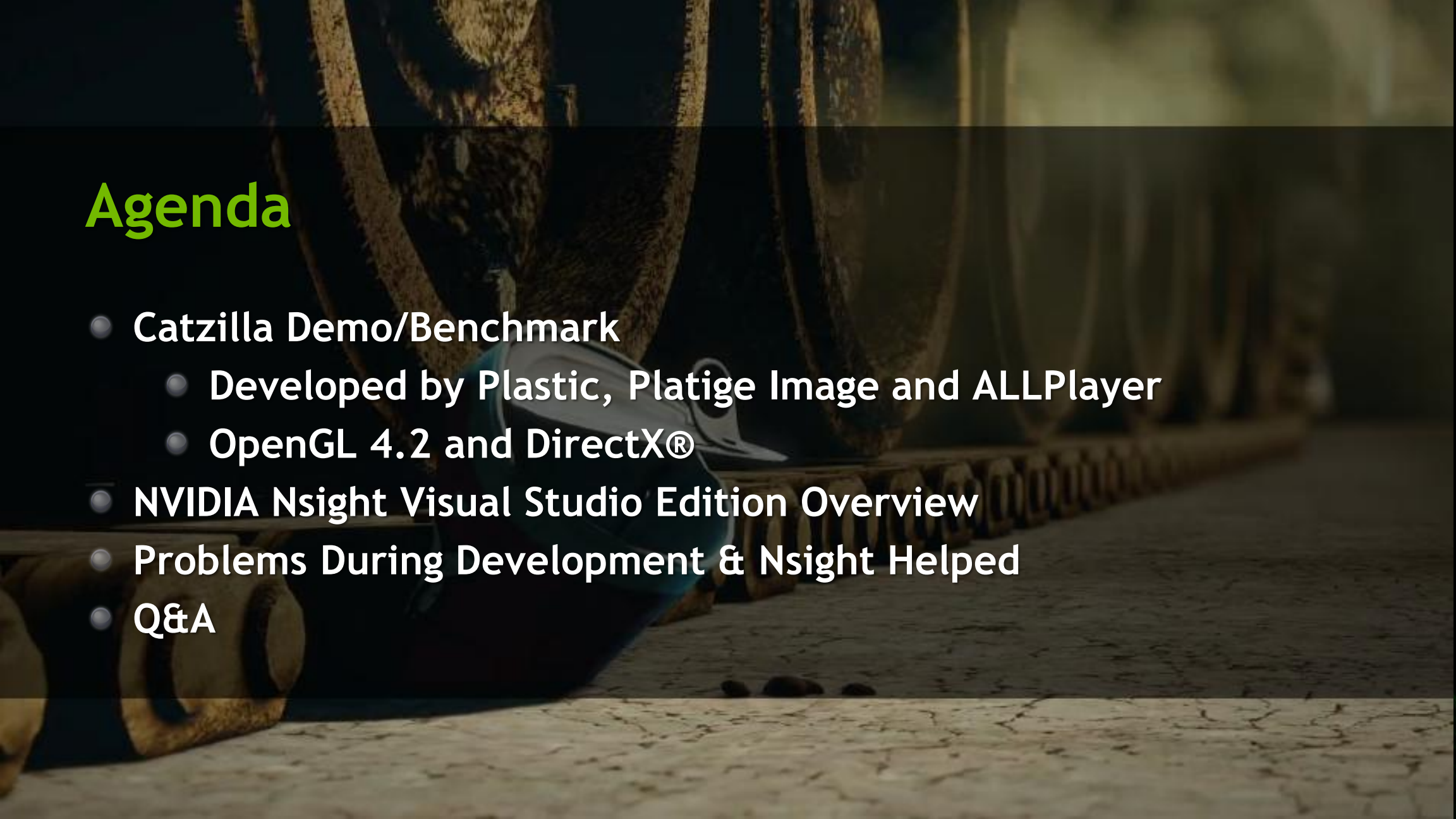
NVIDIA Nsight™ VSE 3.0 Catzilla Engine Development in DirectX® 11 and OpenGL 4.2

Jeff Kiel – NVIDIA Corporation

Michal Szymczyk – CTO, Plastic

Michal Staniszewski – Creative Director, Plastic

NVIDIA



Agenda

- Catzilla Demo/Benchmark
  - Developed by Plastic, Platige Image and ALLPlayer
  - OpenGL 4.2 and DirectX®
- NVIDIA Nsight Visual Studio Edition Overview
- Problems During Development & Nsight Helped
- Q&A

Motivation behind the project

- There was a chance to do it – European Union Competitiveness & Innovation Programme (CIP)
- Old engine was completely tailored for PS3(Datura® development)
- We wanted to prepare new engine for next gen machines

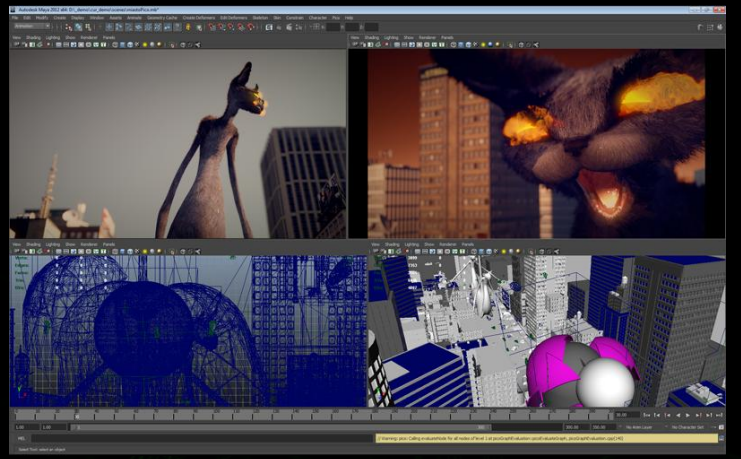
Development timeline

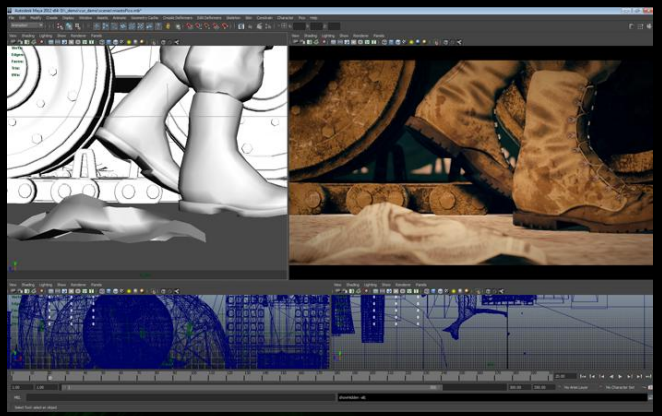
- Engine development before preproduction started - 2 months
- Preproduction - 1 month
- Production - 2 months
- Post Production - 2 months, 6 months total

Demo

Features of the engine

- Post processing (HDR, adaptive luminance, DOF with Bokeh)
- Fur based on geometry shader generated fins
- Physically based lighting
- Engine completely integrated with Autodesk®Maya





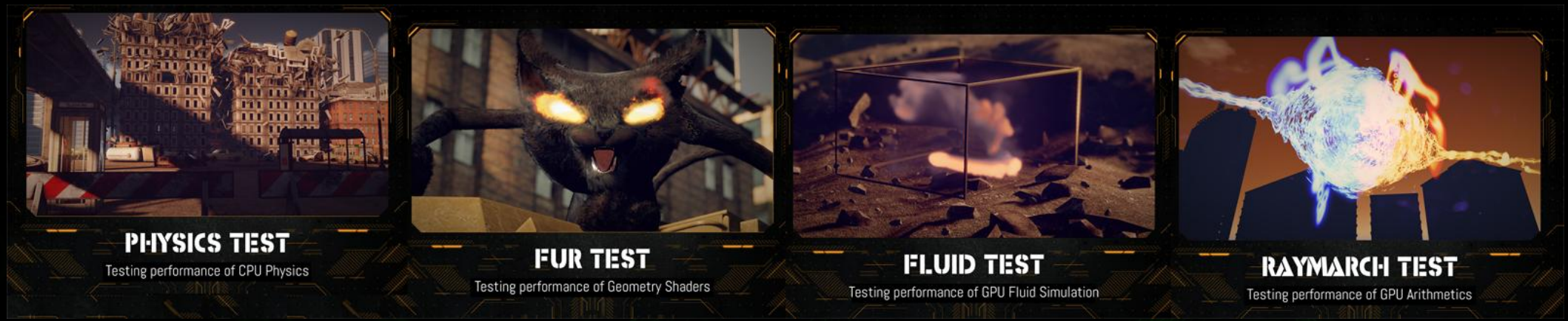
OpenGL/DirectX® 11

- OpenGL needed because of WindowsXP support, possibility to port to MacOSX and Linux
- Still using OpenGL inside of Autodesk®Maya
- Possibility to compare performance between two APIs
- DirectX still faster?



Benchmark modes

- Physics - CPU based using PhysX® - test
- Fur (Geometry Shaders) test
- Fluid (GPU Memory Bandwidth) test
- Raymarching (GPU ALU) test



NVIDIA Nsight Visual Studio Edition

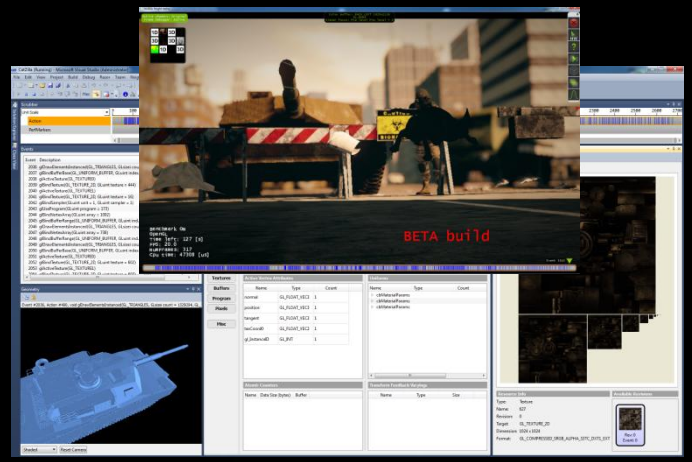
Visual Studio integrated development for GPU and CPU

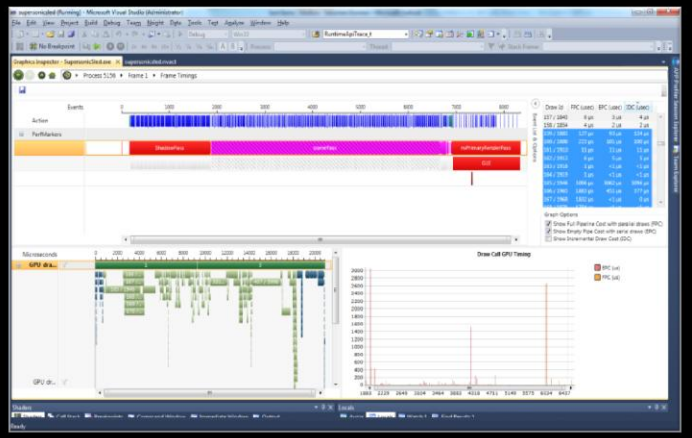
Build **Debug** **Profile**

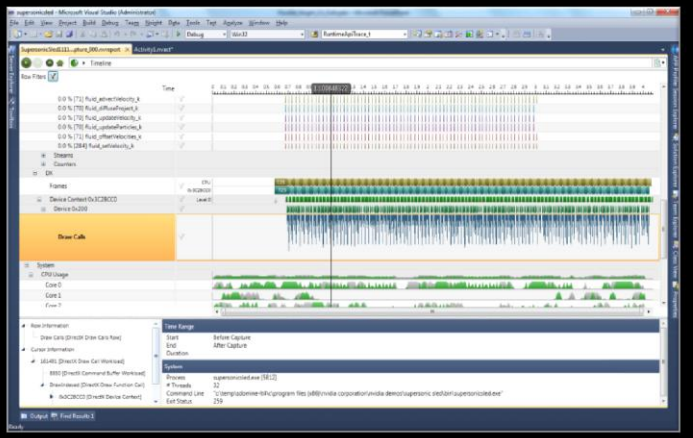
Microsoft Visual Studio PARTNER

NVIDIA Nsight Visual Studio Edition

Supports Direct3D 9/11 and OpenGL 4.2







Frame Debugger

- HUD for scene scrubbing
- State inspection at each draw call
- Real time frame capture and replay
- Source code serialization for D3D9/11

HLSL and GLSL Shader Debugger

- Native GPU shader debugging and GPU memory views
- Complex conditional breakpoints and Pixel History
- Local, single GPU shader debugging

Frame Profiler

- Automatic GPU bottleneck determination
- Draw call and frame timings
- Direct3D Perf Markers and render state grouping/sorting

Application and System Trace

- Inspect Direct3D & OpenGL, CPU & GPU workloads
- Correlate threads, call stack, API calls, WDDM kernel queues and resulting GPU workloads
- Concurrent draw call execution and memory transfer trace

Problems During Development

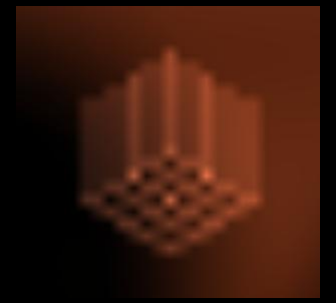
#1 Broken Bokeh Filter

Use similar algorithm to what DICE showed at SIGGRAPH 2011

Bokeh pass has extra bright lines & chess board pattern

Problems During Development

#1 Broken Bokeh Filter



Frame0Part00.cpp | Main.cpp

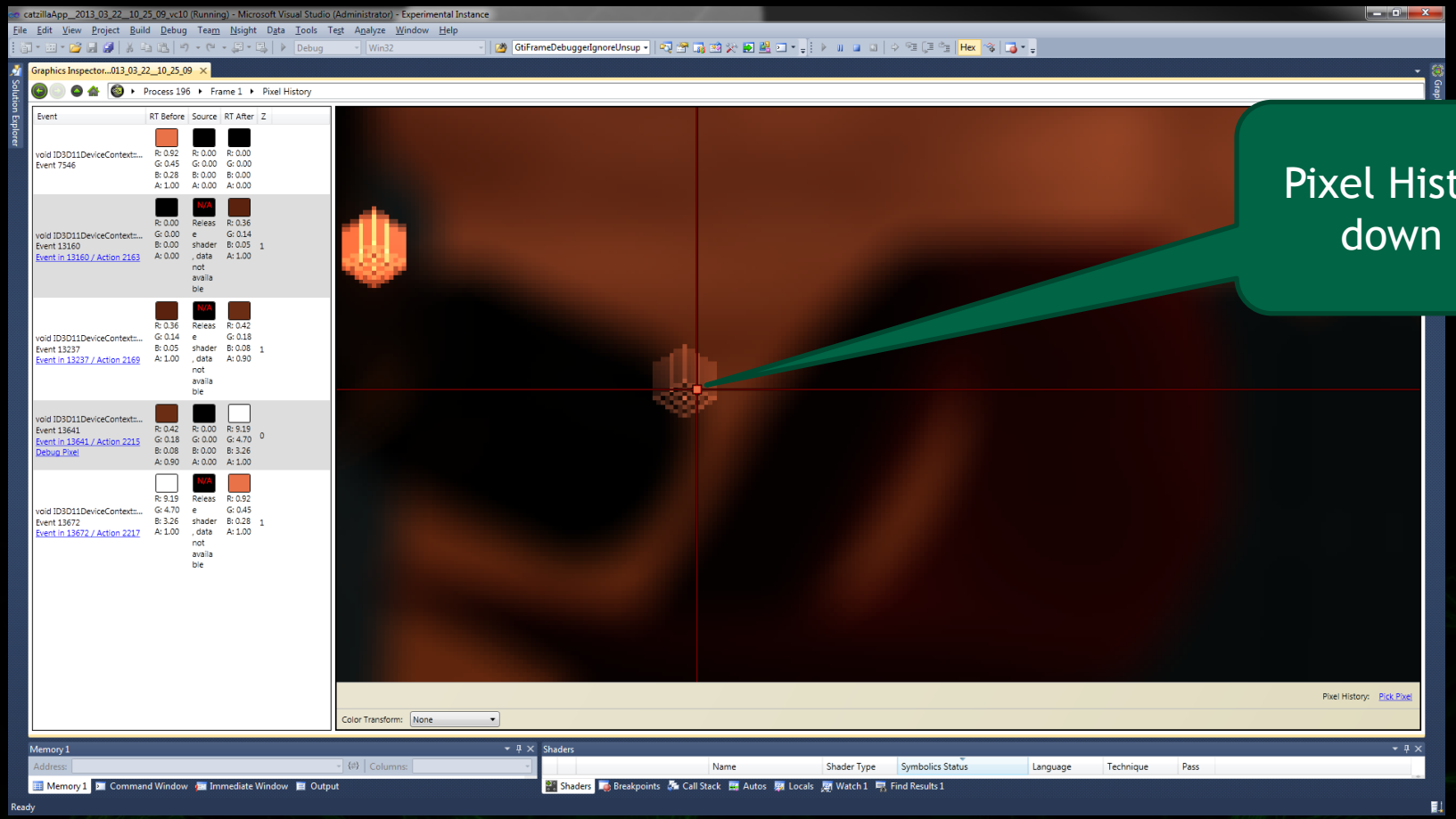
(Global Scope)

```
// Event 49 of 13776
pID3D11DeviceContext_uidof_43->IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_POINTLIST);

// Event 50 of 13776
pID3D11DeviceContext_uidof_43->IASetInputLayout(pID3D11InputLayout_uidof_12484);

// Event 51 of 13776
static ID3D11Buffer* pID3D11Buffer_temp_6[3] = { pID3D11Buffer_uidof_6390, pID3D11Buffer_uidof_6416, pID3D11Buffe
static UINT UINT_temp_2[3] = { 36, 16, 8 };
pID3D11DeviceContext_uidof_43->IASetVertexBuffers(0, 3, pID3D11Buffer_temp_6, UINT_temp_2, ((UINT*)array_of_0s));

// Event 52 of 13776
pID3D11DeviceContext_uidof_43->IASetIndexBuffer(pID3D11Buffer_uidof_6442, DXGI_FORMAT_R16_UINT, 0);

// Event 53 of 13776
static ID3D11RenderTargetView* pID3D11RenderTargetView_temp_1[4] = { pID3D11RenderTargetView
pID3D11DeviceContext_uidof_43->OMSetRenderTargets(4, pID3D11RenderTargetView_temp_1

// Event 54 of 13776, Draw 1 of 2223
pID3D11DeviceContext_uidof_43->DrawInstanced(7706, 1, 0, 0);

// Event 55 of 13776
static ID3D11Buffer* pID3D11Buffer_temp_7[1] = { NULL };
pID3D11DeviceContext_uidof_43->SOSetTargets(1, pID3D11Buffer_temp_7, ((UINT*)array_of_0s));

// Event 56 of 13776
pID3D11DeviceContext_uidof_43->OMSetBlendState(pID3D11BlendState_uidof_46, ((FLOAT*)array_of_0s), 0xFFFFFFFF);

// Event 57 of 13776
pID3D11DeviceContext_uidof_43->OMSetDepthStencilState(pID3D11DepthStencilState_uidof_126, 0);

// Event 58 of 13776
pID3D11DeviceContext_uidof_43->RSSetState(pID3D11RasterizerState_uidof_86);

// Event 59 of 13776
pID3D11DeviceContext_uidof_43->VSSetShader(pID3D11VertexShader_uidof_117, NULL, 0);
```

Bug repro using D3D source generation for captured frames…edit generated code to quickly try debugging ideas

Problems During Development

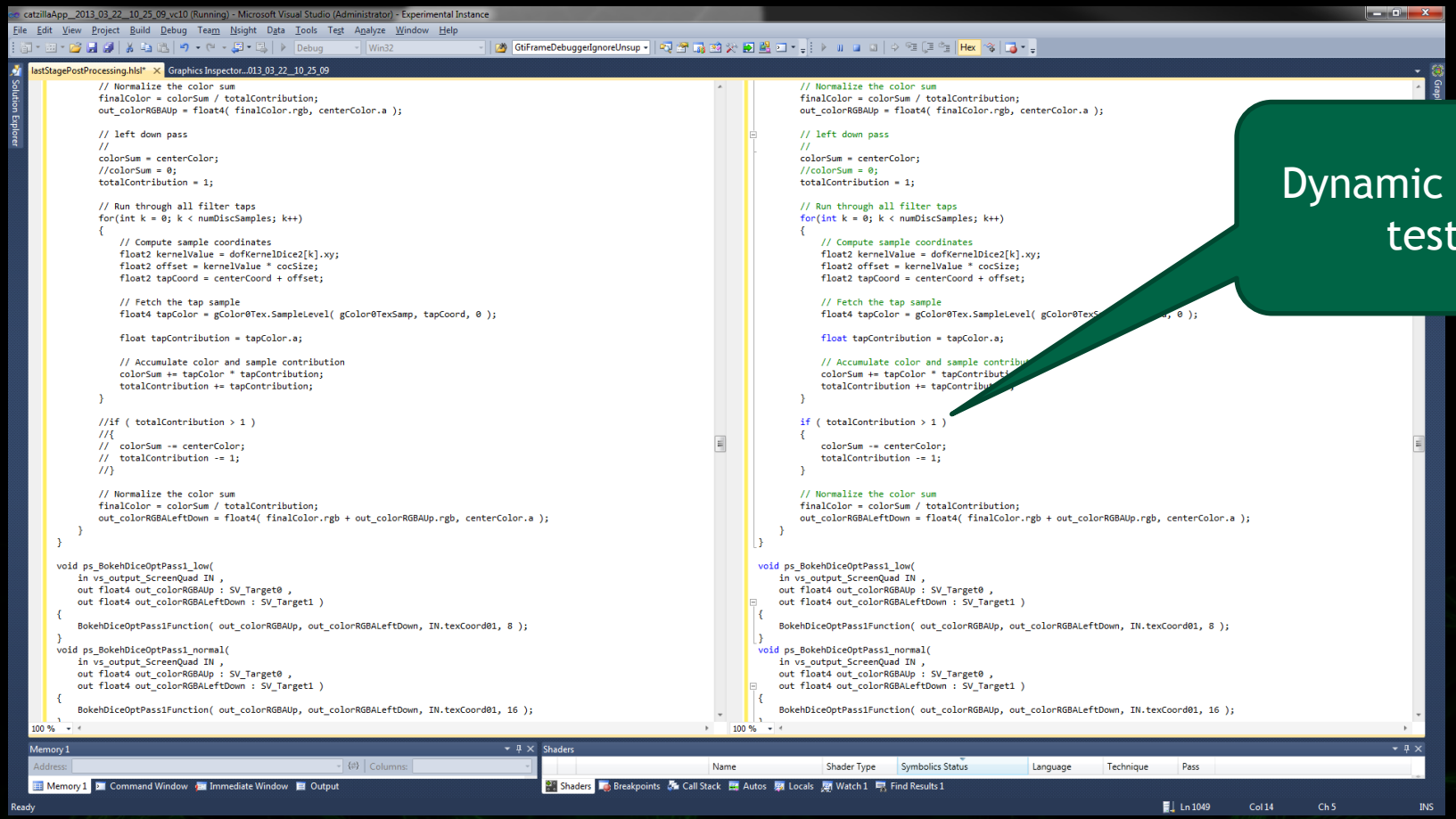
#1 Broken Bokeh Filter



Pixel History helps narrow down the draw calls

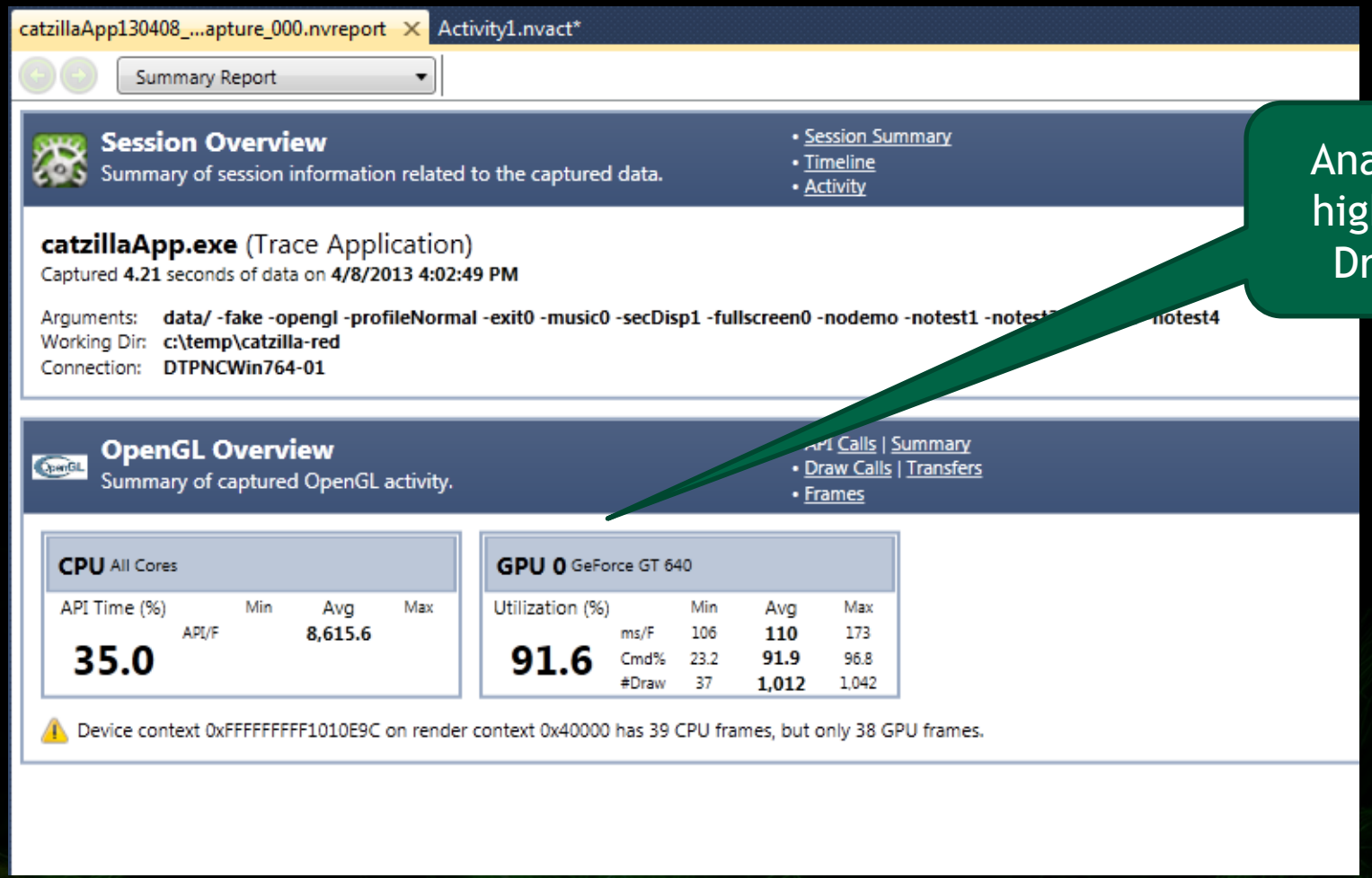
Problems During Development

#1 Broken Bokeh Filter



Problems During Development

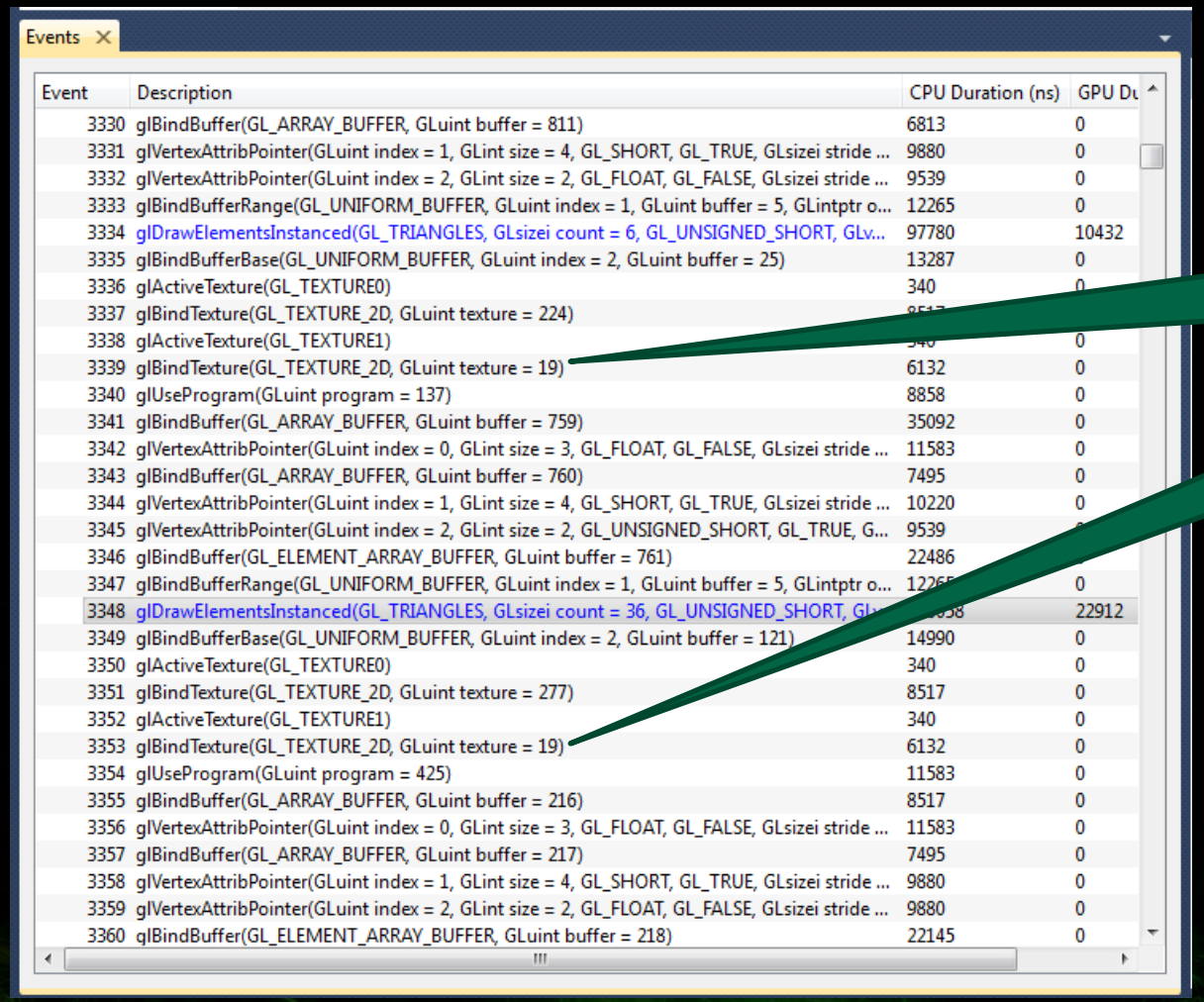
#2 Redundant State Changes



Analysis Summary shows high API Call (8615.6) to Draw Call (1012) ratio

Problems During Development

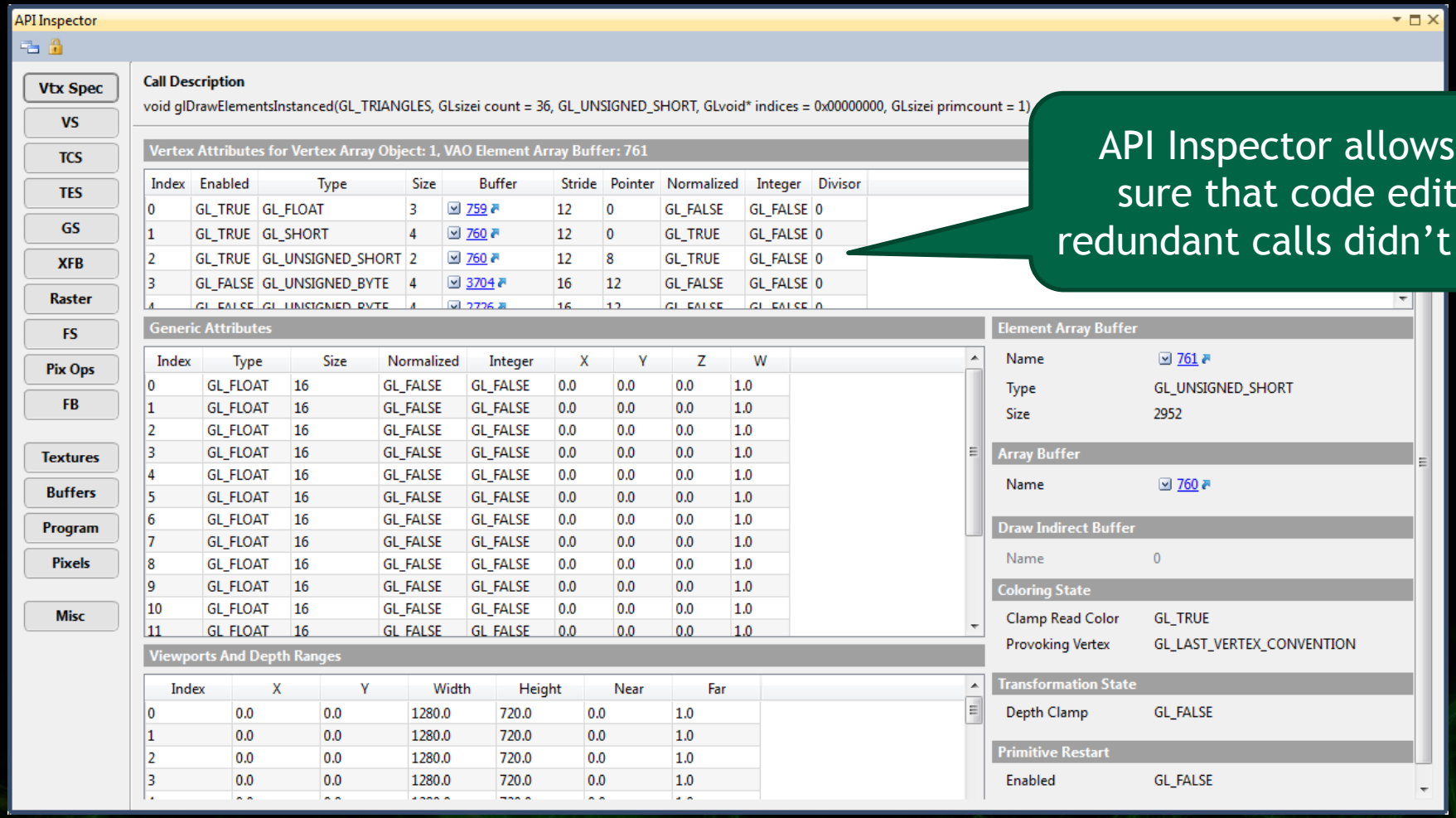
#2 Redundant State Changes



Look at Event List to help confirm redundant state calls

Problems During Development

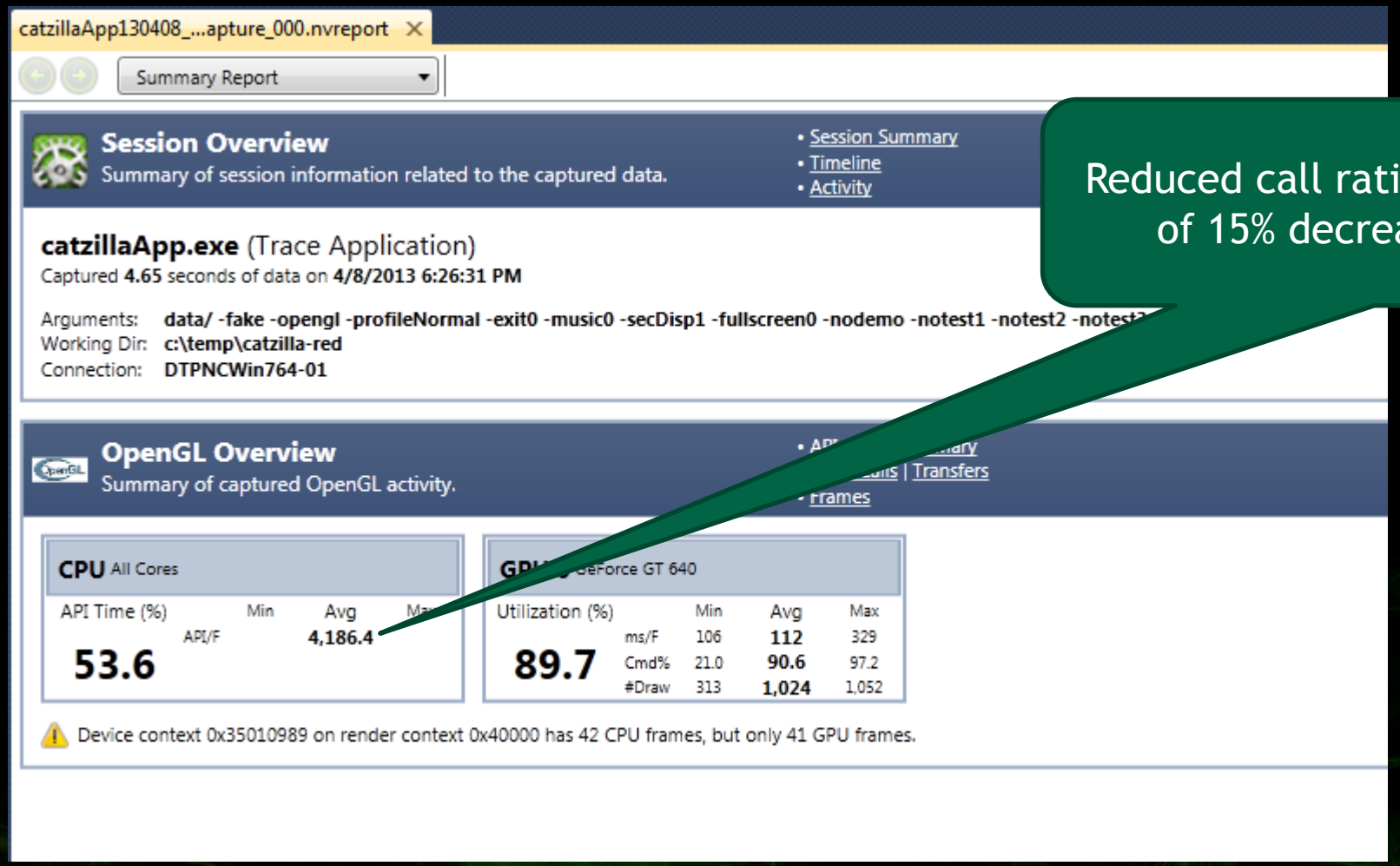
#2 Redundant State Changes



API Inspector allows you to make sure that code edits to remove redundant calls didn't break anything

Problems During Development

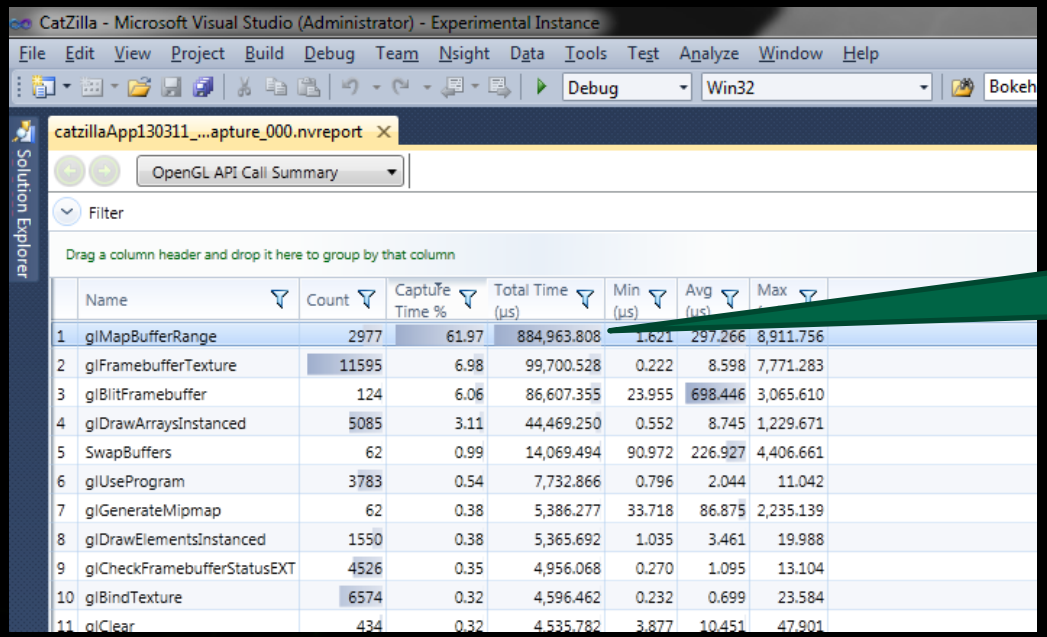
#2 Redundant State Changes



Reduced call ratio resulted in average of 15% decrease in frame time

Problems During Development

#3 glMapBufferRange too expensive

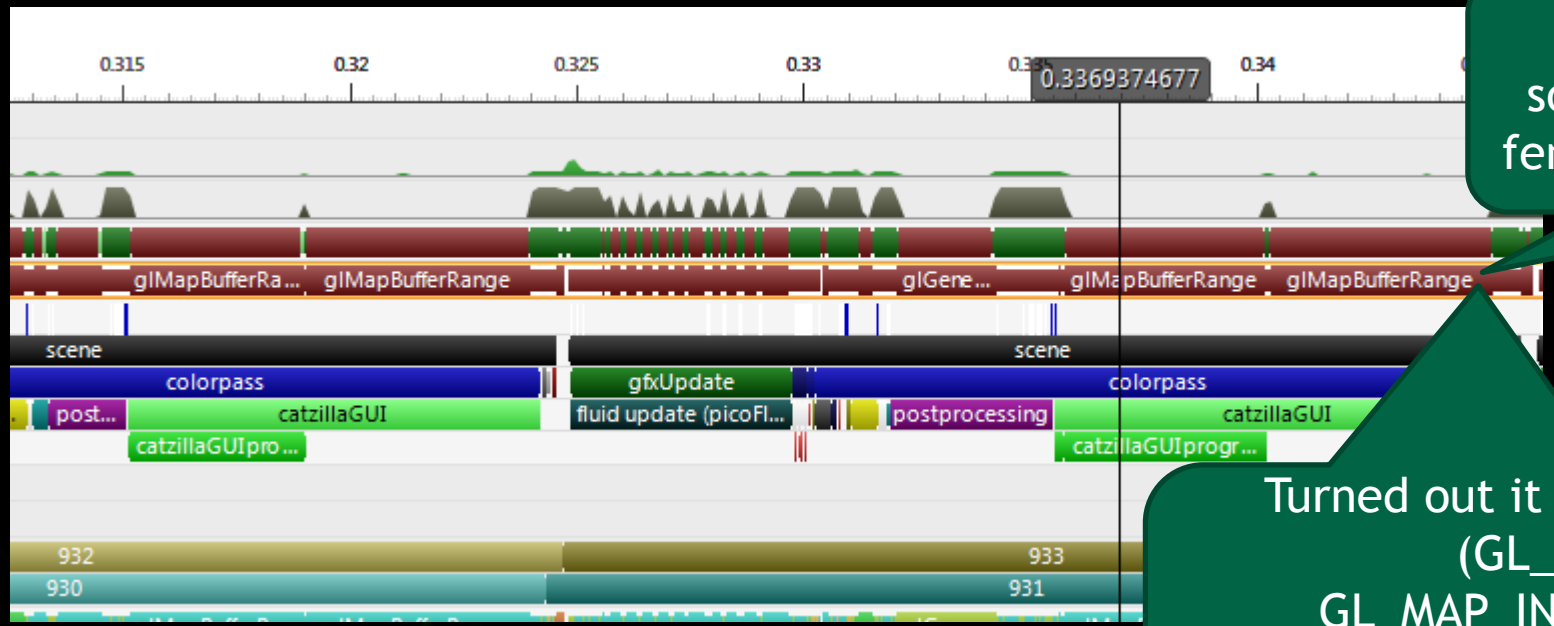


	Name	Count	Capture Time %	Total Time (µs)	Min (µs)	Avg (µs)	Max
1	glMapBufferRange	2977	61.97	884,963.808	1.621	297.266	8,911.756
2	glFramebufferTexture	11595	6.98	99,700.528	0.222	8.598	7,771.283
3	glBlitFramebuffer	124	6.06	86,607.355	23.955	698.446	3,065.610
4	glDrawArraysInstanced	5085	3.11	44,469.250	0.552	8.745	1,229.671
5	SwapBuffers	62	0.99	14,069.494	90.972	226.927	4,406.661
6	glUseProgram	3783	0.54	7,732.866	0.796	2.044	11.042
7	glGenerateMipmap	62	0.38	5,386.277	33.718	86.875	2,235.139
8	glDrawElementsInstanced	1550	0.38	5,365.692	1.035	3.461	19.988
9	glCheckFramebufferStatusEXT	4526	0.35	4,956.068	0.270	1.095	13.104
10	glBindTexture	6574	0.32	4,596.462	0.232	0.699	23.584
11	glClear	434	0.32	4,535.782	3.877	10.451	47.901

3000 calls to glMapBufferRange taking 62% of the frame time

Problems During Development

#3 glMapBufferRange too expensive

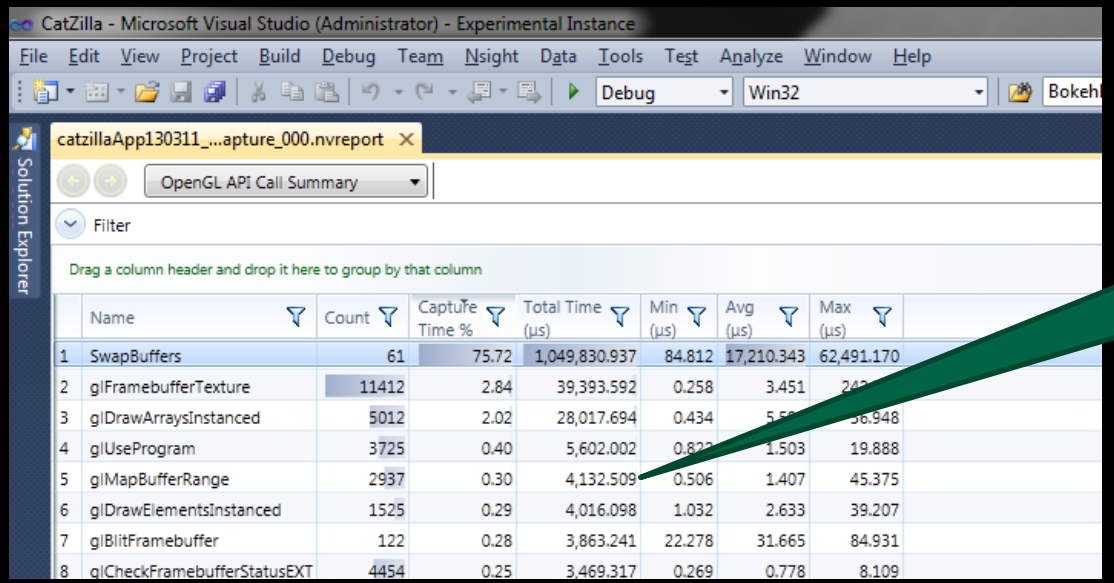


Looked at other possible solutions like buffer pool with fences but hurt SLI performance

Turned out it was bad flags…was passing (GL_MAP_WRITE_BIT | GL_MAP_INVALIDATE_BUFFER_BIT | GL_MAP_UNSYNCHRONIZED_BIT) but the invalidate caused the driver to make too many temporary buffers

Problems During Development

#3 glMapBufferRange too expensive

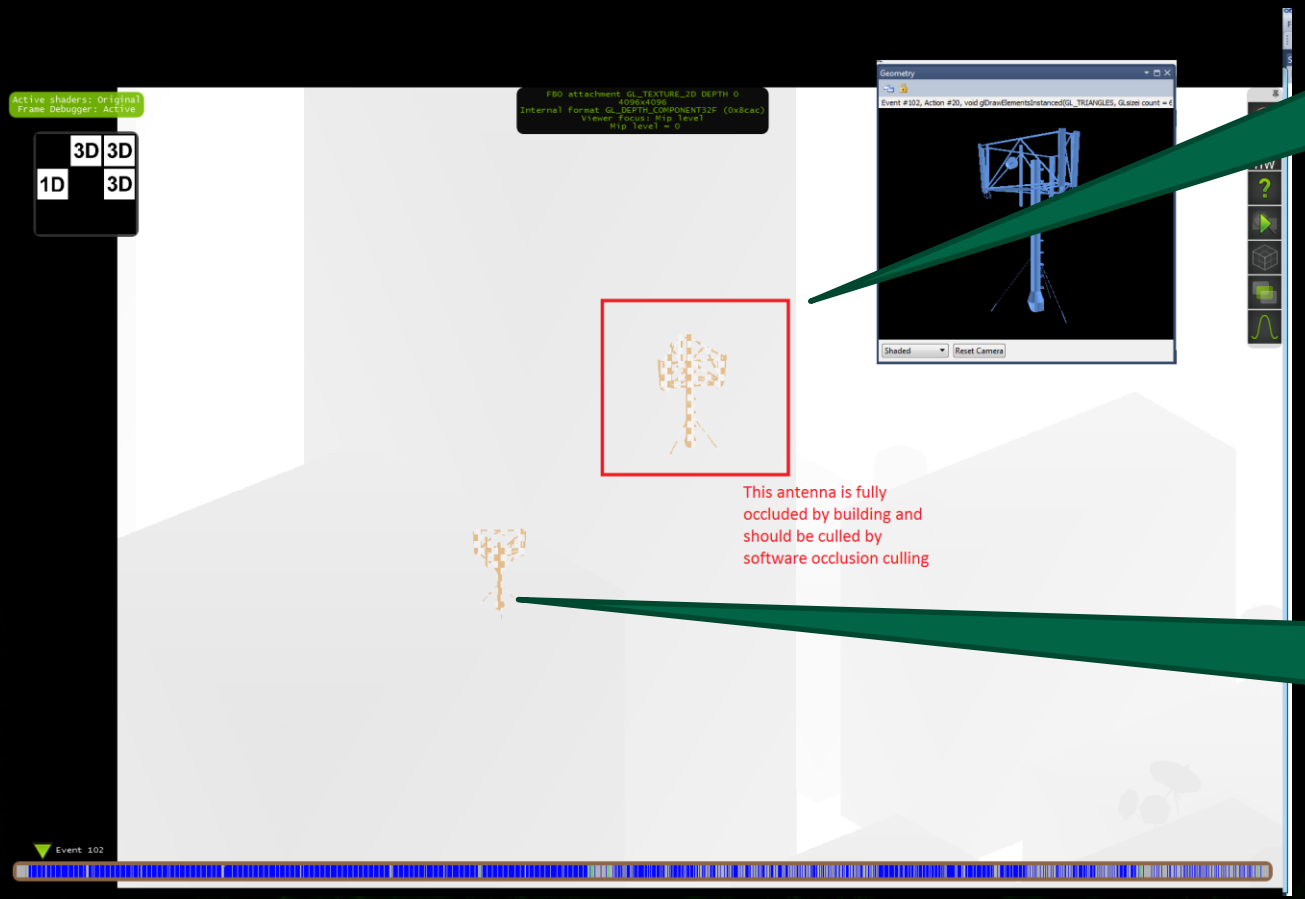


	Name	Count	Capture Time %	Total Time (µs)	Min (µs)	Avg (µs)	Max (µs)
1	SwapBuffers	61	75.72	1,049,830.937	84.812	17,210.343	62,491.170
2	glFramebufferTexture	11412	2.84	39,393.592	0.258	3.451	
3	glDrawArraysInstanced	5012	2.02	28,017.694	0.434		
4	glUseProgram	3725	0.40	5,602.002		1.503	19.888
5	glMapBufferRange	2937	0.30	4,132.509	0.506	1.407	45.375
6	glDrawElementsInstanced	1525	0.29	4,016.098	1.032	2.633	39.207
7	glBlitFramebuffer	122	0.28	3,863.241	22.278	31.665	84.931
8	glCheckFramebufferStatusEXT	4454	0.25	3,469.317	0.269	0.778	8.109

Fix the flags, let the driver manage the memory, and perf improved!

Problems During Development

#4 Bug in SW culling

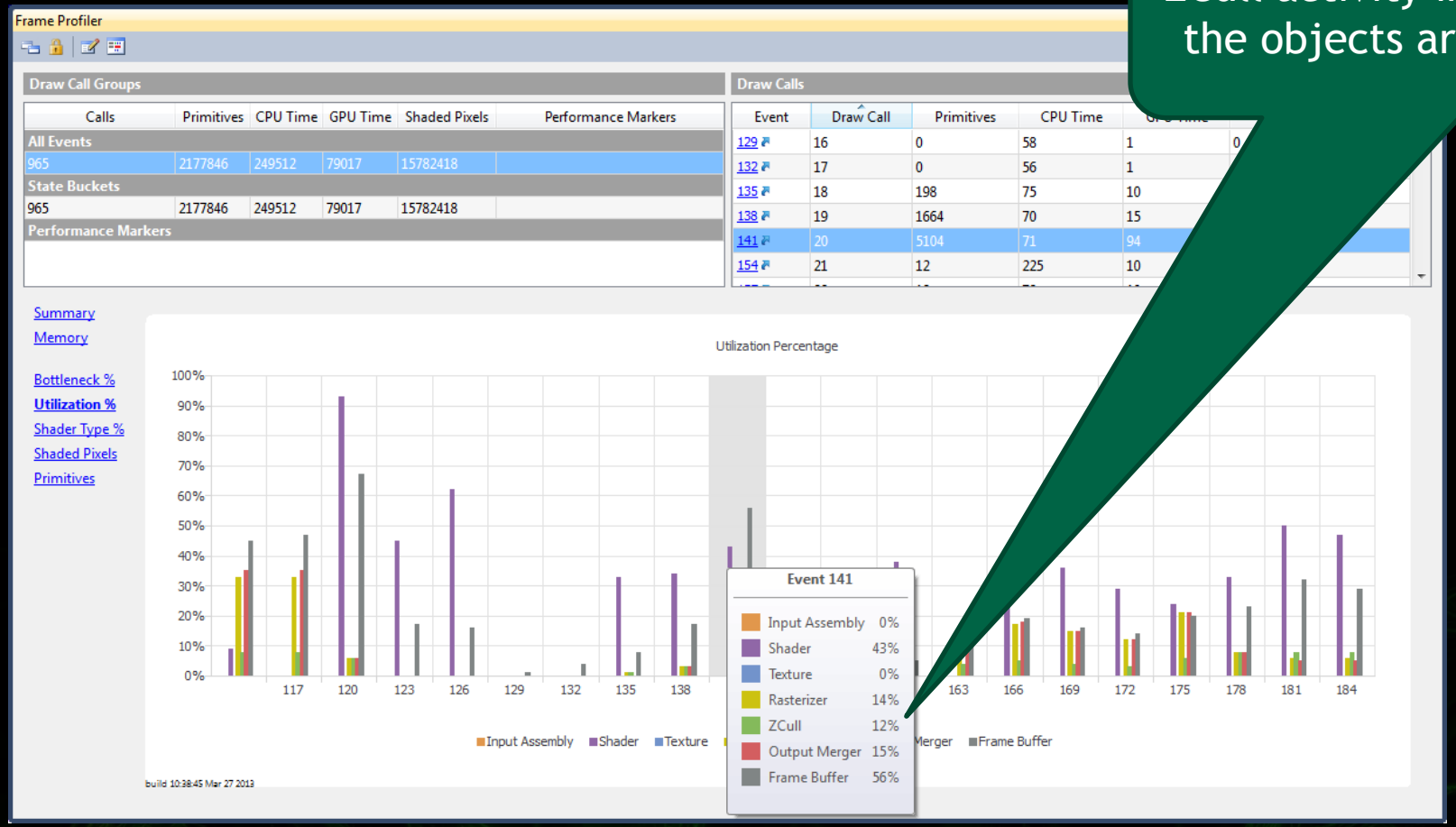


Scrubbing through the scene & saw items showing in depth buffer that never impacted the scene…

Using SW based method based on presentation by Daniel Conlin from DICE at GDC 2011

Problems During Development

#4 Bug in SW culling

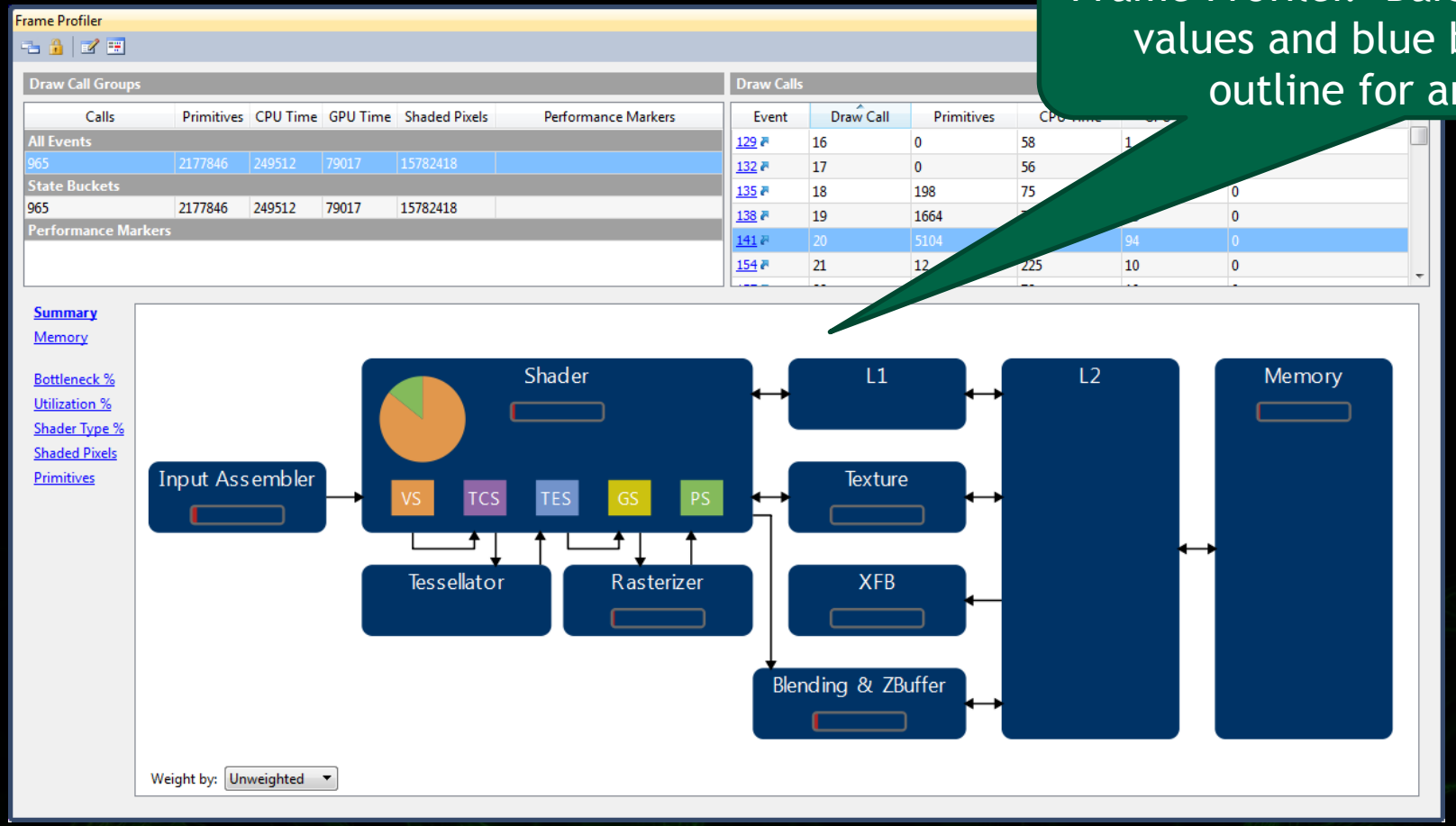


ZCull activity in the profiler confirms the objects are not going to show…

Problems During Development

#4 Bug in SW culling

Note new summary screen for OpenGL Frame Profiler. Bars show unit bottleneck values and blue boxes will have red outline for areas of interest



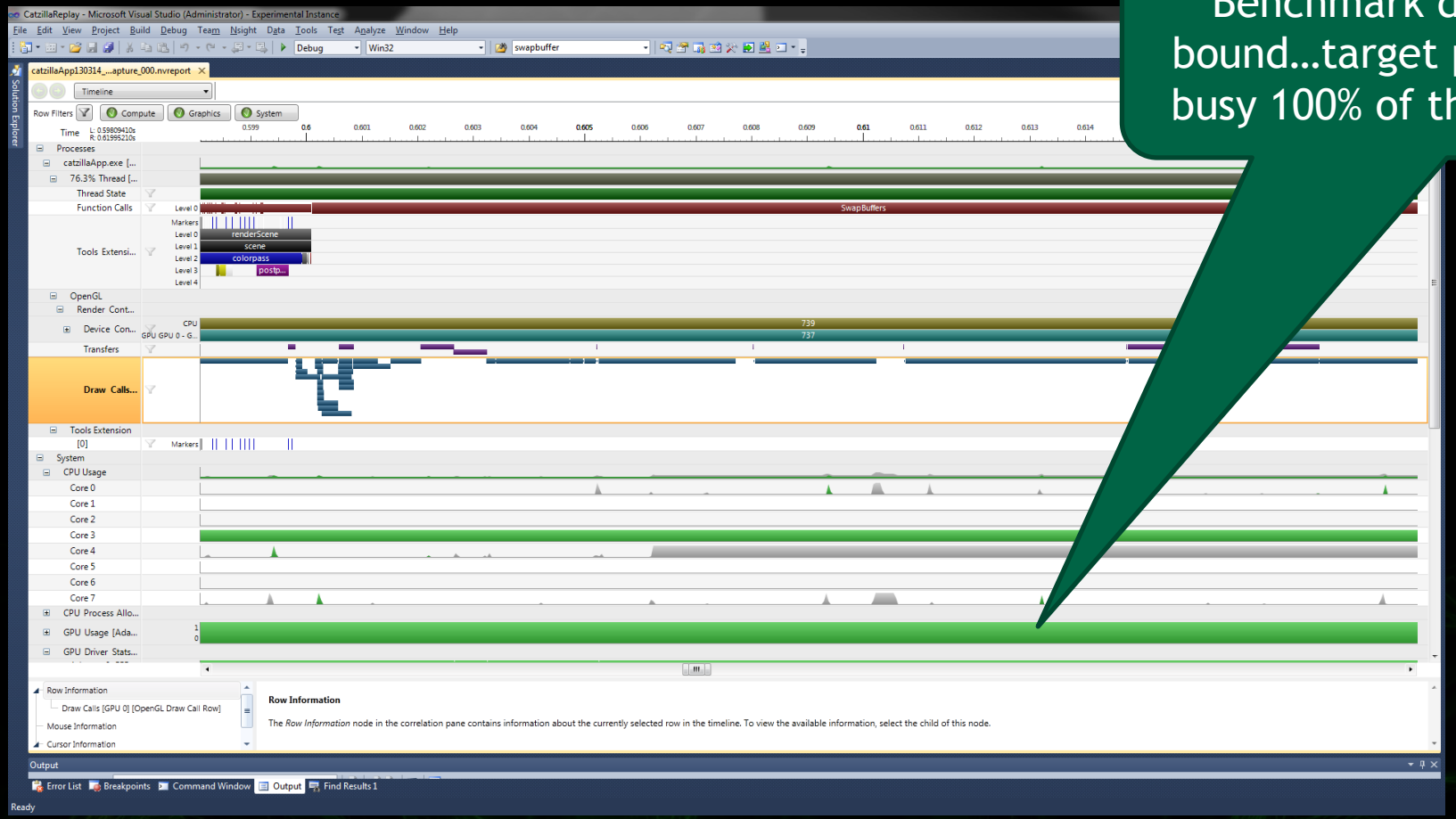
Problems During Development

#4 Bug in SW culling

Found bug in the linear depth calculation, reduced scenes by 400-500 draw calls

Problems During Rendering

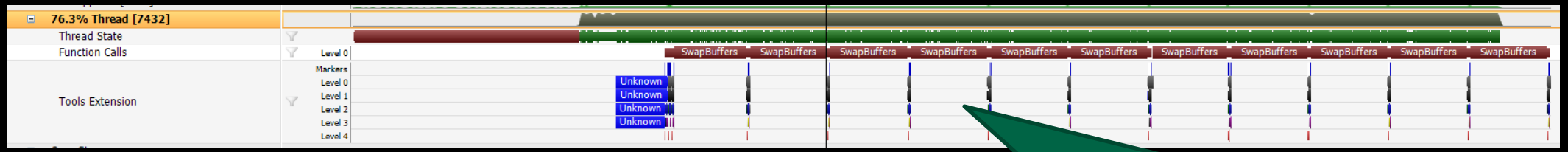
#5 Indepth look at Raymarch Test



Benchmark designed to be GPU bound…target process showing GPU busy 100% of the time…that's good!

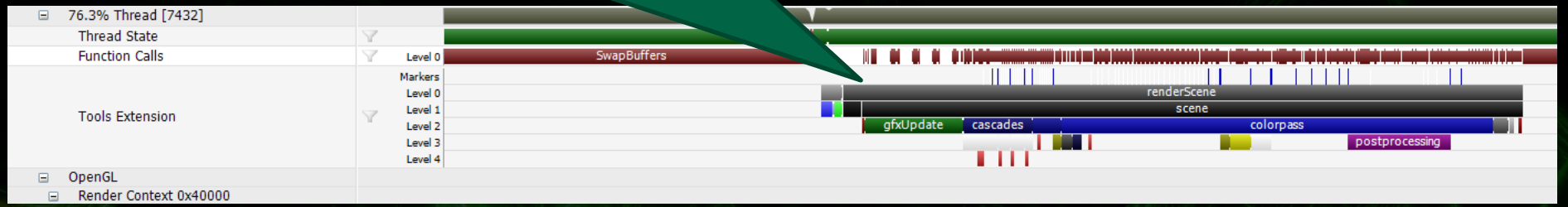
Problems During Rendering

#5 Indepth look at Raymarch Test



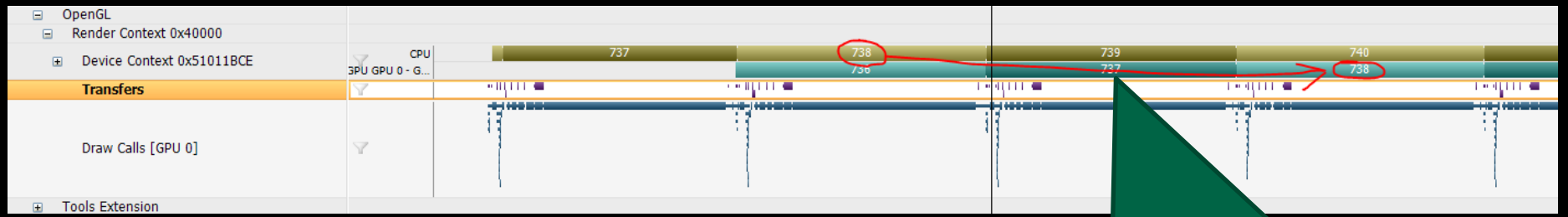
Main thread active all of the time…repeating frame pattern

NVTX used to annotate the frame to help see what was happening in the scene



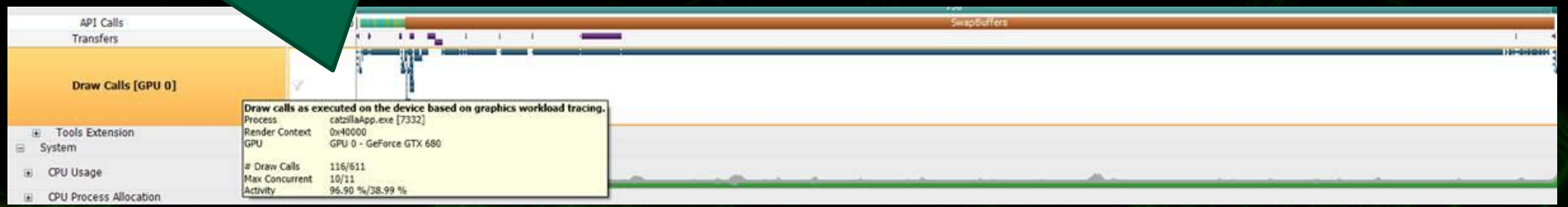
Problems During Rendering

#5 Indepth look at Raymarch Test



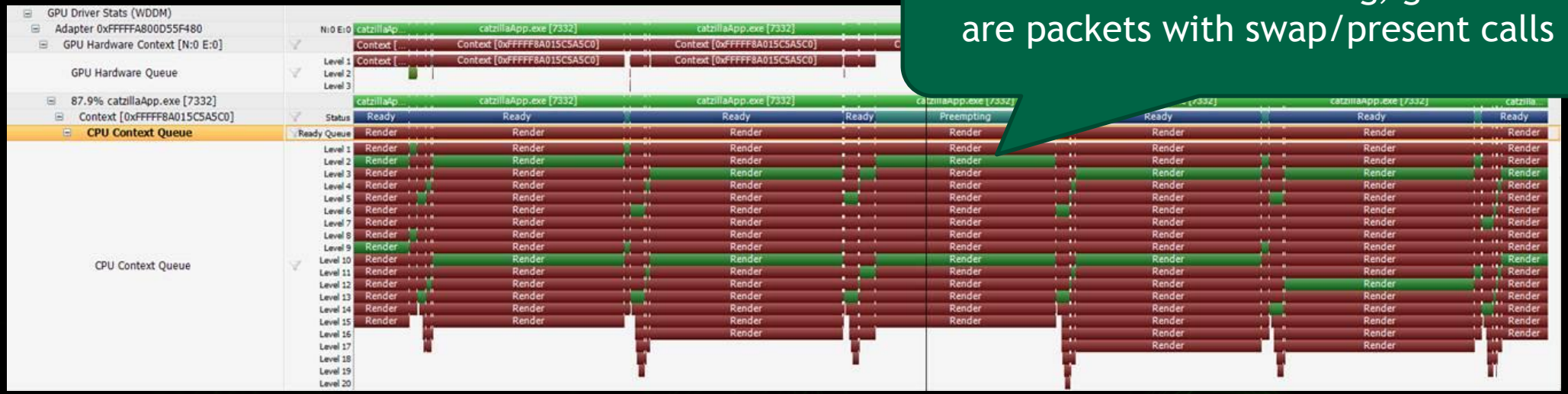
CPU is 2 frames behind the GPU, as expected for GPU bound app

Row tooltips show per frame stats, Draw Call row shows concurrent draw calls



Problems During Rendering

#5 Indepth look at Raymarch Test



3.0 includes WDDM tracing, green bars are packets with swap/present calls

Wrapping Up

- Questions/Comments?
- Resources
  - Info: https://developer.nvidia.com
  - Forums: https://devtalk.nvidia.com
- Downloads
  - http://www.nvidia.com/nsight
  - http://www.allbenchmark.com/download
- GDC Exhibit Booth: #1602