Advances in Real-Time Automotive Visualisation

Chris O’Connor
At ZeroLight, we’ve created the market-leading visualisation and data platform for the automotive industry.
Our Content Maintains the Same High Quality and Real-time Configurability Across All Platforms
Amplifying personalisation through machine learning
Ray Tracing and Next-gen VR
This Section Will Cover

- Why ray tracing is important
- Details of custom tech first shown at GTC18
- Rasterisation and ray tracing
- DXR and RTX
What’s Missing from Automotive Real-time Rendering?

- Audi asked us what's next to improve graphics quality?
- Started development in 2017
- Content already had passed look and quality control
- We had to find a solution that adds quality to the current look
Self-reflection Options

- SSR (Screen Space Reflections) – lots of issues
- Localised Planar Reflections – great for some areas
- Compute ray tracing – too slow for real-time
- Voxels reflections – perfect for GPU acceleration
Voxel Ray-traced Reflections

- GPU can be used to generate voxels
- Quick ray cast and look up in Pixel Shader
- Uses a lot of memory
- Sparse voxels helps a lot
- Difficult to update dynamically
Voxel Ray-traced Reflections
Ray Tracing Demo GTC 2018

- We showed our solution at GTC 2018 on the NVIDIA booth
- Showing the Audi A3 with accessories
- Rendering 4K at 60FPS
- 500 million rays per second (0.5 billion)
- Used 2x GV100 (Volta GPU)
Voxel Ray-traced Reflections
Ray Tracing Demo GTC 2018
DXR + RTX

- DXR and RTX hardware accelerated ray tracing
- Up to 10 billion rays per second
- Resources all managed by DXR
- Native support: rays and pixels can be dispatched in parallel
- Denoiser = fewer rays required per pixel
Rasterisation and Ray Tracing

- Key to maintaining the look across devices that can’t ray trace
- Let each part of the GPU do what it does best
- Ray tracing can be used for shadows, AO, GI and reflections
- Stage 1: graphical improvement without changing the look
- Stage 2: runs at high quality in real-time (4K @ 60FPS)
Rasterisation and Ray Tracing
Ray Tracing
Ray Tracing
Next-gen VR
This Section Will Cover

- Current VR headsets
- Next-gen VR headsets
- Optimising for next-gen VR
- Quality and results
Current VR Headsets
Current VR Headsets

• 1080x1200 / 1440x1440 resolution per eye
• 90 FPS
• 95° - 110° FOV
• 2 rendered viewports, one per eye
Features of New Headsets

- Higher resolution up to 2560x1440 per eye
- 90 FPS
- Up to 210° FOV
- Multiple rendered viewports per eye (wide FOV, foveated rendering)
VIVE Pro and VIVE Pro Eye
StarVR One, Pimax 8K, XTAL
Varjo VR-1
What's the Challenge?

- If you just brute force render viewports and pixels, even with the best GPUs, it will be a challenge to hit performance.
- You must take advantage of GPU software-activated hardware features.
- NVIDIA VR Works with NVAPI.
VR SLI

- Previously, we used AFR (Alternate Frame Rendering)
- Great for GPU Utilisation (80%+)
- AFR adds additional 11ms of latency (bad for VR)
- VR SLI renders one eye per GPU
- Copy of buffer back to GPU0 is the bottleneck (PCI Express)
- Easy to instance per eye, reducing CPU overhead
VR SLI

<table>
<thead>
<tr>
<th>GPUs: 2 NVIDIA</th>
<th>Drivers: 416.81</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU-0: 90%</td>
<td>GPU-1: 84%</td>
</tr>
<tr>
<td>View 0 SinglePassRenderView: 9.896927 [17.51731]</td>
<td></td>
</tr>
<tr>
<td>View 0 RenderUI: 0.03110934 [0.0616]</td>
<td></td>
</tr>
<tr>
<td>RealtimeShadowBlur: 0.6990316 [0.60512]</td>
<td></td>
</tr>
<tr>
<td>VRAM: 7208/12238</td>
<td></td>
</tr>
<tr>
<td>View 0 PostPass: 0.8722379 [1.727328]</td>
<td></td>
</tr>
<tr>
<td>RealtimeShadowRender: 3.098377 [4.358176]</td>
<td></td>
</tr>
<tr>
<td>Milliseconds Waited For Result: 2</td>
<td></td>
</tr>
</tbody>
</table>
VR SLI

- We have to use tricks to copy less data and prevent GPUs waiting for copies

Copy Data Async

```cpp
m_pMultiGPUDevice->CopySubresourceRegion(deviceContext, rtcd.m_RenderTargetTexture, 0, dstGPU, rtcd.m_Left, rtcd.m_Top, 0, rtcd.m_RenderTargetTexture, 0, srcGPU, &srcBox, NVAPI_COPYASYNCHRONOUSLY);
```
Next-gen VR

VR SLI

Viewport cut and copy
Single Pass Stereo

- Reduce CPU overhead and render in a single pass for both eyes
- Requires shader modifications

```cpp
//Standard
float4 standardPos = mul(MATRIX_MVP, vertex);
outPos.position = standardPos;

//Single Pass Stereo
float4 eyePos0 = mul(g_ViewProjMatrix[1], worldPos);
float4 eyePos1 = mul(g_ViewProjMatrix[2], worldPos);

outPos.position = eyePos0;
outPos.posClipRight = eyePos1;
outPos.ViewportMask = NV_SINGLERT_VIEWPORT_MASK;
```

- Mid-frame effects become more complex
New with Turing GPUs

VARIABLE RATE SHADING

MULTI-VIEW RENDERING
NVLINK2

- Reduces copy time from 4ms to 0.1ms (StarVR One)
- We can now share additional data between the GPUs
Multi-view Rendering

- Very similar to SPS but supports 4 views
- Big CPU optimisation on HMDs that require >2 Viewports

```c
//Standard
float4 standardPos = mul(MATRIX_MVP, vertex);
outPos.position = standardPos;

//Single Pass Stereo
float4 eyePos0 = mul(g_ViewProjMatrix[1], worldPos);
float4 eyePos1 = mul(g_ViewProjMatrix[2], worldPos);

outPos.position = eyePos0;
outPos.posClipRight = eyePos1;
outPos.ViewportMask = NV_SINGLERT_VIEWPORT_MASK;

//Multi-View Rendering
float4 eyePos0 = mul(g_ViewProjMatrix[1], worldPos);
float4 eyePos1 = mul(g_ViewProjMatrix[2], worldPos);
float4 eyePos2 = mul(g_ViewProjMatrix[3], worldPos);
float4 eyePos3 = mul(g_ViewProjMatrix[4], worldPos);

outPos.position = eyePos0;
outPos.position_v1 = eyePos1;
outPos.position_v2 = eyePos2;
outPos.position_v3 = eyePos3;
outPos.ViewportMask = NV_SINGLERT_VIEWPORT_MASK;
outPos.ViewportMask2 = NV_SINGLERT_VIEWPORT_MASK_2;
```
Variable Rate Shading

- Vary the pixel density of your render target
- Big fill rate improvements possible
- Lens-optimised shading and foveated rendering
- The larger the render target the more that can be saved
- Supersampling antialiasing makes VR look great
Variable Rate Shading

VRS Supersampling Performance

- 16x
- 9x
- 4x
- 1x

- Base (No VRS)
- VRS Lens Optimised Mask
- VRS Foveated Rendering
Foveated Rendering
Variable Rate Shading
Variable Rate Shading

```c
void Plugin_Initialize()
{
    // Initialize NVAPI
    NvAPI_Initialize();

    // Check VRS Support
    NV_D3D1x_GRAPHICS_CAPS caps = {}
    if (NvAPI_D3D1x_GetGraphicsCapabilities(d3d11NativeDevice,
      NV_D3D1x_GRAPHICS_CAPS_VER, &caps) == NVAPI_OK
      && caps.bVariablePixelRateShadingSupported)
    {
        isTuringCard = true;
    }
}

// Post main camera render
void Plugin_DisableVRS()
{
    // Create look up table with shading rates and set
    vsrViewportDescription.numViewports = 0;
    NvAPI_D3D11_RSSetViewportsPixelShadingRates(deviceContext,
      &vsrViewportDescription);
}

// Pre main camera Render
void Plugin_EnableVRS(

    void* VRStextureResourceView
)
{
    // Create look up table with shading rates and set
    vsrViewportDescription.numViewports = 2;
    NvAPI_D3D11_RSSetViewportsPixelShadingRates(deviceContext,
      &vsrViewportDescription);

    // Set texture to be used as index into shading rate table
    NvAPI_D3D11_RSSetShadingRateResourceView(deviceContext,
      VRStextureResourceView);
}

// Engine main camera render
GameEngineCamera.Render();

// Update texture using compute shader or render
void RenderFullScreenWithShader(VRSTexture)
{
    // Create look up table with shading rates and set
    vsrViewportDescription.numViewports = 0;
    NvAPI_D3D11_RSSetViewportsPixelShadingRates(deviceContext,
      &vsrViewportDescription);
}
```
Porsche StarVR One
Porsche StarVR One
Porsche StarVR One

4 Viewport Split over wide FOV configuration

4 Viewport Foveated Rendering Configuration
Porsche StarVR One
Porsche StarVR One

GPU Profile Standard 4 Viewport wide FOV rendering

GPU Profile Foveated Rendering 4 Viewports
Audi Varjo VR-1
Audi Varjo VR-1
BMW VIVE Pro Eye

The New VIVE Pro
With Precision Eye Tracking
BMW VIVE Pro Eye

**Base (No VRS)**

<table>
<thead>
<tr>
<th>Supersampling Level</th>
<th>Frame Time (ms)</th>
<th>GPU Load (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x</td>
<td>16</td>
<td>100</td>
</tr>
<tr>
<td>9x</td>
<td>14</td>
<td>75</td>
</tr>
<tr>
<td>4x</td>
<td>12</td>
<td>50</td>
</tr>
<tr>
<td>1x</td>
<td>10</td>
<td>25</td>
</tr>
</tbody>
</table>

**VRS Foveated Rendering**

<table>
<thead>
<tr>
<th>Supersampling Level</th>
<th>Frame Time (ms)</th>
<th>GPU Load (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x</td>
<td>8</td>
<td>100</td>
</tr>
<tr>
<td>9x</td>
<td>7</td>
<td>75</td>
</tr>
<tr>
<td>4x</td>
<td>6</td>
<td>50</td>
</tr>
<tr>
<td>1x</td>
<td>5</td>
<td>25</td>
</tr>
</tbody>
</table>
BMW VIVE Pro Eye
Find Out More

https://devblogs.nvidia.com/
Profiling

NVIDIA® Nsight™

INTEL® GRAPHICS PERFORMANCE ANALYZERS

https://software.intel.com/gpa
Follow Our Tech Updates on Twitter

@TechZeroLight

Questions?

chris.oconnor@zerolight.com

www.linkedin.com/in/chris-zerolight