Improved Geometric Specular Antialiasing

YUSUKE TOKUYOSHI (SQUARE ENIX CO., LTD.)

ANTON S. KAPLANYAN (FACEBOOK REALITY LABS)

Aliasing of Specular Highlights

```
CPU   0.272 ms, GPU   2.836 ms, 324 Hz
Engine Tuning
+ Application/...
[X]                 Display Frame Rate
[X]                 Display Profiler
- Experiments/...
   Disabled            SAA
   [-]                 SSAA
+ Graphics/...
[-]                 Load Settings
[-]                 Save Settings
+ Timing/...
```

with a bloom posteffect (1920×1080 pixels)

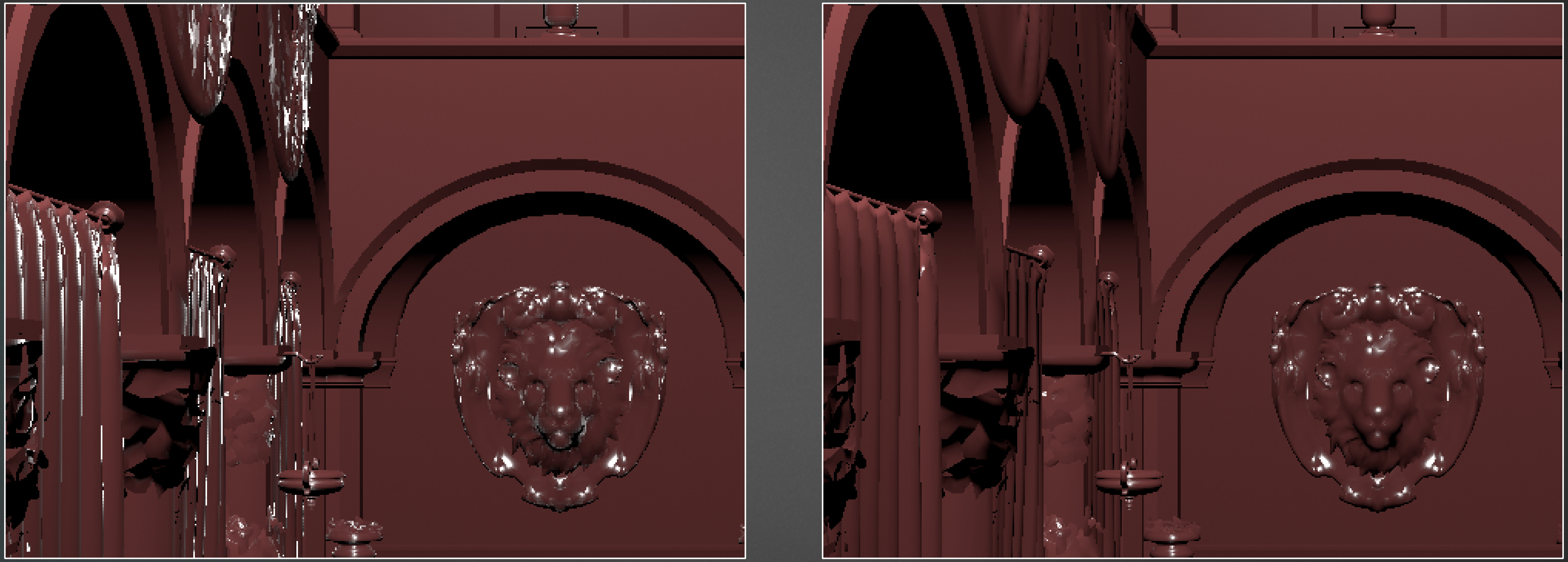
Geometric Specular AA [Kaplanyan16]

- Simple & fast ☺
  - NDF filtering in pixel shader
  - Just increase the roughness parameter of the microfacet BRDF [Cook82]

- Limitations:
  - Suppress only the specular aliasing
  - Require high-quality tangent frames
  - Numerical error for grazing angles

Filtering Error (Non-Axis-Aligned Filtering)

GGX microfacet BRDF (roughness: 0.01) [Walter07]

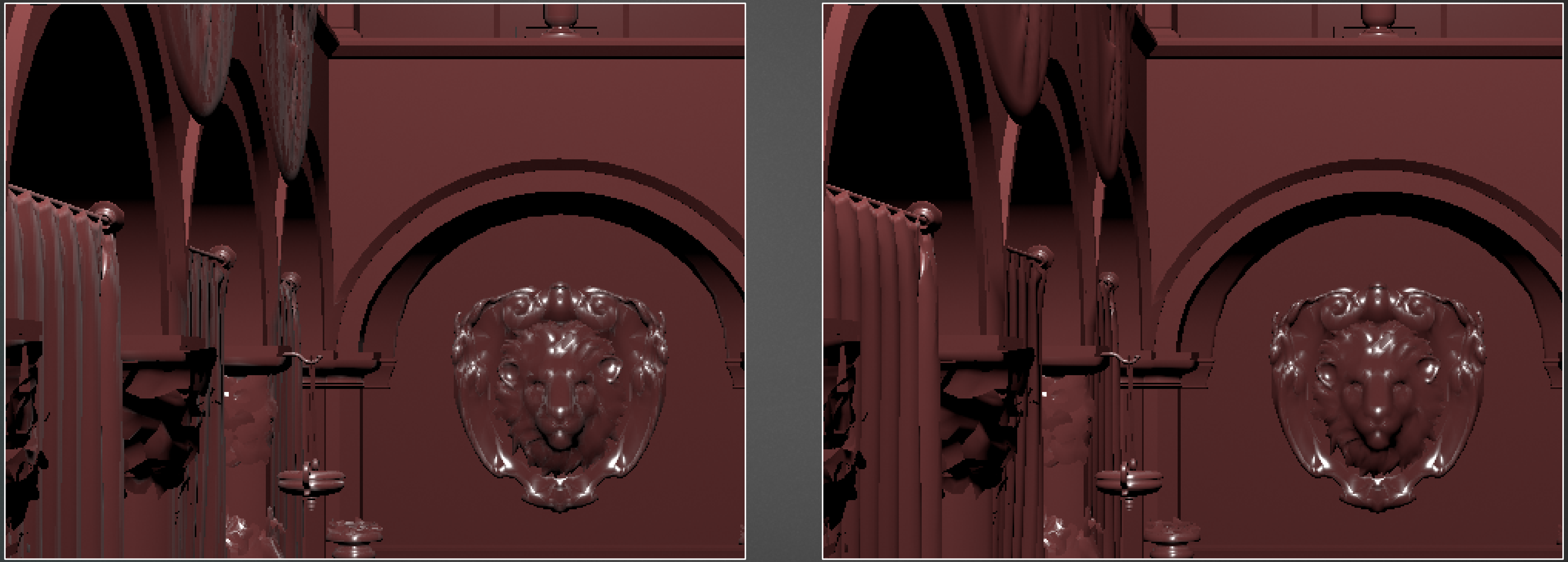


[Kaplanyan16]

with our modification

Filtering Error (Biased Axis-Aligned Filtering)

GGX microfacet BRDF (roughness: 0.01) [Walter07]



[Kaplanyan16]

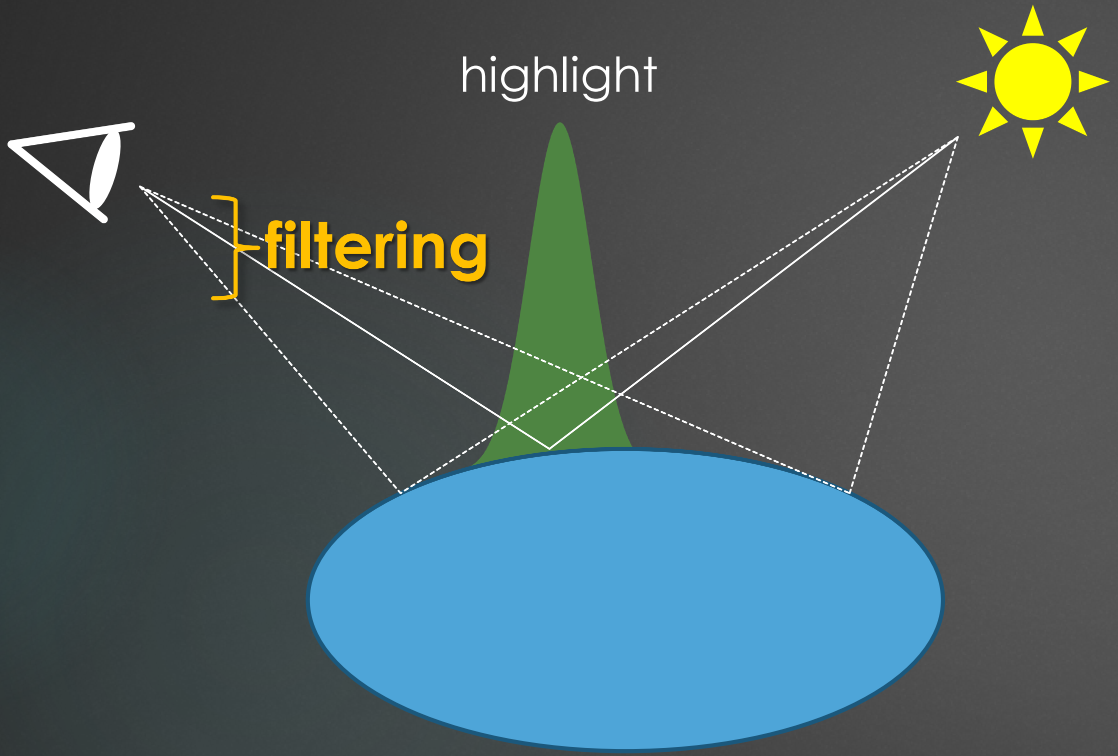
with our modification

Our Contributions

- Error analysis of geometric specular AA
- Efficient filter kernel taking the error into account
  - Simpler than the previous method
- Simplification for deferred rendering
  - 12 lines of code → 4 lines of code

NDF Filtering

Specular AA

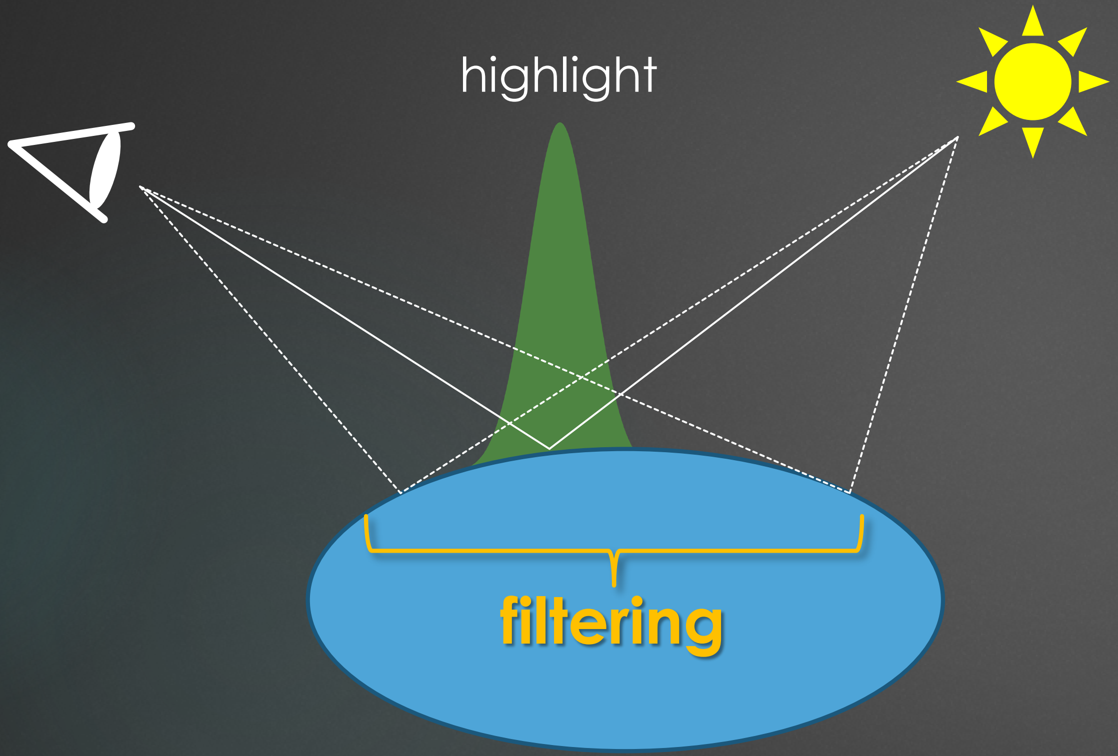


Antialiasing

||

Filtering in screen space

Specular AA



Antialiasing

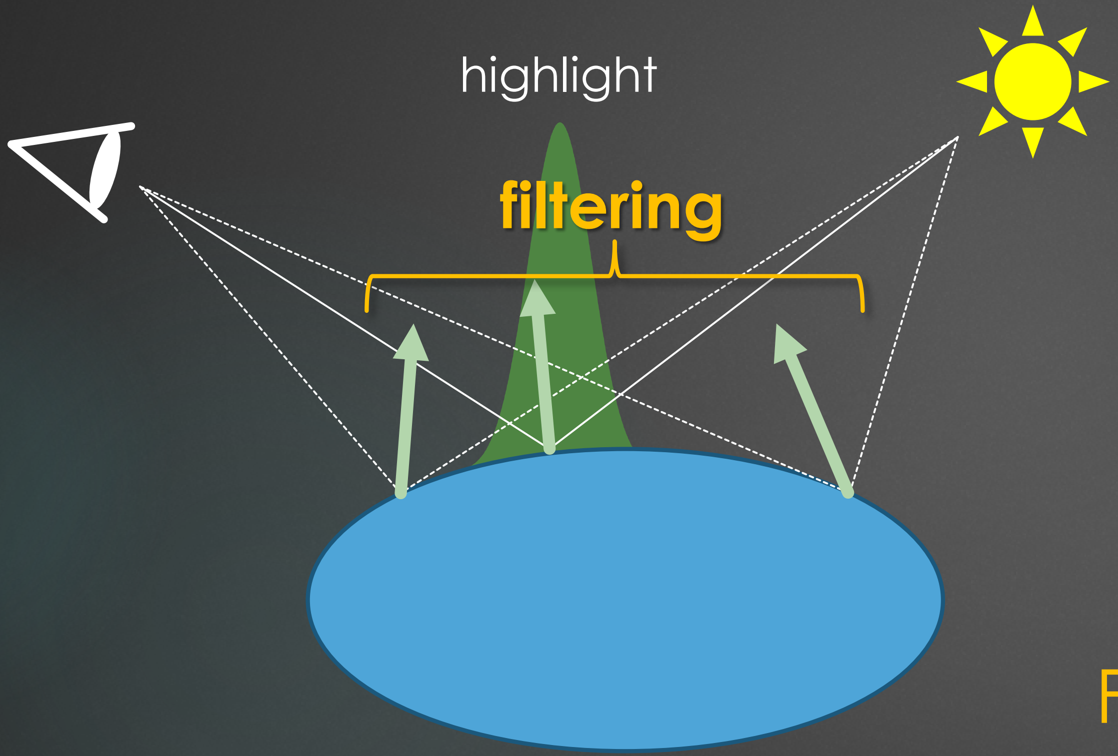
||

Filtering in screen space

⬇

Filtering in world space

Specular AA



Antialiasing

||

Filtering in screen space

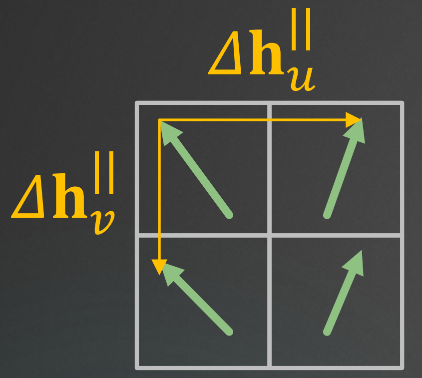
⬇

Filtering in world space

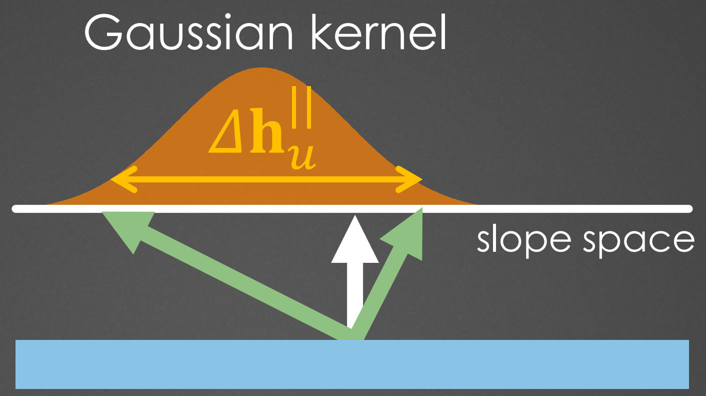
⬇

Filtering in halfvector-slope space

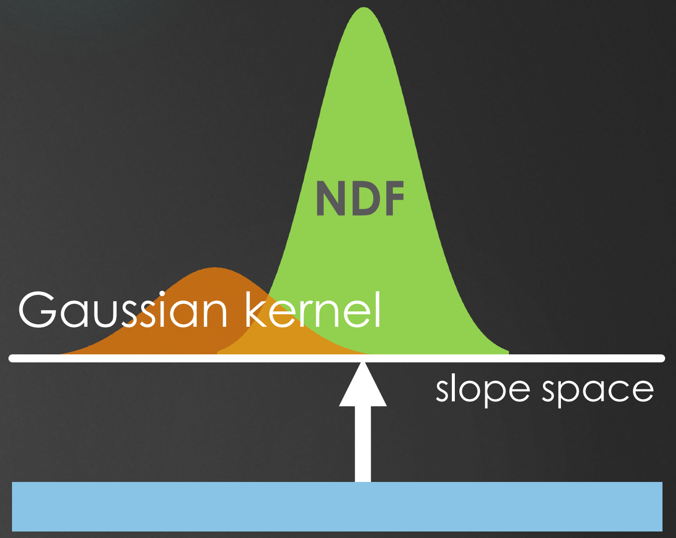
NDF Filtering in Pixel Shader



Derivative estimation



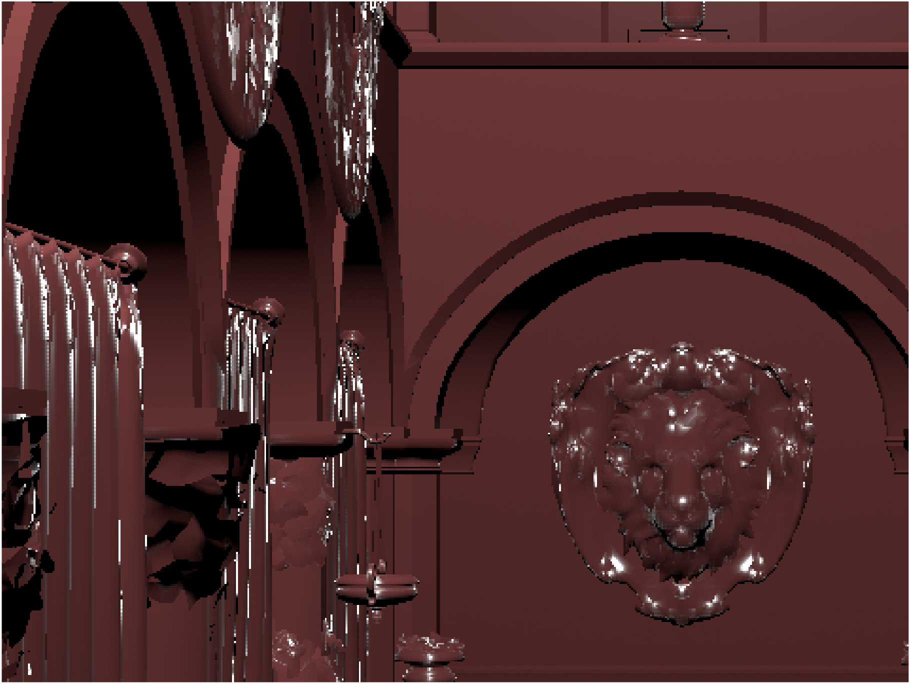
Pixel footprint in slope space



Convolution in slope space

- Estimate the derivatives of halfvector slopes
  - Rough estimation using the difference between contiguous pixels (i.e., ddx/ddy)
- Compute a 2×2 covariance matrix (i.e., Gaussian kernel) using the derivatives
- Filter the NDF using this Gaussian kernel by assuming the Beckmann NDF [1963]
  - Add the covariance matrix into the NDF variance (i.e., surface roughness)

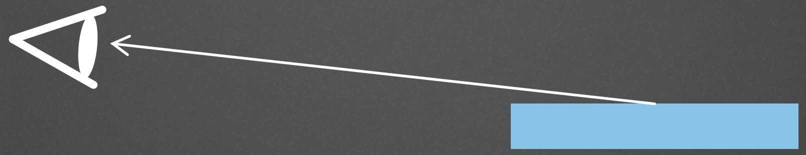
Estimation Error of Derivatives



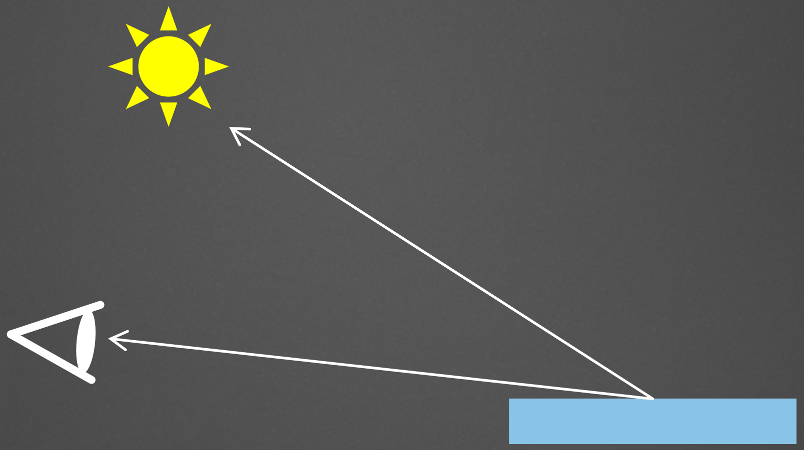
GGX NDF

- Artifacts for grazing angles
- Noticeable especially for the GGX NDF
  - Due to a heavier tail than the Beckmann NDF

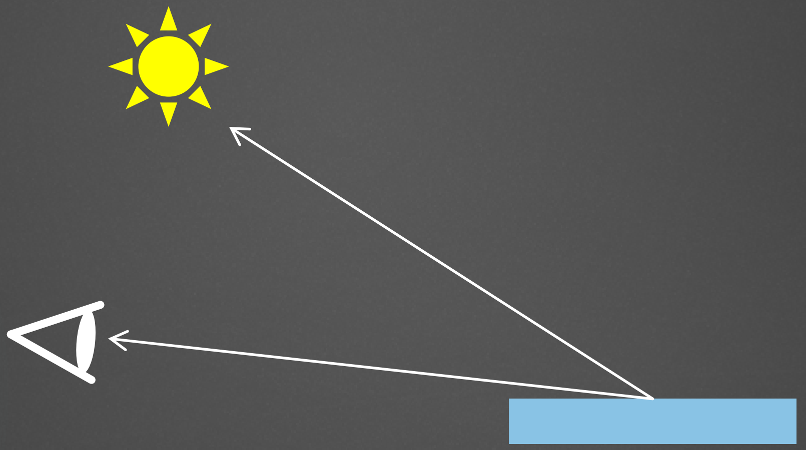
Estimation Error for Grazing Angles

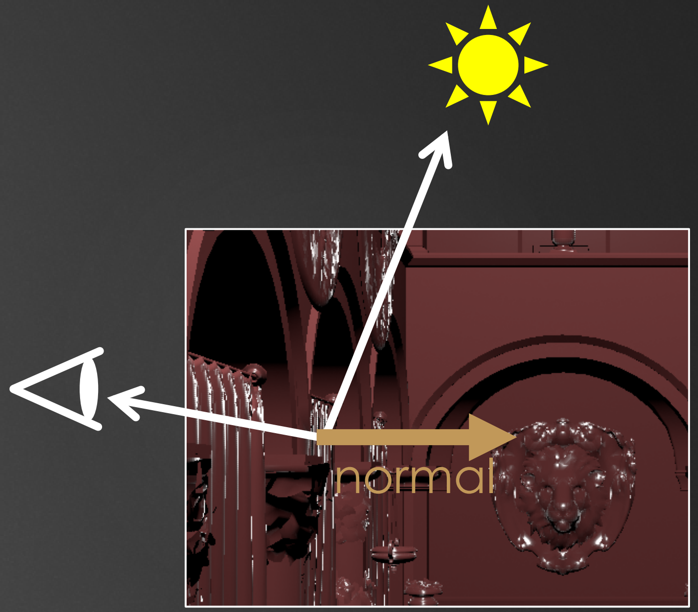


Estimation Error for Grazing Angles

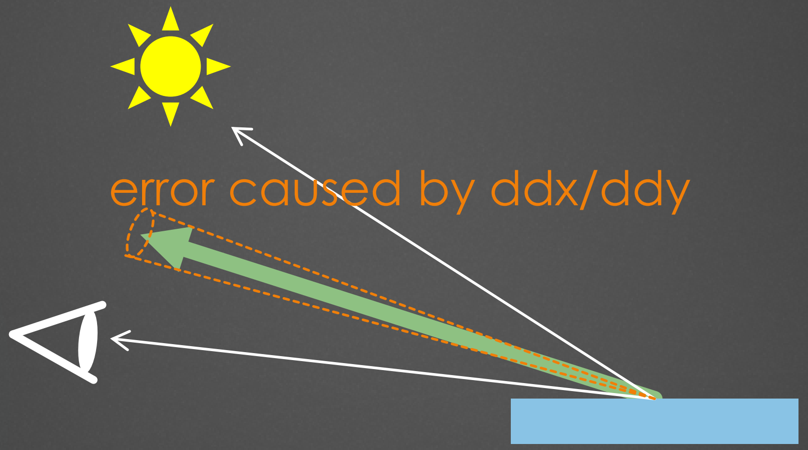


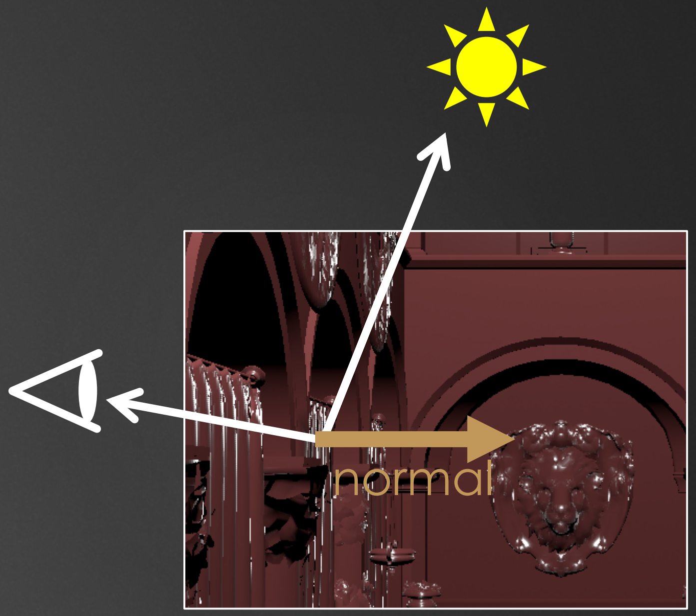
Estimation Error for Grazing Angles



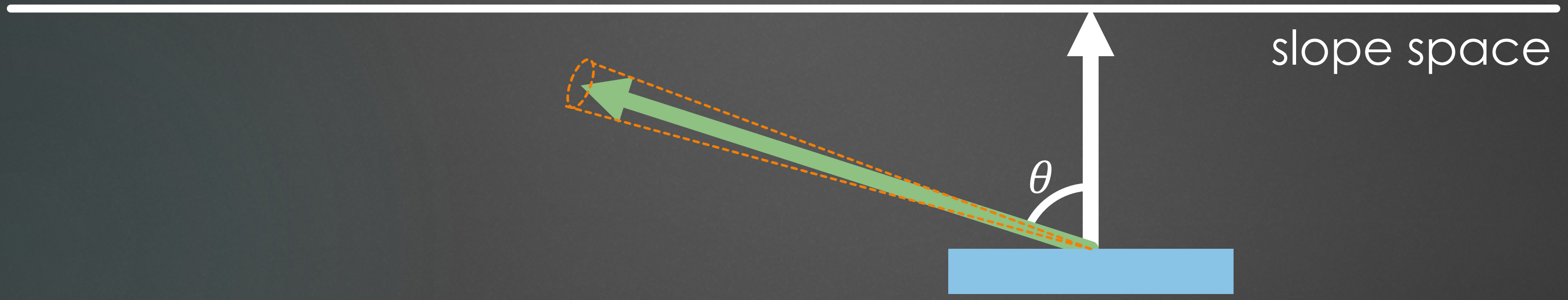


Estimation Error for Grazing Angles

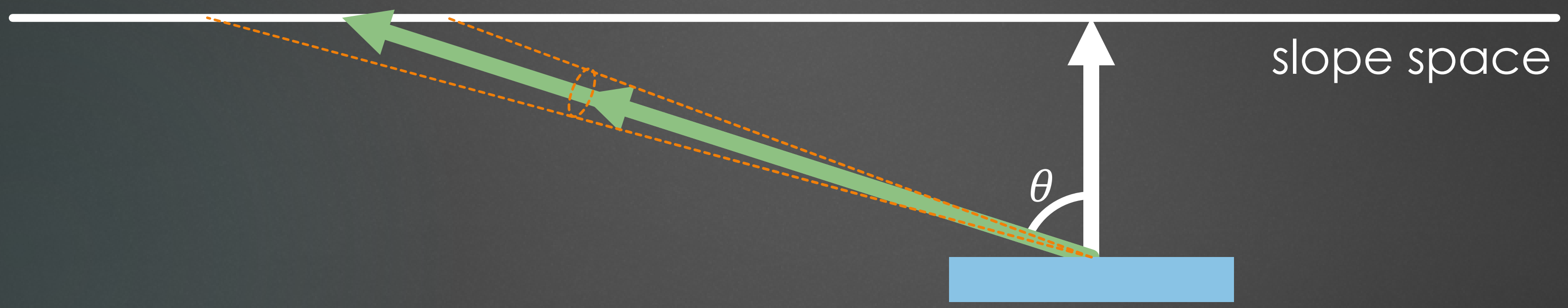




Estimation Error for Grazing Angles



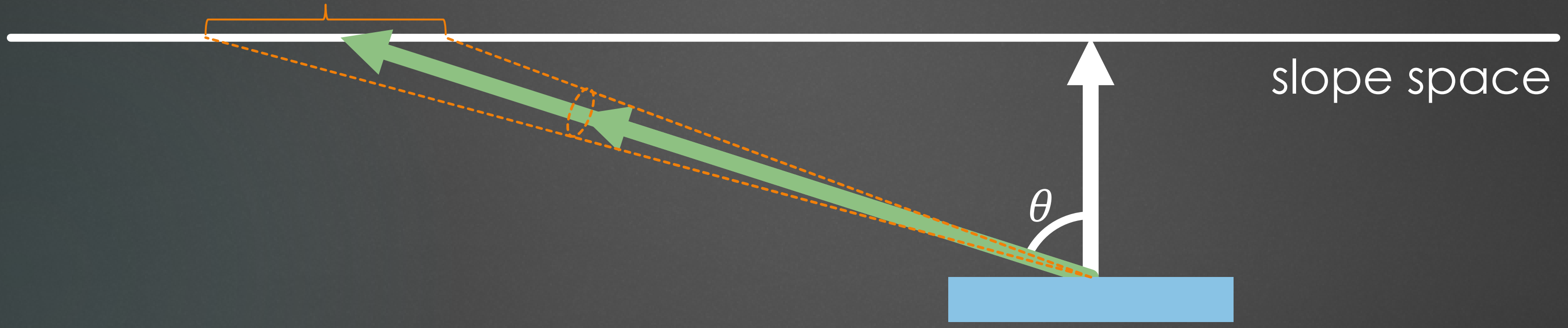
Estimation Error for Grazing Angles



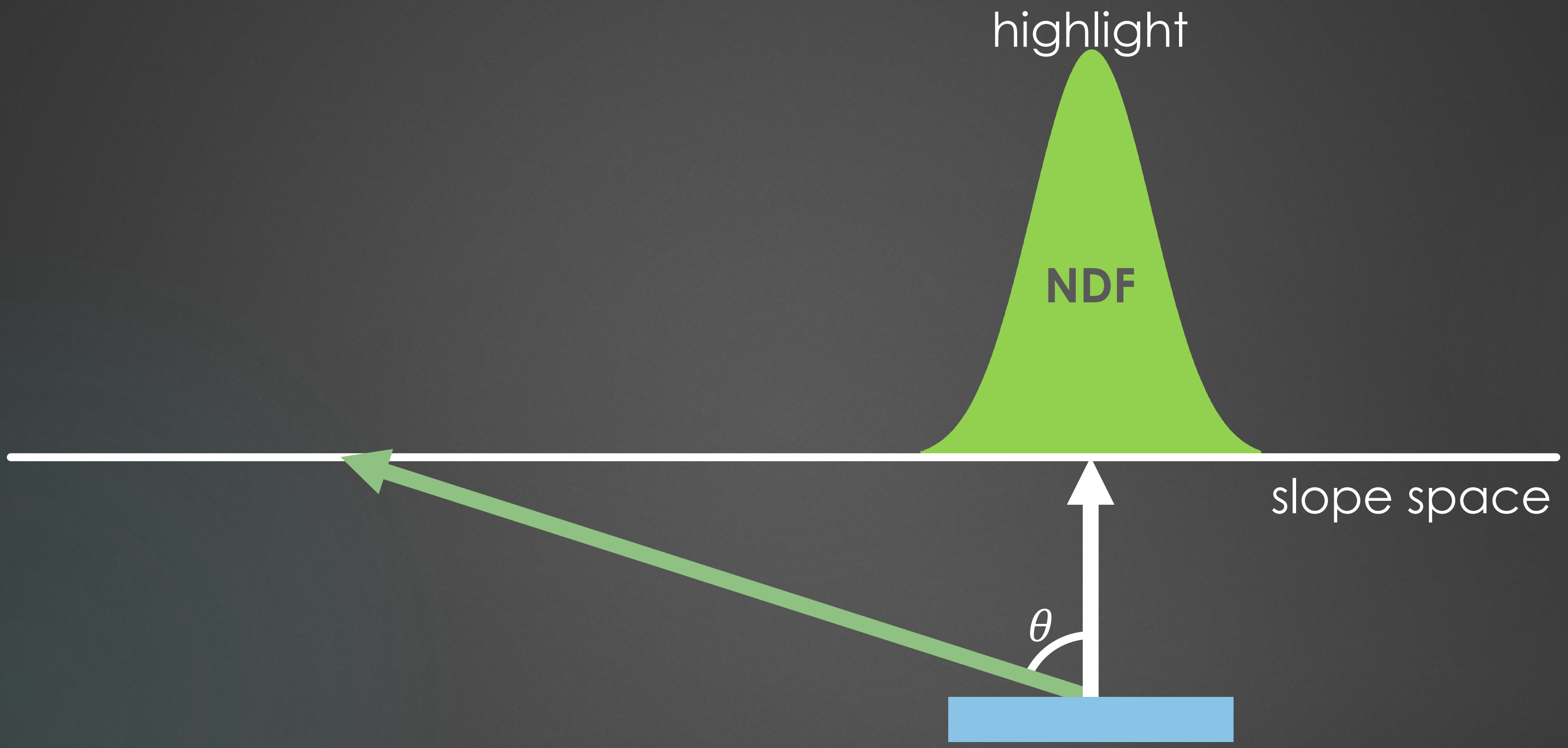
Estimation Error for Grazing Angles

Jacobian from directions to slopes

$$\text{error in slope space} \propto \frac{1}{\cos^3\theta}$$



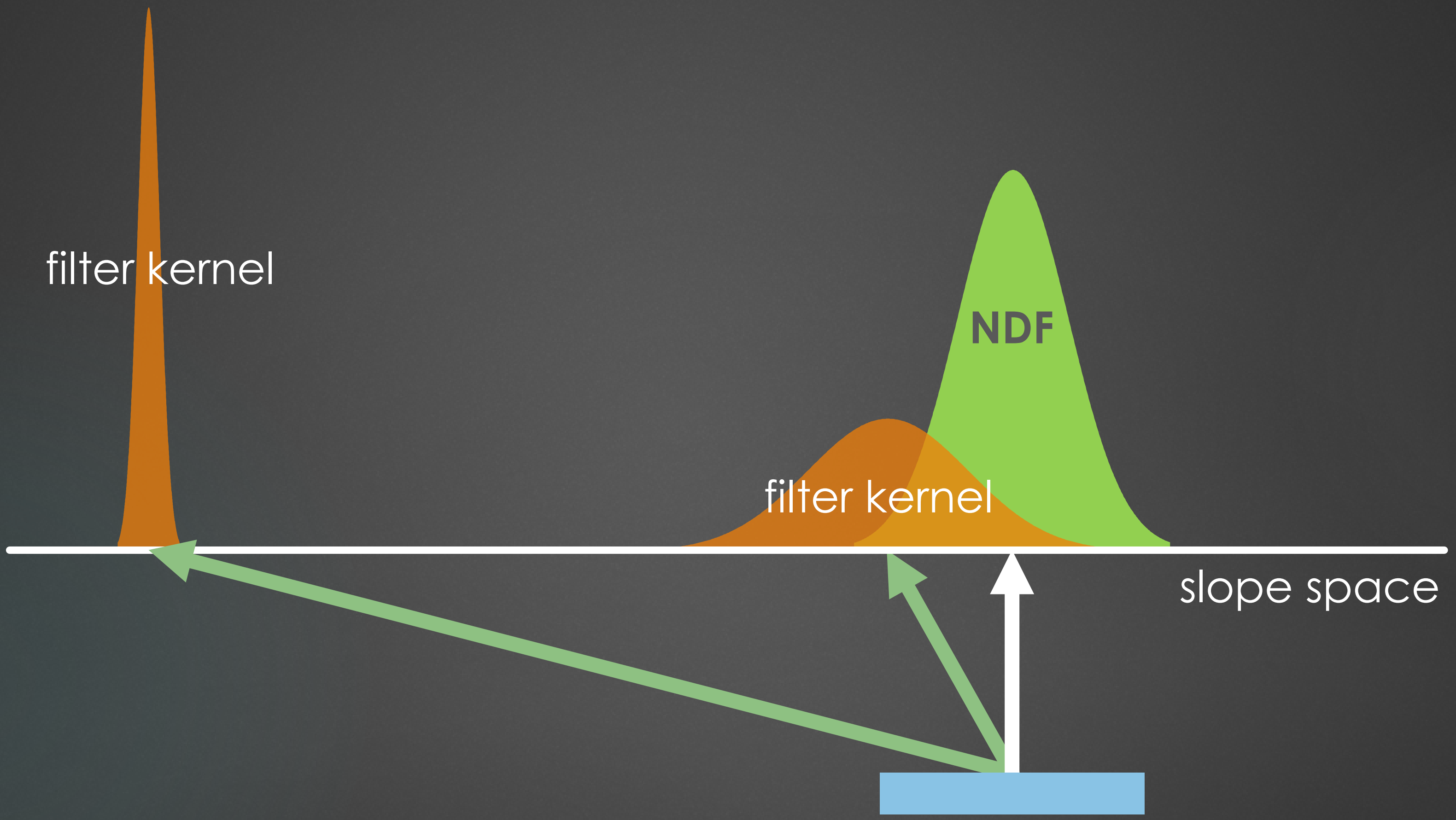
Estimation Error for Grazing Angles



Actually, NDF filtering is unnecessary for grazing angles, because they don’t produce highlights

Our Improvement

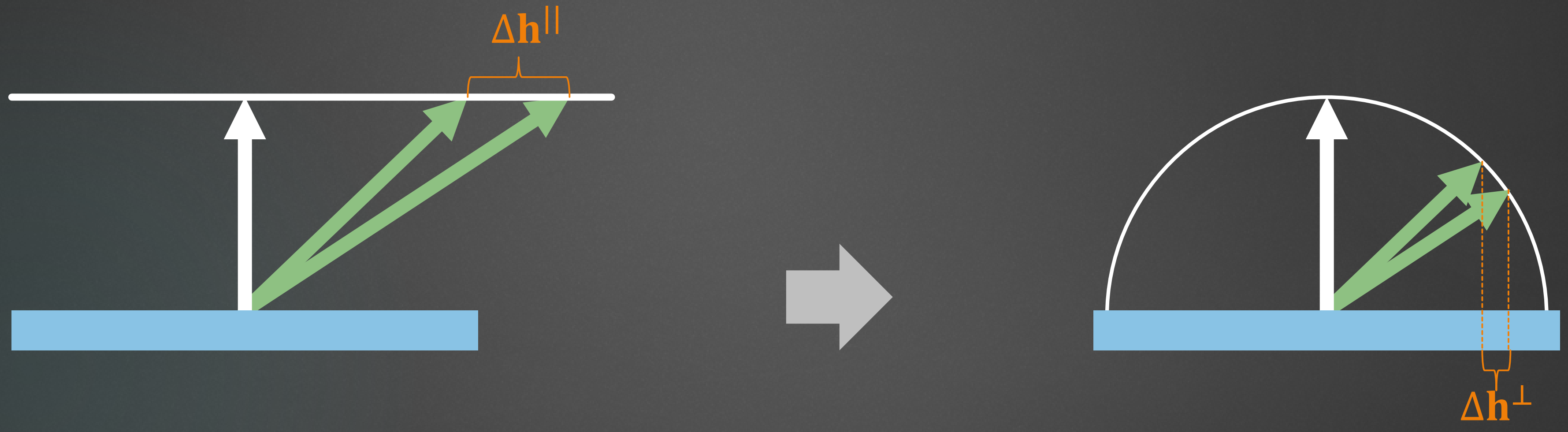
Our Filter Kernel



Higher-frequency kernel for a shallower halfvector angle

Projection onto a Unit Disk

Shrink the kernel size by estimating derivatives in a projected space



[Kaplanyan16]

Ours

Code of Derivative Estimation

```
float3 halfvector = normalize( viewDirection + lightDirection );
float3 halfvectorTS = mul( tangentFrame, halfvector );
float2 halfvector2D = halfvectorTS.xy / abs( halfvectorTS.z );
float2 deltaU = ddx( halfvector2D );
float2 deltaV = ddy( halfvector2D );
```

Code of Derivative Estimation

```
float3 halfvector = normalize( viewDirection + lightDirection );
float3 halfvectorTS = mul( tangentFrame, halfvector );
float2 halfvector2D = halfvectorTS.xy ~~/ abs( halfvectorTS.z )~~;
float2 deltaU = ddx( halfvector2D );
float2 deltaV = ddy( halfvector2D );
```

Remove from the [Kaplanyan16]'s implementation

Code of Derivative Estimation

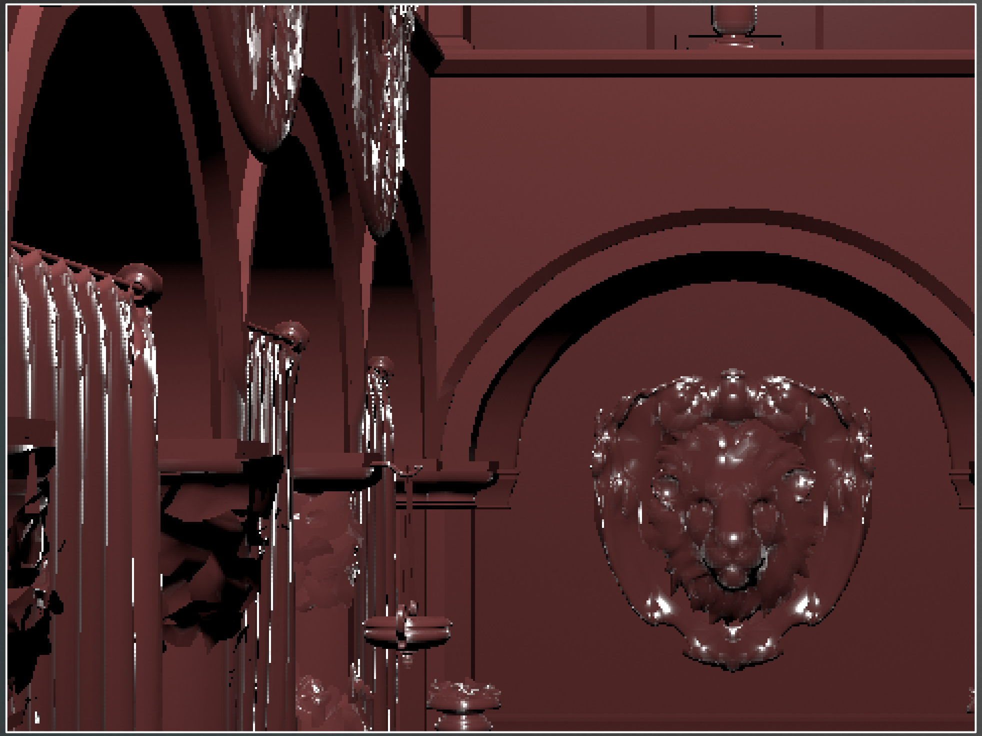
```
float3 halfvector = normalize( viewDirection + lightDirection );
float3 halfvectorTS = mul( tangentFrame, halfvector );
float2 halfvector2D = halfvectorTS.xy ~~/ abs( halfvectorTS.z )~~;
float2 deltaU = ddx( halfvector2D );
float2 deltaV = ddy( halfvector2D );
```

Remove from the [Kaplanyan16]'s implementation

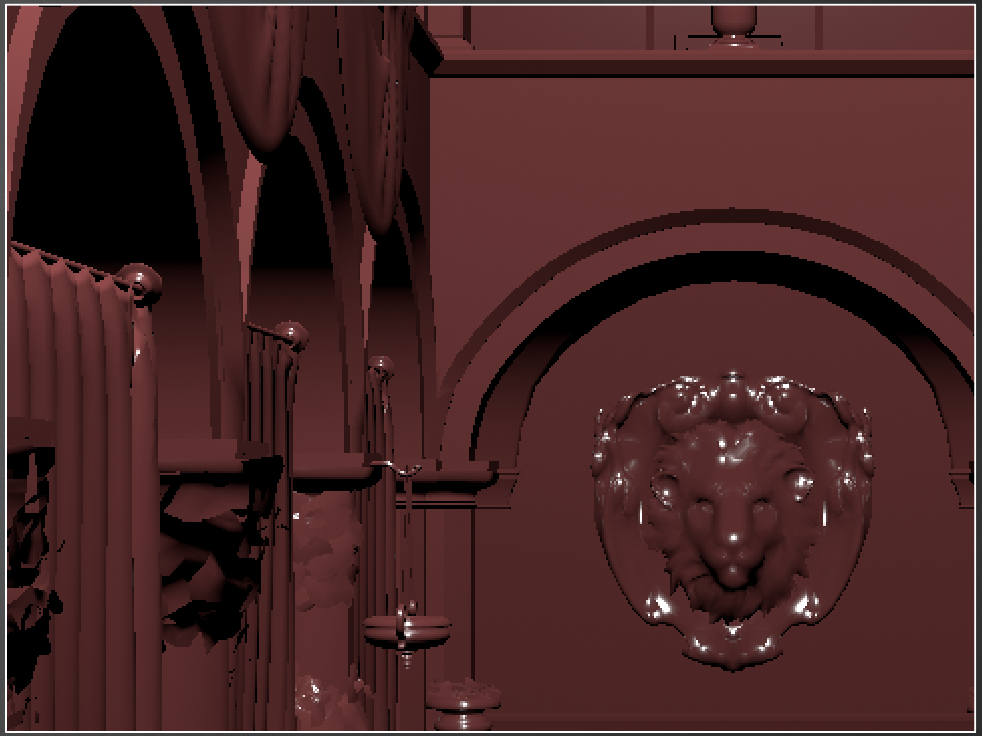
Simple ☺

Results (Non-Axis-Aligned Filtering)

GGX microfacet BRDF (roughness: 0.01)



[Kaplanyan16]



Ours

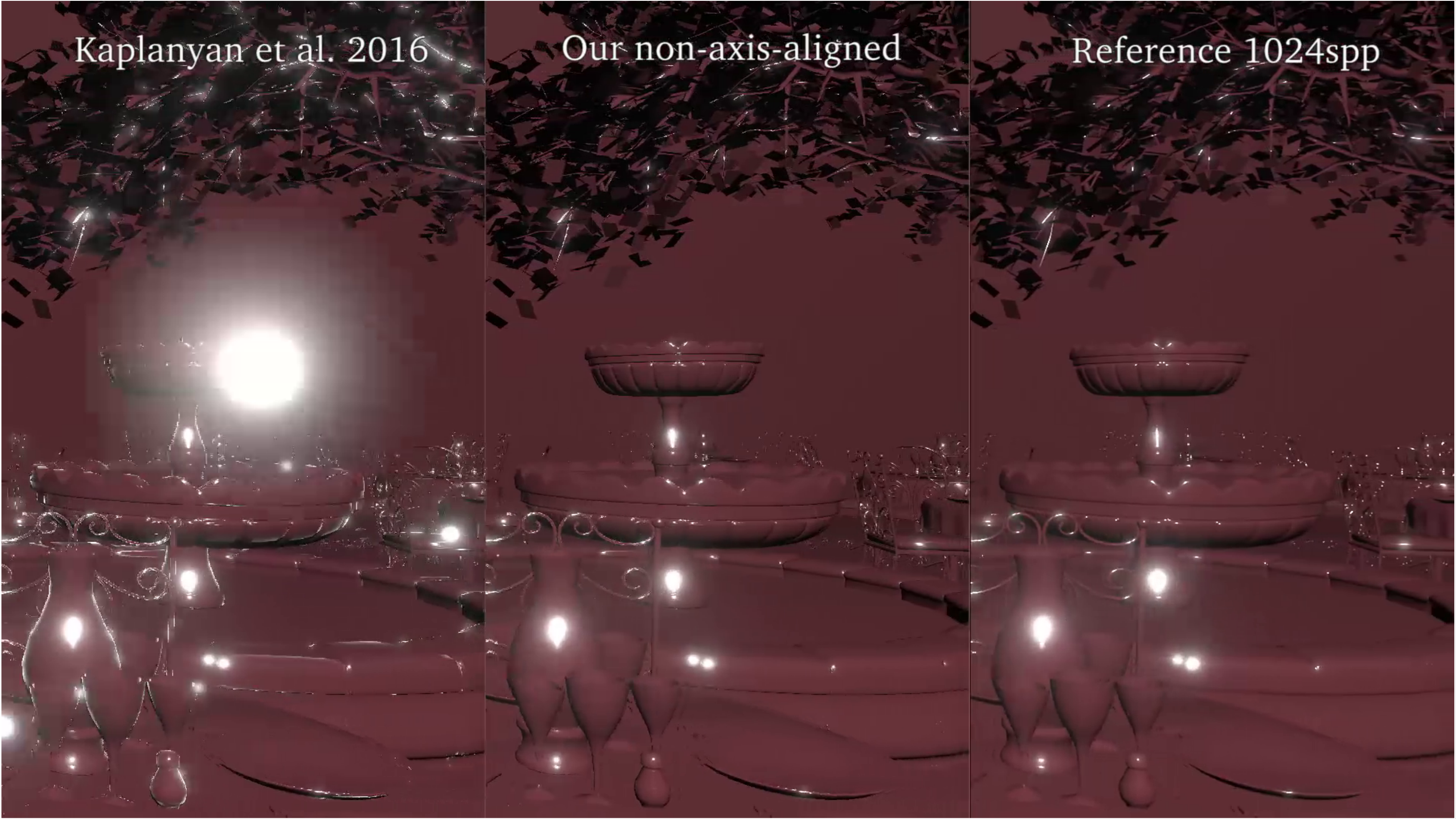
Results

```
CPU   0.261 ms, GPU   2.854 ms, 325 Hz

Engine Tuning
+ Application/...
[X]                 Display Frame Rate
[X]                 Display Profiler
- Experiments/...
   Our Axis-Aligned  SAA
   [-]               SSAA
+ Graphics/...
[-]                 Load Settings
[-]                 Save Settings
+ Timing/...
```

with a bloom posteffect (1920×1080 pixels)

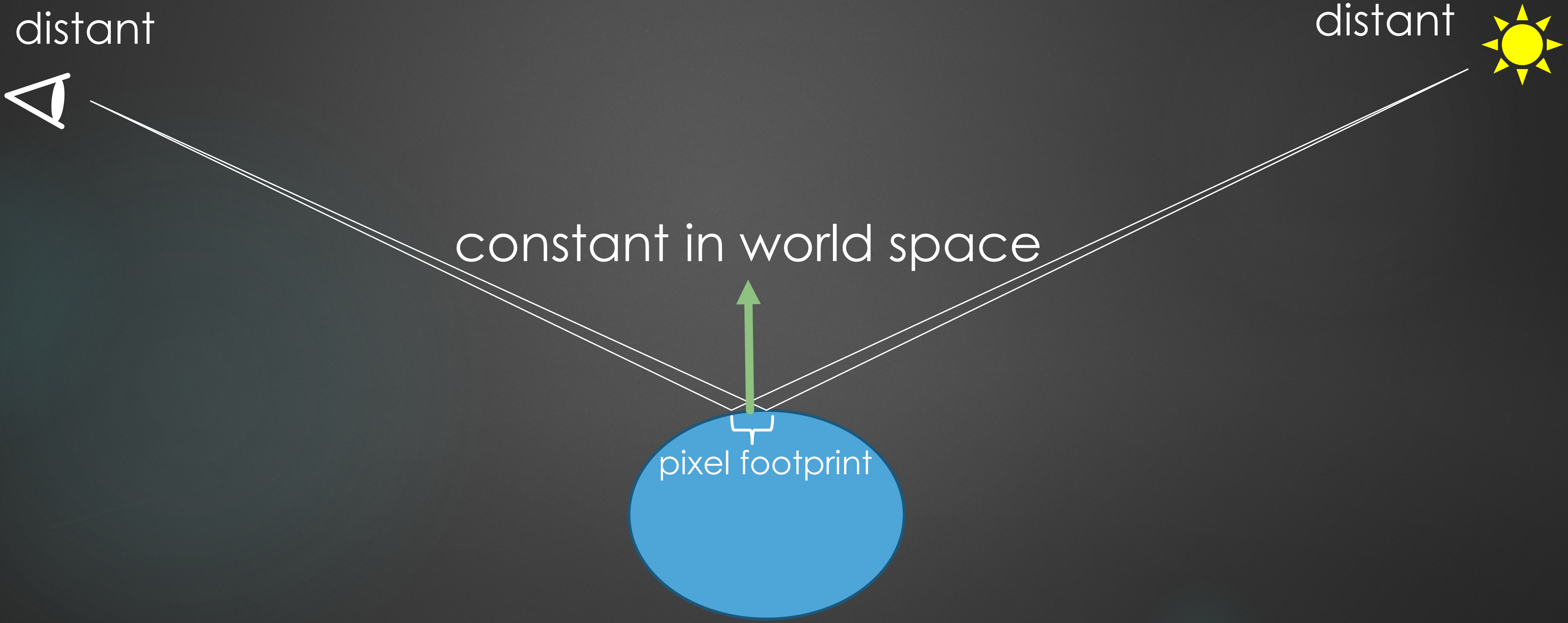
Comparison with the Reference (1024 spp)



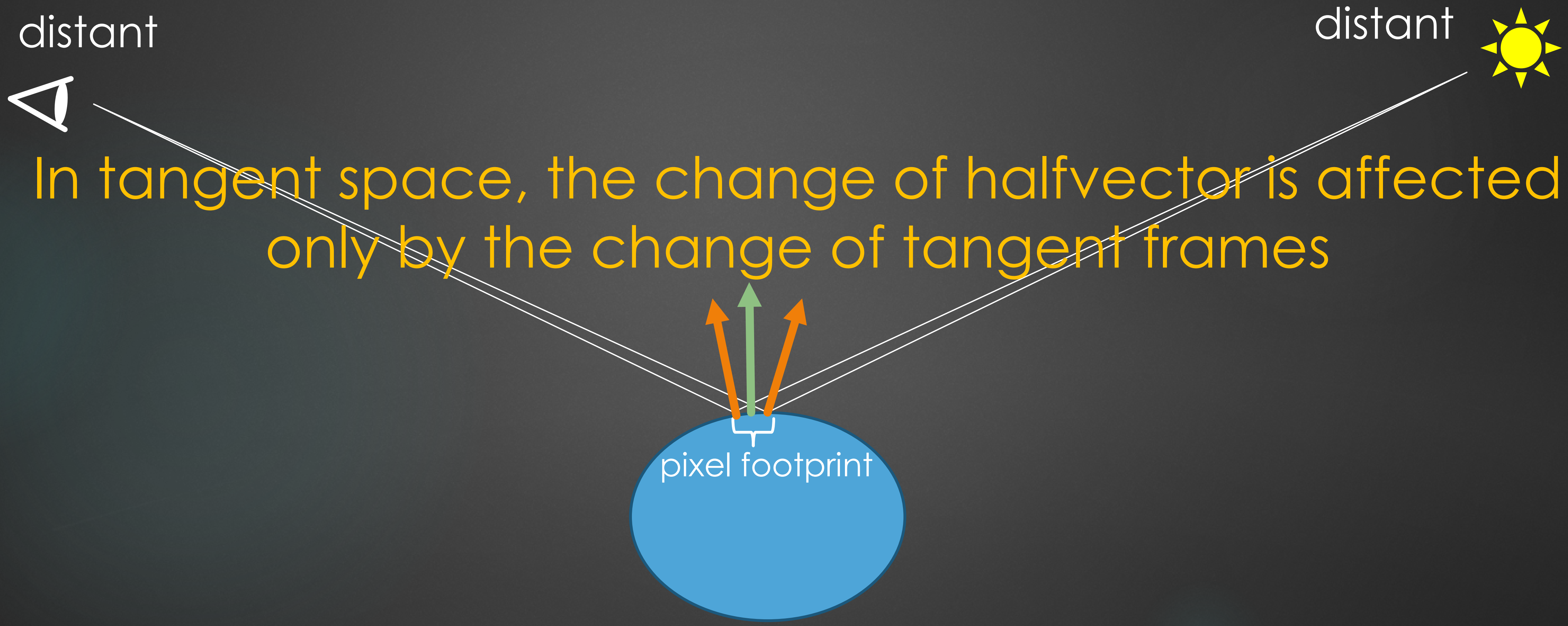
with a bloom posteffect (1920×1080 pixels)

Simplification for Deferred Rendering

Approximation for Deferred Rendering

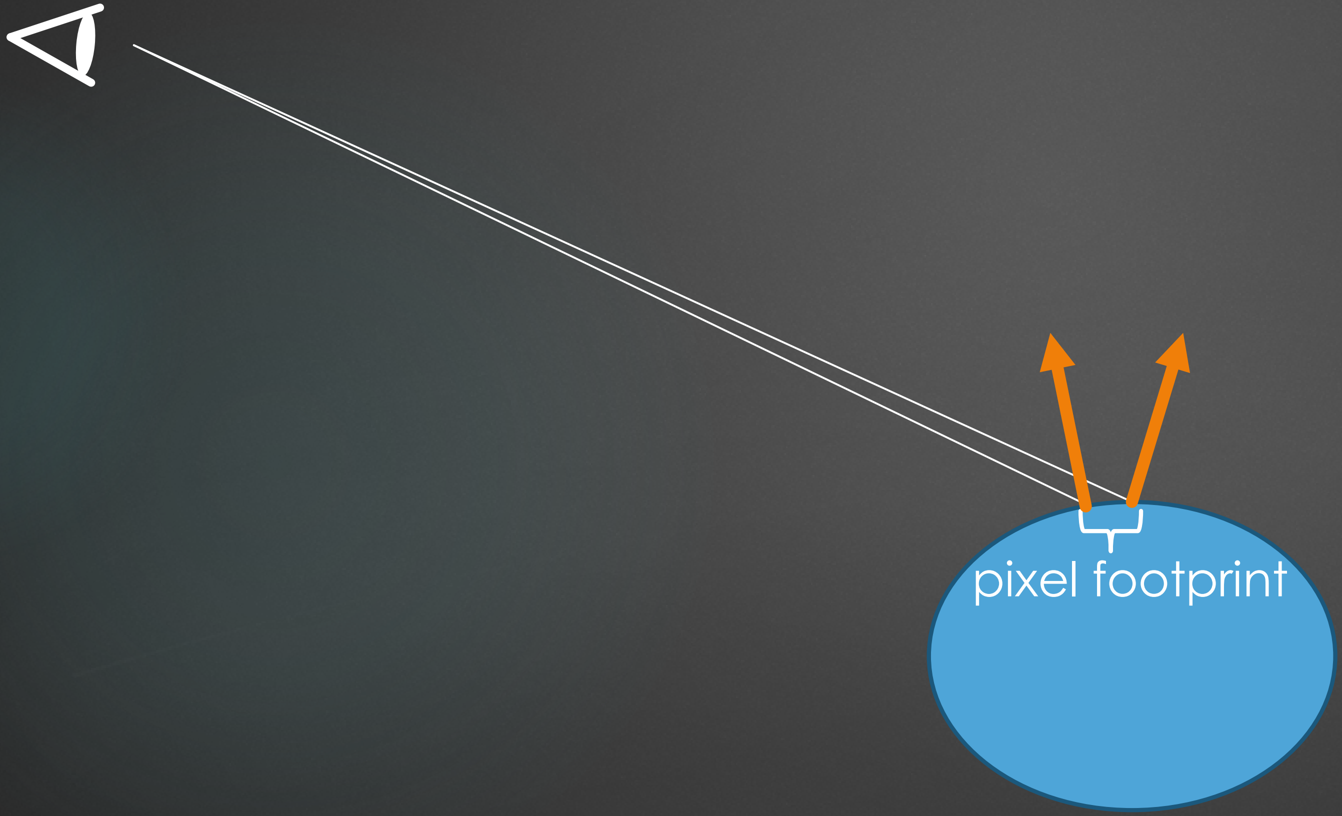


Approximation for Deferred Rendering



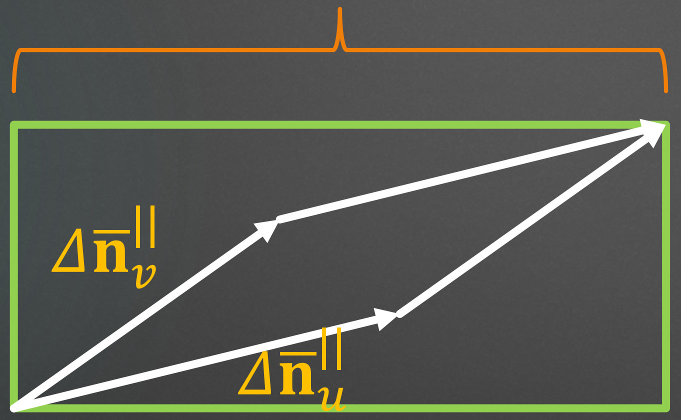
In tangent space, the change of halfvector is affected only by the change of tangent frames

Approximation for Deferred Rendering



Previous Approximation

- Average normal in the shading quad instead of the halfvector
- Isotropic filtering for a compact G-buffer (i.e., scalar roughness)
- Conservative (i.e., overfiltering)
  - Kernel size = Maximum width of the axis-aligned filter kernel



Axis-aligned kernel in slope space

Our Approach

- Based on the average eigenvalue of the 2×2 covariance matrix
- Eliminate the computation of average normal in tangent space
- Balance overfiltering and underfiltering

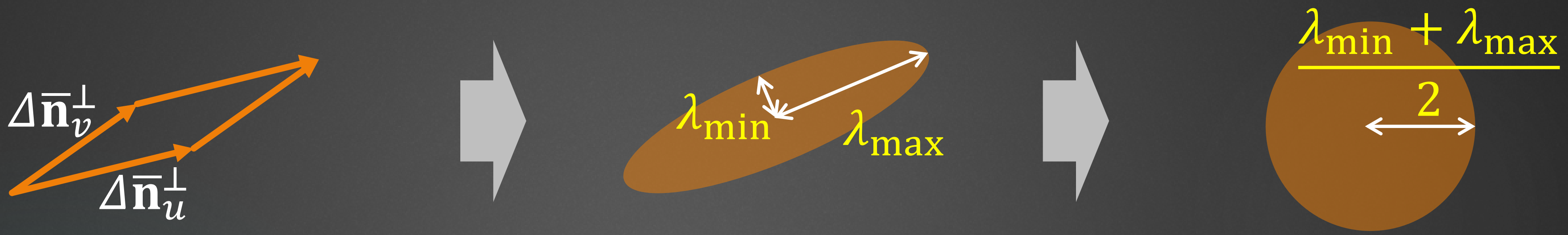
```
float2 neighboringDir = 0.5 - 2.0 * frac( pixelPosition * 0.5 );
float3 deltaNormalX = ddx_fine( normal ) * neighboringDir.x;
float3 deltaNormalY = ddy_fine( normal ) * neighboringDir.y;
float3 avgNormal = normal + deltaNormalX + deltaNormalY;
float3 avgNormalTS = mul( tangentFrame, avgNormal );
float2 avgNormal2D = avgNormalTS.xy / abs( avgNormalTS.z );
float2 deltaU = ddx( avgNormal2D ), deltaV = ddy( avgNormal2D );
float2 boundingRectangle = abs( deltaU ) + abs( deltaV );
float maxWidth = max( boundingRectangle.x, boundingRectangle.y );
float variance = SIGMA2 * maxWidth * maxWidth;
float kernelRoughness2 = min( 2.0 * variance, KAPPA );
float filteredRoughness2 = saturate( roughness2 + kernelRoughness2 );
```

Previous code

```
float3 dndu = ddx( normal ), dndv = ddy( normal );
float variance = SIGMA2 * ( dot( dndu, dndu ) + dot( dndv, dndv ) );
float kernelRoughness2 = min(variance, KAPPA );
float filteredRoughness2 = saturate( roughness2 + kernelRoughness2 );
```

Our code

Kernel Size Using the Average Eigenvalue



Pixel footprint in tangent space

Gaussian kernel
(non-axis-aligned)

Isotropic Gaussian kernel

Kernel Size Using the Average Eigenvalue

Pixel footprint in tangent space

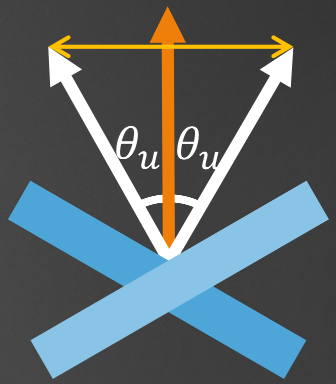
Isotropic Gaussian kernel

$$\lambda_{\min}+\lambda_{\max}=\sigma^2\left(\left\|\Delta\bar{\mathbf{n}}_u^\perp\right\|^2+\left\|\Delta\bar{\mathbf{n}}_v^\perp\right\|^2\right)$$

- Sum of eigenvalues is given by the trace of the covariance matrix
- Use only the norms of derivatives

Norms of Derivatives

$$\left\|\Delta \bar{\mathbf{n}}_u^{\perp}\right\| = \left\|\Delta \mathbf{n}_u\right\|$$

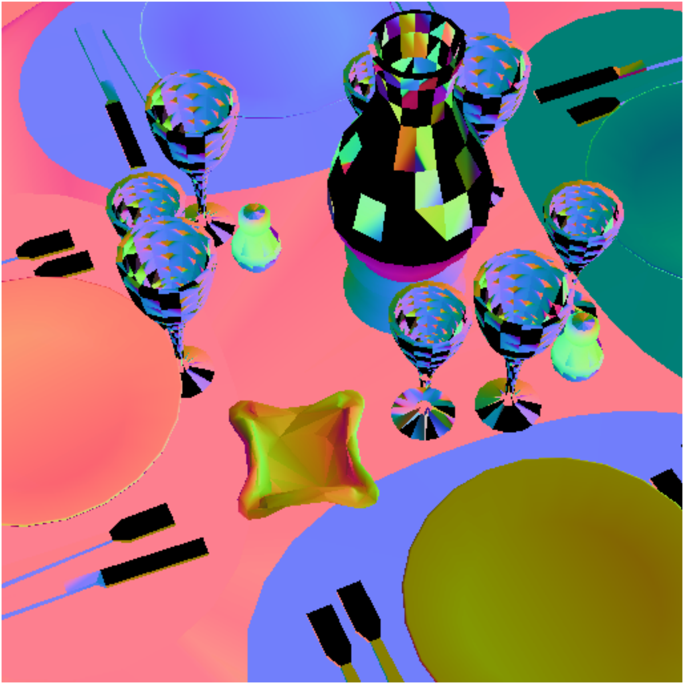


Derivative of average normals in tangent space

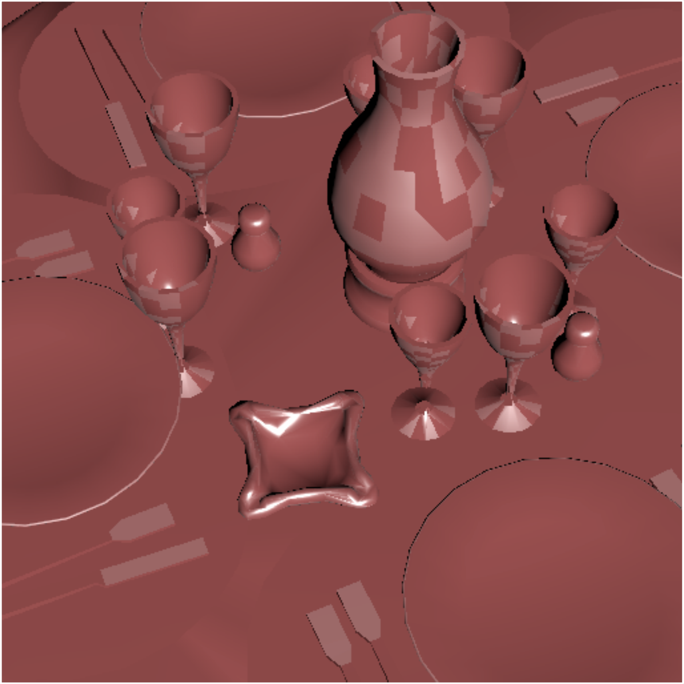
Derivative of world-space normals

- Replace by the norms of world-space derivatives
  - Using the average normal of two contiguous pixels for each screen axis
- No need to compute the average normal in tangent space ☺

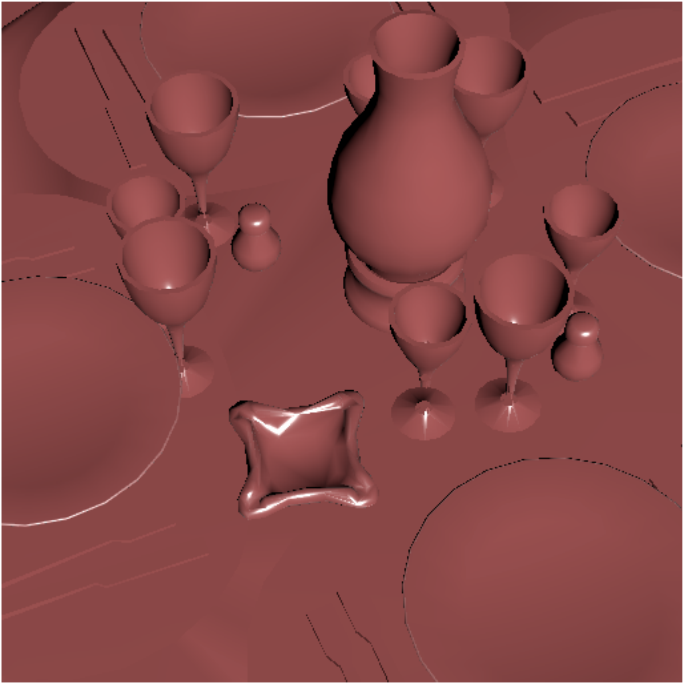
Objects with Invalid Tangent Vectors



Tangent vectors

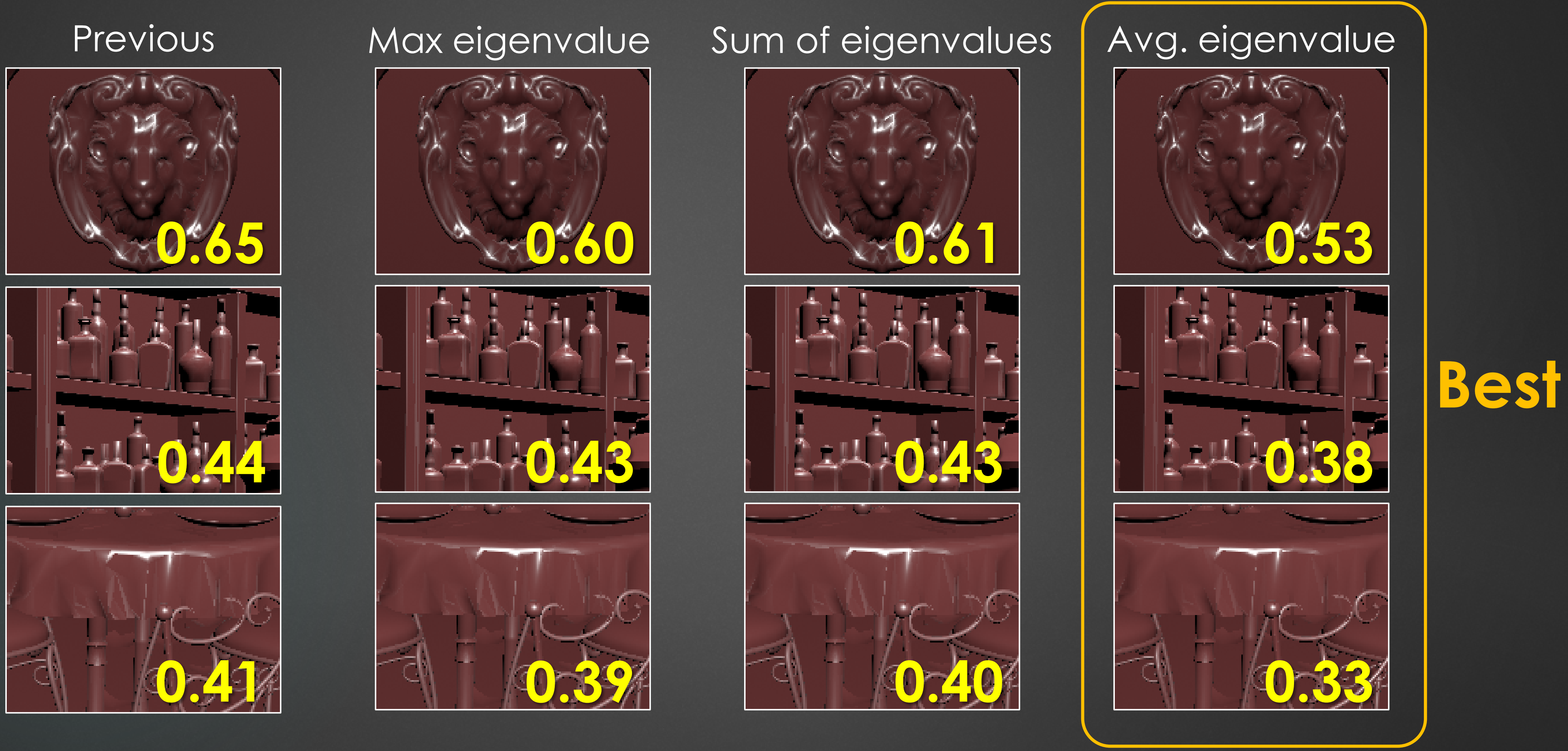


Previous



Ours

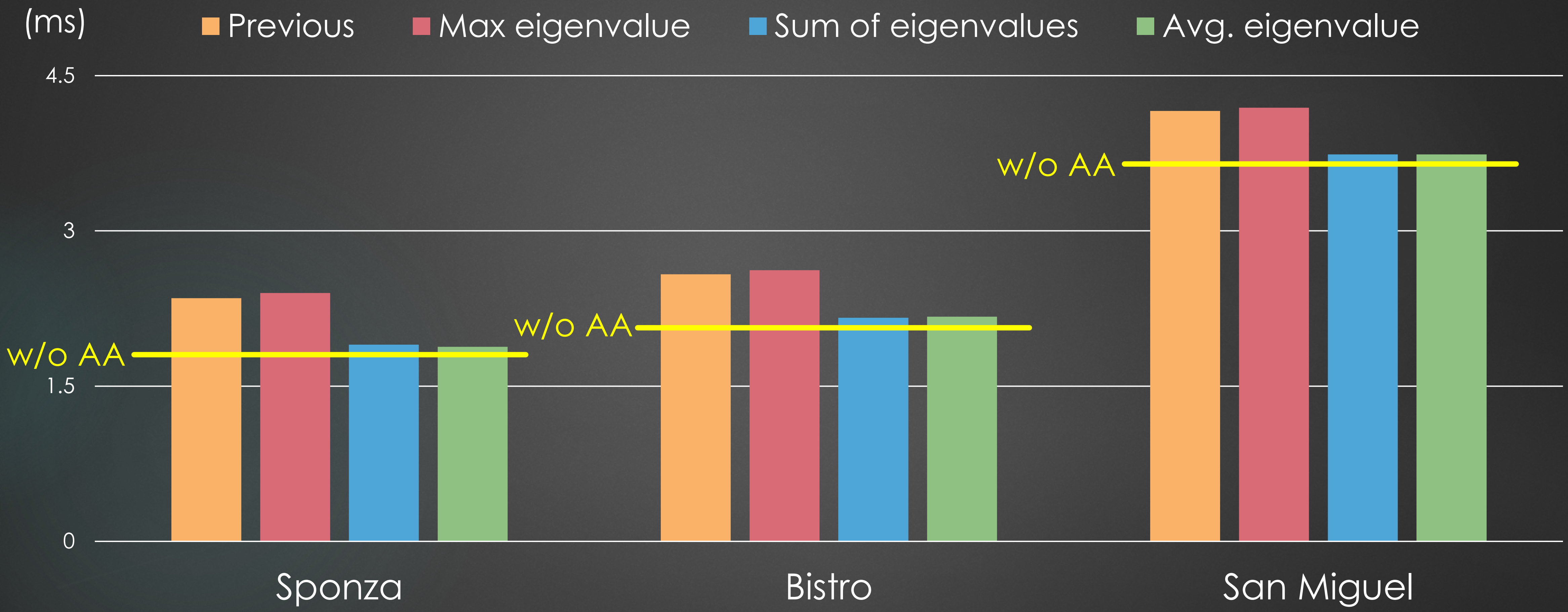
Filtering Quality (RMSE)



Application to Forward Rendering

- NDF filtering is not a bottleneck when rendering a G-buffer
- However, normal-based filtering can also be desirable to use for forward rendering
  - Constant filtering cost for many lights
  - Applicable to any real-time approximations
  - E.g., area lights, IBL, and indirect illumination

Performance (8K, Forward Rendering)



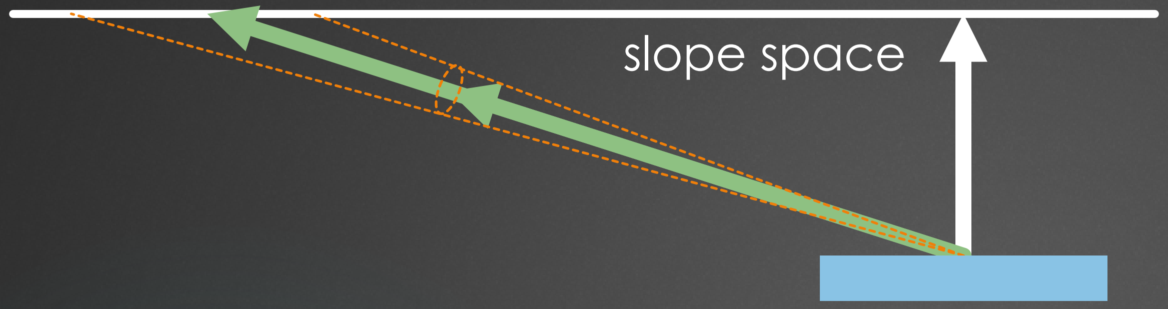
7680×4320 screen resolution, GPU: AMD Radeon^{TM} RX Vega 56

Limitations & Conclusions

Limitations

- Inherited from [Kaplanayan16]
  - Geometric discontinuities
  - Bias introduced by approximating the pixel footprint
  - Bias introduced by approximating the GGX NDF with the Beckmann NDF
  - Require high-quality tangent frames for anisotropic filtering
    - For our isotropic filtering, this limitation is alleviated to high-quality shading normals
- Underfiltering for grazing halfvectors
  - Usually not a problem
  - Aliasing is small for grazing halfvectors

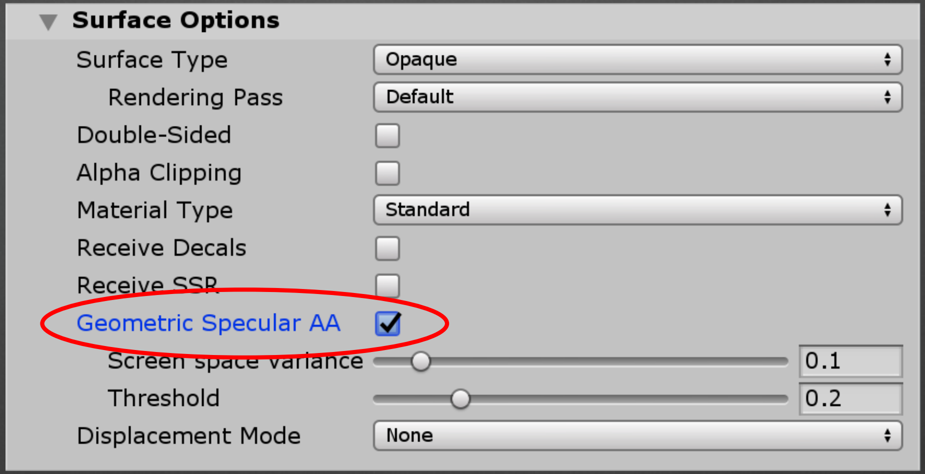
Conclusions



- Estimation error of slope derivatives is increased for grazing halfvectors
- Reduced the filtering error using a higher-frequency kernel for a shallower halfvector
  - Slope → projected halfvector (orthographic projection)
  - Simpler than the previous method
- Optimized normal-based isotropic NDF filtering (4 lines of code)

Application

- Already implemented in Unity HDRP
  - Based on our technical report [Tokuyoshi17]
  - Isotropic normal-based filtering
  - Source code: https://github.com/Unity-Technologies/ScriptableRenderPipeline



Surface Options

Surface Type: Opaque
Rendering Pass: Default
Double-Sided: ☐
Alpha Clipping: ☐
Material Type: Standard
Receive Decals: ☐
Receive SSR: ☐
Geometric Specular AA: ☑
Screen space variance: 0.1
Threshold: 0.2
Displacement Mode: None

References

- P. Beckmann and A. Spizzichino. 1963. The Scattering of Electromagnetic Waves from Rough Surfaces. Pergamon Press.
- R. L. Cook and K. E. Torrance. 1982. A Reflectance Model for Computer Graphics. ACM Trans. Graph. 1, 1 (1982), 7–24.
- A. S. Kaplanyan, S. Hill, A. Patney, and A. Lefohn. 2016. Filtering Distributions of Normals for Shading Antialiasing. In HPG ’16. 151–162.
- Y. Tokuyoshi. 2017. Error Reduction and Simplification for Shading Anti-aliasing. Technical Report.
- B.Walter, S. Marschner, H. Li, and K. Torrance. 2007. Microfacet Models for Refraction through Rough Surfaces. In EGSR ’07. 195–206.

Bonus

HLSL Code (Non-Axis-Aligned Filtering)

```
float3 halfvector = normalize( viewDirection + lightDirection );
float3 halfvectorTS = mul( tangentFrame, halfvector );
float2 halfvector2D = halfvectorTS.xy;
float2 deltaU = ddx( halfvector2D );
float2 deltaV = ddy( halfvector2D );
float2x2 delta = { deltaU, deltaV };
float2x2 covarianceMatrix = SIGMA2 * mul( transpose( delta ), delta );
float2x2 roughnessMatrix = { roughness2.x, 0.0, 0.0, roughness2.y };
float2x2 filteredRoughnessMatrix = roughnessMatrix + 2.0 * covarianceMatrix;
```

roughness2: squared surface roughness (i.e., α_x^2, α_y^2 in the paper)
SIGMA2: screen-space variance (i.e., $\sigma^2 = 0.25$ in the paper)

HLSL Code (Biased Axis-Aligned Filtering)

```
float3 halfvector = normalize( viewDirection + lightDirection );
float3 halfvectorTS = mul( tangentFrame, halfvector );
float2 halfvector2D = halfvectorTS.xy;
float2 deltaU = ddx( halfvector2D );
float2 deltaV = ddy( halfvector2D );
float2 boundingRectangle = abs( deltaU ) + abs( deltaV );
float2 variance = SIGMA2 * ( boundingRectangle * boundingRectangle );
float2 kernelRoughness2 = min( 2.0 * variance, KAPPA );
float2 filteredRoughness2 = saturate( roughness2 + kernelRoughness2 );
```

roughness2: squared surface roughness (i.e., α_x^2, α_y^2 in the paper)
SIGMA2: screen-space variance (i.e., $\sigma^2 = 0.25$ in the paper)
KAPPA: clamping threshold (i.e., $\kappa = 0.18$ in the paper)