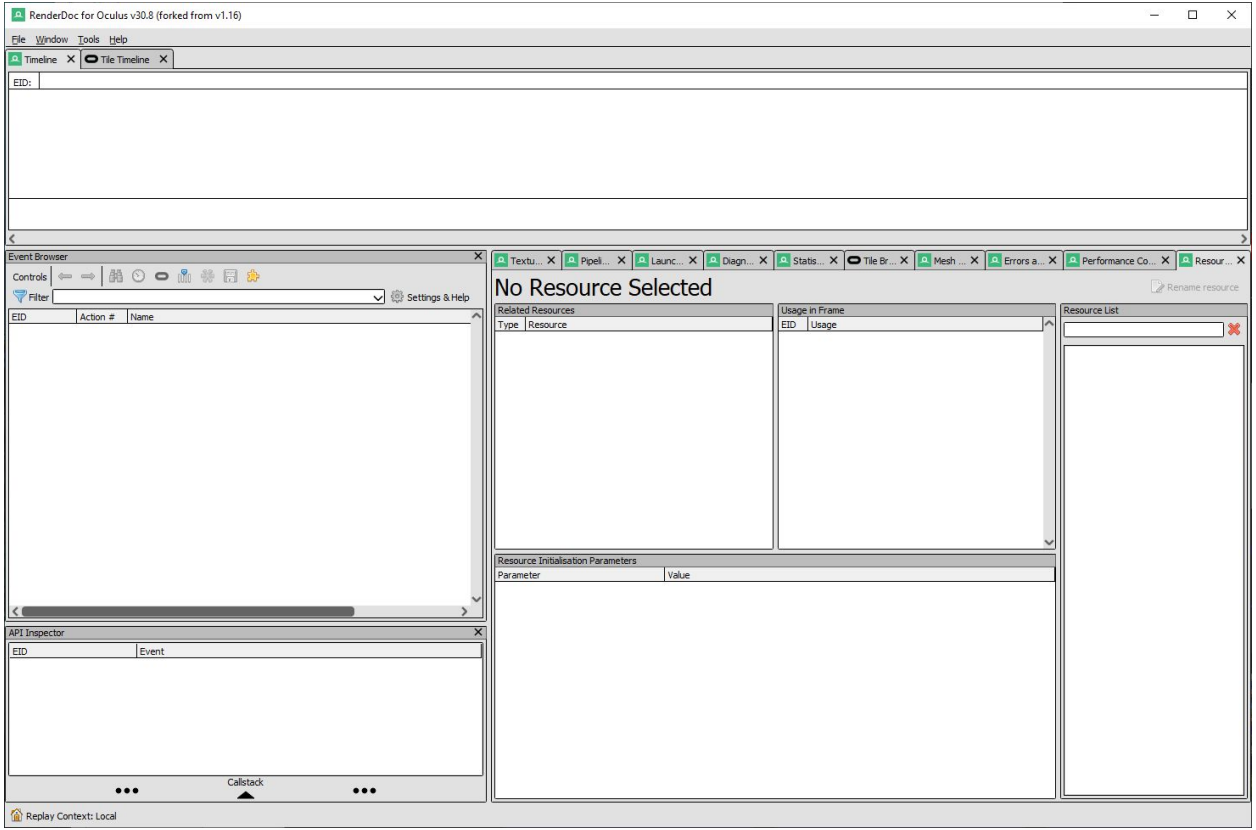
Render Doc - Meta Fork!

Why or when would I use it?

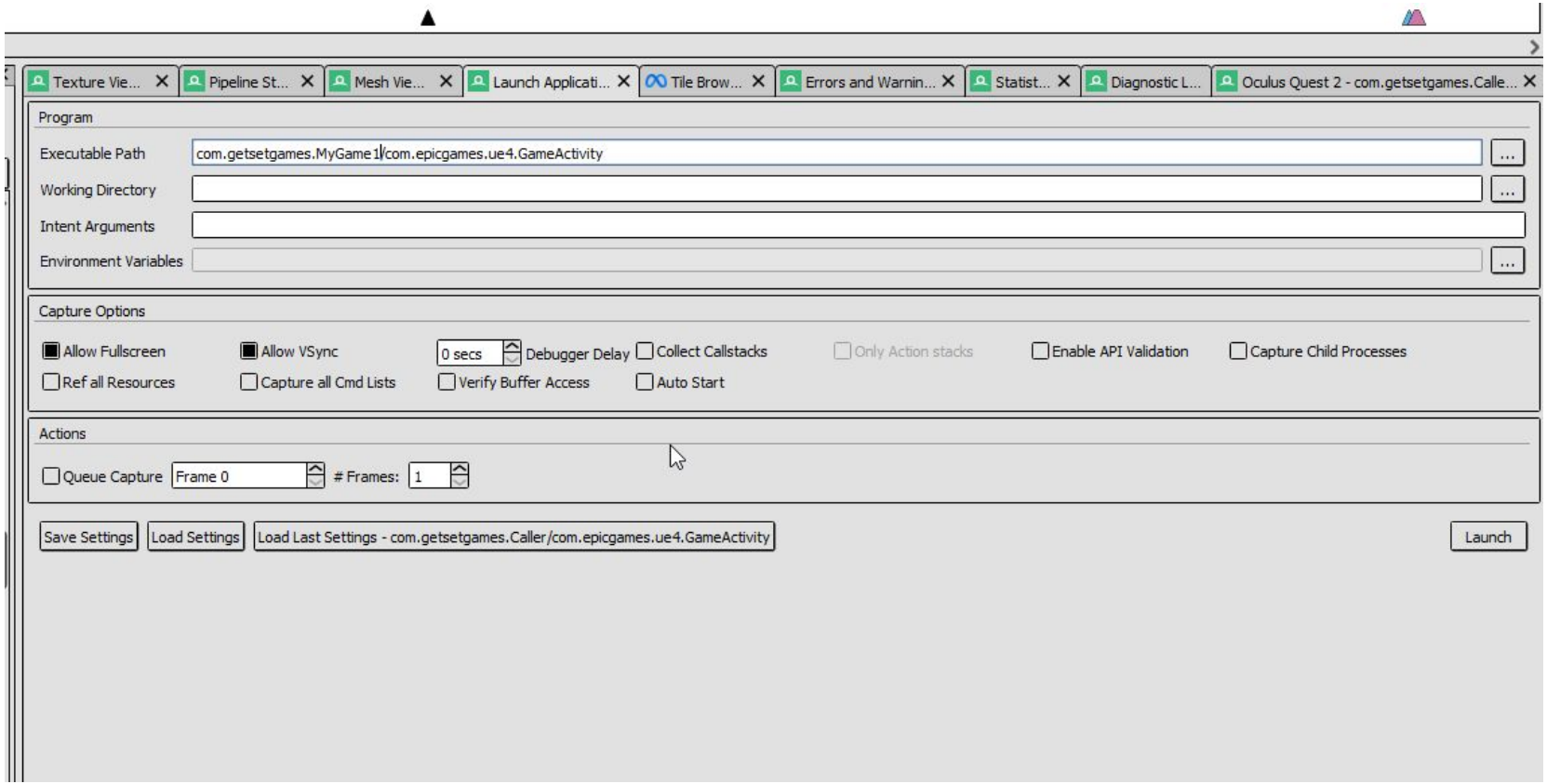
- GPU bound
- Determining draw call count
- Review contents of render buffers (stencil, depth, etc.)
- Interested/curious about what the GPU is doing

What can it do?

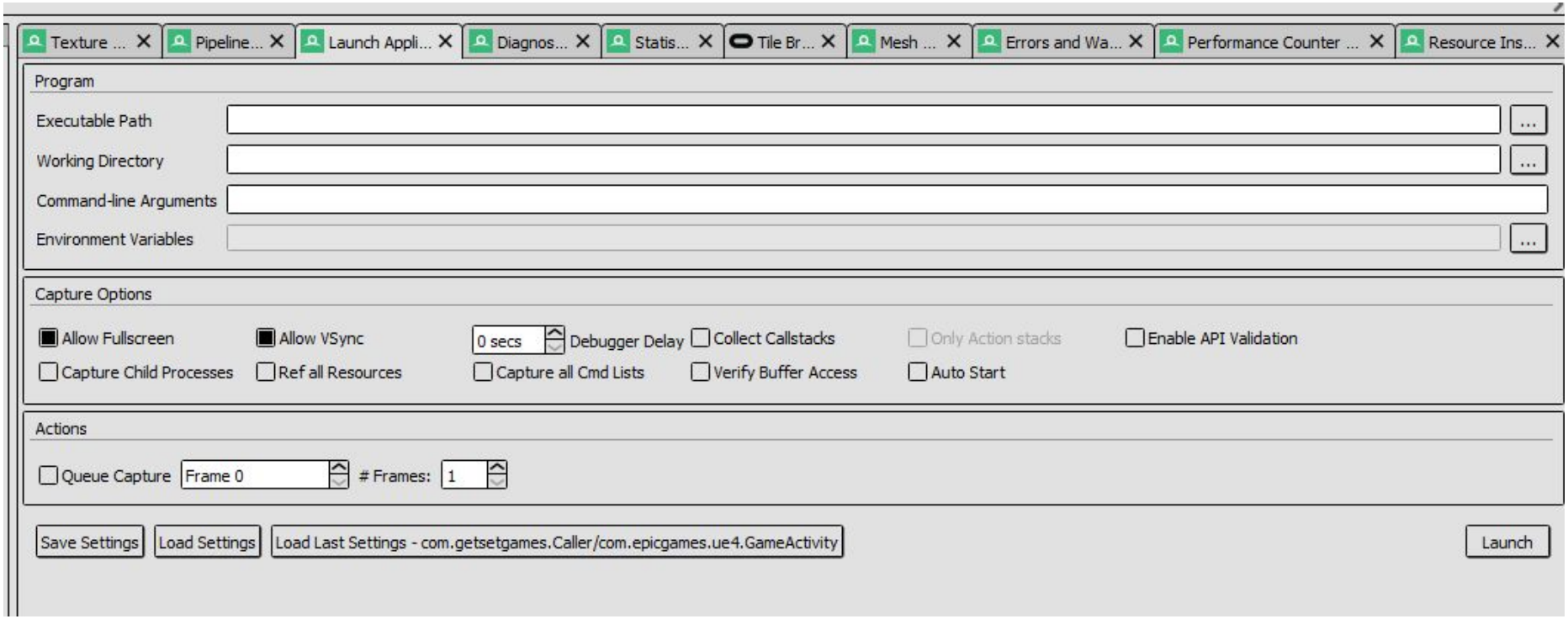
- Review frame draw call count
- Review buffer usage (stencil, depth, etc.)
- Shader debugging/analysis
- Verify GPU instancing of shared meshes
- Texture format and resolution verification



Render Doc main window



Launching a VR Application



Launching a VR Application

While profiling tools can provide data about how your project is performing in VR, there are some points to consider when interpreting those values.

Delays can appear in odd places in the GPU and CPU profilers, most often as occlusion or scene graph traversal time. If those numbers are abnormally high, it's possible that it's a false positive.

> **TIP**
> Often when profiling, you'll notice that you pop between 90 Hz and 45 Hz. The difference is accounted for because the Compositor acts much like a vsync. If you miss framerate, it delays you until the next frame entirely. So, you tend to go down in brackets of 90 / n, where n is a whole number. Because the fps reporting is an average, it won't always report as a whole number, unless the drop is sustained. If you're bouncing back and forth between making framerate and not, you may see some fraction of a jump.

If you'd like to remove this for testing, sometimes it's useful to run the game emulating stereo rendering, instead of running in the device itself. To do so:

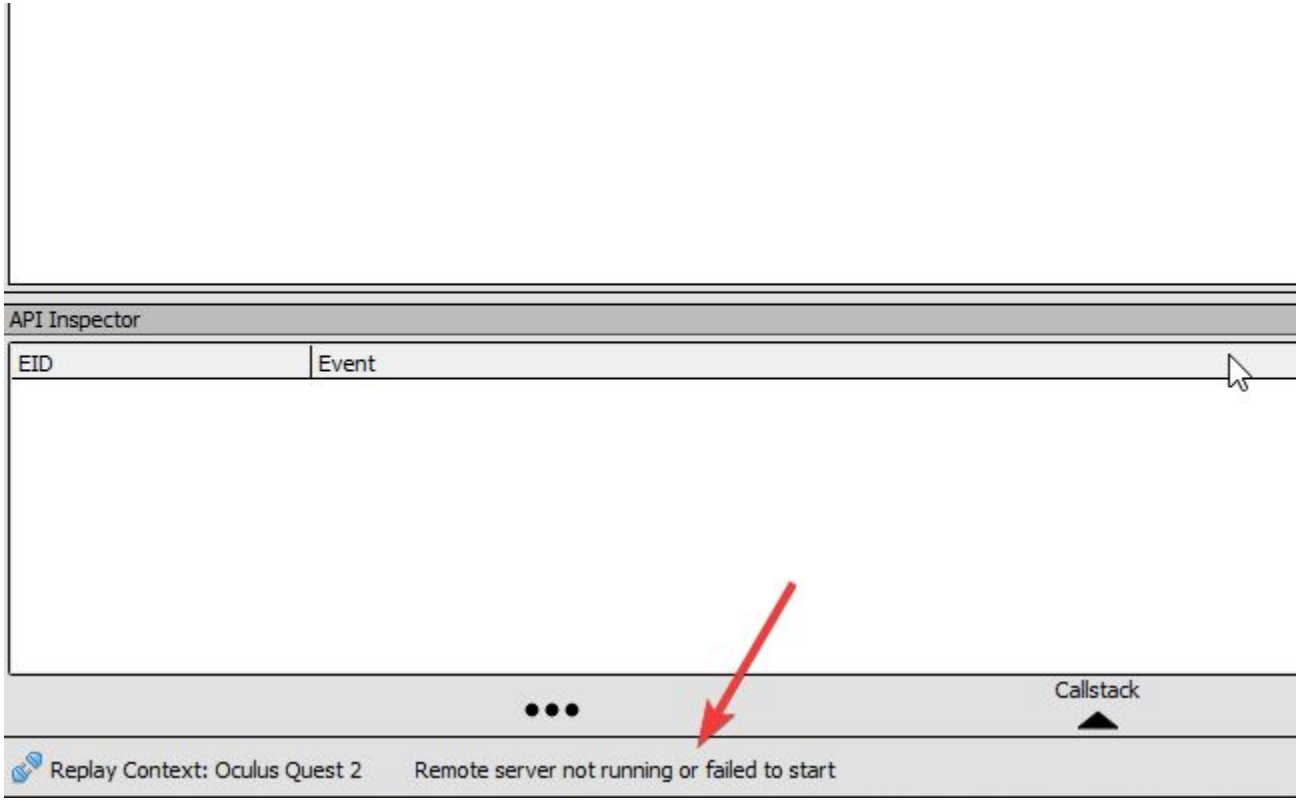
- Launch the game with `-game -emulatestereo -res=2160x1200` on the commandline
- Ensure vsync is off with `r.vsync 0` in the console
- Update the screen percentage to emulate the oversampling we have to do for VR with `r.screenpercentage 137` in the console

This will emulate the GPU and CPU performance characteristics without the annoyances of the variable framerate.

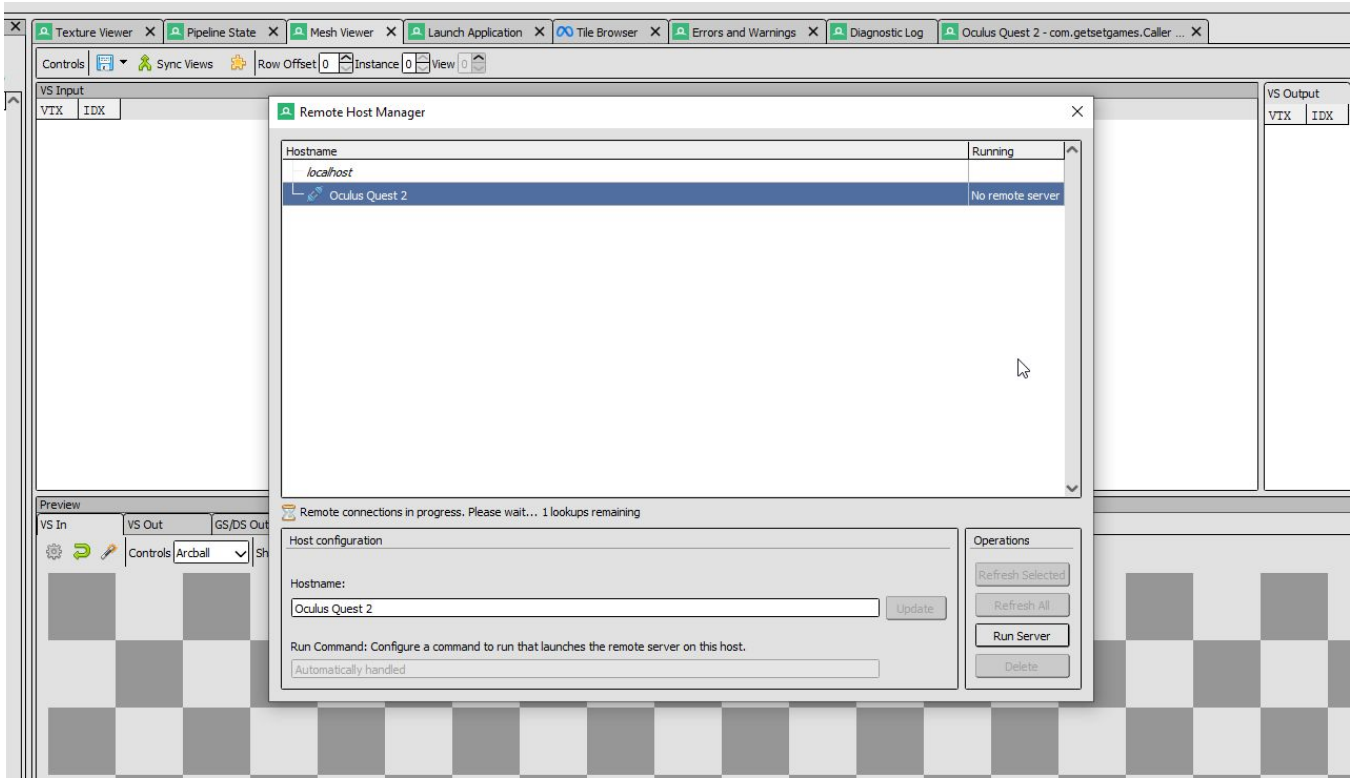
It's worth explicitly noting that if you're hitting 90 frames a second most of the time, and then make a change and notice a drastic drop, you're probably falling prey to the Compositor "vsync" issue noted above. As soon as you tip over the cliff, you'll see drastic changes in numbers.

Launching a VR Application - PC

Render Doc service running on device



Remote server not running or failed to start

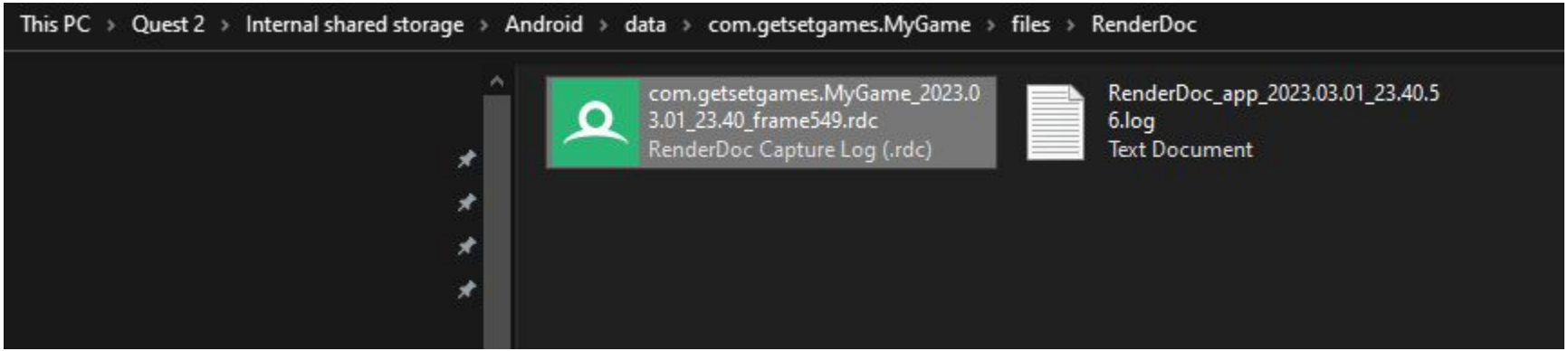


Attach to running instance

Starting the Render Doc service

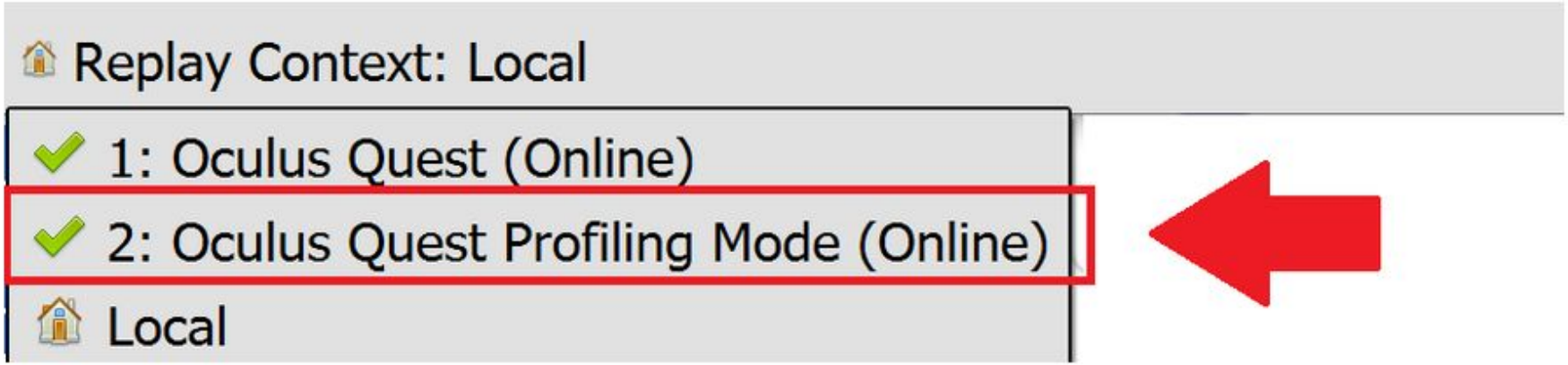
Launching via Android Debug Bridge (ADB) shell...

```
adb '-s 1XXXXXXXXXX shell am start -n com.oculus.renderdoccmd.arm64/.Loader -e renderdoccmd
remoteserver'
```

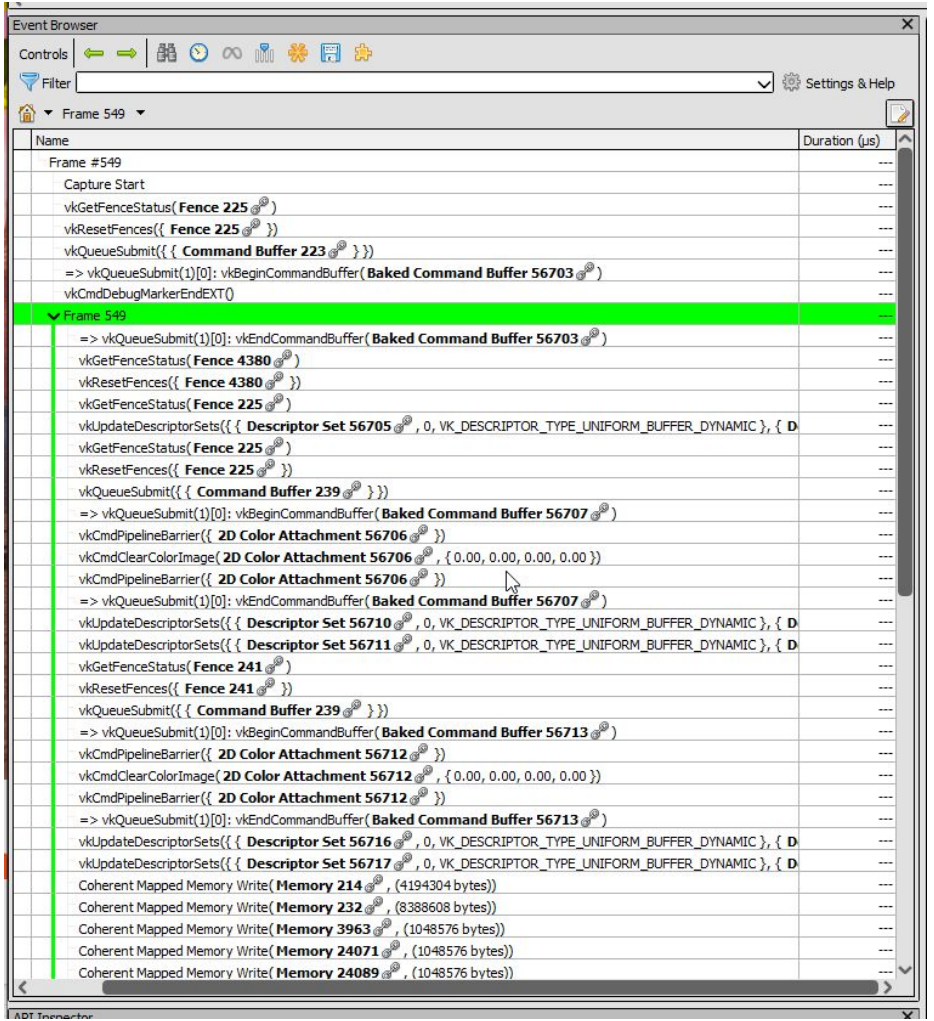


Render Doc capture on headset

Profiling mode

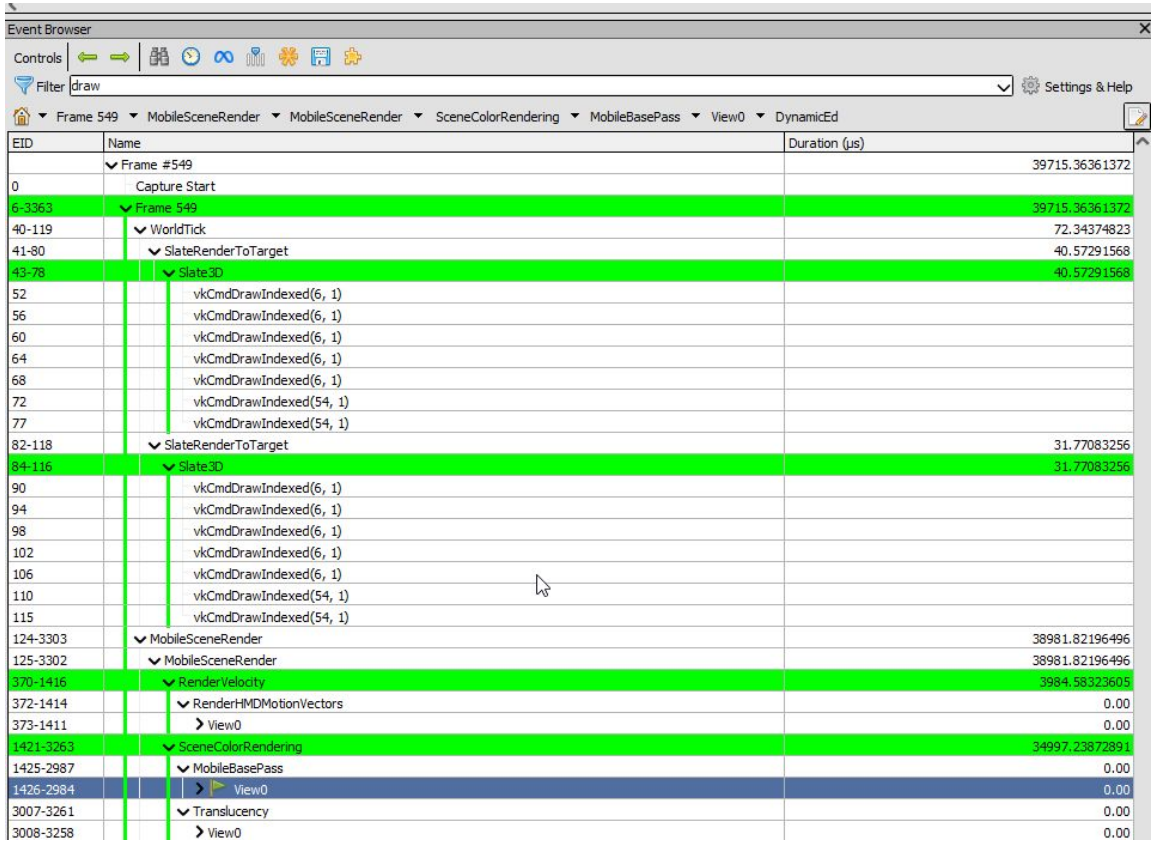


There is alot of noise....



Much better...

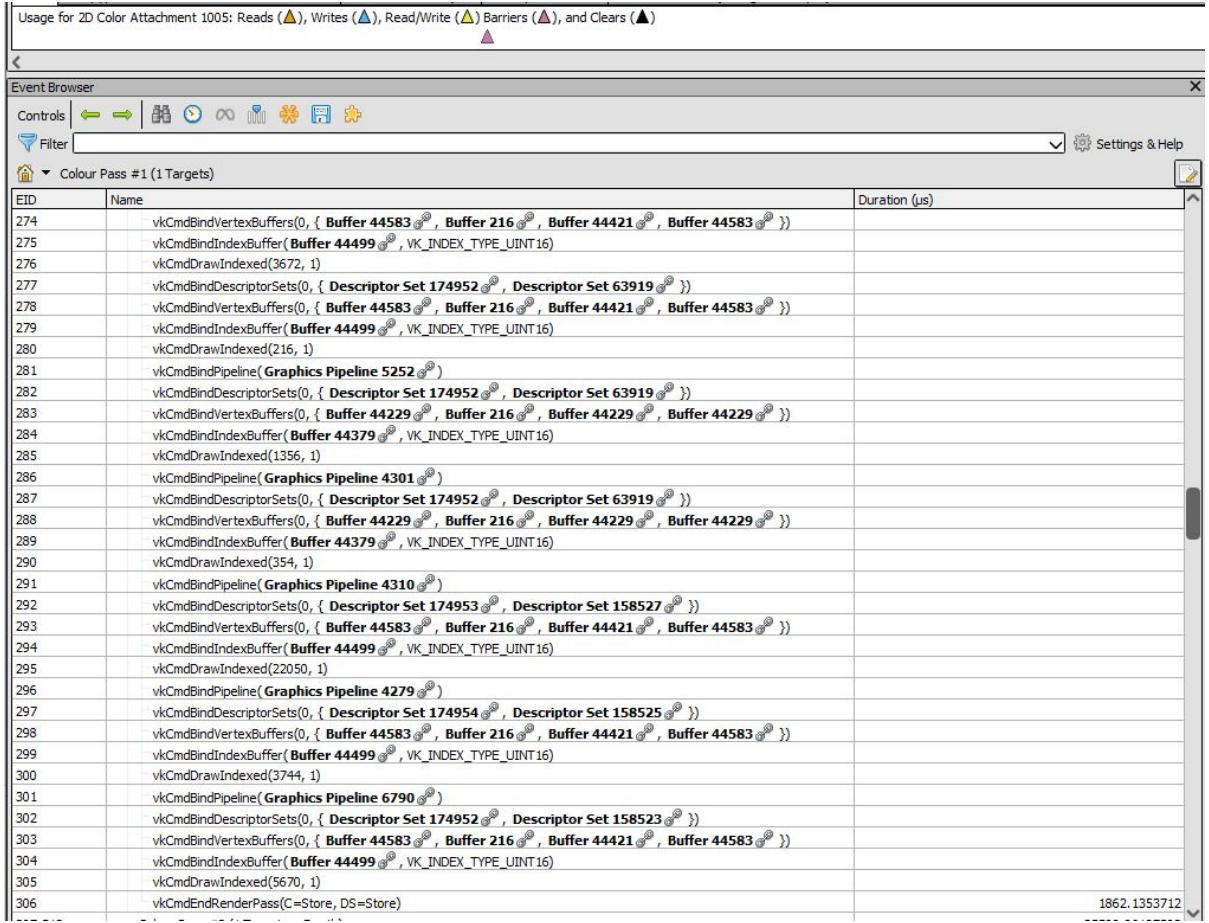
Filter using keyword “draw”



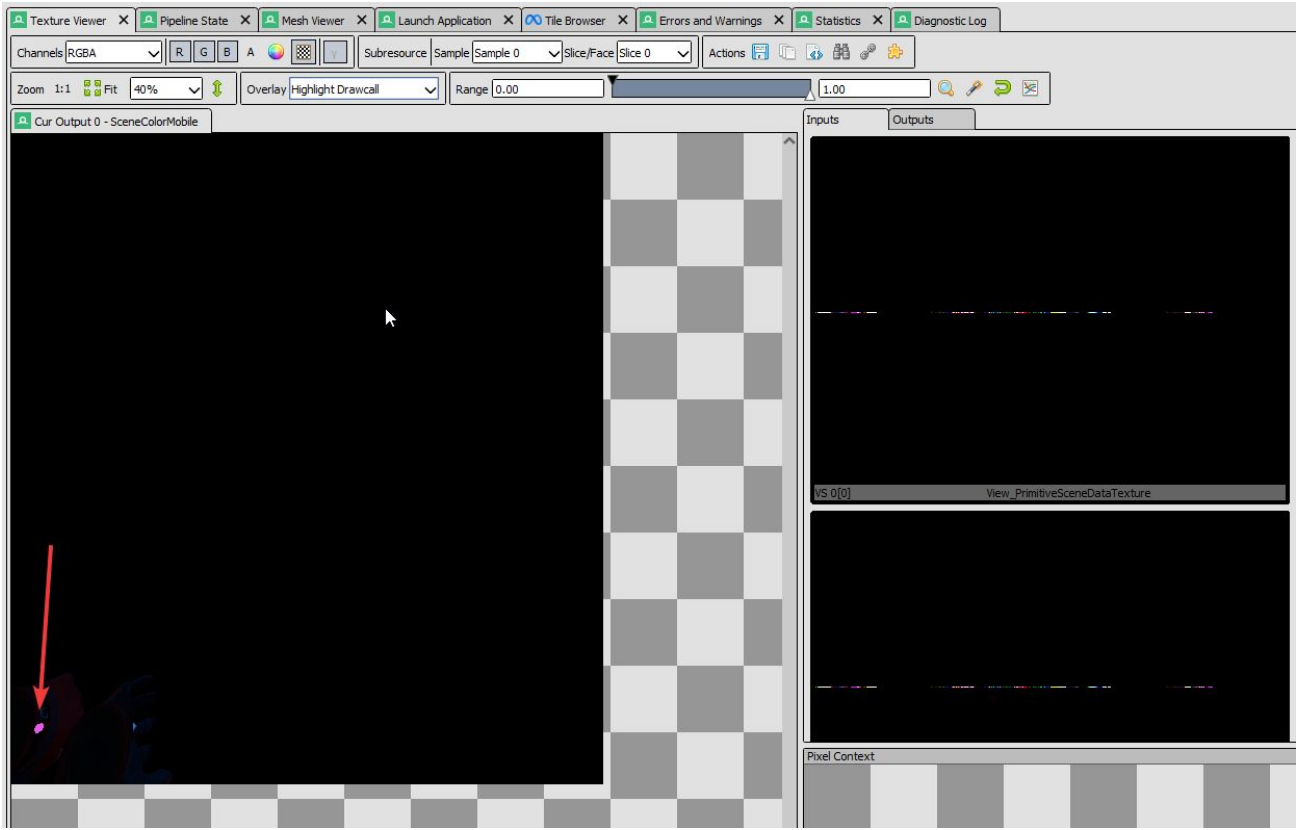
Use Development builds for captures

Less good...

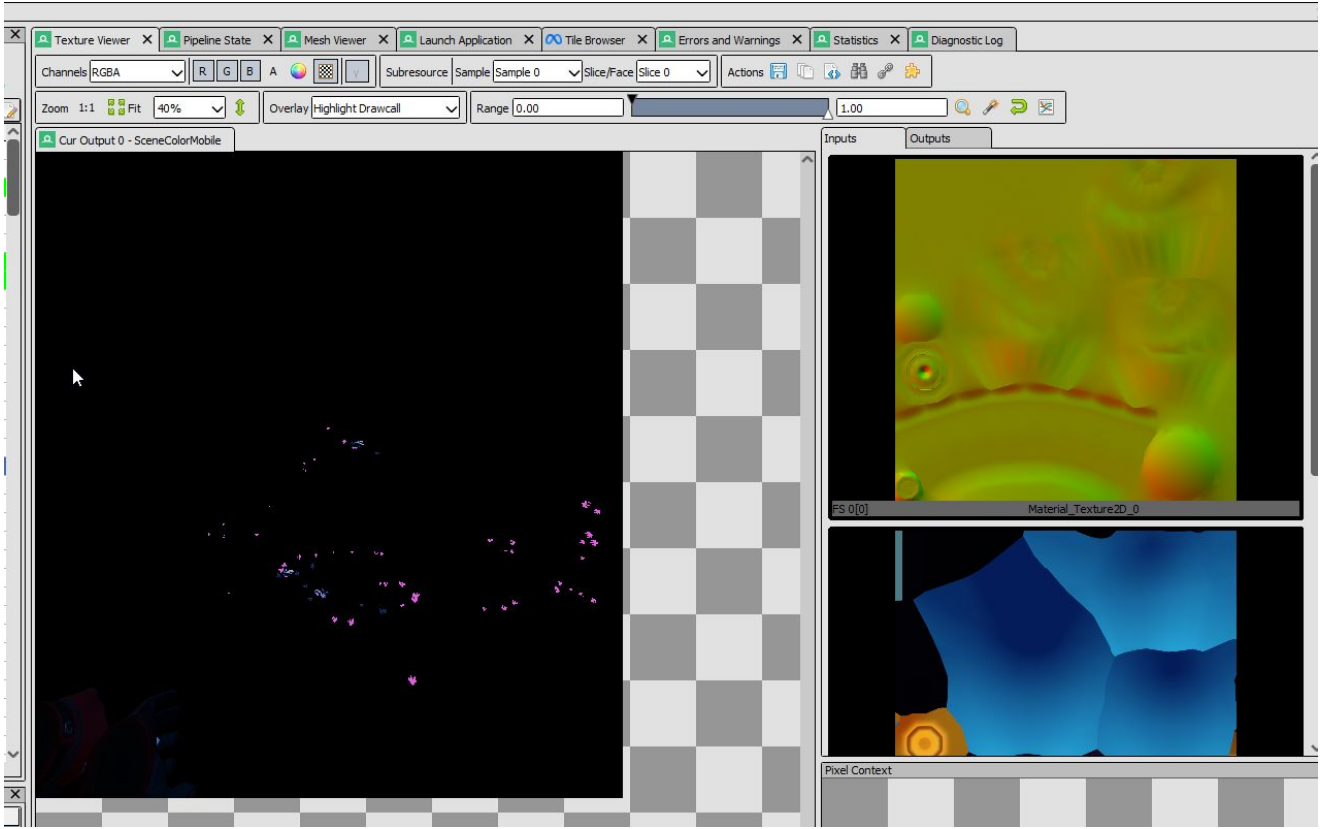
Filter using keyword “draw”



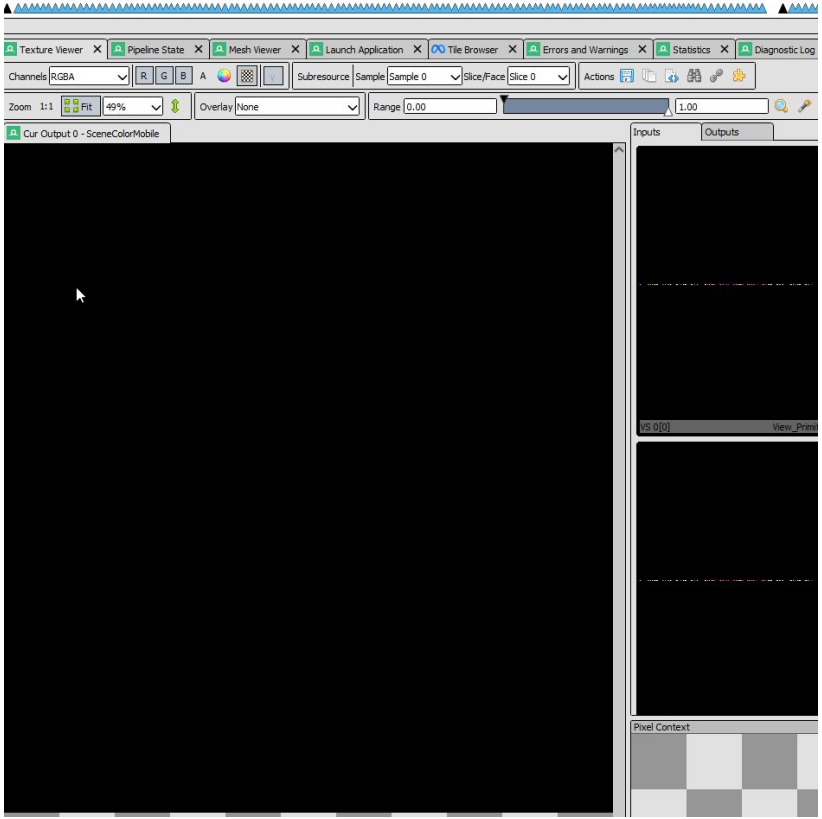
EID	Name	Duration (µs)
274	vkCmdBindVertexBuffers(0, { **Buffer 44583**, **Buffer 216**, **Buffer 44421**, **Buffer 44583** })	
275	vkCmdBindIndexBuffer(**Buffer 44499**, VK_INDEX_TYPE_UINT16)	
276	vkCmdDrawIndexed(3672, 1)	
277	vkCmdBindDescriptorSets(0, { **Descriptor Set 174952**, **Descriptor Set 63919** })	
278	vkCmdBindVertexBuffers(0, { **Buffer 44583**, **Buffer 216**, **Buffer 44421**, **Buffer 44583** })	
279	vkCmdBindIndexBuffer(**Buffer 44499**, VK_INDEX_TYPE_UINT16)	
280	vkCmdDrawIndexed(216, 1)	
281	vkCmdBindPipeline(**Graphics Pipeline 5252**)	
282	vkCmdBindDescriptorSets(0, { **Descriptor Set 174952**, **Descriptor Set 63919** })	
283	vkCmdBindVertexBuffers(0, { **Buffer 44229**, **Buffer 216**, **Buffer 44229**, **Buffer 44229** })	
284	vkCmdBindIndexBuffer(**Buffer 44379**, VK_INDEX_TYPE_UINT16)	
285	vkCmdDrawIndexed(1356, 1)	
286	vkCmdBindPipeline(**Graphics Pipeline 4301**)	
287	vkCmdBindDescriptorSets(0, { **Descriptor Set 174952**, **Descriptor Set 63919** })	
288	vkCmdBindVertexBuffers(0, { **Buffer 44229**, **Buffer 216**, **Buffer 44229**, **Buffer 44229** })	
289	vkCmdBindIndexBuffer(**Buffer 44379**, VK_INDEX_TYPE_UINT16)	
290	vkCmdDrawIndexed(354, 1)	
291	vkCmdBindPipeline(**Graphics Pipeline 4310**)	
292	vkCmdBindDescriptorSets(0, { **Descriptor Set 174953**, **Descriptor Set 158527** })	
293	vkCmdBindVertexBuffers(0, { **Buffer 44583**, **Buffer 216**, **Buffer 44421**, **Buffer 44583** })	
294	vkCmdBindIndexBuffer(**Buffer 44499**, VK_INDEX_TYPE_UINT16)	
295	vkCmdDrawIndexed(22050, 1)	
296	vkCmdBindPipeline(**Graphics Pipeline 4279**)	
297	vkCmdBindDescriptorSets(0, { **Descriptor Set 174954**, **Descriptor Set 158525** })	
298	vkCmdBindVertexBuffers(0, { **Buffer 44583**, **Buffer 216**, **Buffer 44421**, **Buffer 44583** })	
299	vkCmdBindIndexBuffer(**Buffer 44499**, VK_INDEX_TYPE_UINT16)	
300	vkCmdDrawIndexed(3744, 1)	
301	vkCmdBindPipeline(**Graphics Pipeline 6790**)	
302	vkCmdBindDescriptorSets(0, { **Descriptor Set 174952**, **Descriptor Set 158523** })	
303	vkCmdBindVertexBuffers(0, { **Buffer 44583**, **Buffer 216**, **Buffer 44421**, **Buffer 44583** })	
304	vkCmdBindIndexBuffer(**Buffer 44499**, VK_INDEX_TYPE_UINT16)	
305	vkCmdDrawIndexed(5670, 1)	
306	vkCmdEndRenderPass(C=Store, DS=Store)	1862.1353712



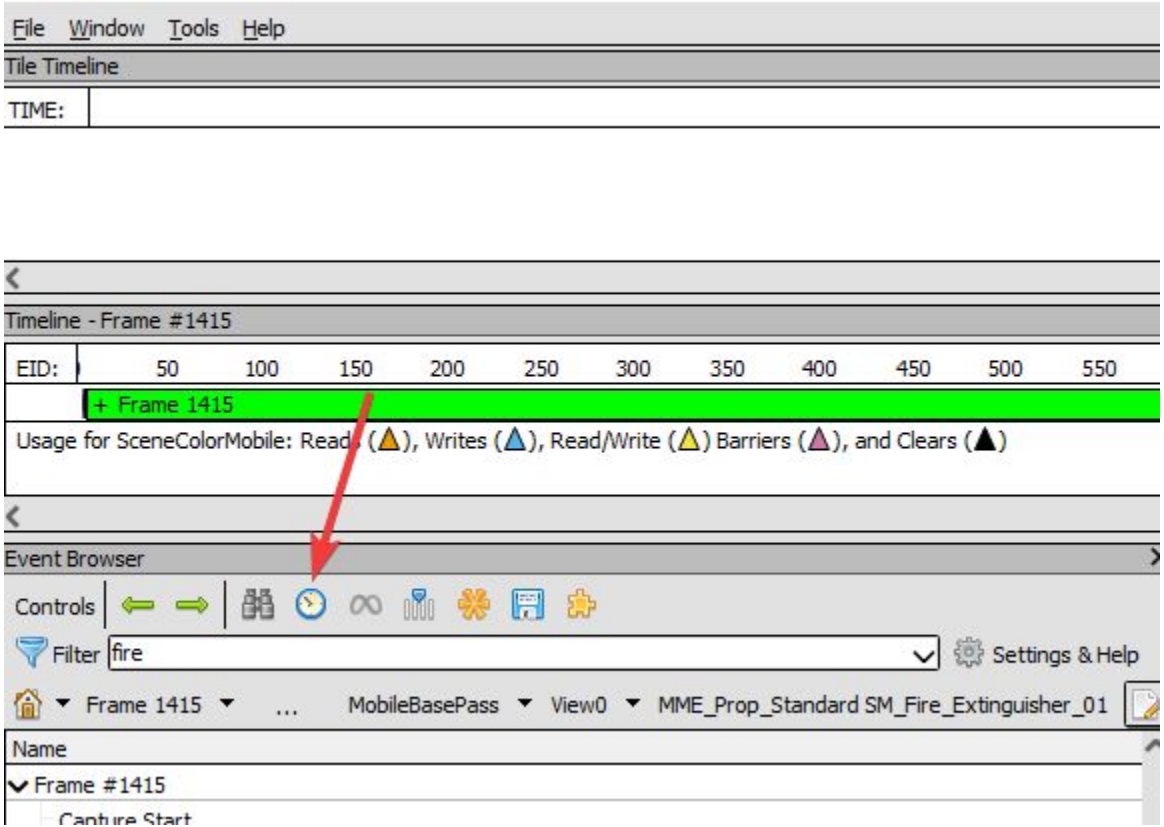
Highlight draw call



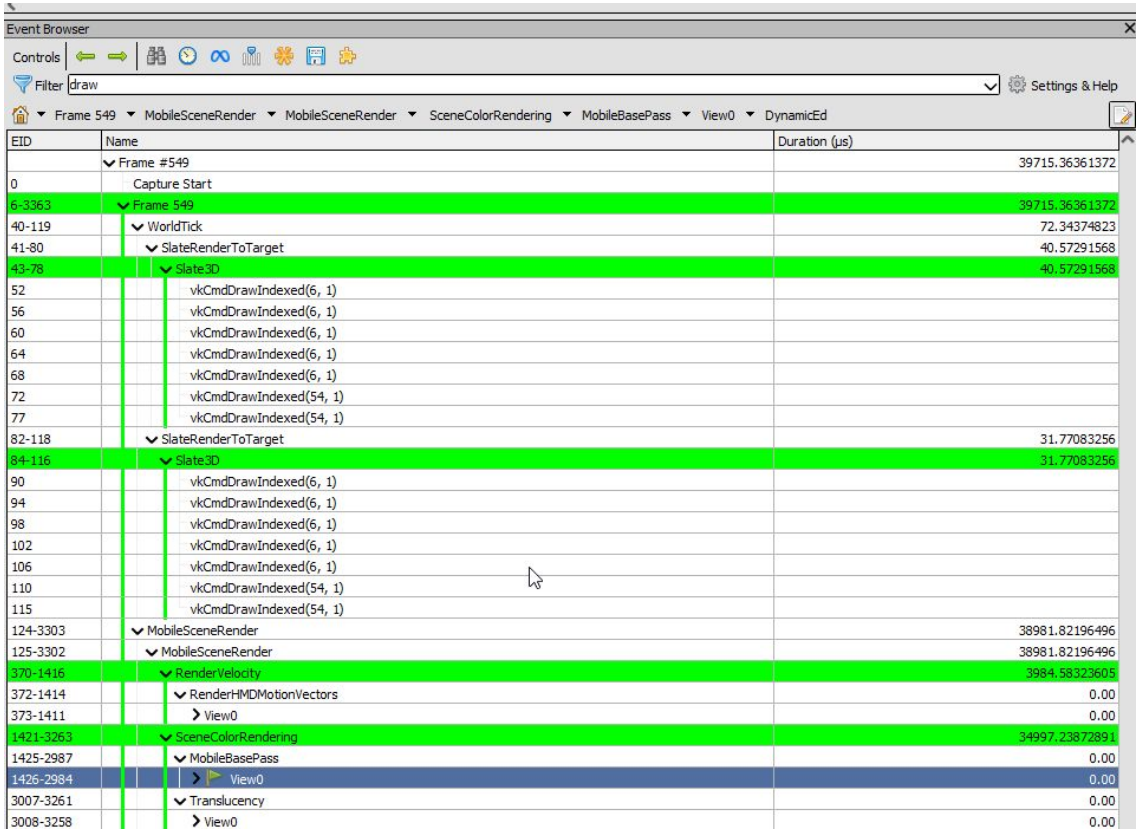
Highlight draw call - auto-instancing



Highlight draw call - nothing!

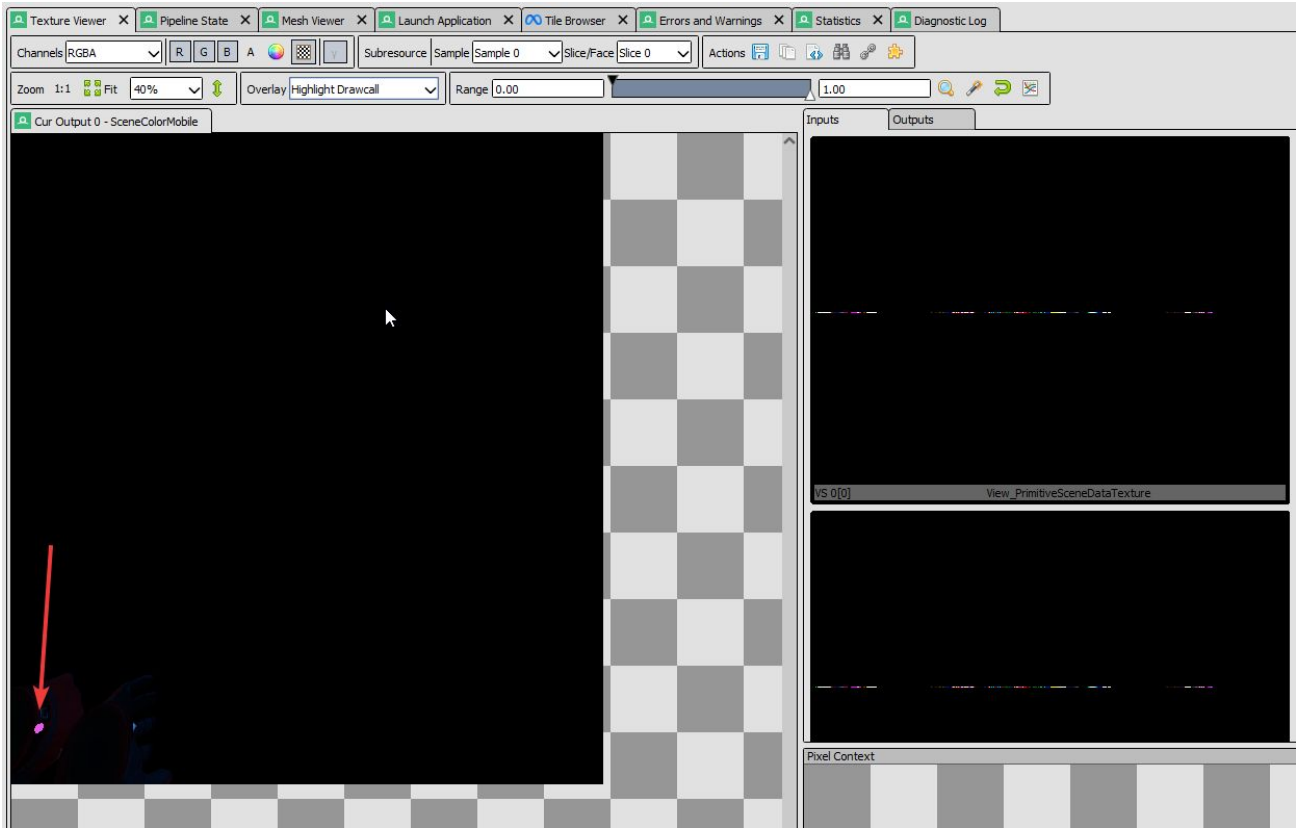


Time duration



Time duration

Distance Culling



Do we really need to draw this?

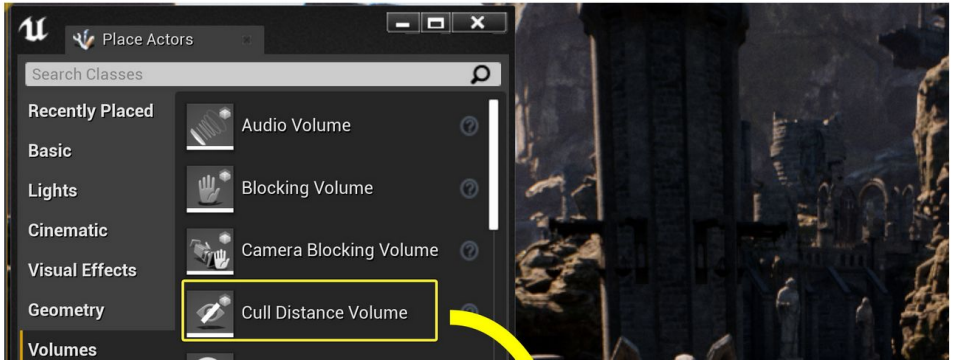
Cull Distance Volume

An overview of how to use the Cull Distance Volume to cull Actors in your Levels as specified distances based on their size.

Cull Distance Volumes are a useful optimization tool that defines what distance to draw (or make visible) any Actor within the volume. These volumes store any number of size and distance combinations called **Cull Distance Pairs**. The Actor (along its longest dimension) and then assigned to that Actor instance in the level. Cull Distance Volumes are most useful for optimizing large outdoor levels that have detailed interiors. The interiors can be culled when they are sm unimportant.

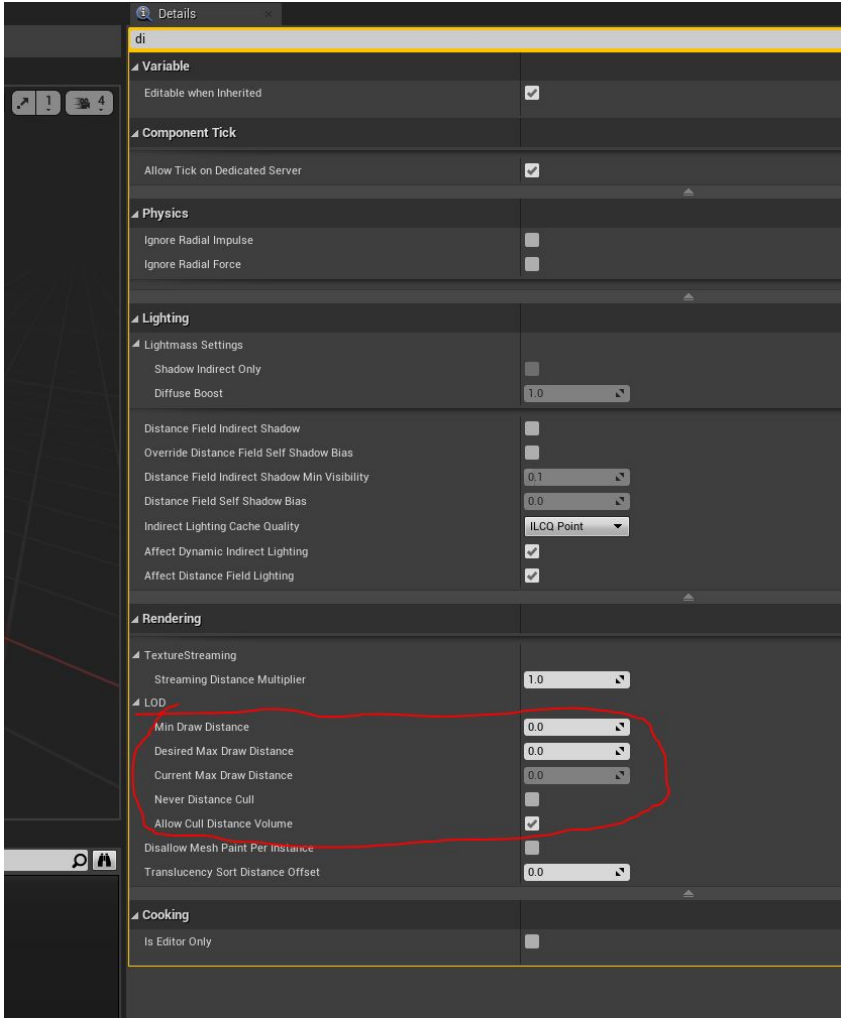
Setup and Usage

Cull Distance Volumes can be added to your level through the **Place Actors** panel in the Editor, and you can scale it to fit the level.



Cull distance volume

Draw distance



Auto-Instancing

Auto-Instancing is a feature that automatically combines multiple draw calls into one instanced draw call.

Mesh Auto-Instancing on Mobile

Learn to use this Beta feature, but use caution when shipping with it.

Mesh Auto-Instancing on Mobile

How to enable mesh auto-instancing on mobile.

Intermediate

ON THIS PAG

SEE ALSO

The UE4 **Mesh Drawing Pipeline** implements a mesh auto-instancing feature that merges draw calls, which can greatly improve graphics performance. This functionality is now available for mobile devices with some additional settings configuration.

Steps

1. Locate the **Config** folder for your project and open **DefaultEngine.ini.**
2. Add the following lines:

```
r.Mobile.SupportGPUScene=1
r.Mobile.UseGPUSceneTexture=1
```

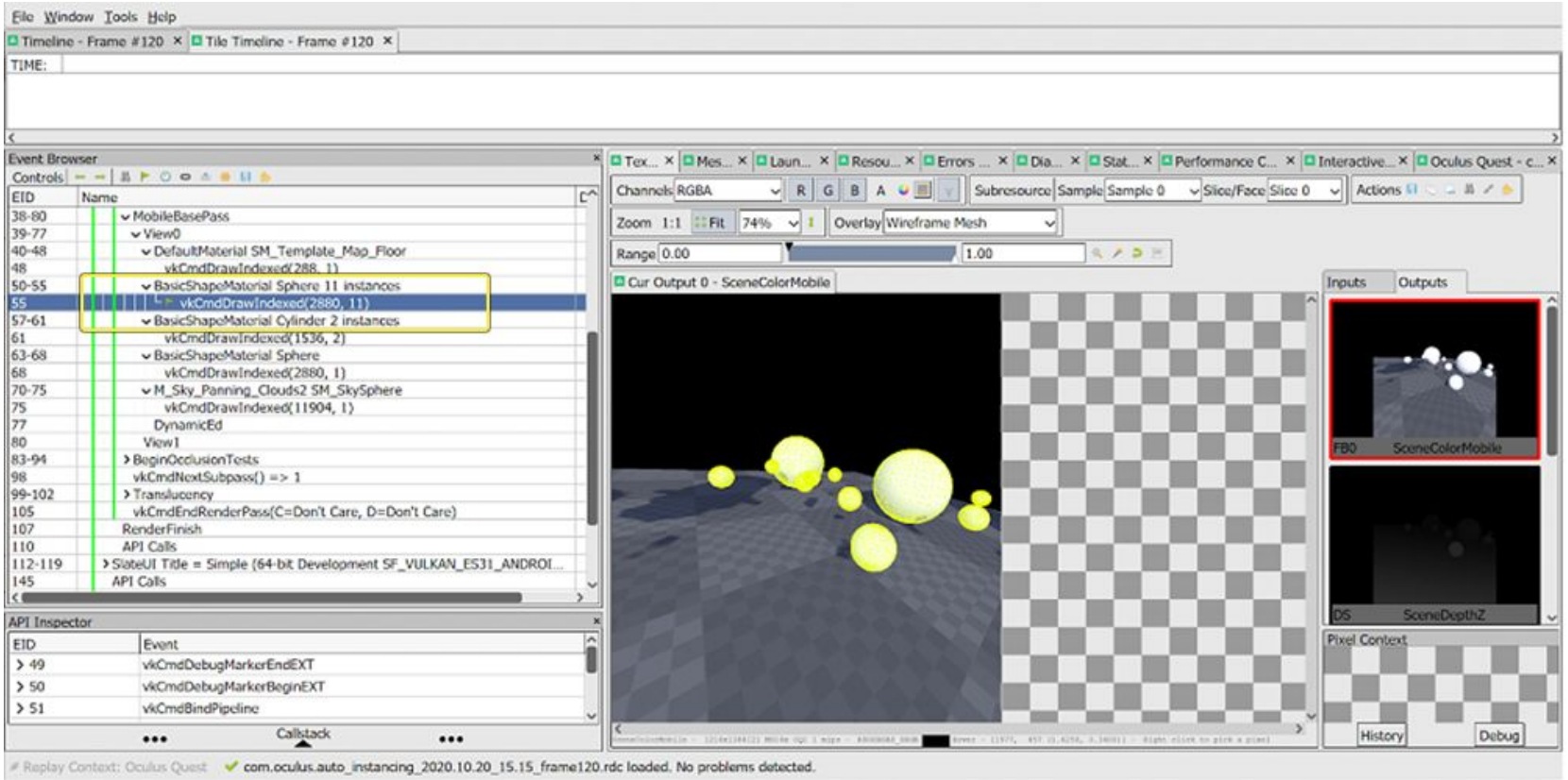
Save your changes and close the file.

NOTE

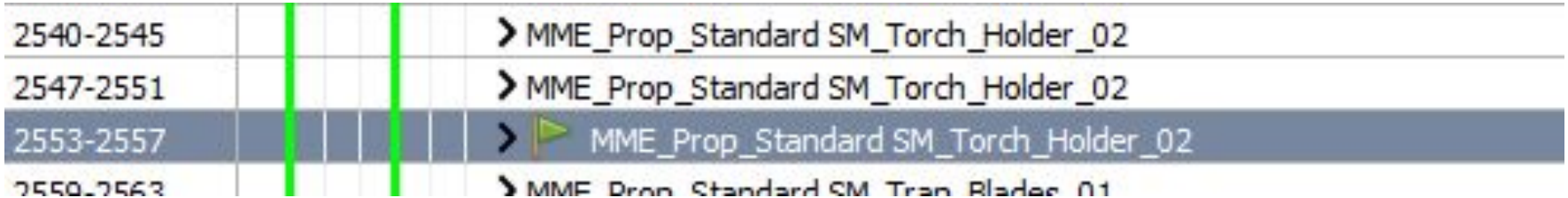
Enabling this feature will cause shaders to be rebuilt for mobile platforms. If you have Unreal Editor set to Android Preview mode, the editor will recompile shaders accordingly. Large projects may have a long iteration time.

Result

By enabling the above settings for your project, auto-instancing will be enabled for all devices. `r.Mobile.SupportGPUScene` enables auto-instancing on mobile devices. However, they will use the same buffer as a desktop build. Mali devices only support buffers of up to 64 kb and are unable to support this feature normally. `r.Mobile.UseGPUSceneTexture` will make the auto-instancing process use a texture instead of a buffer to store the required information, enabling Mali devices to use



Instanced draw calls



SM_Torch_Holder_02 - Same mesh but multiple draw calls?

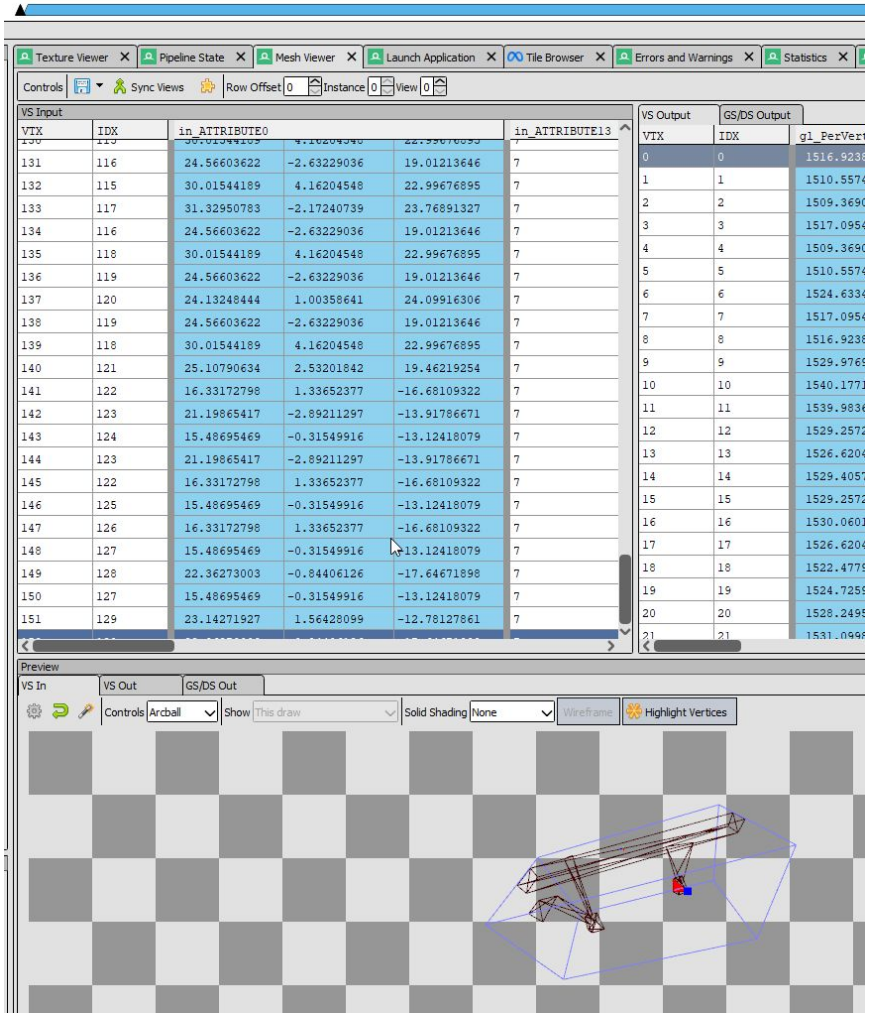
Statistics - Pre Changes

Stats for com.getsetgames.MyGame1_2023.03.01_23.40_frame549.rdc.
File size: 270.18MB (435.17MB uncompressed, compression ratio 1.61:1)
Persistent Data (approx): 77.76MB, Frame-initial data (approx): 353.51MB

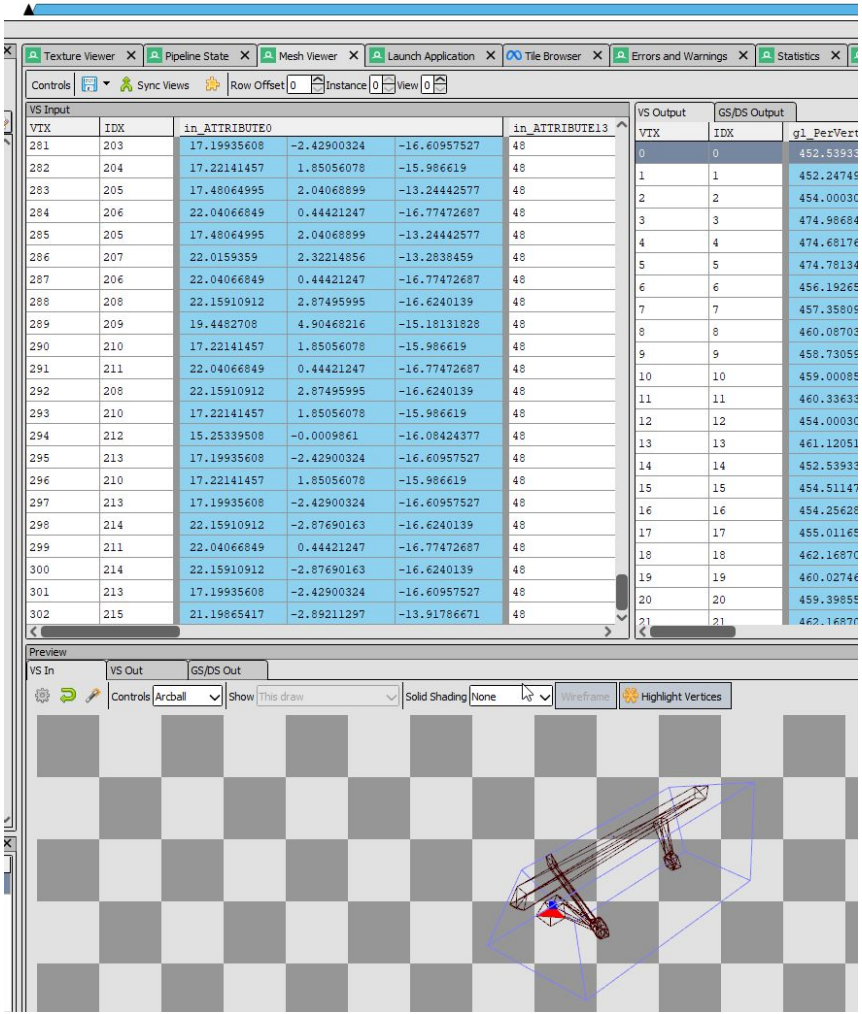
*** Summary ***
Draw calls: **464**
Dispatch calls: 6
API calls: 1878
API:Draw/Dispatch call ratio: 3.99574

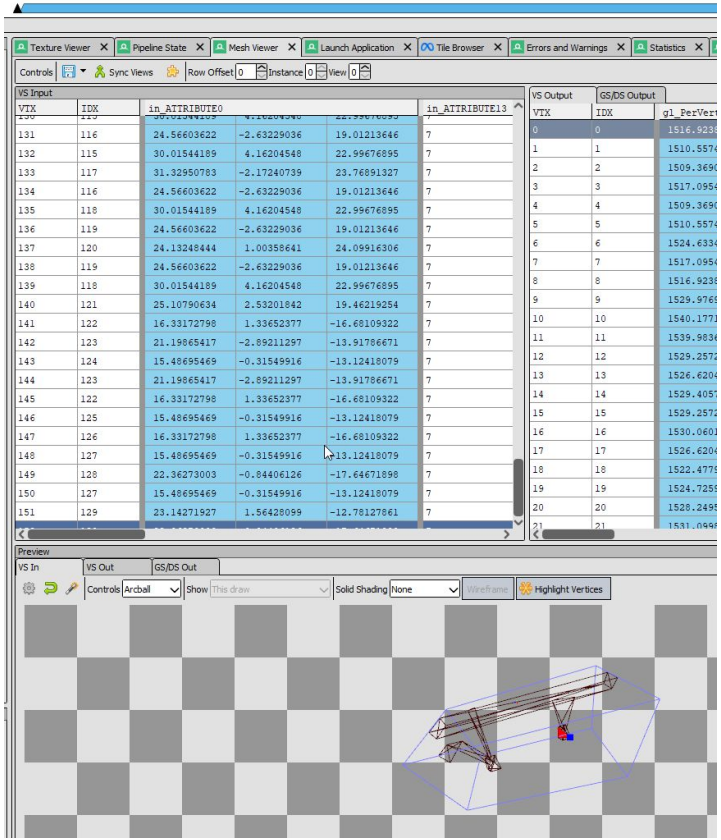
320 Textures - 179.90 MB (179.77 MB over 32x32), 11 RTs - 167.72 MB.
Avg. tex dimension: 581.068x577.149 (594.763x593.927 over 32x32)
66 Buffers - 76.00 MB total 25.00 MB IBs 46.00 MB VBs.
423.62 MB - Grand total GPU buffer + texture load.

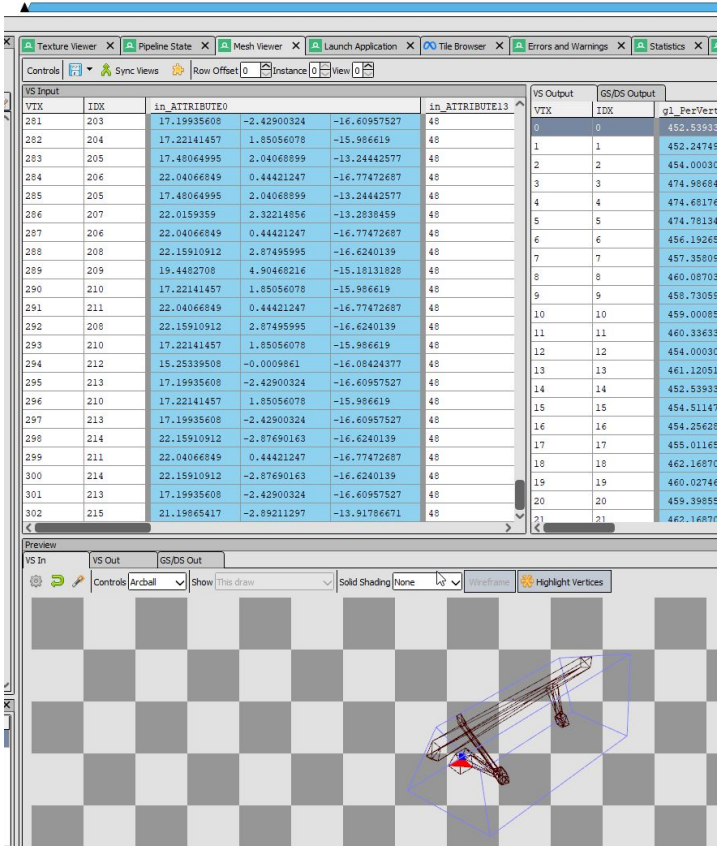
Torch holder instance #1



Torch holder instance #2







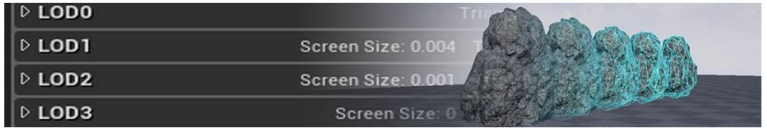
What's the difference?

Automatic LOD Generation

Setting Up Automatic LOD Generation

How To use the Automatic LOD Generation system in UE4.

Intermediate



The Automatic LOD generation system allows you to automatically reduce the polygon count of your Static Meshes to create LODs with the Unreal Engine 4 (UE4) Editor. Automatic LOD generation uses what is called quadratic mesh simplification to help generate the LODs for Static Meshes. Quadratic mesh simplification works by calculating the amount of visual difference that collapsing an edge (by merging two vertices) would generate. It then picks the edge with the least amount of visual impact and collapses it. When this happens, the tool will pick the best place to put the newly merged vertex, removing any triangles that have also collapsed along with the edge. It will continue to collapse edges until it reaches the requested target number of triangles. In the following guide, we'll show you how-to setup and use the automatic LOD generation system in your UE4 projects.

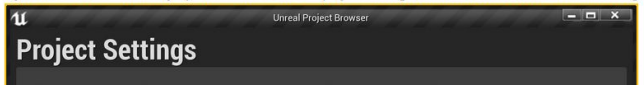
Setup

In the following section, we will create a new project that has Starter Content, and then open up a Static Mesh asset to work with.

NOTE

For this part of the how-to guide, we will use the **SM_Rock** Static Mesh that comes with the Starter Content. However, feel free to use any Static Mesh of your choosing to follow along.

1. If you have not done so already, open, or create a new UE4 project, making sure that the **With Starter Content** setting is enabled.



Statistics - Post Changes

Stats for com.getsetgames.mygame1_2023.02.15_10.58_frame765.rdc.
File size: 258.49MB (522.77MB uncompressed, compression ratio 2.02:1)
Persistent Data (approx): 66.96MB, Frame-initial data (approx): 452.58MB

*** Summary ***
Draw calls: 266
Dispatch calls: 30
API calls: 1225
API:Draw/Dispatch call ratio: 4.13851

217 Textures - 287.92 MB (287.79 MB over 32x32), 21 RTs - 174.00 MB.
Avg. tex dimension: 875.469x867.969 (902.053x899.354 over 32x32)
56 Buffers - 66.00 MB total 22.00 MB IBs 39.00 MB VBs.
527.93 MB - Grand total GPU buffer + texture load.

References

- TechArtAid - Rendering Passes
- Tech Art Aid - Dissecting a Frame with RenderDoc
- RenderDoc for Oculus
- How to Optimize your Oculus Quest App w/ RenderDoc: Getting Started + Frame Capture
- Auto Instancing On Oculus
- VR Profiling Interpretations and Considerations
- How to Optimize your Oculus Quest App w/ RenderDoc: Walkthroughs of Key Usage Scenarios and Optimization Tips Part 1
- How to Optimize your Oculus Quest App w/ RenderDoc: Walkthroughs of Key Usage Scenarios and Optimization Tips Part 2

Questions?

rob@getsetgames.com

@robertfsegal

sarcasticcoder.com