Raise your VR game with NVIDIA® GeForce® Tools

Yan An - Graphics Tools QA Manager
AGENDA

Introduction & tour of Nsight
Analyze a geometry corruption bug
VR debugging
System Analysis Tracing
GPU Range Profiling
NSIGHT VISUAL STUDIO EDITION 5.1
VR Development and Advanced Graphics Profiling

New Range Profiler, including DirectX12
New Geometry View
Oculus VR SDK support
CUDA 8.0 support
UE4’s VR Sample - Showdown

Released at Oculus Connect

Immersive firefight experience ala The Matrix

The engine completes a render pass on the left eye, then proceeds to the right

The same render target is used for each eye
UE4’s VR Engine

VR Processing

Once complete, the frame is submitted to Oculus for processing
UE4’s VR Engine

Render pass per eye

View 0
Left
View 1
Right

View 0
Left
View 1
Right

Depth Pass
Lighting Pass

Time

Frame (16ms)
PrePass (31ms)
All Events (104ms)
BasePass (151ms)
Scene (164ms)

View 0
View 1

PostOps
Static
Static/Opaque/Unlit
Dynamic
Static
Static/Opaque/Unlit
Dynamic

PostOps
View 0
View 1

UE4’s VR Engine uses a render pass per eye architecture. This approach allows for optimized rendering of each eye separately, improving the performance and visual quality of VR experiences. The diagram illustrates the timeline and pass breakdown for a single frame, highlighting the depth pass and lighting pass stages for both View 0 and View 1, ensuring a seamless rendering experience.
Walkthrough
Walkthrough

- Configure your project’s Nsight User Settings
- Nsight -> Start Graphics Debugging
- Look at the HUD
  - Status - FPS, etc.
  - Performance Graphs
- Hit hot key Ctrl-Z - now Nsight has keyboard
  - Can move graphs around, etc.
Walkthrough

• Hit space bar to capture frame
• Real time capture, app is still running, and pause the loop
• HUD capabilities
  • Scrub scene
  • Inspect textures & RTs
• All of these features are available w/o VS, can drag & drop to desktop icon
• Can serialize a frame to generated C++ source code, support VR API!
Walkthrough

- Alt+Tab back to VS
- Scrubber
  - Scrub through scene, get context via Perf Markers
- Event View
  - Display API call with parameters
  - Connected to scrubber
  - Filtering - ‘draw\|pssetshader\|’
- If you are leveraging middleware, you may not have a choice of the perfmarker that they use. You can specify your perfmarker to investigate your own code.
- Current Targets - view all bound, can switch different RTVs and DSVs

GDC

gameworks.nvidia.com
Walkthrough

• Make sure you are on a draw call, let’s look at the “API Inspector”

• Basic Graphics/GPU pipeline on left

• Each stage contains the state of the API at current draw call
  
  • IA: all of your vertex setup
  
  • VS, HS, DS, PS, CS: shader stages
  
  • SO
  
  • Raster
  
  • OM
Walkthrough

• Open up the “API Statistics” view just over 1k draw calls but almost 30k “other”
• AddRef/Release is ~13k calls!
• If you filtered for “GetDesc”…600 Calls but Descs are immutable. This might be worth caching
• Sort the object column
• Look at the textures object ids; note that some are called 8-12 times
• Caching/storage cost for D3D11_TEXTURE2D_DESC is 44 bytes
Walkthrough

• Some of the normal features in Nsight are useful for VR Debugging

• New scrubber feature: add rows with ranges that have different characteristics

• Click the “+” button and select “viewport”, offers a high-level view of each eye’s render pass

• This is especially useful if you haven’t configured your own perfmarkers
Welcome to Nsight!

Utilize basic views to find potential CPU perf issues

HUD performance graphs & scrubbing

Visual Studio integrated views

Scrubber, Events & Current Target

API Inspector & API Statistics

Geometry & Resources
Bad Geometry
Bad Geometry

• Start Nsight as we described in “Walkthrough” section

• Not a real bug from UE, but representative of a class of problems users have reported: what happens to my render result?
Bad Geometry

• Pause and capture the frame, scrub on the HUD to the *bad draw call*

• Many causes to this type of problem, use Nsight to check things that can go wrong
  • API Inspector - VS stage
  • API Inspector - RS stage
  • Etc.
Bad Geometry

- API Inspector - VS stage, collapse all and check buffers, is anything wrong?
Bad Geometry

- API Inspector - RS stage, check fill mode, cull mode, depth bias, etc. is anything wrong?
Bad Geometry

• Looks like our processing is set up properly, going to guess it might be a data input problem

• Let us open Geometry View!
Bad Geometry

• Open the Geometry View [new view with Nsight 5.1]
  • Can already see that Nsight detected some FP specials (let us discuss later)

• Geometry Graphical View
  • Ability to map any attribute to one of 3 characteristics for the rendering: pos/color/normal
  • Visualize things like normals or UV as color
  • Settings persisted according to the vertex layout

• Geometry Memory Table View
  • Each vertex is a row with the columns as each attribute
  • Can see in index buffer order or vertex buffer order
  • Takes into account any offsets being used
Bad Geometry

- Back to the “FP specials detection”
- Could use the “tag” feature in the resources view to show all uses of this buffer in the scene. Not only tag the vertex buffers, but also support any resources, like texture.
BAD GEOMETRY RECAP

Corruption in vertex data
API Inspector to check pipeline setup
Geometry View to inspect pre-transform data
Resources View + ‘tag’ to show all uses
VR...not so Reality
Parallax & Binocular Vision

- pick an object in the foreground, like your finger or someone’s head
- Now alternate covering your left and right eye
- Notice how the object ‘moves’ relative to the background as you switch eyes
- This is called “parallax” and is part of the nature of our binocular vision system
- This gives us the ability to perceive depth, and is what is used in stereo/VR to give a much more immersive experience
Parallax & Binocular Vision
VR...not so Reality

- Sometimes, stereo can break down in your VR rendering and kill the experience.
- Ever seen cheesy “3D” where everything just looks like several different depths of billboards?
- In this scene, look at the cement chunks that are flying through the air.
- If you had an HMD, you would see them floating unnaturally in space, almost like a HUD element or reticle site.
VR...not so Reality

- Start Nsight as we described in “Walkthrough” section
- Capture the frame (Ctrl-Z and Space as shortcut keys)
- Let’s use the Resources View to dig into the texture used here to confirm what we are seeing
- First, let us find the drawcalls for the “not so reality” effect
VR...not so Reality

- Go to event that render the left eye
- Hit Ctrl+F2 to bookmark it in Event View
- Go to event that render the right eye
- Hit Ctrl+F2 to bookmark it in Event View
- Again, if we didn’t have Perf Markers, you could use a new Viewport row on the scrubber to confirm that these calls go to different eyes
- Go to CTV (Current Target View) and click on the depth texture, it will open Resources View for detail inspect
VR...not so Reality

- Check the current DSV
- Note that you can see just black image.
- Open up the histogram and renormalize (click the little arrow on the bottom right side of texture display), and drag the max arrow to adjust the range of your depth, so that you can see details
- Note the depth of the chunk for each eye
VR...not so Reality

- Use API Inspector as in previous section and confirm the setup of rendering, and that it is all OK.
- What’s else? We need to look at the VR setup!
- Open VR Inspector view
- You will see we show Swap Chains, Mirror Textures, Render Desc Queries, info about the HMD
### Swap Chains

<table>
<thead>
<tr>
<th>Resources</th>
<th>CurrentIndex</th>
<th>Type</th>
<th>Format</th>
<th>ArraySize</th>
<th>Width</th>
<th>Height</th>
<th>MipLevels</th>
<th>SampleCount</th>
<th>StaticImage</th>
<th>MiscFlags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texture2D 342</td>
<td>0</td>
<td>owTexture_2D</td>
<td>OVR_FORMAT_B8G8R8A8_UNORM_SRGB</td>
<td>1</td>
<td>16</td>
<td>16</td>
<td>5</td>
<td>1</td>
<td>owTrue</td>
<td>owTextureMisc_AllowGenerateMip</td>
</tr>
<tr>
<td>Texture2D 343</td>
<td>1</td>
<td>owTexture_2D</td>
<td>OVR_FORMAT_B8G8R8A8_UNORM_SRGB</td>
<td>1</td>
<td>2432</td>
<td>1448</td>
<td>1</td>
<td>1</td>
<td>owFalse</td>
<td>owTextureMisc_DX_Typeless</td>
</tr>
</tbody>
</table>

### Mirror Textures

<table>
<thead>
<tr>
<th>Resource</th>
<th>Format</th>
<th>Width</th>
<th>Height</th>
<th>MiscFlags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Texture2D 346</td>
<td>OVR_FORMAT_B8G8R8A8_UNORM_SRGB</td>
<td>1920</td>
<td>1080</td>
<td>owTextureMisc_DX_Typeless</td>
</tr>
</tbody>
</table>

### Render Desc Queries

<table>
<thead>
<tr>
<th>Eye</th>
<th>DistortedViewport</th>
<th>Fov</th>
<th>PlaneFarPlaneNearPlaneAtCenter</th>
<th>HmdIdVelOffset</th>
</tr>
</thead>
<tbody>
<tr>
<td>owEye_Left</td>
<td>(1.33, 1.33, 1.06, 1.09)</td>
<td>(0.00, 0.00, 0.00, 0.00)</td>
<td>(560.00, 1080.00)</td>
<td>(549.62, 549.62)</td>
</tr>
<tr>
<td>owEye_Left</td>
<td>(1.33, 1.33, 1.09, 1.06)</td>
<td>(0.00, 0.00, 0.00, 0.00)</td>
<td>(560.00, 1080.00)</td>
<td>(549.62, 549.62)</td>
</tr>
</tbody>
</table>

### HMD Description

- **Type**: oculus DK2
- **ProductLine**: Oculus Rift DK2
- **Manufacturer**: Oculus VR
- **VendorId**: 10291
- **ProductId**: 33
- **SerialNumber**: 29909500004M
- **FirmwareMajor**: 2
- **FirmwareMinor**: 12
- **RuntimeVersion**: 1.60
- **AvailableHmdCaps**: 0x00000000
- **DefaultHmdCaps**: 0x00000000
- **DefaultEyeFov**: (1.33, 1.33, 1.06, 1.09) (1.33, 1.33, 1.09, 1.06)
- **MaxEyeFov**: (1.87, 1.87, 1.62, 1.62)
- **Resolution**: (1920.00, 1080.00)
- **DisplayRefreshRate**: 75.00
VR...not so Reality

Click through to ‘Show API Usage’ to show VR APIs in the events view
VR...not so Reality

- Look closer at the Render Desc Queries section
- Note that the app queried for the left eye twice
- Must have been a typo where they meant to query for the right eye...copy paste strikes again
- Change to query the right eye, problem solved!
VR...NOT SO R RECAP

Bookmarking in the Events View
Resources View image inspection
Histogram viewer and remapping
API Inspector for the rendering setup
VR Inspector for the VR setup
Where’s the bottleneck?
Where’s the bottleneck?

- Start Nsight as we described in “Walkthrough” section
- If you are GPU bound, you want to use a profiler to determine what part of the pipeline is overtaxed and leading to your performance drop
- Right click on scrubber and select “Profile Frame” to open “Range Profiler” view
- Profiler is draw call centric, allowed grouping of draw calls into state buckets
- New is “range” centric: a consecutive list of draw calls.
Where’s my bottleneck?

- New paradigm is “ranges”, which can be based on rendering state
  - Perf Markers
  - Program Ranges (all 6 shader stages)
  - Viewport (as we saw in the VR section)
  - Render targets (see how expensive each one is)
- User - create a range on the fly
Where’s my bottleneck?

- There are 3 sections for more detail profiler information
  - Range Info
    - Shows basic information about current range (primitives, pixels, compute shader, etc.)
    - Combo in upper right about ‘Action Details’ allows you to select per draw call info
  - One of the key improvement is the ability to selectively profile different sections for just the data you want. This will help in modern APIs where the draw call counts are going from a few thousand to 10s of thousands
Where’s my bottleneck?

Pipeline Overview

“Summary”, We show you the top 4 inefficient or bottlenecked stages

“Range Details” shows the classic pipeline diagram with each unit’s bottleneck

“Action Details” shows the detail information per drawcall
Where’s my bottleneck?

- **Memory**
  - “Summary”, we show shows some basic memory statistics including L2 usage, etc.
  - “Range Details” shows the requests/bytes between each one of the stages that utilize the memory subsystem
  - “Action Details” shows more detail info per draw call
Where’s my bottleneck?

- Range Profiler and Dynamic Shader Edit
  - Pick really expensive draw call, make a user range around it and select
  - Go to API Inspector
  - Click shader link, see source code
  - Right click and use Shader Edit, code for shader optimization
  - Lock profiler and Clone a new profiler view
  - Rerun profiler and make sure the values for that draw call have dropped as your optimization
BOTTLENECK RECAP

Range Profiler

Scrubber with ranges

New data view(s)

Could optimize and rerun profile

edit your shader on the fly, make it more efficient

rerun Profiler, prove the optimize works as expected
Questions?

- Support forum of Nsight:

- Support Email:
  - devtools-support@nvidia.com

Any question, discussion, suggestion is welcome!
Thanks! 😊