Advanced VR Rendering

Alex Vlachos, Valve

Alex@ValveSoftware.com

GAME DEVELOPERS CONFERENCE®

MOSCONE CENTER · SAN FRANCISCO, CA

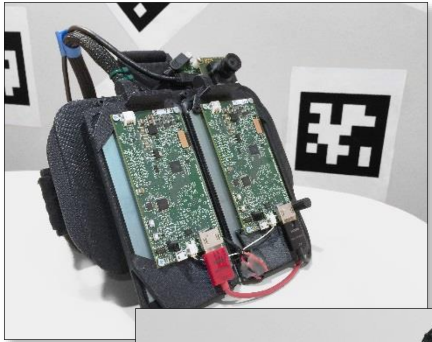
MARCH 2-6, 2015 · EXPO: MARCH 4-6, 2015

Outline

- VR at Valve
- Methods for Stereo Rendering
- Timing: Scheduling, Prediction, VSync, GPU Bubbles
- Specular Aliasing & Anisotropic Lighting
- Miscellaneous VR Rendering Topics

VR at Valve

- Began VR research 3+ years ago
- Both hardware and software engineers
- Custom optics designed for VR
- Display technology – low persistence, global display
- Tracking systems
  - Fiducial-based positional tracking
  - Desktop dot-based tracking and controllers
  - Laser-tracked headset and controllers
- SteamVR API – Cross-platform, OpenVR

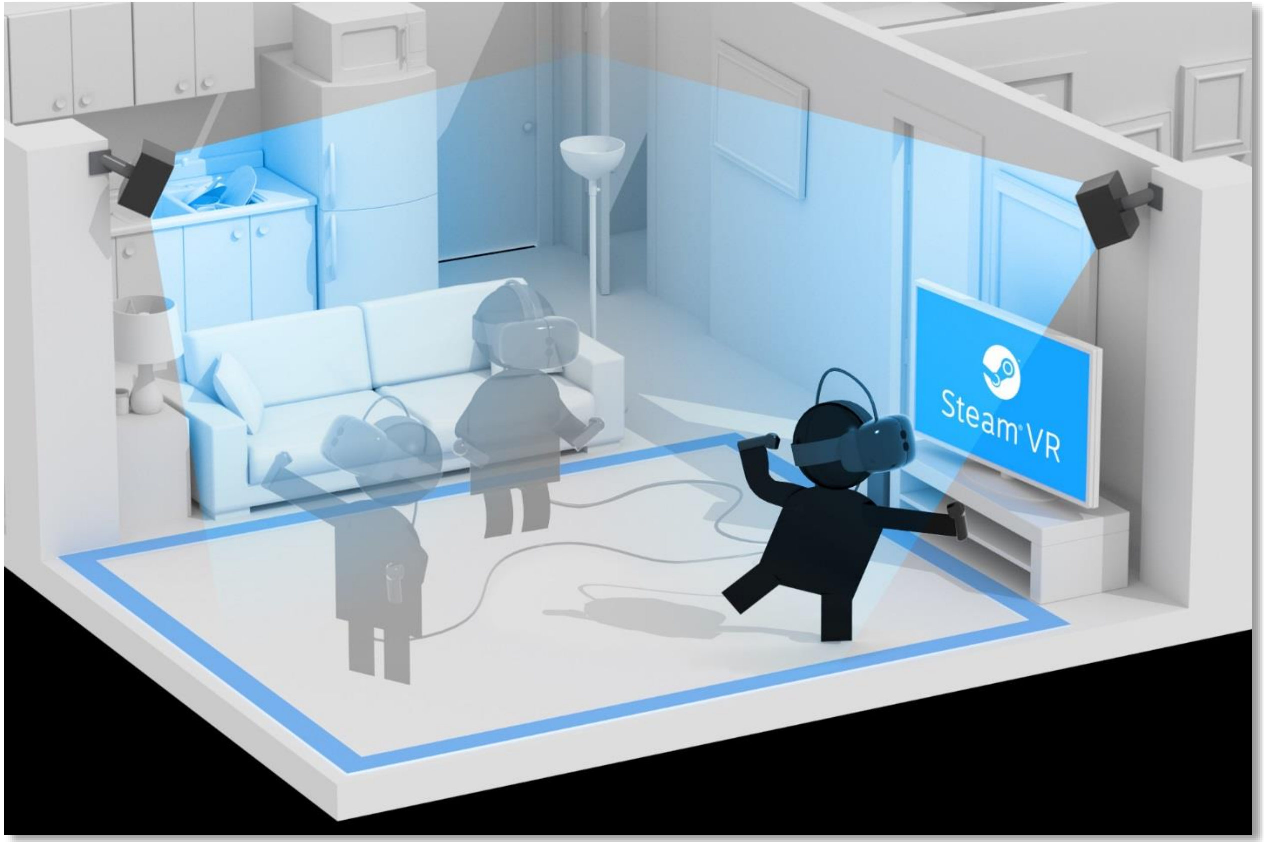


HTC Vive Developer Edition Specs

VALVE®

- Refresh rate: 90 Hz (11.11 ms per frame)
- Low persistence, global display
- Framebuffer: 2160x1200 (1080x1200 per-eye)
- Off-screen rendering ~1.4x in each dimension:
  - 1512x1680 per-eye = 2,540,160 shaded pixels per-eye (brute-force)
- FOV is about 110 degrees
- 360º room-scale tracking
- Multiple tracked controllers and other input devices

Room-Scale Tracking



Optics & Distortion (Pre-Warp)

Warp pass uses 3 sets of UVs for RGB separately to account for spatial and chromatic distortion

(Visualizing 1.4x render target scalar)

Optics & Distortion (Post-Warp)

Warp pass uses 3 sets of UVs for RGB separately to account for spatial and chromatic distortion

(Visualizing 1.4x render target scalar)

Shaded Visible Pixels per Second

VALVE

- 720p @ 30 Hz: **27** million pixels/sec
- 1080p @ 60 Hz: **124** million pixels/sec
- 30” Monitor 2560x1600 @ 60 Hz: **245** million pixels/sec
- 4k Monitor 4096x2160 @ 30 Hz: **265** million pixels/sec
- VR 1512x1680x2 @ 90 Hz: **457** million pixels/sec
  - We can reduce this to **378** million pixels/sec (later in the talk)
  - Equivalent to 30” Monitor @ 100 Hz for a non-VR renderer

There Are No "Small" Effects

VALVE

- Tracking allows users to get up close to anything in the tracked volume
- Can't implement a super expensive effect and claim "it's just this small little thing in the corner"
- Even your floors need to be higher fidelity than we have traditionally authored
- If it's in your tracked volume, it must be high fidelity

VR Rendering Goals

VALVE

- Lowest GPU min spec possible
  - We want VR to succeed, but we need customers
  - The lower the min spec, the more customers we have
- Aliasing should not be noticeable to customers
  - Customers refer to aliasing as “sparkling”
- Algorithms should scale up to multi-GPU installations
  - Ask yourself, “Will ‘X’ scale efficiently to a 4-GPU machine?”

Outline

- VR at Valve
- **Methods for Stereo Rendering**
- Timing: Scheduling, Prediction, VSync, GPU Bubbles
- Specular Aliasing & Anisotropic Lighting
- Miscellaneous VR Rendering Topics

Stereo Rendering (Single-GPU)

VALVE®

- Brute-force run your CPU code twice (BAD)
- Use geometry shader to amplify geometry (BAD)
- Resubmit command buffers (GOOD, our current solution)
- Use instancing to double geo (BETTER. Half the API calls, improved cache coherency for VB/IB/texture reads)
    - *"High Performance Stereo Rendering For VR"*, Timothy Wilson, San Diego Virtual Reality Meetup

Stereo Rendering (Multi-GPU)

VALVE

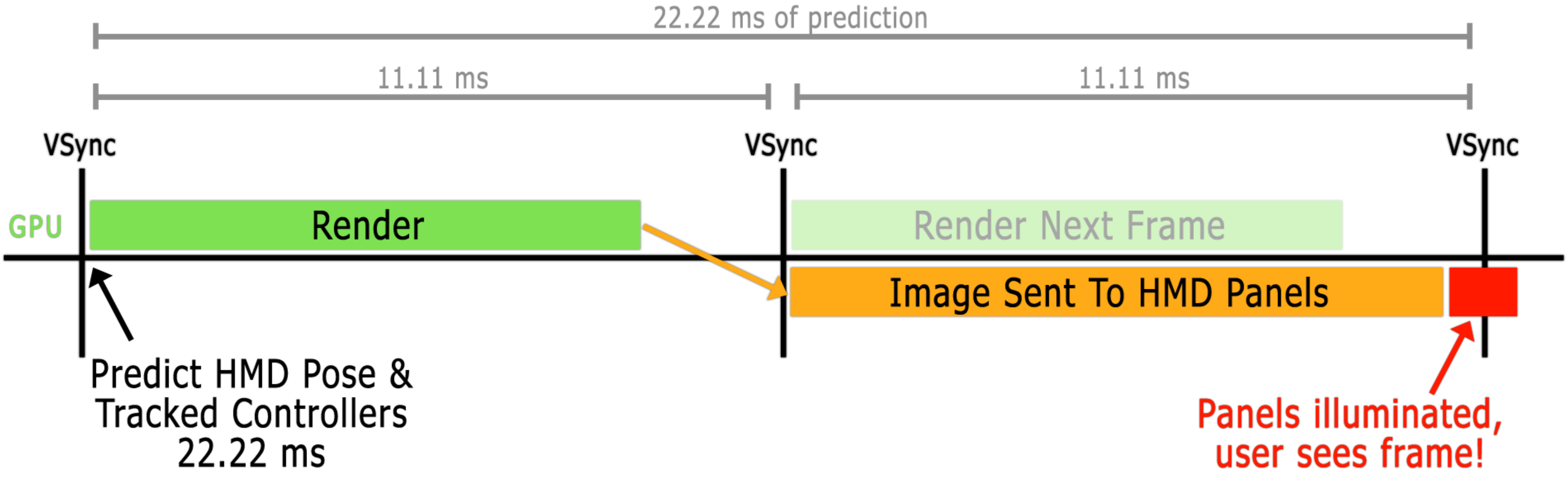
- AMD and NVIDIA both provide DX11 extensions to accelerate stereo rendering across multiple GPUs
  - We have already tested the AMD implementation and it nearly doubles our framerate – have yet to test the NVIDIA implementation but will soon
- Great for developers
  - Everyone on your team can have a multi-GPU solution in their dev box
  - This allows you to break framerate without uncomfortable low-framerate VR
  - But lie to your team about framerate and report single-GPU fps :)

Outline

- VR at Valve
- Methods for Stereo Rendering
- **Timing: Scheduling, Prediction, VSync, GPU Bubbles**
- Specular Aliasing & Anisotropic Lighting
- Miscellaneous VR Rendering Topics

Prediction

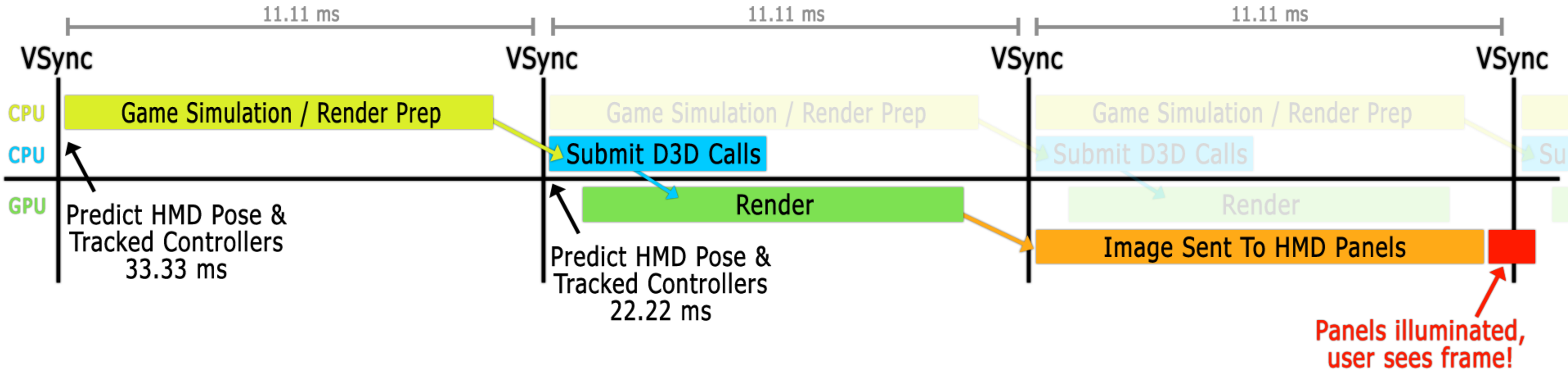
- We aim to keep prediction times (render to photons) for the HMD and controller transforms as short as possible (accuracy is more important than total time)
- Low persistence global displays: panel is lit for only ~2 ms of the 11.11 ms frame



NOTE: Image above is not optimal VR rendering, but helps describe prediction (See later slides)

Pipelined Architectures

- Simulating next frame while rendering the current frame



- We re-predict transforms and update our global cbuffer right before submit
- VR practically requires this due to prediction constraints
- You must conservatively cull on the CPU by about 5 degrees

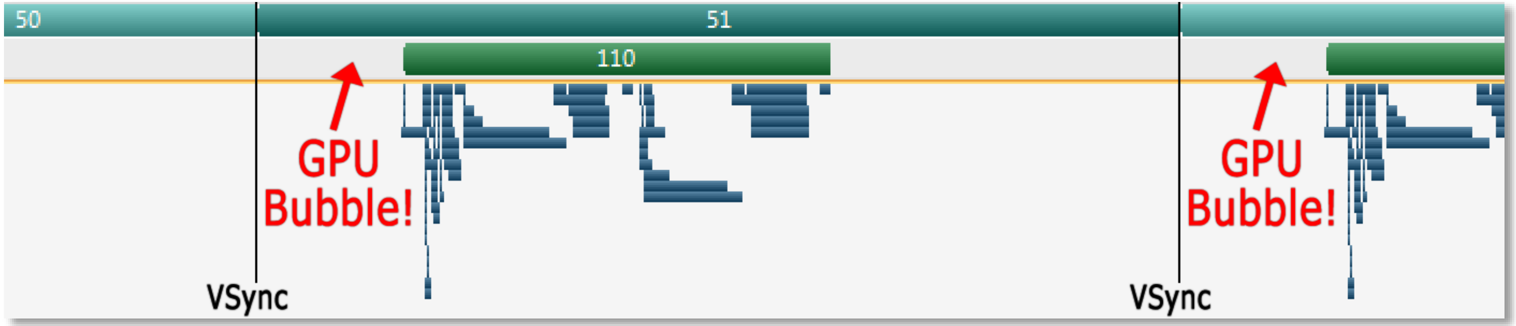
Waiting for VSync

VALVE

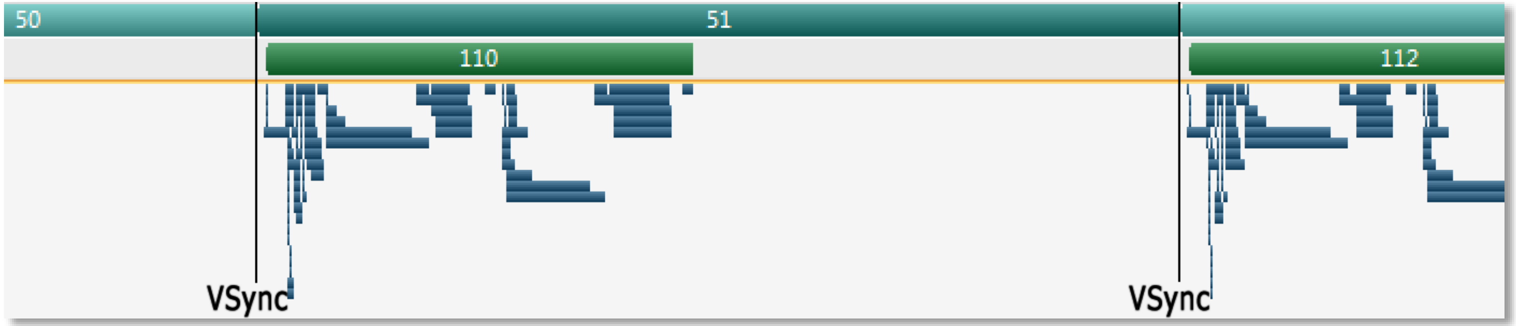
- Simplest VR implementation, predict right after VSync
  - Pattern #1: Present(), clear back buffer, read a pixel
  - Pattern #2: Present(), clear back buffer, spin on a query
- Great for initial implementation, but please DO NOT DO THIS. GPUs are not designed for this.
- See John McDonald's talk:
  - *"Avoiding Catastrophic Performance Loss: Detecting CPU-GPU Sync Points"*, John McDonald, NVIDIA, GDC 2014

GPU Bubbles

- If you start submitting draw calls after VSync:



- Ideally, your capture should look like this:

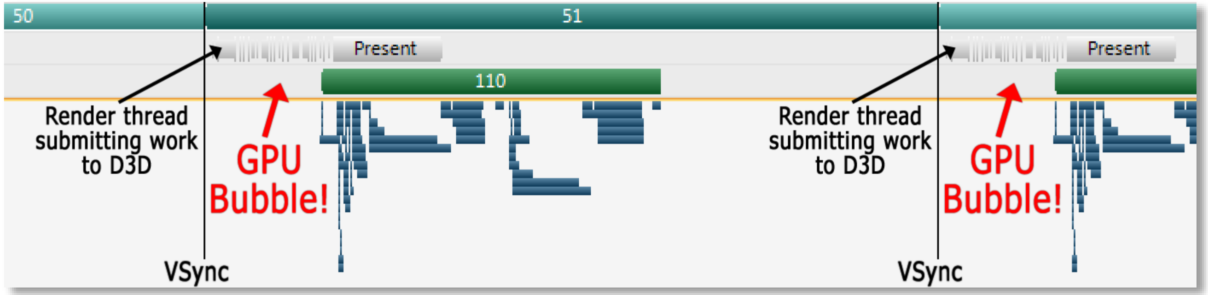


(Images are screen captures of NVIDIA Nsight)

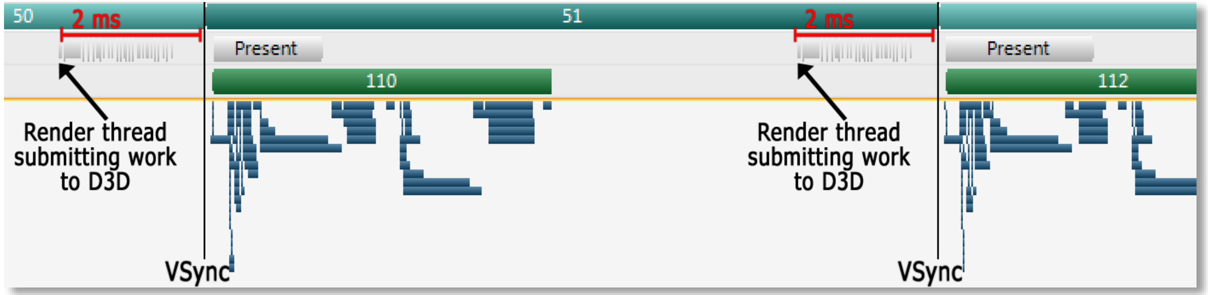
“Running Start”

VALVE

- If you start to submit D3D calls after VSync:



- Instead, start submitting D3D calls 2 ms before VSync. (2 ms is a magic number based on the 1.5-2.0ms GPU bubbles we measured on current GPUs):



- But, you end up predicting another 2 ms (24.22 ms total)

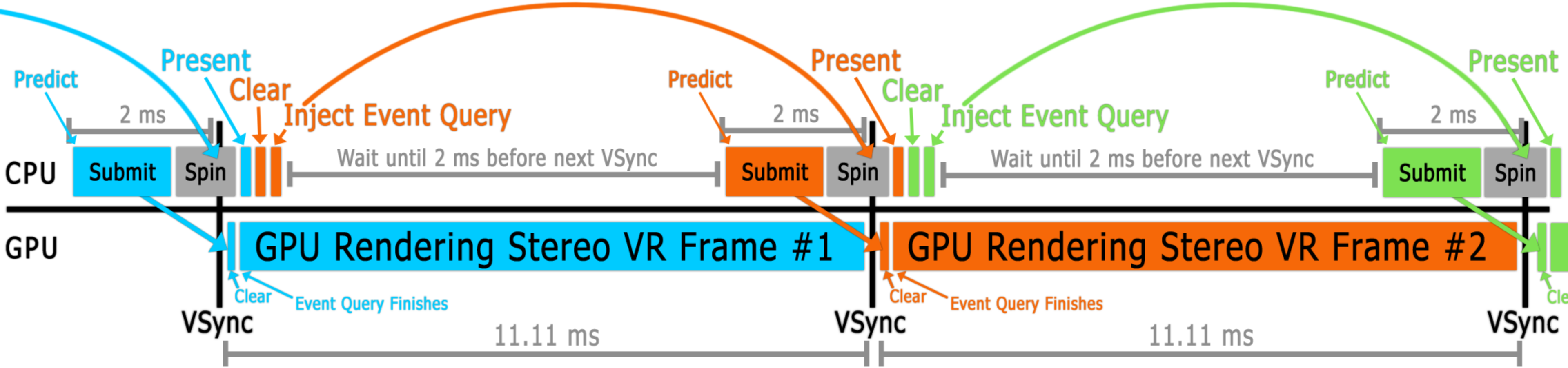
"Running Start" VSync

VALVE

- Question: How do you know how far you are from VSync?
- Answer: It's tricky. Rendering APIs don't directly provide this.
- The SteamVR/OpenVR API on Windows in a separate process spins on calls to IDXGIOutput::WaitForVBlank() and notes the time and increments a frame counter. The application can then call GetTimeSinceLastVSync() that also returns a frame ID.
- GPU vendors, HMD devices, and rendering APIs should provide this

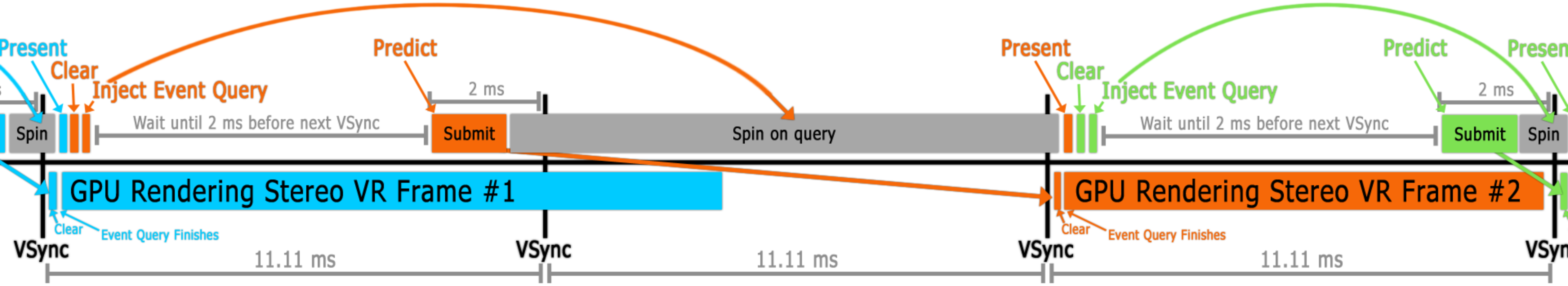
"Running Start" Details

- To deal with a bad frame, you need to partially synchronize with the GPU
- We inject a query after clearing the back buffer, submit our entire frame, spin on that query, then call Present()
- This ensures we are on the correct side of VSync for the current frame, and we can now spin until our running start time



Why the Query Is Critical

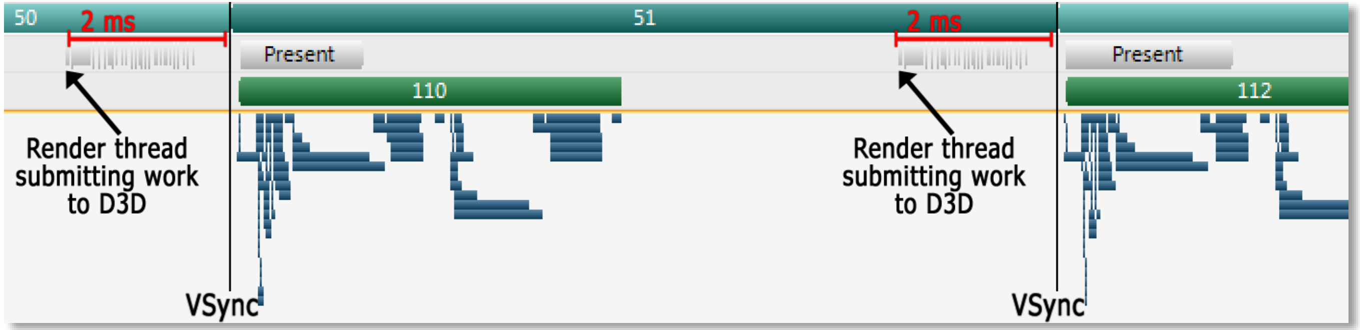
- If a frame is late, the query will keep you on the right side of VSync for the following frame ensuring your prediction remains accurate



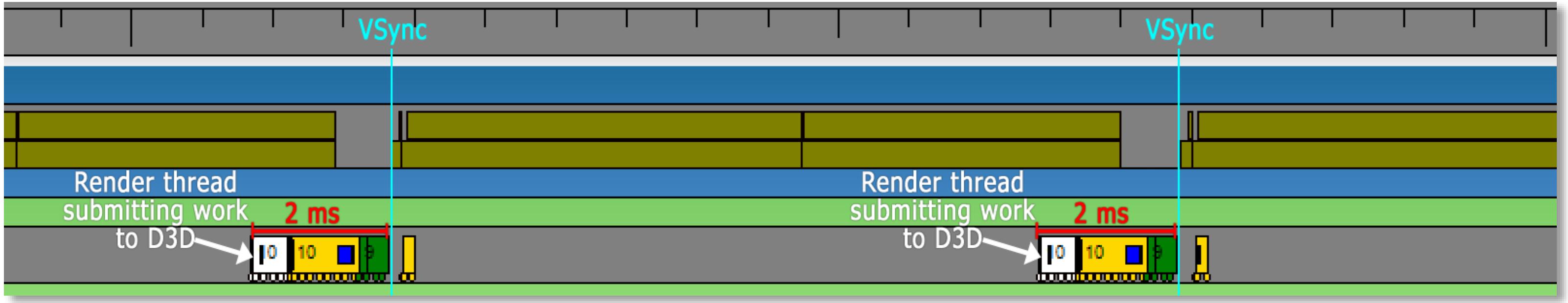
Running Start Summary

VALVE

- **This is a solid 1.5-2.0ms GPU perf gain!**
- You want to see this in NVIDIA Nsight:



- You want to see this in Microsoft's GPUView:



Outline

- VR at Valve
- Methods for Stereo Rendering
- Timing: Scheduling, Prediction, VSync, GPU Bubbles
- **Specular Aliasing & Anisotropic Lighting**
- Miscellaneous VR Rendering Topics

Aliasing Is Your Enemy

VALVE

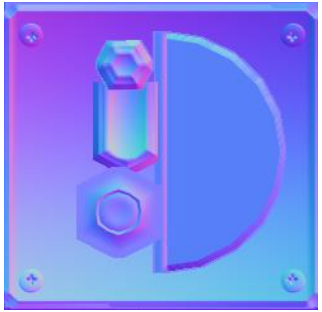
- The camera (your head) never stops moving. Aliasing is amplified because of this.
- While there are more pixels to render, each pixel fills a larger angle than anything we've done before. Here are some averages:
  - 2560x1600 30" monitor: ~**50** pixels/degree (50 degree H fov)
  - 720p 30" monitor: ~**25** pixels/degree (50 degree H fov)
  - VR: ~**15.3** pixels/degree (110 degree fov w/ 1.4x)
- We must increase the quality of our pixels

4xMSAA Minimum Quality

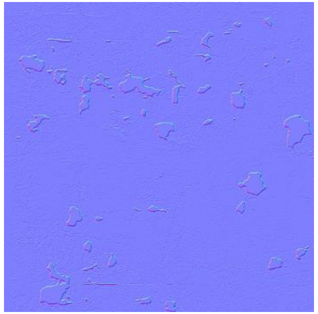
VALVE

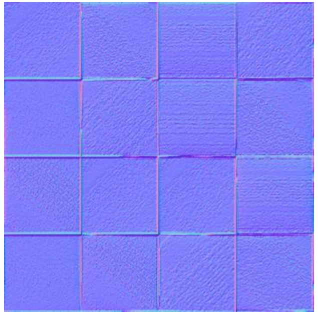
- Forward renderers win for antialiasing because MSAA just works
- We use 8xMSAA if perf allows
- Image-space antialiasing algorithms must be compared side-by-side with 4xMSAA and 8xMSAA to know how your renderer will compare to others in the industry
- Jittered SSAA is obviously the best using the HLSL 'sample' modifier, but only if you can spare the perf

Normal Maps Are Not Dead

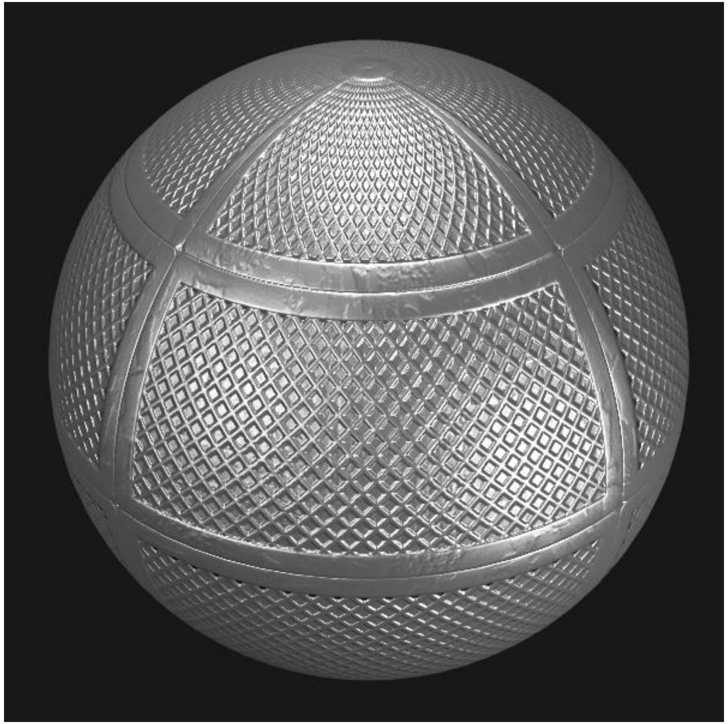


- Most normal maps work great in VR...mostly.
- What doesn't work:
  - Feature detail larger than a few cm inside tracked volume is bad
  - Surface shape inside a tracked volume can't be in a normal map
- What does work:
  - Distant objects outside the tracked volume you can't inspect up close
  - Surface "texture" and fine details:



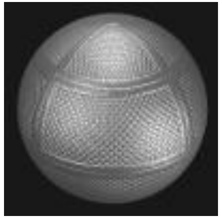


Normal Map Mipping Error



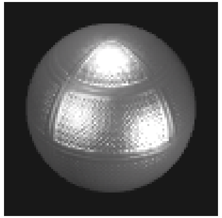
Blinn-Phong Specular

Expected glossiness



Zoomed out
super-sampled
36 samples

Incorrect glossiness

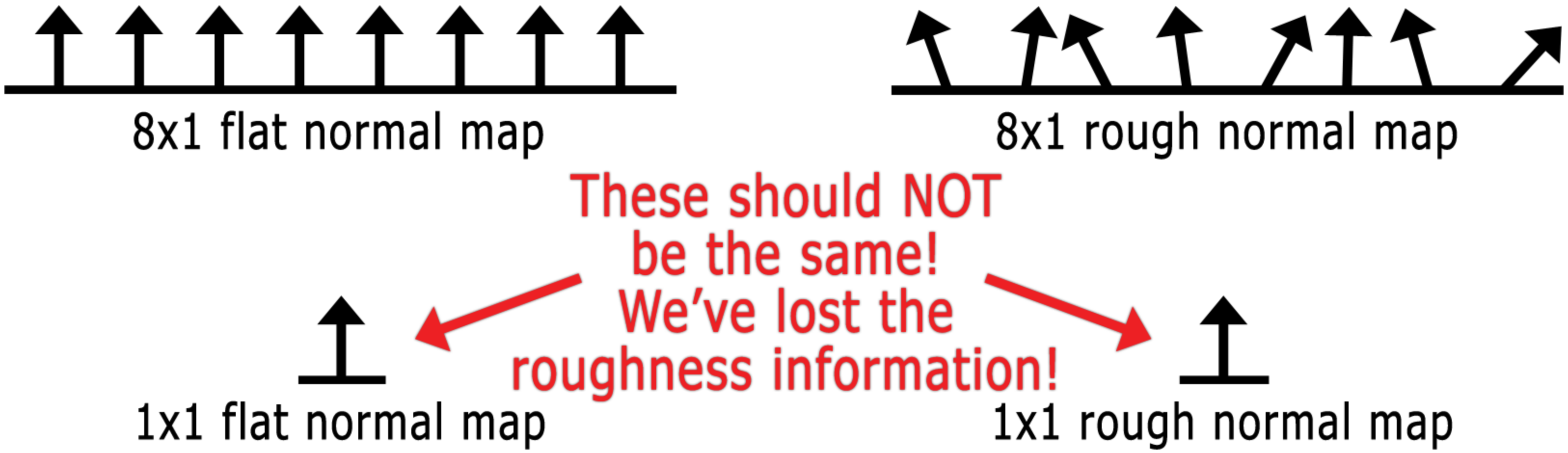


Zoomed out
normal map
box filtered mips

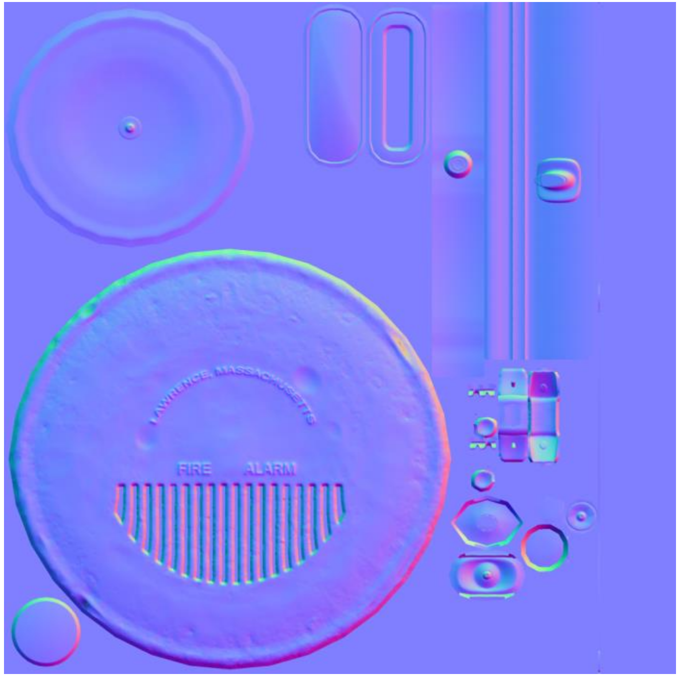
Normal Map Mipping Problems

VALVE®

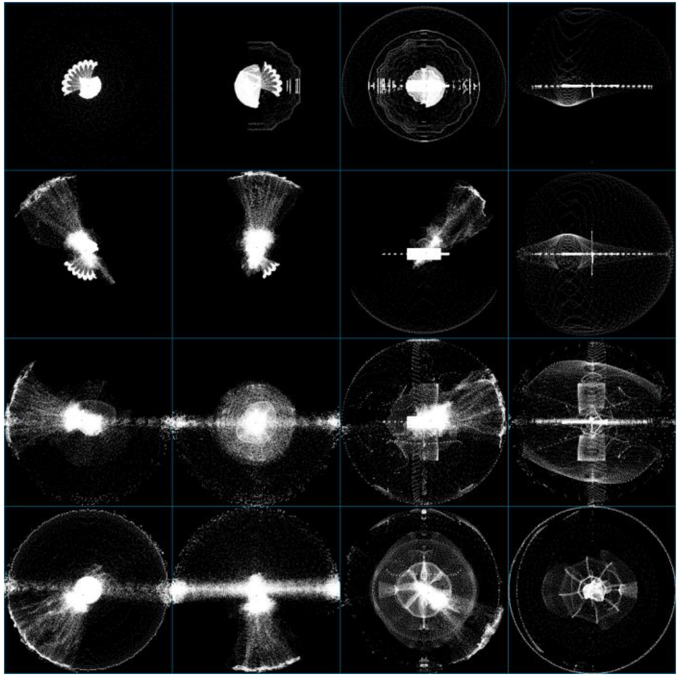
- Any mip filter that just generates an averaged normal loses important roughness information



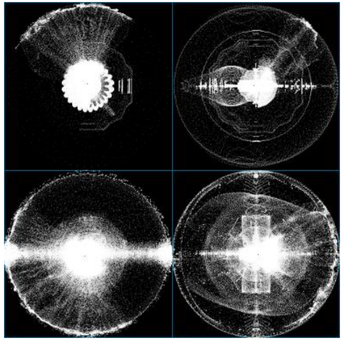
Normal Map Visualization



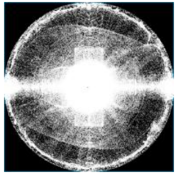
4096x4096 Normal Map
Fire Alarm



4x4 Mip Visualization



2x2 Mip

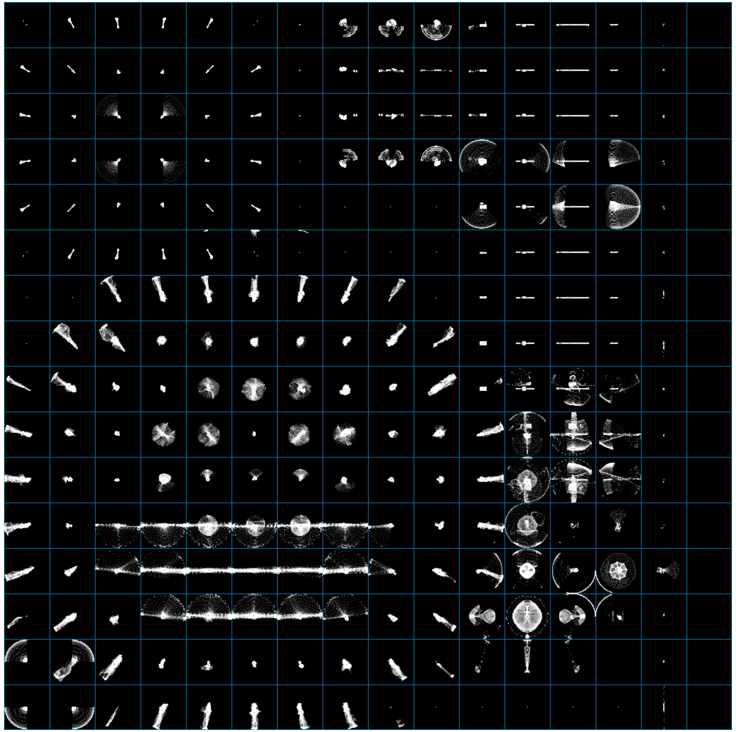


1x1 Mip

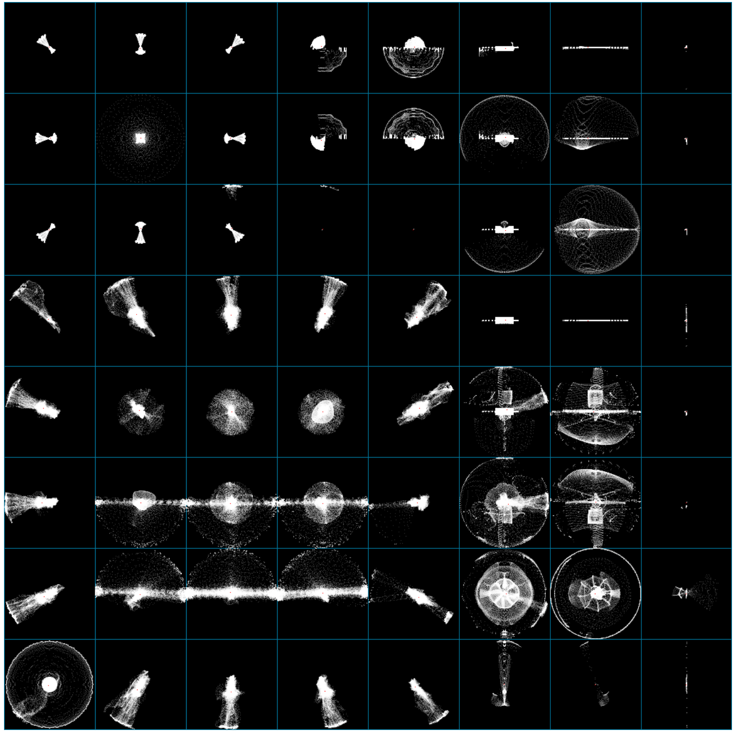
Normal Map Visualization

VALVE

4096x4096 Normal Map
Fire Alarm

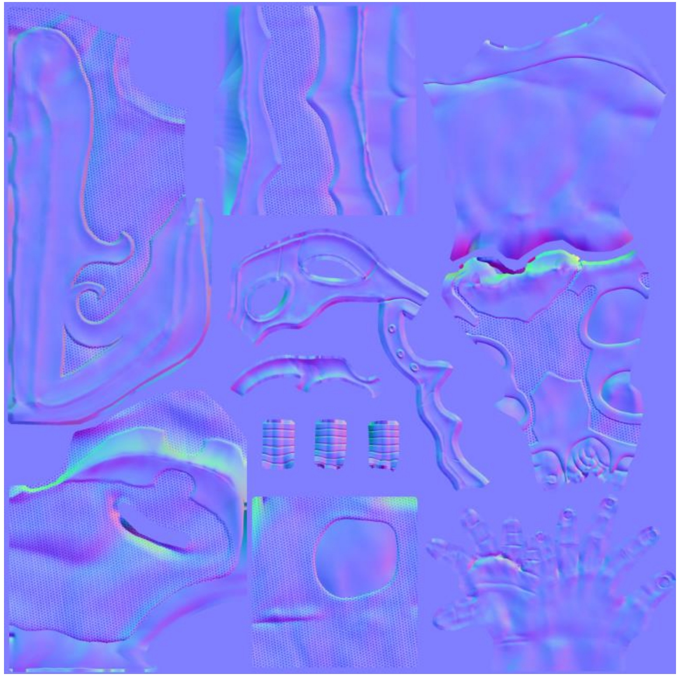


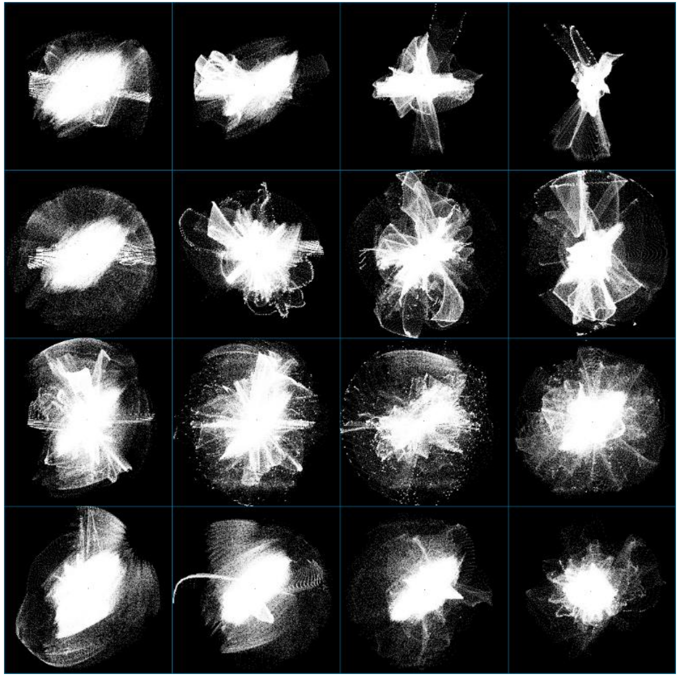
16x16 Mip Visualization

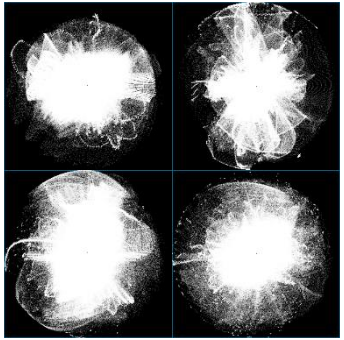


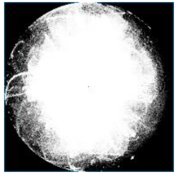
8x8 Mip Visualization

Normal Map Visualization

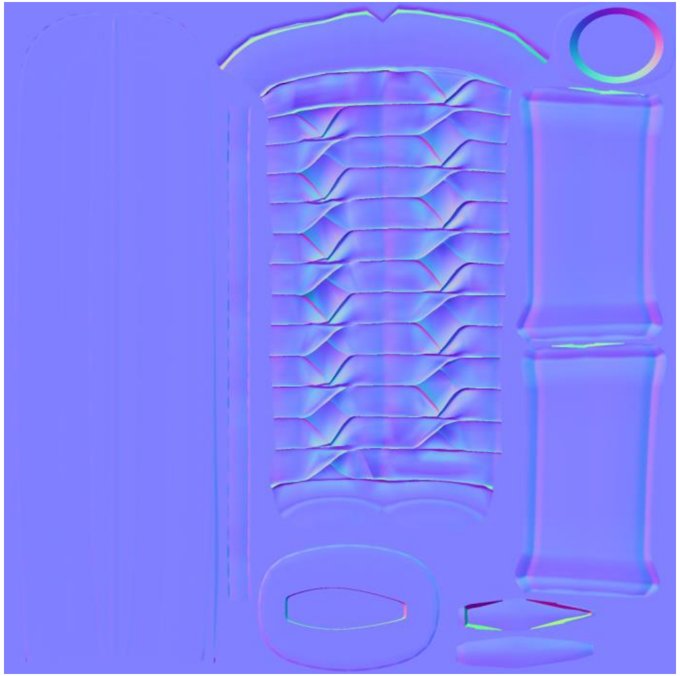

4096x4096 Normal Map
Dota 2 Mirana Body


4x4 Mip Visualization

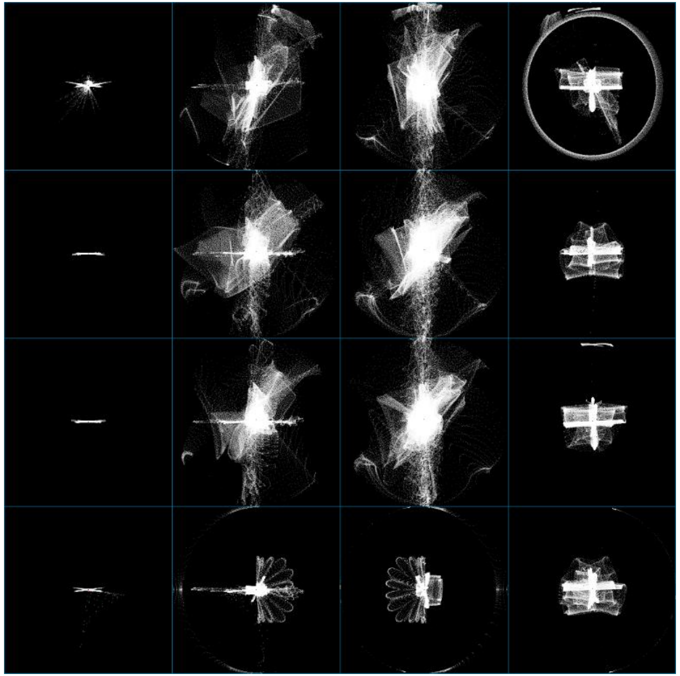

2x2 Mip


1x1 Mip

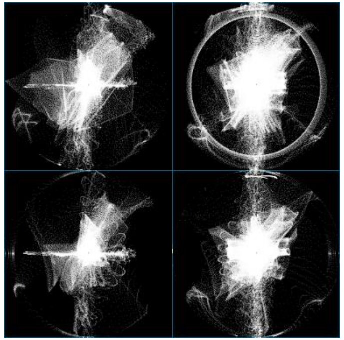
Normal Map Visualization



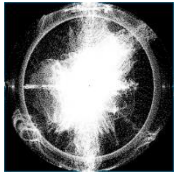
4096x4096 Normal Map
Dota 2 Juggernaut Sword Handle



4x4 Mip Visualization

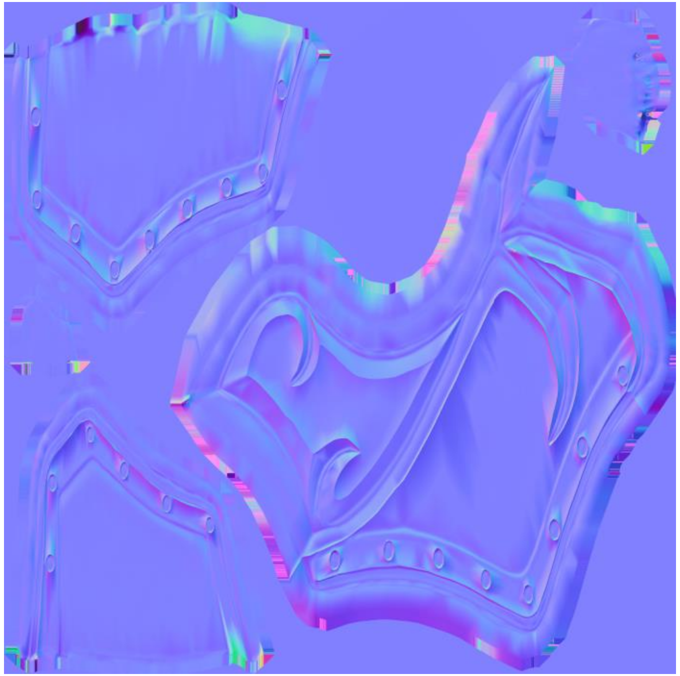


2x2 Mip

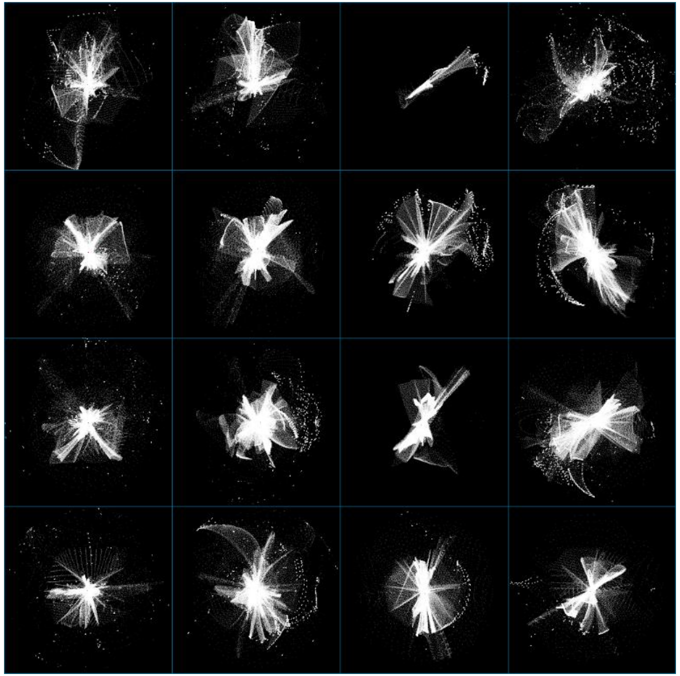


1x1 Mip

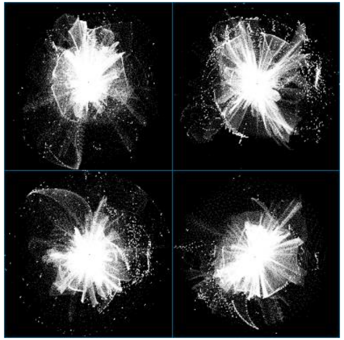
Normal Map Visualization



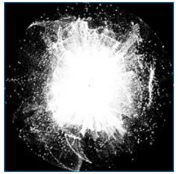
4096x4096 Normal Map
Shoulder Armor



4x4 Mip Visualization

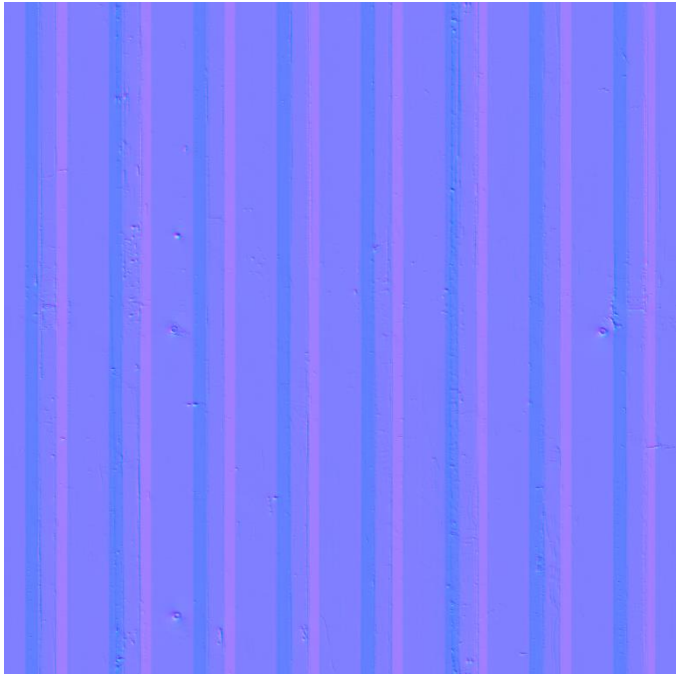


2x2 Mip

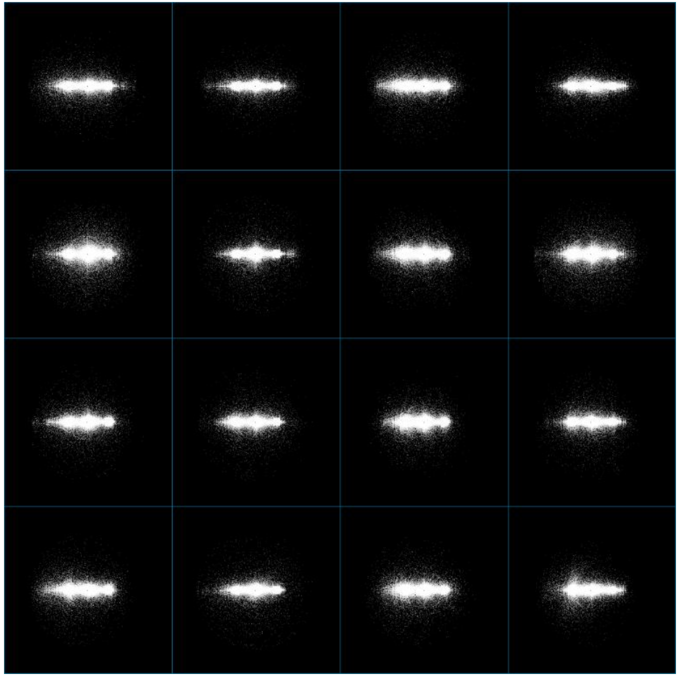


1x1 Mip

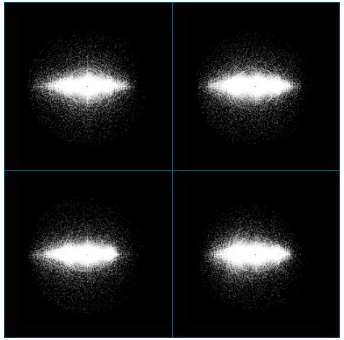
Normal Map Visualization



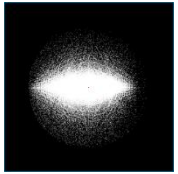
4096x4096 Normal Map
Metal Siding



4x4 Mip Visualization



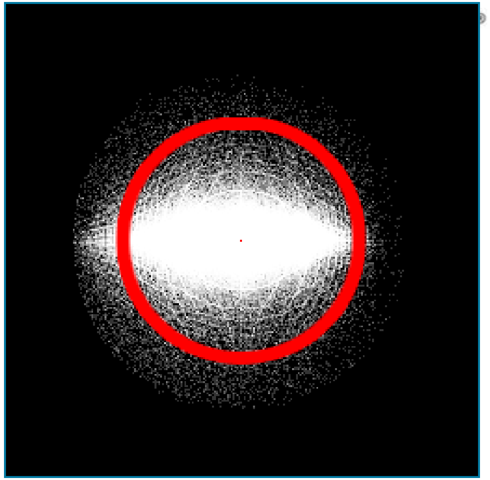
2x2 Mip



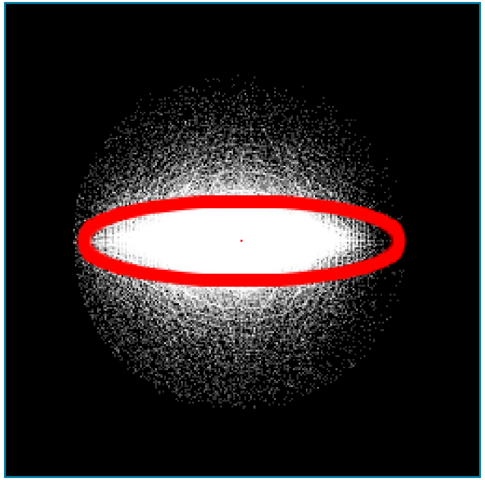
1x1 Mip

Roughness Encoded in Mips

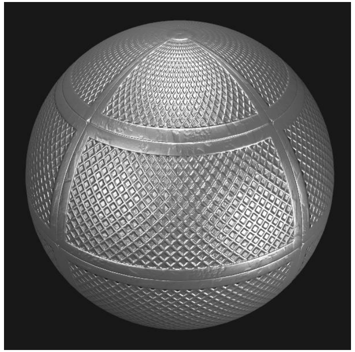
- We can store a single isotropic value (visualized as the radius of a circle) that is the standard deviation of all 2D tangent normals from the highest mip that contributed to this texel



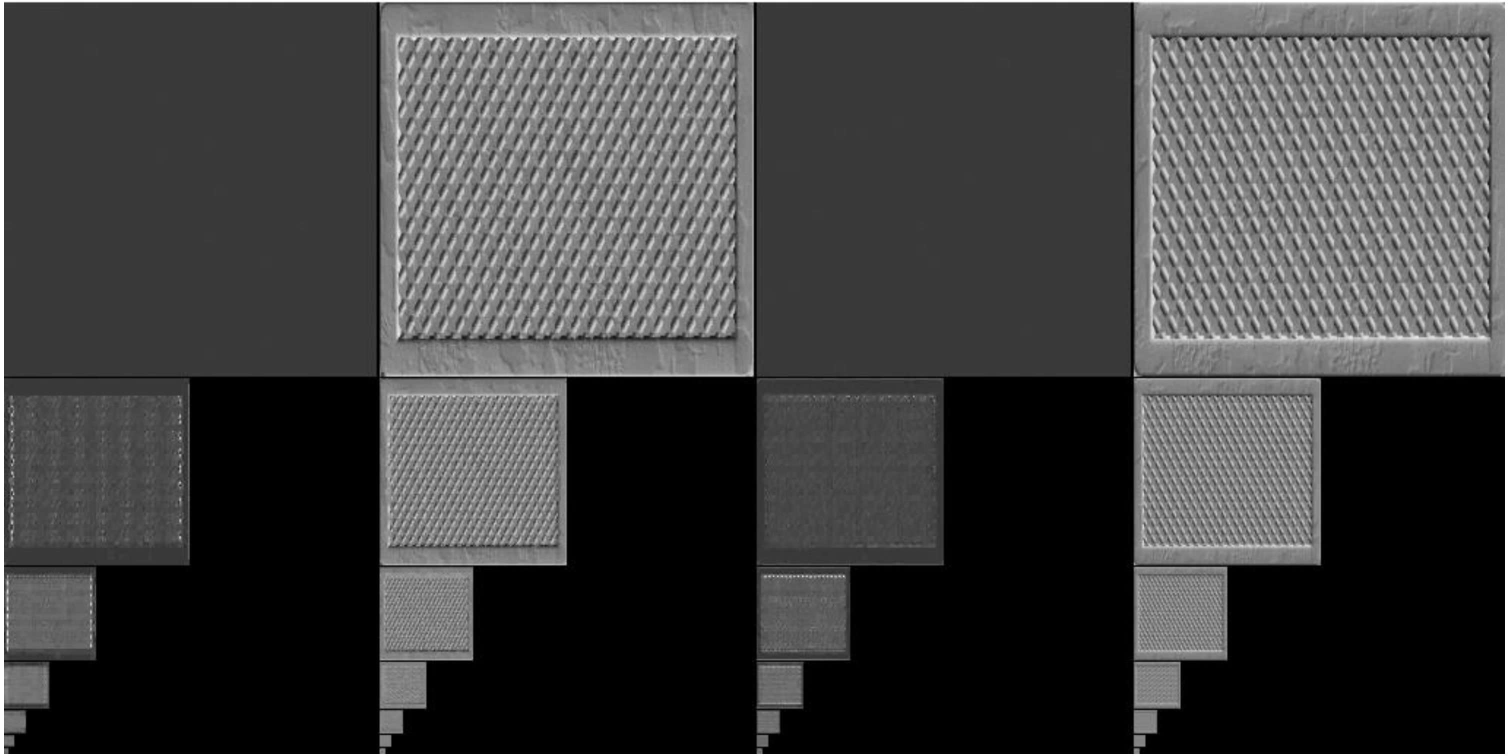
- We can also store a 2D anisotropic value (visualized as the dimensions of an ellipse) for the standard deviation in X and Y separately that can be used to compute **tangent-space axis-aligned anisotropic lighting**!



Final Mip Chain



R = Roughness X G = Normal Y B = Roughness Y A = Normal X



Add Artist-Authored Roughness

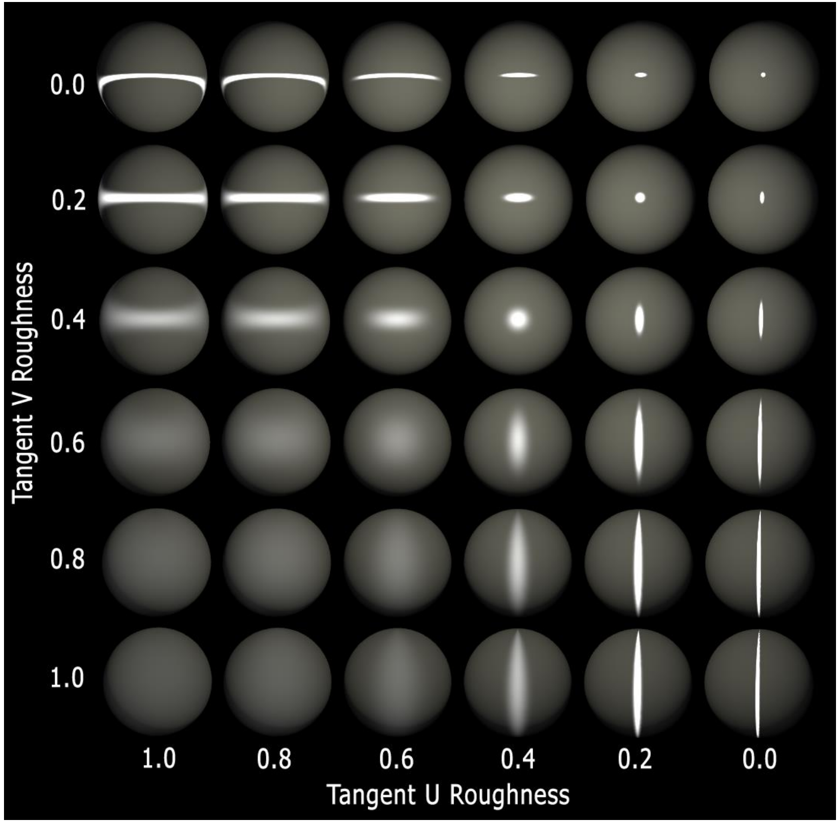
VALVE®

- We author 2D gloss = 1.0 – roughness
- Mip with a simple box filter
- Add/sum it with the normal map roughness at each mip level
- Because we have anisotropic gloss maps anyway, storing the generated normal map roughness is FREE

Isotropic Gloss

Anisotropic Gloss

Tangent-Space Axis-Aligned Anisotropic Lighting

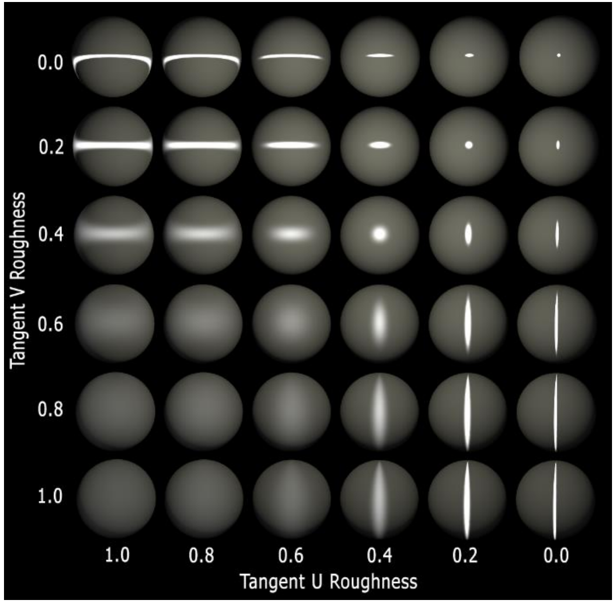


- Standard isotropic lighting is represented along the diagonal
- Anisotropy is aligned with either of the tangent-space axes
- Requires only 2 additional values paired with a 2D tangent normal = Fits into an RGBA texture (DXT5 >95% of the time)

Roughness to Exponent Conversion

VALVE

- Diffuse lighting is Lambert raised to exponent ($N.L^k$) where k is in the range 0.6-1.4
- Experimented with anisotropic diffuse lighting, but not worth the instructions
- Specular exponent range is 1-16,384 and is a modified Blinn-Phong with anisotropy (more on this later)

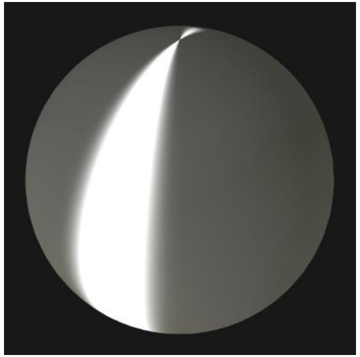



```
void RoughnessEllipseToScaleAndExp( float2 vRoughness,
                                    out float o_flDiffuseExponentOut, out float2 o_vSpecularExponentOut, out float2 o_vSpecularScaleOut )
{
   o_flDiffuseExponentOut = ( ( 1.0 - ( vRoughness.x + vRoughness.y ) * 0.5 ) * 0.8 ) + 0.6; // Outputs 0.6-1.4
   o_vSpecularExponentOut.xy = exp2( pow( 1.0 - vRoughness.xy, 1.5 ) * 14.0 ); // Outputs 1-16384
   o_vSpecularScaleOut.xy = 1.0 - saturate( vRoughness.xy * 0.5 ); // This is a pseudo energy conserving scalar for the roughness exponent
}
```

How Anisotropy Is Computed

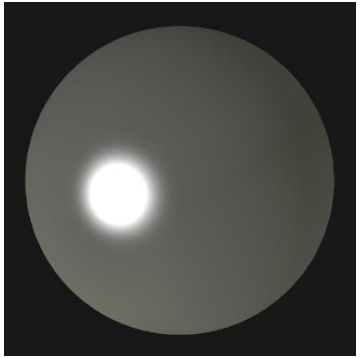
VALVE

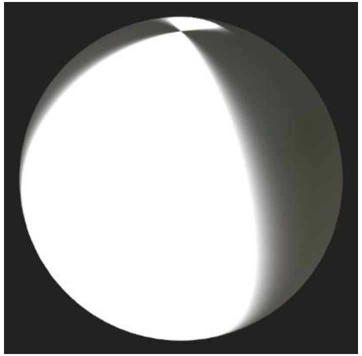
Tangent U Lighting | Tangent V Lighting | Final Lighting



*

=





*

=

Shader Code

VALVE®

```
void RoughnessEllipseToScaleAndExp( float2 vRoughness,
                                    out float o_flDiffuseExponentOut, out float2 o_vSpecularExponentOut, out float2 o_vSpecularScaleOut )
{
   o_flDiffuseExponentOut = ( ( 1.0 - ( vRoughness.x + vRoughness.y ) * 0.5 ) * 0.8 ) + 0.6; // Outputs 0.6-1.4
   o_vSpecularExponentOut.xy = exp2( pow( 1.0 - vRoughness.xy, 1.5 ) * 14.0 ); // Outputs 1-16384
   o_vSpecularScaleOut.xy = 1.0 - saturate( vRoughness.xy * 0.5 ); // This is a pseudo energy conserving scalar for the roughness exponent
}
```

Isotropic Diffuse Lighting:

```
float flDiffuseTerm = pow( flNDotL, flDiffuseExponent ) * ( ( flDiffuseExponent + 1.0 ) * 0.5 );
```

Anisotropic Specular Lighting:

```
float3 vHalfAngleDirWs = normalize( vPositionToLightDirWs.xyz + vPositionToCameraDirWs.xyz );

float3 vSpecularNormalX = vHalfAngleDirWs.xyz - ( vTangentUWs.xyz * dot( vHalfAngleDirWs.xyz, vTangentUWs.xyz ) );
float3 vSpecularNormalY = vHalfAngleDirWs.xyz - ( vTangentVWs.xyz * dot( vHalfAngleDirWs.xyz, vTangentVWs.xyz ) );

float flNDotHX = max( 0.0, dot( vSpecularNormalX.xyz, vHalfAngleDirWs.xyz ) );
float flNDotHkX = pow( flNDotHX, vSpecularExponent.x * 0.5 );
flNDotHkX *= vSpecularScale.x;

float flNDotHY = max( 0.0, dot( vSpecularNormalY.xyz, vHalfAngleDirWs.xyz ) );
float flNDotHkY = pow( flNDotHY, vSpecularExponent.y * 0.5 );
flNDotHkY *= vSpecularScale.y;

float flSpecularTerm = flNDotHkX * flNDotHkY;
```

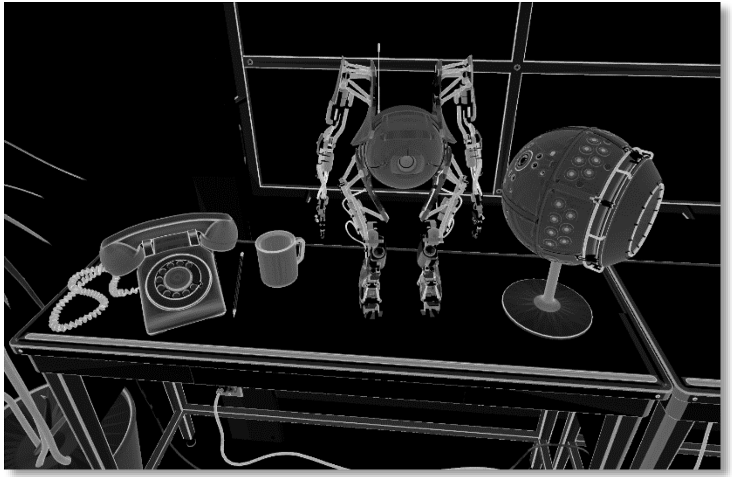
Isotropic Specular Lighting:

```
float flNDotH = saturate( dot( vNormalWs.xyz, vHalfAngleDirWs.xyz ) );
float flNDotHk = pow( flNDotH, dot( vSpecularExponent.xy, float2( 0.5, 0.5 ) ) );
flNDotHk *= dot( vSpecularScale.xy, float2( 0.33333, 0.33333 ) ); // 0.33333 is to match the spec intensity of the aniso algorithm above
float flSpecularTerm = flNDotHk;
```

Geometric Specular Aliasing

- Dense meshes without normal maps also alias, and roughness mips can't help you!
- We use partial derivatives of interpolated vertex normals to generate a geometric roughness term that approximates curvature. Here is the hacky math:

```
float3 vNormalWsDdx = ddx( vGeometricNormalWs.xyz );
float3 vNormalWsDdy = ddy( vGeometricNormalWs.xyz );
float flGeometricRoughnessFactor = pow( saturate( max( dot( vNormalWsDdx.xyz, vNormalWsDdx.xyz ), dot( vNormalWsDdy.xyz, vNormalWsDdy.xyz ) ) ), 0.333 );
vRoughness.xy = max( vRoughness.xy, flGeometricRoughnessFactor.xx ); // Ensure we don't double-count roughness if normal map encodes geometric roughness
```



Visualization of flGeometricRoughnessFactor

Geometric Specular Aliasing Part 2

VALVE®

- MSAA center vs centroid interpolation: It's not perfect
- Normal interpolation can cause specular sparkling at silhouettes due to over-interpolated vertex normals
- Here's a trick we are using:
  - Interpolate normal twice: once with centroid, once without

```
float3 vNormalWs : TEXCOORD0;
centroid float3 vCentroidNormalWs : TEXCOORD1;
```

  - In the pixel shader, choose the centroid normal if normal length squared is greater than 1.01

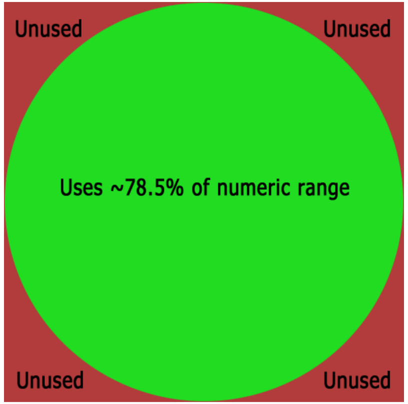
```
if ( dot( i.vNormalWs.xyz, i.vNormalWs.xyz ) >= 1.01 )
{
    i.vNormalWs.xyz = i.vCentroidNormalWs.xyz;
}
```

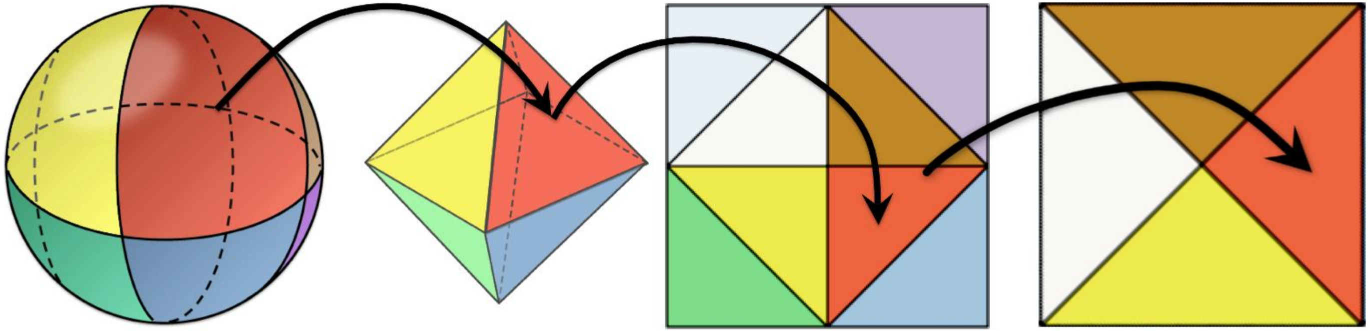
Outline

- VR at Valve
- Methods for Stereo Rendering
- Timing: Scheduling, Prediction, VSync, GPU Bubbles
- Specular Aliasing & Anisotropic Lighting
- **Miscellaneous VR Rendering Topics**

Normal Map Encoding

- Projecting tangent normals onto Z plane only uses 78.5% of the range of a 2D texel
- Hemi-octahedron encoding uses the full range of a 2D texel
  - *"A Survey of Efficient Representations for Independent Unit Vectors"*, Cigolle et al., Journal of Computer Graphics Techniques Vol. 3, No. 2, 2014





(Image modified from above paper)

Scale Render Target Resolution

VALVE®

- Turns out, 1.4x is just a recommendation for the HTC Vive (Each HMD design has a different recommended scalar based on optics and panels)
- On slower GPUs, scale the recommended render target scalar down
- On faster GPUs, scale the recommended render target scalar up
- If you've got GPU cycles to burn, BURN THEM

Anisotropic Texture Filtering

VALVE

- Increases the perceived resolution of the panels (don't forget, we only have fewer pixels per degree)
- Force this on for color and normal maps
  - We use 8x by default
- Disable for everything else. Trilinear only, but measure perf. Anisotropic filtering may be "free" if you are bottlenecked elsewhere.

Noise Is Your Friend

VALVE®

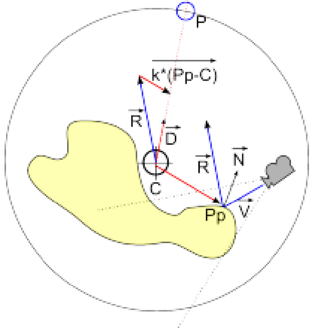
- Gradients are horrible in VR. Banding is more obvious than LCD TVs.
- We add noise on the way into the framebuffer when we have floating-point precision in the pixel shader

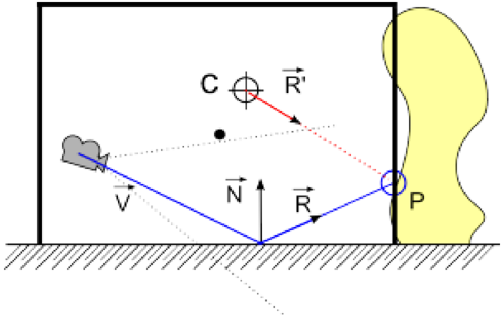
```
float3 ScreenSpaceDither( float2 vScreenPos )
{
    // Iestyn's RGB dither (7 asm instructions) from Portal 2 X360, slightly modified for VR
    float3 vDither = dot( float2( 171.0, 231.0 ), vScreenPos.xy + g_flTime ).xxx;
    vDither.rgb = frac( vDither.rgb / float3( 103.0, 71.0, 97.0 ) ) - float3( 0.5, 0.5, 0.5 );
    return ( vDither.rgb / 255.0 ) * 0.375;
}
```

Environment Maps

VALVE®

- Standard implementation at infinity = only works for sky
- Need to use some type of distance remapping for environment maps
  - Sphere is cheap
  - Box is more expensive
  - Both are useful in different situations
- Read this online article:
  - *"Image-based Lighting approaches and parallax-corrected cubemaps"*, Sébastien Lagarde, 2012





Stencil Mesh (Hidden Area Mesh)

VALVE

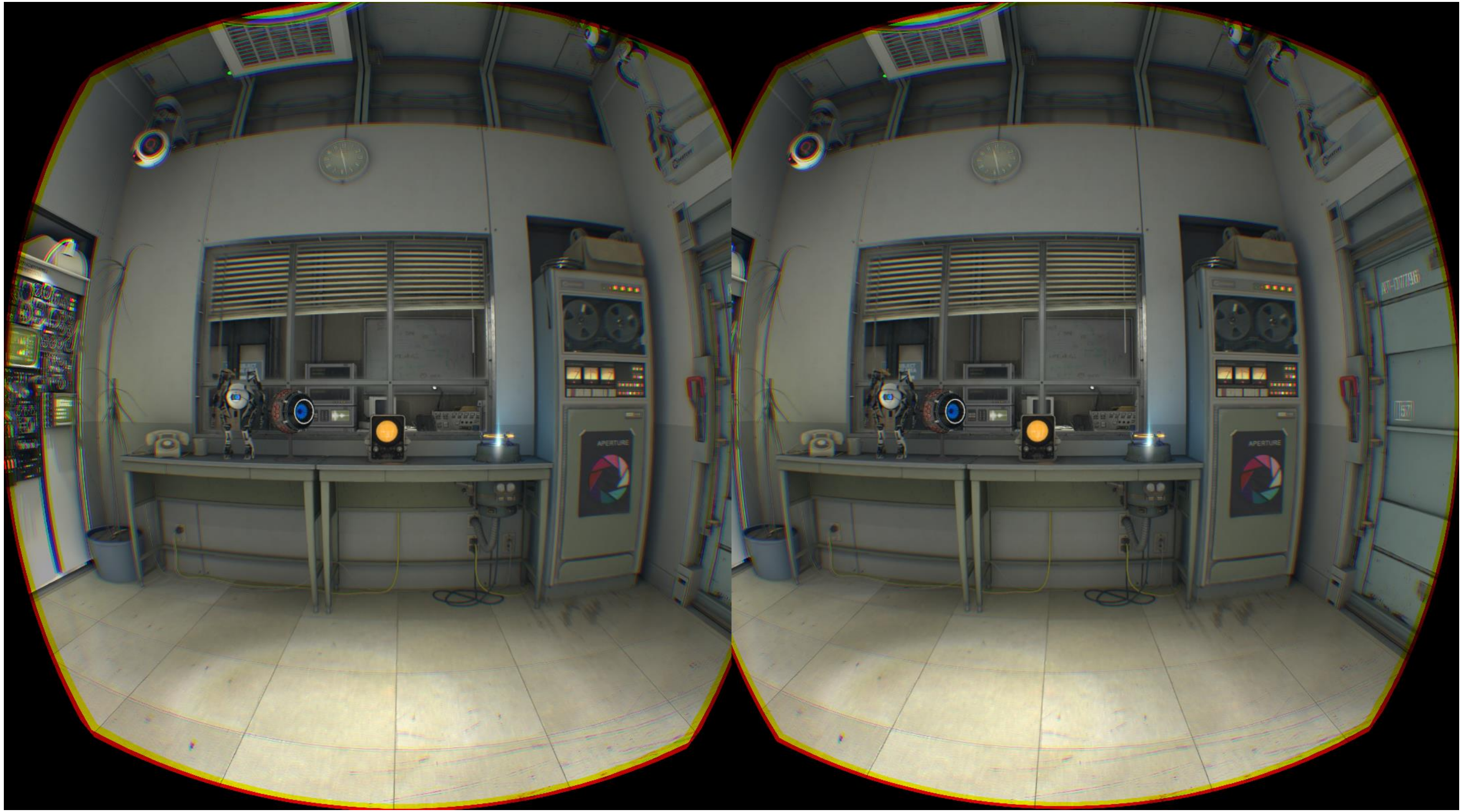
- Stencil out the pixels you can't actually see through the lenses. GPUs are fast at early stencil-rejection.

- Alternatively you can render to the depth buffer at near z so everything early z-rejects instead

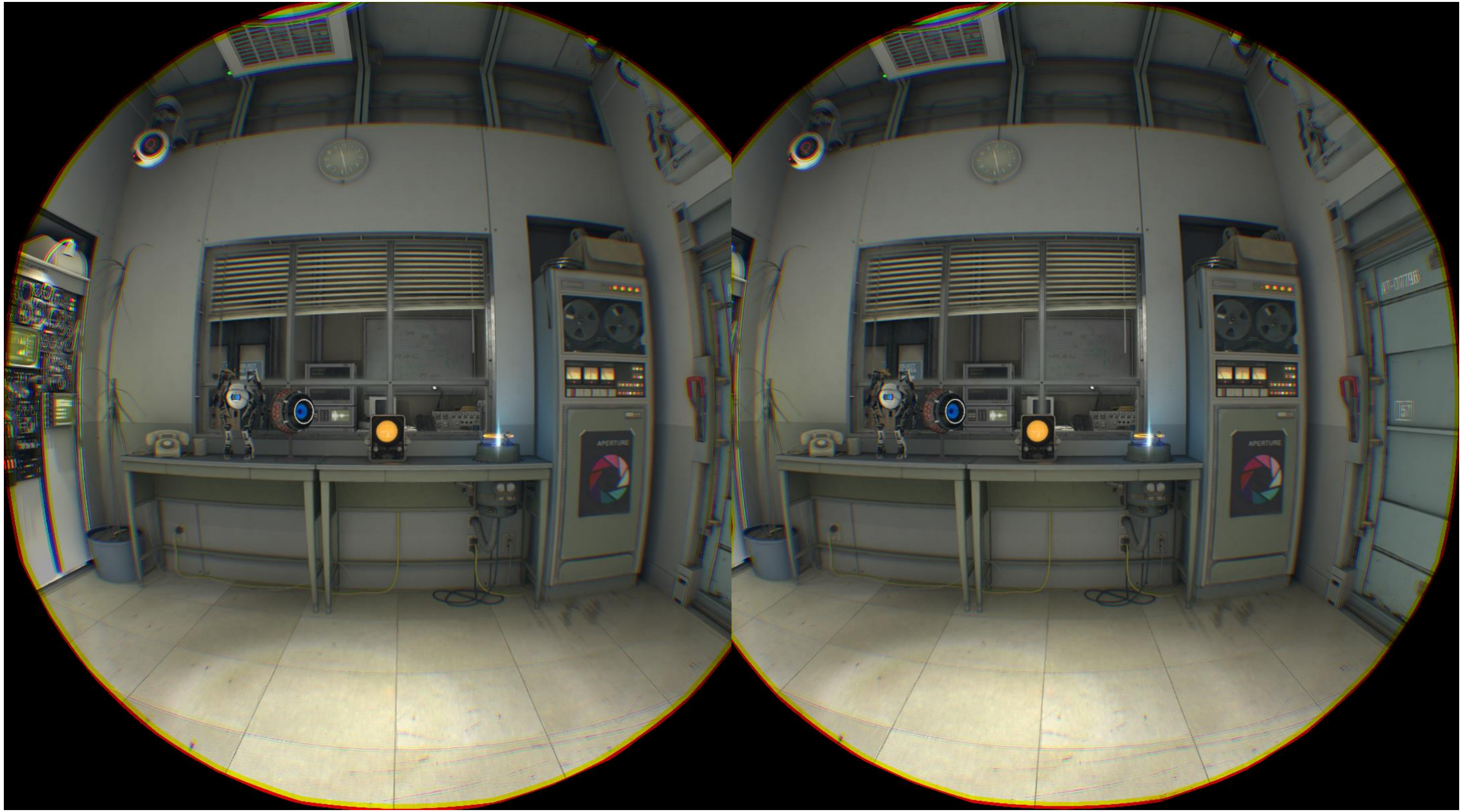
- Lenses produce radially symmetric distortion which means you effectively see a circular area projected on the panels

Stencil Mesh (Warped View)

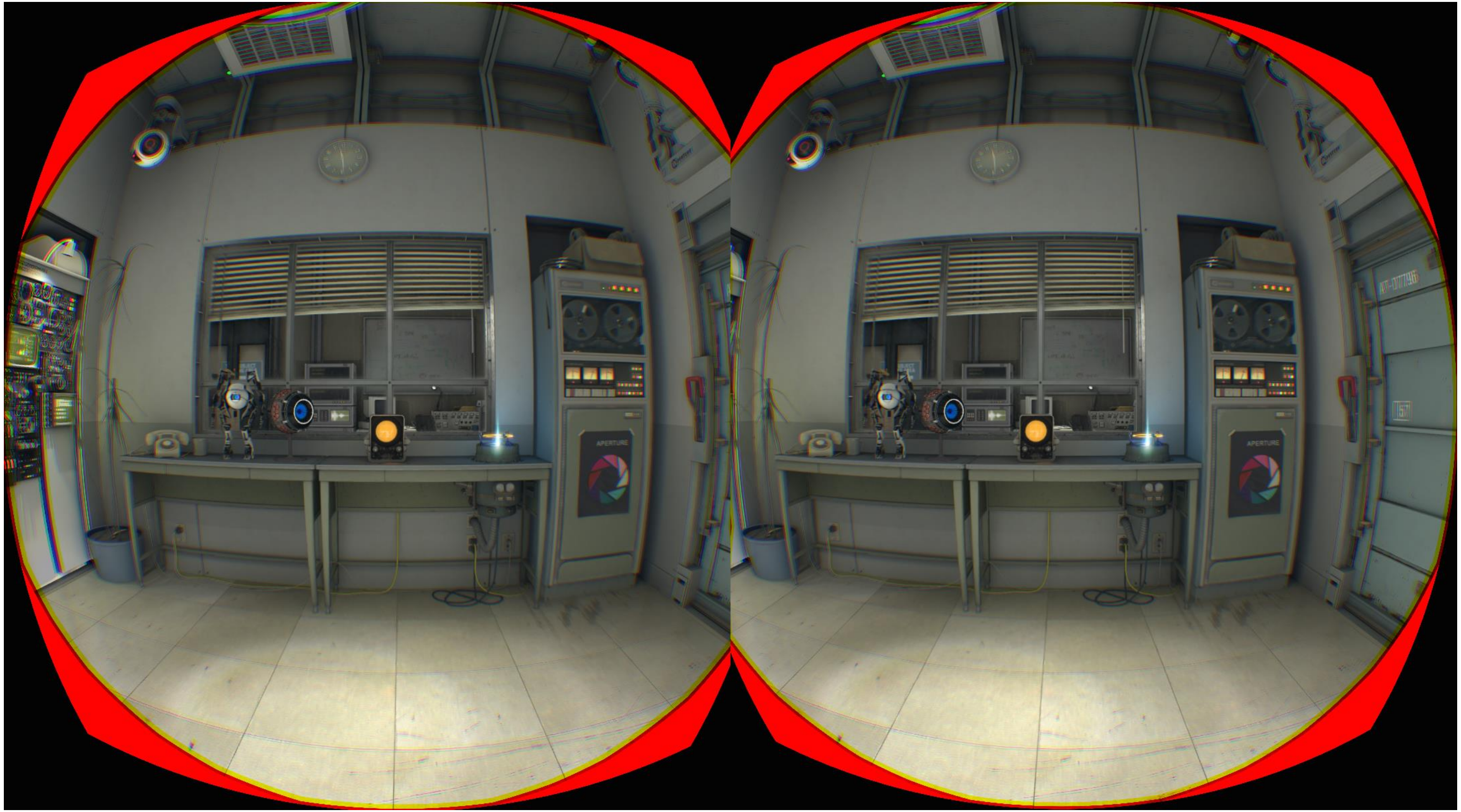
VALVE



Stencil Mesh (Ideal Warped View)



Stencil Mesh (Wasted Pixels)



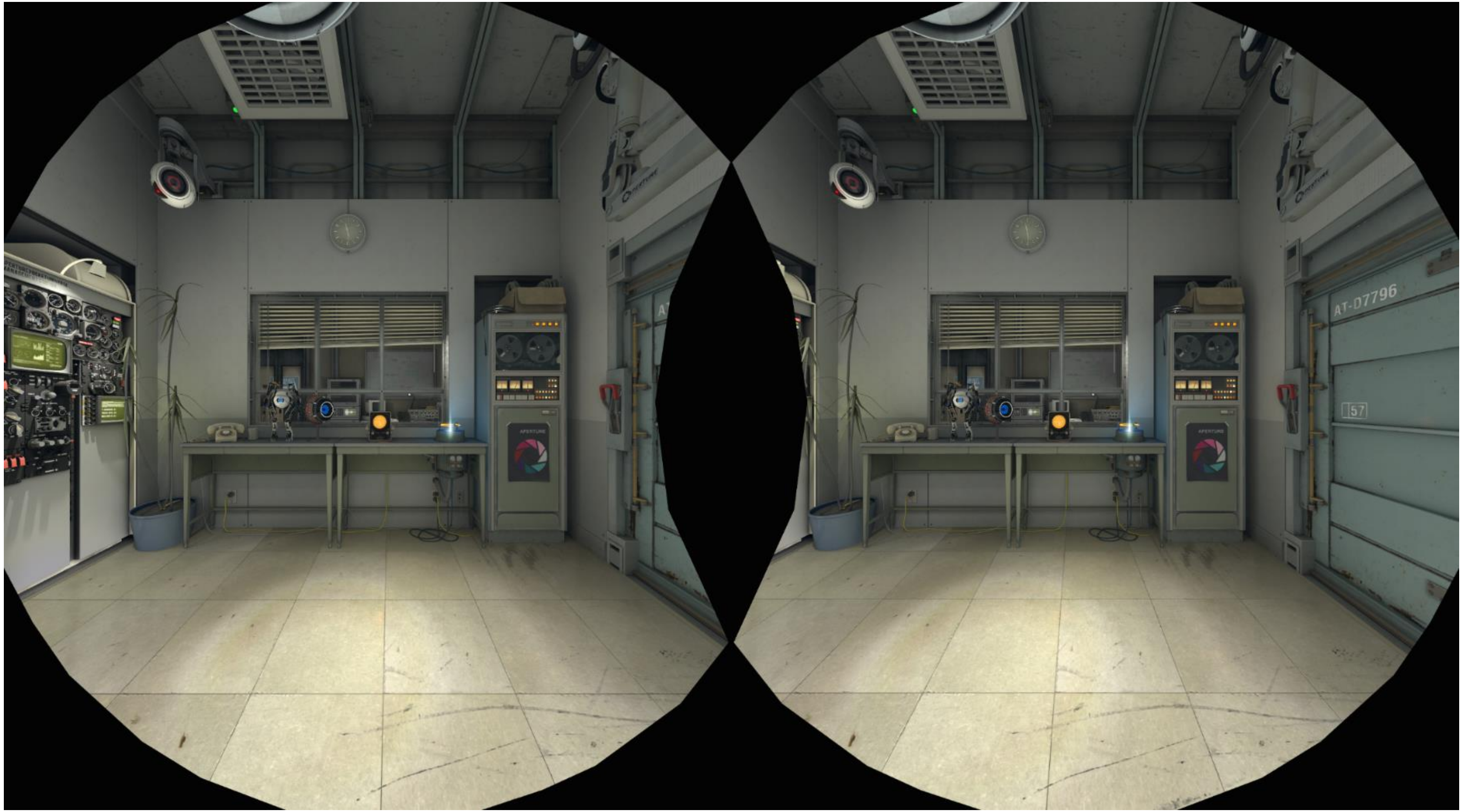
Stencil Mesh (Unwarped View)

VALVE

Stencil Mesh (Unwarped View)

VALVE

Stencil Mesh (Final Unwarped View)



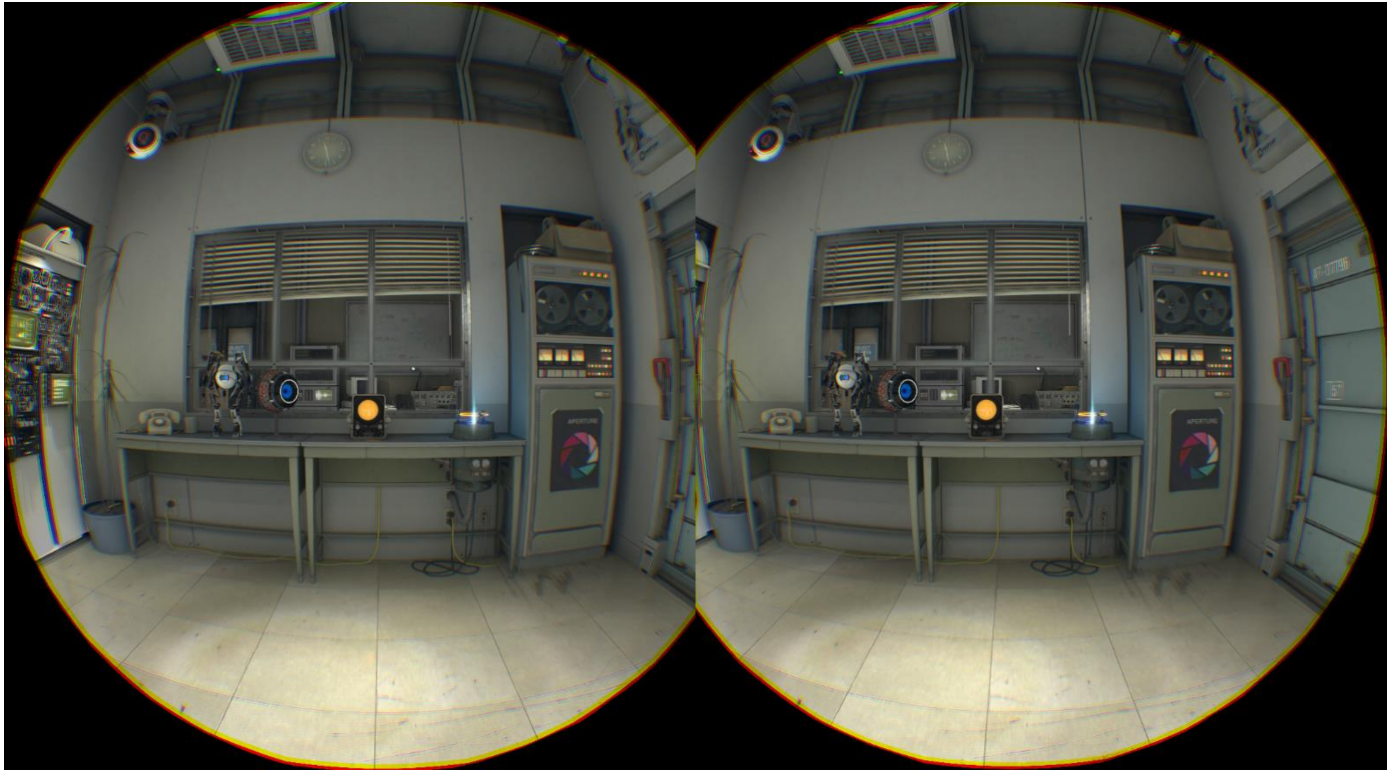
Stencil Mesh (Final Warped View)

VALVE

Stencil Mesh (Hidden Area Mesh)

- SteamVR/OpenVR API will provide this mesh to you
- Results in a 17% fill rate reduction!
- No stencil mesh: VR 1512x1680x2 @ 90Hz: **457** million pixels/sec
  - 2,540,160 pixels per eye (5,080,320 pixels total)
- With stencil mesh: VR 1512x1680x2 @ 90Hz: **378** million pixels/sec
  - **About 2,100,000 pixels per eye (4,200,000 pixels total)**

Warp Mesh (Lens Distortion Mesh)



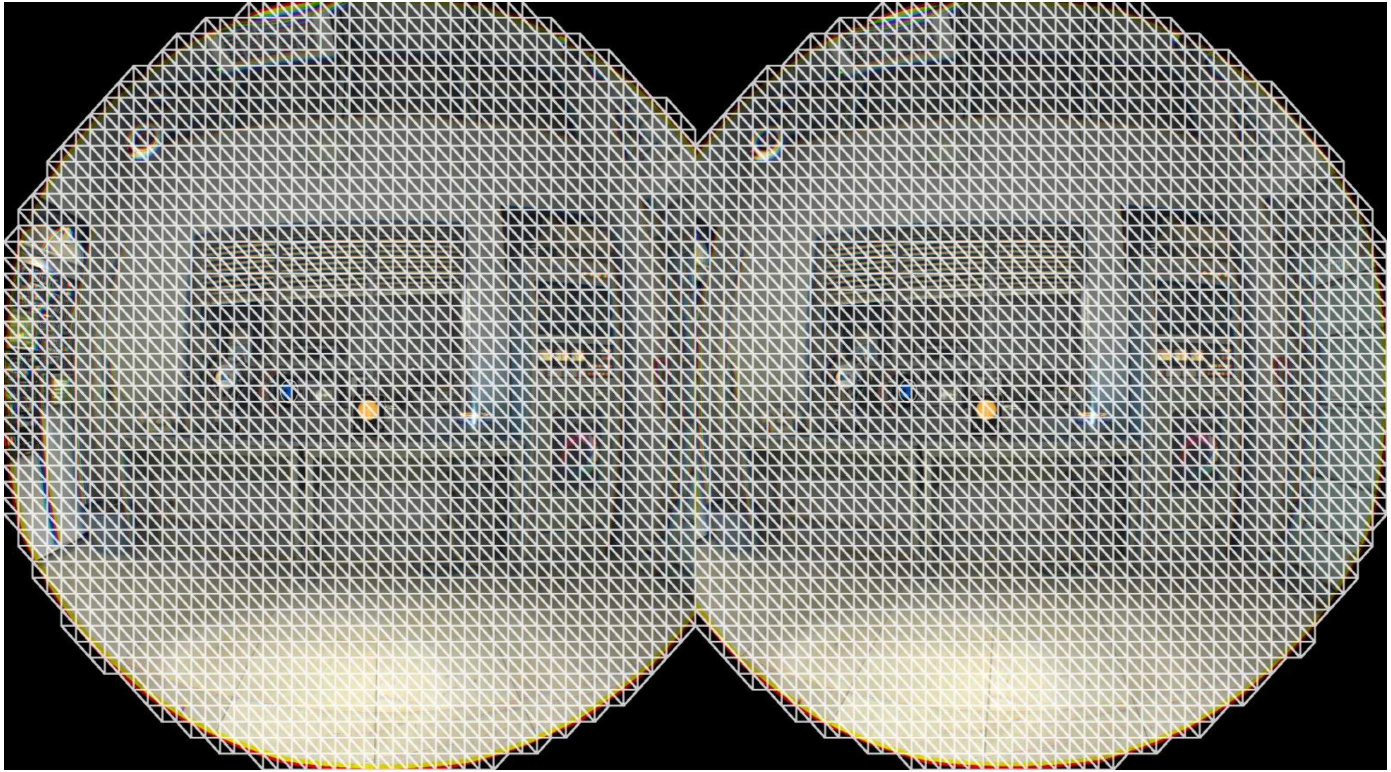
Warp Mesh (Brute-Force)

VALVE

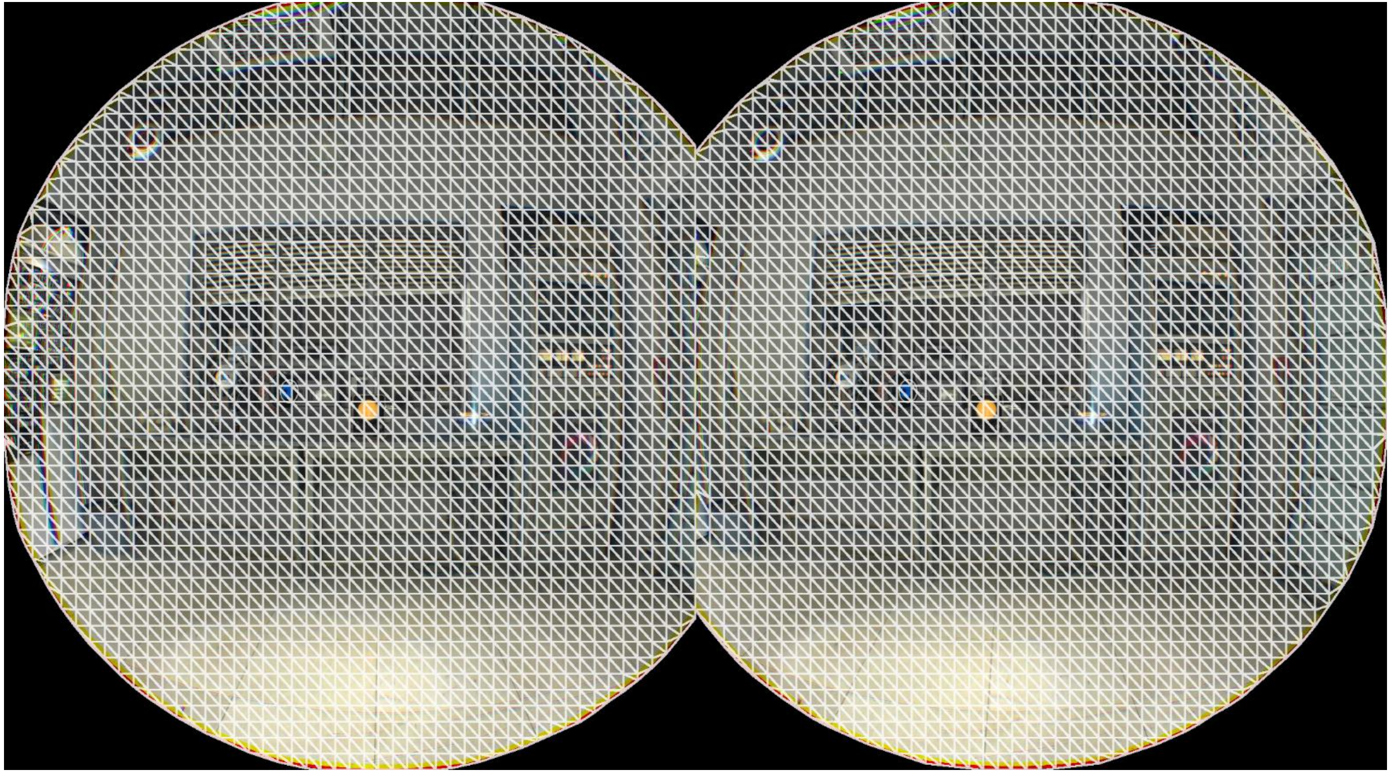
Warp Mesh (Cull UV's Outside 0-1)

VALVE

Warp Mesh (Cull Stencil Mesh)



Warp Mesh (Shrink Wrap)



15% of pixels culled from the warp mesh

Performance Queries Required!

VALVE

- You are always VSync'd
- Disabling VSync to see framerate will make you dizzy
- Need to use performance queries to report GPU workload
- Simplest implementation is to measure first to last draw call
- Ideally measure these things:
  - Idle time from Present() to first draw call
  - First draw call to last draw call
  - Idle time from last draw call to Present()

VALVE®

Summary

- Stereo Rendering
- Prediction
- “Running Start” (Saves 1.5-2.0 ms/frame)
- Anisotropic Lighting & Mipping Normal Maps
- Geometric Specular Antialiasing
- Stencil Mesh (Saves 17% pixels rendered)
- Optimized Warp Mesh (Reduces cost by 15%)
- Etc.

Thank You!

Alex Vlachos, Valve

Alex@ValveSoftware.com