# Hierarchical Light Sampling with Accurate Spherical Gaussian Lighting (Supplementary Document)

YUSUKE TOKUYOSHI, Advanced Micro Devices, Inc., Japan
SHO IKEDA, Advanced Micro Devices, Inc., Japan
PARITOSH KULKARNI*, Advanced Micro Devices, Inc., Canada
TAKAHIRO HARADA, Advanced Micro Devices, Inc., USA

## 1 Derivation for the Jacobian Matrix J

Stam [2001] derived the Jacobian matrix for the transformation between halfvectors and light directions on a tangent plane by assuming that the view direction $\mathbf{i}$ is aligned with the $x$-axis of the plane. At $\mathbf{h} = \mathbf{n}$, it is given by

$$\mathbf{J}_\perp = \begin{cases} \frac{1}{2(\mathbf{i}\cdot\mathbf{h})}\mathbf{E} & \text{for reflection} \\ \frac{\eta}{(\mathbf{i}\cdot\mathbf{h})+\eta(\mathbf{o}\cdot\mathbf{h})}\mathbf{E} & \text{for refraction} \end{cases},$$

where $\eta \in (0, \infty)$ is the relative refractive index. By orthographically projecting the light direction $\mathbf{o}$ onto the plane, we get the Jacobian matrix for the transformation between halfvectors and light directions on the unit sphere:

$$\mathbf{J}_\vee = \mathbf{J}_\perp \begin{bmatrix} |\mathbf{o}\cdot\mathbf{h}| & 0 \\ 0 & 1 \end{bmatrix}.$$

where we assume that the $x$-axis of the plane is aligned with $\mathbf{i} - \mathbf{o}$. To extend this Jacobian matrix for arbitrary $\mathbf{i}$, $\mathbf{h}$, and $\mathbf{o}$, we transform the plane perpendicular to $\mathbf{h}$ into the tangent space. We first rotate the plane to align the $x$-axis with $\mathbf{h} - \mathbf{n}$, and then project it onto the tangent plane. After that, we rotate it to align the tangent space. With these transformations, we obtain the following Jacobian matrix:

$$\mathbf{J} = \underbrace{\begin{bmatrix} \cos\phi_h & -\sin\phi_h \\ \sin\phi_h & \cos\phi_h \end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix} |\cos\theta_h| & 0 \\ 0 & 1 \end{bmatrix}}_{\substack{\text{projection to} \\ \text{tangent plane}}} \underbrace{\begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix}}_{\text{rotation}} \mathbf{J}_\vee.$$

where $[\theta_h, \phi_h] \in \mathbb{R}^2$ is the polar coordinates of the tangent-space halfvector (i.e., $\cos\theta_h = h_z$), and $\psi \in \mathbb{R}$ is the angle between $\mathbf{n} \times \mathbf{h}$ and $\mathbf{i} \times \mathbf{o}$. If $\mathbf{h} = \mathbf{n}$, we use $[\cos\phi_h, \sin\phi_h] = [i_x, i_y]/\|[i_x, i_y]\|$ and $\psi = 0$ as in the case where $\mathbf{i}$, $\mathbf{o}$, $\mathbf{h}$, and $\mathbf{n}$ are on the same great circle.

## 2 Non-Axis-Aligned Anisotropic Microfacet BRDFs

The microfacet bidirectional reflectance distribution function (BRDF) model [Cook and Torrance 1982] is defined as

$$f(\mathbf{i}, \mathbf{o}) = \frac{F(\mathbf{i}\cdot\mathbf{h})G_2(\mathbf{i}, \mathbf{o}, \mathbf{h})D(\mathbf{h})}{4|\mathbf{i}\cdot\mathbf{n}||\mathbf{o}\cdot\mathbf{n}|},$$

*Paritosh Kulkarni is now at Qualcomm.

Authors' Contact Information: Yusuke Tokuyoshi, Advanced Micro Devices, Inc., Japan, yusuke.tokuyoshi@amd.com; Sho Ikeda, Advanced Micro Devices, Inc., Japan, sho.ikeda@amd.com; Paritosh Kulkarni, Advanced Micro Devices, Inc., Canada; Takahiro Harada, Advanced Micro Devices, Inc., USA, takahiro.harada@amd.com.

where $F(\mathbf{i}\cdot\mathbf{h}) \in [0,1]$ is the Fresnel factor, $G_2(\mathbf{i},\mathbf{o},\mathbf{h}) \in [0,1]$ is the masking-shadowing function, and $D(\mathbf{h}) \in [0,\infty)$ is the microfacet normal distribution function (NDF). For the Smith microsurface model [1967], the height-correlated masking-shadowing function [Heitz 2014] is given by

$$G_2(\mathbf{i},\mathbf{o},\mathbf{h}) = \frac{H(\mathbf{i}\cdot\mathbf{h})H(\mathbf{o}\cdot\mathbf{h})}{1+\Lambda(\mathbf{i})+\Lambda(\mathbf{o})},$$

where

$$\Lambda(\mathbf{i}) = \frac{\int_{S^2} D(\boldsymbol{\omega})\max(\mathbf{i}\cdot\boldsymbol{\omega},0)\mathrm{d}\boldsymbol{\omega}}{|\mathbf{i}\cdot\mathbf{n}|} - 1 \quad \text{for } \mathbf{i}\cdot\mathbf{n} > 0.$$

Although microfacet BRDFs are typically used with axis-aligned anisotropy $[\alpha_x, \alpha_y]$, these BRDFs are straightforward to generalize for non-axis-aligned anisotropy [Heitz 2014]. In this paper, we use non-axis-aligned anisotropic microfacet BRDFs for NDF filtering.

### 2.1 Non-axis-aligned GGX NDF

The generalized GGX NDF is defined as

$$D(\mathbf{h};\mathbf{A}) = \frac{\chi^+(h_z)}{\pi\sqrt{\det(\mathbf{A})}\left(\left[h_x,h_y\right]\mathbf{A}^{-1}\left[h_x,h_y\right]^\top + h_z^2\right)^2},$$

where $\mathbf{A}$ is a 2×2 nonnegative-definite matrix representing the roughness for non-axis-aligned anisotropy. For this NDF, the masking-shadowing function is obtained using the following function:

$$\Lambda(\mathbf{i}) = \frac{\sqrt{k^2+1}-1}{2}, \quad \text{where } k = \sqrt{\left[\frac{i_x}{i_z},\frac{i_y}{i_z}\right]\mathbf{A}\left[\frac{i_x}{i_z},\frac{i_y}{i_z}\right]^\top}.$$

### 2.2 Non-axis-aligned Beckmann NDF

The generalized Beckmann NDF is defined as

$$D(\mathbf{h};\mathbf{A}) = \frac{\chi^+(h_z)}{\pi\sqrt{\det(\mathbf{A})}h_z^4}\exp\left(-\left[\frac{h_x}{h_z},\frac{h_y}{h_z}\right]\mathbf{A}^{-1}\left[\frac{h_x}{h_z},\frac{h_y}{h_z}\right]^\top\right).$$

For this NDF, the masking-shadowing function is obtained using the following function:

$$\Lambda(\mathbf{i}) = \frac{1}{2}\left(\mathrm{erf}\left(\frac{1}{k}\right) + \frac{k}{\sqrt{\pi}}\mathrm{e}^{-1/k^2} - 1\right).$$

For the visible normal distribution function (VNDF) of symmetric microflakes, the normalization factor (i.e., projected microflake area) is derived by

$$\begin{aligned}\int_{\mathrm{S}^2} D(\boldsymbol{\omega};\mathbf{A})|\mathbf{i}\cdot\boldsymbol{\omega}|\mathrm{d}\boldsymbol{\omega} &= \int_{\mathrm{S}^2} D(\boldsymbol{\omega};\mathbf{A})\max(\mathbf{i}\cdot\boldsymbol{\omega},0)\mathrm{d}\boldsymbol{\omega} + \int_{\mathrm{S}^2} D(\boldsymbol{\omega};\mathbf{A})\max(-\mathbf{i}\cdot\boldsymbol{\omega},0)\mathrm{d}\boldsymbol{\omega}\\ &= \frac{i_z \mathrm{erfc}\left(-\frac{i_z}{\sqrt{[i_x,i_y]\mathbf{A}[i_x,i_y]^\top}}\right) + \sqrt{\frac{[i_x,i_y]\mathbf{A}[i_x,i_y]^\top}{\pi}}\exp\left(-\frac{1}{k^2}\right)}{2} +\\ &\quad \frac{-i_z \mathrm{erfc}\left(\frac{i_z}{\sqrt{[i_x,i_y]\mathbf{A}[i_x,i_y]^\top}}\right) + \sqrt{\frac{[i_x,i_y]\mathbf{A}[i_x,i_y]^\top}{\pi}}\exp\left(-\frac{1}{k^2}\right)}{2}\\ &= i_z \mathrm{erf}\left(\frac{i_z}{\sqrt{[i_x,i_y]\,\mathbf{A}\,[i_x,i_y]^\top}}\right) + \sqrt{\frac{[i_x,i_y]\,\mathbf{A}\,[i_x,i_y]^\top}{\pi}}\exp\left(-\frac{1}{k^2}\right).\end{aligned}$$

For all other NDF models that have a closed-form Smith $\Lambda(\mathbf{i})$ function, the normalization factor is derived analytically in the same manner.

## 3 Previous Spherical Gaussian Approximations

### 3.1 Anisotropic Spherical Gaussians

An anisotropic spherical Gaussian (ASG) [Xu et al. 2013] is defined as

$$\dot{g}\left(\mathbf{o};\left[\dot{\boldsymbol{\xi}}_x,\dot{\boldsymbol{\xi}}_y,\dot{\boldsymbol{\xi}}_z\right],\left[\tau_x,\tau_y\right]\right) = \max\left(\mathbf{o}\cdot\dot{\boldsymbol{\xi}}_z,0\right)\exp\left(-\tau_x\left(\mathbf{o}\cdot\dot{\boldsymbol{\xi}}_x\right)^2-\tau_y\left(\mathbf{o}\cdot\dot{\boldsymbol{\xi}}_y\right)^2\right),$$

where $\dot{\boldsymbol{\xi}}_x\in\mathrm{S}^2, \dot{\boldsymbol{\xi}}_y\in\mathrm{S}^2, \dot{\boldsymbol{\xi}}_z\in\mathrm{S}^2$ are orthonormal vectors, and $\tau_x\in[0,\infty)$, $\tau_y\in[0,\infty)$ are the sharpness parameters for $\dot{\boldsymbol{\xi}}_x$-axis and $\dot{\boldsymbol{\xi}}_y$-axis, respectively. For glossy SG lighting, Xu et al. [2013] first approximated the NDF with an ASG, and then they warped the ASG lobe to the reflection vectors as follows:

$$f(\mathbf{i},\mathbf{o})|\mathbf{o}\cdot\mathbf{n}| \propto\!\!\propto \dot{g}\left(\mathbf{o};\left[\dot{\boldsymbol{\xi}}_x,\dot{\boldsymbol{\xi}}_y,\dot{\boldsymbol{\xi}}_z\right],\left[\tau_x,\tau_y\right]\right),$$

where the lobe axis $\dot{\boldsymbol{\xi}}_z = 2(\mathbf{i}\cdot\mathbf{n})\mathbf{n}-\mathbf{i}$ is the perfect specular reflection vector. To obtain $\dot{\boldsymbol{\xi}}_x,\dot{\boldsymbol{\xi}}_y,\tau_x,\tau_y$, they introduced a *spherical warping* method for the perfect specular reflection vector $\boldsymbol{\xi}_z$. To convolve the ASG reflection lobe with an isotropic SG light, they also introduced the following approximation:

$$\int_{\mathrm{S}^2}\dot{g}\left(\mathbf{o};\left[\dot{\boldsymbol{\xi}}_x,\dot{\boldsymbol{\xi}}_y,\dot{\boldsymbol{\xi}}_z\right],\left[\tau_x,\tau_y\right]\right)g(\mathbf{o};\boldsymbol{\xi},\kappa)\mathrm{d}\mathbf{o} \propto\!\!\propto \dot{g}\left(\boldsymbol{\xi};\left[\dot{\boldsymbol{\xi}}_x,\dot{\boldsymbol{\xi}}_y,\dot{\boldsymbol{\xi}}_z\right],\left[\frac{2\tau_x\kappa}{2\tau_x+\kappa},\frac{2\tau_y\kappa}{2\tau_y+\kappa}\right]\right).$$

Since the ASG has the clamped cosine term $\max(\boldsymbol{\xi}\cdot\dot{\boldsymbol{\xi}}_z,0)$, the above approximation produces zero for $\boldsymbol{\xi}\cdot\dot{\boldsymbol{\xi}}_z\le 0$ (please see Fig. 6a in the main document). In addition, it cannot represent lower-frequency lobes than the cosine (e.g., uniform distribution). Thus, the ASG approximation can be inaccurate for low-frequency NDFs and low-frequency light distributions. Our glossy SG lighting considers such low-frequency distributions as well as high-frequency distributions.

### 3.2 Product Integral of an SG and a Clamped Cosine

Meder and Brüderlin [2018] approximated the product integral of an SG and a clamped cosine as follows:

$$\int_{S^2} g(\mathbf{o};\boldsymbol{\xi},\kappa)\max(\mathbf{o}\cdot\mathbf{n},0)\mathrm{d}\mathbf{o} \approx a\int_{S^2} H(\mathbf{o}\cdot\mathbf{n})g(\mathbf{o};\boldsymbol{\xi},\kappa)g(\mathbf{o};\mathbf{n},\epsilon)\mathrm{d}\mathbf{o} - b\int_{S^2} H(\mathbf{o}\cdot\mathbf{n})g(\mathbf{o};\boldsymbol{\xi},\kappa)\mathrm{d}\mathbf{o}, \tag{1}$$

where Meder and Brüderlin [2018] obtained constant parameters $a = 32.708$, $b = 31.703$, and $\epsilon = 0.0315$ by non-linear fitting. However, these constants were not optimal in practice. Tokuyoshi [2022] derived $a = be^{\epsilon}$, $b = \epsilon/(2e^{\epsilon}-2-2\epsilon)$, and $\epsilon \longrightarrow 0$. For single-precision floating-point arithmetic, he also derived a near-optimal value $\epsilon = 0.00084560872241480124$ by analyzing the numerical error bound.

The hemispherical integral of an SG is computed using the following interpolation:

$$\int_{S^2} H(\mathbf{o}\cdot\mathbf{n})g(\mathbf{o};\boldsymbol{\xi},\kappa)\mathrm{d}\mathbf{o} = \hat{C}(\kappa)v + \check{C}(\kappa)(1-v), \tag{2}$$

where $v \in [0,1]$ is the interpolation factor, and

$$\hat{C}(\kappa) = \int_{S^2} H(\mathbf{o}\cdot\mathbf{n})g(\mathbf{o};\mathbf{n},\kappa)\mathrm{d}\mathbf{o} = 2\pi\frac{1-e^{-\kappa}}{\kappa},$$
$$\check{C}(\kappa) = \int_{S^2} H(\mathbf{o}\cdot\mathbf{n})g(\mathbf{o};-\mathbf{n},\kappa)\mathrm{d}\mathbf{o} = 2\pi e^{-\kappa}\frac{1-e^{-\kappa}}{\kappa}.$$

For the interpolation factor $v$, Meder and Brüderlin [2018] empirically approximated it with a normalized logistic function:

$$v \approx \frac{e^{t_{\mathrm{logistic}}(\kappa)}e^{t_{\mathrm{logistic}}(\kappa)(\boldsymbol{\xi}\cdot\mathbf{n})}-1}{\left(e^{t_{\mathrm{logistic}}(\kappa)}-1\right)\left(e^{t_{\mathrm{logistic}}(\kappa)(\boldsymbol{\xi}\cdot\mathbf{n})}+1\right)}, \quad \text{where } t_{\mathrm{logistic}} \approx \sqrt{\kappa}\frac{1.6988\kappa^2+10.8438\kappa}{\kappa^2+6.2201\kappa+10.2415}.$$

Wang et al. [2009] also used a logistic function for their SSDF-based visibility. Tokuyoshi [2022] improved the quality by using the error function as follows:

$$v \approx \frac{1}{2} + \frac{\mathrm{erf}(t_{\mathrm{erf}}(\kappa)(\boldsymbol{\xi}\cdot\mathbf{n}))}{2\mathrm{erf}(t_{\mathrm{erf}}(\kappa))}, \quad \text{where } t_{\mathrm{erf}}(\kappa) \approx \kappa\sqrt{\frac{0.5\kappa+0.65173288269070562}{\kappa^2+1.3418280033141288\kappa+7.2216687798956709}}. \tag{3}$$

Eq. 1 requires an SG product and two interpolations to approximate the hemispherical integrals. In addition, the difference of approximated hemispherical integrals can induce a negative value. Our method uses only one interpolation, which is more accurate and guarantees positive results.

## 4 Numerically Stable Implementations for Spherical Gaussian Operators

Numerically stable implementations of SG operators are available as open source [Tokuyoshi 2024]. In this section, we show the implementation details for these existing SG operators.

### 4.1 SG Integral

The integral of an SG is given by the following equation:

$$\int_{S^2} g(\mathbf{o};\boldsymbol{\xi},\kappa)\mathrm{d}\mathbf{o} = 2\pi\frac{1-e^{-2\kappa}}{\kappa} = 4\pi\mathrm{expm1\_over\_x}(-2\kappa), \tag{4}$$

where $\mathrm{expm1\_over\_x}(x) = (e^x-1)/x$. A straightforward implementation of this integral can produce a large numerical error due to floating-point arithmetic (i.e., *catastrophic cancellation*) if the sharpness $\kappa$ is small. Therefore, we use a numerically stable algorithm [Higham 2002] for $\mathrm{expm1\_over\_x}(x)$ shown in Listing 1.

Listing 1. expm1_over_x($x$) = $(e^x - 1)/x$ with cancellation of rounding errors [Higham 2002] (HLSL).

```
float expm1_over_x(float x) {
 float u = exp(x);
 if (u == 1.0) {
  return 1.0;
 }
 float y = u - 1.0;
 if (abs(x) < 1.0) {
  return y / log(u);
 }
 return y / x;
}
```

### 4.2 SG Product

The product of two SGs is given by

$$g(\omega;\xi_1,\kappa_1)g(\omega;\xi_2,\kappa_2) = e^{\kappa_3-\kappa_1-\kappa_2} g\left(\omega;\frac{\kappa_1\xi_1+\kappa_2\xi_2}{\kappa_3},\kappa_3\right),$$

where $\kappa_3 = \|\kappa_1\xi_1 + \kappa_2\xi_2\|$. However, if the sharpness $\kappa_1$ or $\kappa_2$ is large, $\kappa_3 - \kappa_1 - \kappa_2$ can cause catastrophic cancellation. To improve the numerical stability for large $\kappa_1$ or $\kappa_2$, we use the following form:

$$\kappa_3 - \kappa_1 - \kappa_2 = \frac{2\kappa_{\min}((\xi_1\cdot\xi_2)-1)}{1+\frac{\kappa_{\min}}{\kappa_{\max}}+\sqrt{2\frac{\kappa_{\min}}{\kappa_{\max}}(\xi_1\cdot\xi_2)+\left(\frac{\kappa_{\min}}{\kappa_{\max}}\right)^2+1}},$$

where $\kappa_{\min} = \min(\kappa_1,\kappa_2)$ and $\kappa_{\max} = \max(\kappa_1,\kappa_2)$. The HLSL code for this SG product is shown in Listing 2.

Listing 2. Numerically stable SG product (HLSL).

```
struct SGLobe {
 float3 axis;
 float sharpness;
 float logAmplitude;
};

SGLobe sg_product(float3 axis1, float sharpness1, float3 axis2, float sharpness2) {
 float3 axis = axis1 * sharpness1 + axis2 * sharpness2;
 float sharpness = length(axis);
 float cosine = clamp(dot(axis1, axis2), -1.0, 1.0);
 float sharpnessMin = min(sharpness1, sharpness2);
 float sharpnessRatio = sharpnessMin / max(sharpness1, sharpness2);
 float logAmplitude = 2.0 * sharpnessMin * (cosine - 1.0) / (1.0 + sharpnessRatio + sqrt(2.0 * sharpnessRatio * cosine
      + sharpnessRatio * sharpnessRatio + 1.0));
 SGLobe result = { axis / max(sharpness, FLT_MIN), sharpness, logAmplitude };
 return result;
}
```

### 4.3 Hemispherical Integral of an SG

The hemispherical integral of an SG is given by Eq. 2. To improve the numerical stability, we use expm1_over_x($-\kappa$) for $\hat{C}(\kappa)$ and $\check{C}(\kappa)$ as follows:

$$\hat{C}(\kappa) = \int_{S^2} H(\mathbf{o}\cdot\mathbf{n}) g(\mathbf{o};\mathbf{n},\kappa)\mathrm{d}\mathbf{o} = 2\pi\frac{1-\mathrm{e}^{-\kappa}}{\kappa} = 2\pi\mathrm{expm1\_over\_x}(-\kappa),$$
$$\check{C}(\kappa) = \int_{S^2} H(\mathbf{o}\cdot\mathbf{n}) g(\mathbf{o};-\mathbf{n},\kappa)\mathrm{d}\mathbf{o} = 2\pi\mathrm{e}^{-\kappa}\frac{1-\mathrm{e}^{-\kappa}}{\kappa} = 2\pi\mathrm{e}^{-\kappa}\mathrm{expm1\_over\_x}(-\kappa).$$

Since $\hat{C}(\kappa) > 0$ and $\check{C}(\kappa) > 0$, this interpolation does not yield zero. Listing 3 shows an HLSL implementation using the interpolation factor $v$ derived by Tokuyoshi [2022] (Eq. 3).

Listing 3. Numerically stable hemispherical integral of an SG (HLSL).

```
float sg_hemispherical_integral(float cosine, float sharpness) {
 // Interpolation factor [Tokuyoshi 2022].
 float steepness = sharpness * sqrt((0.5 * sharpness + 0.65173288269070562) / ((sharpness + 1.3418280033141288) *
      sharpness + 7.2216687798956709));
 float v = saturate(0.5 + 0.5 * (erf(steepness * clamp(cosine, -1.0, 1.0)) / erf(steepness)));
 // Interpolation between upper and lower hemispherical integrals.
 return 2.0 * M_PI * lerp(exp(-sharpness), 1.0, v) * expm1_over_x(-sharpness);
}
```

### 4.4 Hemispherical Integral of a Normalized SG

The hemispherical integral of a normalized SG (a.k.a., vMF) is obtained from Eq. 4 and Eq. 2 as follows:

$$\frac{\int_{S^2} H(\mathbf{o}\cdot\mathbf{n}) g(\mathbf{o};\xi,\kappa)\mathrm{d}\mathbf{o}}{\int_{S^2} g(\mathbf{o};\xi,\kappa)\mathrm{d}\mathbf{o}} = \frac{v + \mathrm{e}^{-\kappa}(1-v)}{\mathrm{e}^{-\kappa}+1}.$$

Listing 4 shows an HLSL implementation for this hemispherical integral. In this paper, we use this implementation for the visibility term $V$ in our glossy SG lighting approximation.

Listing 4. Numerically stable hemispherical integral of a normalized SG (HLSL).

```
float vmf_hemispherical_integral(float cosine, float sharpness) {
 // Interpolation factor [Tokuyoshi 2022].
 float steepness = sharpness * sqrt((0.5 * sharpness + 0.65173288269070562) / ((sharpness + 1.3418280033141288) *
      sharpness + 7.2216687798956709));
 float v = saturate(0.5 + 0.5 * (erf(steepness * clamp(cosine, -1.0, 1.0)) / erf(steepness)));
 // Interpolation between upper and lower hemispherical integrals.
 float e = exp(-sharpness);
 return lerp(e, 1.0, v) / (e + 1.0);
}
```

## 5 Numerically Stable Implementation for Our SG Lighting

### 5.1 Diffuse SG Lighting

For our diffuse SG lighting, we interpolate the following two product integrals of an SG and a clamped cosine:

$$\hat{B}(\kappa) = \int_{S^2} g(\mathbf{o};\mathbf{n},\kappa)\max(\mathbf{o}\cdot\mathbf{n},0)\mathrm{d}\mathbf{o} = \frac{2\pi\left(\mathrm{e}^{-\kappa}-1+\kappa\right)}{\kappa^2},$$
$$\check{B}(\kappa) = \int_{S^2} g(\mathbf{o};-\mathbf{n},\kappa)\max(\mathbf{o}\cdot\mathbf{n},0)\mathrm{d}\mathbf{o} = \frac{2\pi\mathrm{e}^{-\kappa}\left(1-\mathrm{e}^{-\kappa}-\kappa\mathrm{e}^{-\kappa}\right)}{\kappa^2}.$$

To compute these integrals in a numerically stable manner, we use Listings 5 and 6. In these codes, we use polynomial approximations using the Taylor series to reduce a numerical error due to catastrophic cancellation for a small SG sharpness $\kappa$. A faster approximation is a future work. Listing 7 shows the code for the product integral of an arbitrary SG and the clamped cosine used for our diffuse SG lighting.

Listing 5. Numerically stable implementation for $\hat{B}(\kappa)/(2\pi)$ (HLSL).

```
float upper_sg_clamped_cosine_integral_over_two_pi(float sharpness) {
 if (sharpness <= 0.5) {
  // Taylor-series approximation for numerical stability.
  return (((((((-1.0 / 362880.0) * sharpness + 1.0 / 40320.0) * sharpness - 1.0 / 5040.0) * sharpness + 1.0 / 720.0) *
        sharpness - 1.0 / 120.0) * sharpness + 1.0 / 24.0) * sharpness - 1.0 / 6.0) * sharpness + 0.5;
 }
 return (expm1(-sharpness) + sharpness) / (sharpness * sharpness);
}
```

Listing 6. Numerically stable implementation for $\check{B}(\kappa)/(2\pi)$ (HLSL).

```
float lower_sg_clamped_cosine_integral_over_two_pi(float sharpness) {
 float e = exp(-sharpness);
 if (sharpness <= 0.5) {
  // Taylor-series approximation for numerical stability.
  return e * ((((((((1.0 / 403200.0) * sharpness - 1.0 / 45360.0) * sharpness + 1.0 / 5760.0) * sharpness - 1.0 /
        840.0) * sharpness + 1.0 / 144.0) * sharpness - 1.0 / 30.0) * sharpness + 1.0 / 8.0) * sharpness - 1.0 / 3.0) *
         sharpness + 0.5);
 }
 return e * (-expm1(-sharpness) - sharpness * e) / (sharpness * sharpness);
}
```

Listing 7. Numerically stable implementation for the product integral of an arbitrary SG and a clamped cosine over $\pi$ (HLSL).

```
float sg_clamped_cosine_integral_over_pi(float z, float sharpness) { // z = dot(sg_axis, n).
 // Fitted approximation for t(sharpness).
 float A = 2.736083161127255802824720376520 4;
 float B = 17.021297781741875355455305 31145;
 float C = 4.010082672851042140393929003 0394;
 float D = 15.219156263147210594866010069381;
 float E = 76.087896272360737270901154261082;
 float t = sharpness * sqrt(0.5 * ((sharpness + A) * sharpness + B) / (((sharpness + C) * sharpness + D) * sharpness +
       E));
 float tz = t * z;
 float INV_SQRTPI = 0.564189583547756286948079451560 77;
 float lerpFactor = saturate(0.5 * (z * erfc(-tz) + erfc(t)) - 0.5 * INV_SQRTPI * exp(-tz * tz) * expm1(t * t * (z * z
       - 1.0)) / t);
 // Interpolation between upper and lower hemispherical integrals.
 float lowerIntegral = lower_sg_clamped_cosine_integral_over_two_pi(sharpness);
 float upperIntegral = upper_sg_clamped_cosine_integral_over_two_pi(sharpness);
 return 2.0 * lerp(lowerIntegral, upperIntegral, lerpFactor);
}
```

### 5.2 Roughness Conversion for NDF Filtering

In NDF filtering, we convert the filtered covariance matrix $\bar{\Sigma}_D$ to the roughness matrix $\bar{\mathbf{A}}$ using the following equation:

$$\bar{\mathbf{A}} \approx \left( (2\bar{\Sigma}_D)^{-1} + \mathbf{E} \right)^{-1}. \tag{5}$$

To obtain $(2\bar{\Sigma}_D)^{-1}$, we have to compute $\det(2\bar{\Sigma}_D)$ in a numerically stable manner. This determinant must be positive. However, a straightforward implementation can produce a negative value due to catastrophic cancellation, especially at grazing angles. Therefore, we use the following equation:

$$\begin{aligned}\det\left(2\bar{\Sigma}_D\right) \approx \det\left(2\Sigma_D + \frac{2}{\kappa}\mathbf{J}\mathbf{J}^\top\right) &= \left(\beta_x + \frac{2}{\kappa}\gamma_{11}\right)\left(\beta_y + \frac{2}{\kappa}\gamma_{22}\right) - \frac{4}{\kappa^2}\gamma_{12}\gamma_{21} \\ &= \boxed{\beta_x\beta_y + \frac{2}{\kappa}(\beta_x\gamma_{11} + \beta_y\gamma_{22}) + \frac{4}{\kappa^2}\det\left(\mathbf{J}\mathbf{J}^\top\right)},\end{aligned}$$

where $\gamma_{11} \in [0, \infty)$ and $\gamma_{22} \in [0, \infty)$ are diagonal components of $\mathbf{J}\mathbf{J}^\top = \begin{bmatrix}\gamma_{11} & \gamma_{12} \\ \gamma_{21} & \gamma_{22}\end{bmatrix}$, and $\beta_x \in (0, \infty]$ and $\beta_y \in (0, \infty]$ are diagonal components of $2\Sigma_D$ as follows:

$$2\Sigma_D = \begin{bmatrix}\beta_x & 0 \\ 0 & \beta_y\end{bmatrix} \approx \left(\mathbf{A}^{-1} - \mathbf{E}\right)^{-1} = \begin{bmatrix}\frac{\alpha_x^2}{1-\alpha_x^2} & 0 \\ 0 & \frac{\alpha_y^2}{1-\alpha_y^2}\end{bmatrix}.$$

The determinant of $\mathbf{J}\mathbf{J}^\top$ is given by

$$\det\left(\mathbf{J}\mathbf{J}^\top\right) = \begin{cases}\frac{(\mathbf{h}\cdot\mathbf{n})^2}{16(\mathbf{i}\cdot\mathbf{h})^2} & \text{for reflection} \\ \frac{\eta^4(\mathbf{o}\cdot\mathbf{h})^2(\mathbf{h}\cdot\mathbf{n})^2}{((\mathbf{i}\cdot\mathbf{h})+\eta(\mathbf{o}\cdot\mathbf{h}))^4} & \text{for refraction}\end{cases}.$$

This form yields a positive value in floating points, unlike a straightforward calculation: $\det\left(\mathbf{J}\mathbf{J}^\top\right) = \gamma_{11}\gamma_{22} - \gamma_{12}\gamma_{21}$ that induces catastrophic cancellation. Using this form, we also avoid catastrophic cancellation in Eq. 5 as follows:

$$\bar{\mathbf{A}} \approx \left((2\bar{\Sigma}_D)^{-1} + \mathbf{E}\right)^{-1} = \frac{\operatorname{adj}\left((2\bar{\Sigma}_D)^{-1} + \mathbf{E}\right)}{\det\left((2\bar{\Sigma}_D)^{-1} + \mathbf{E}\right)} = \boxed{\frac{2\bar{\Sigma}_D + \det\left(2\bar{\Sigma}_D\right)\mathbf{E}}{1 + \operatorname{tr}(2\bar{\Sigma}_D) + \det\left(2\bar{\Sigma}_D\right)}}.$$

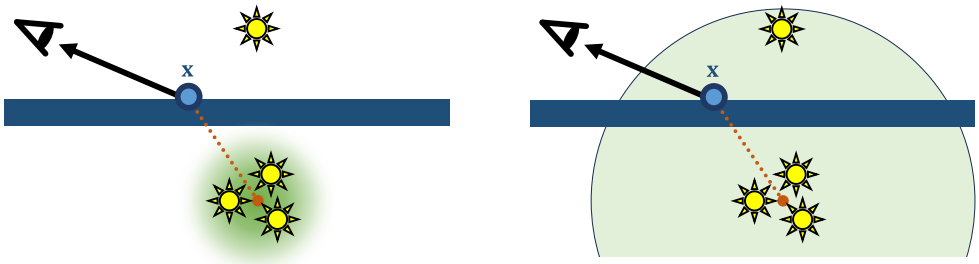
We use the above equation in our implementation.

## 6 Conservative Spatial Variance for Outliers

Although our SG light tree can represent an all-frequency distribution of lights within a cluster in each node, it is a single lobe. Thus, it may underestimate outlier lights in the cluster when most lights are dense and a few are distant from the cluster center. Such outliers are insignificant for the importance approximation if the cluster center is visible to a shading point and most lights are considered. However, if the cluster center is invisible (i.e., below the surface) and only a few outlier lights are visible (i.e., above the surface) as shown in Fig. 1, Gaussian approximation may induce firefly noise (Fig. 2a) due to the underestimation of the outliers. Therefore, to conservatively approximate the importance of such outlier lights, we overstate the spatial variance $\sigma^2$ by using a bounding sphere of the cluster. By assuming that lights are uniformly distributed on a plane, we can obtain $\sigma^2 = 0.5r^2$ using the bounding sphere radius $r \in [0, \infty)$. However, this can increase noise due to the false assumption in most cases (Fig. 2b). Therefore, we blend this bound-based spatial variance $0.5r^2$ and the original spatial variance $\sigma_s^2$ based on the angle of the cluster center $\boldsymbol{\mu}$ viewed from the shading point $\mathbf{x}$ as follows:

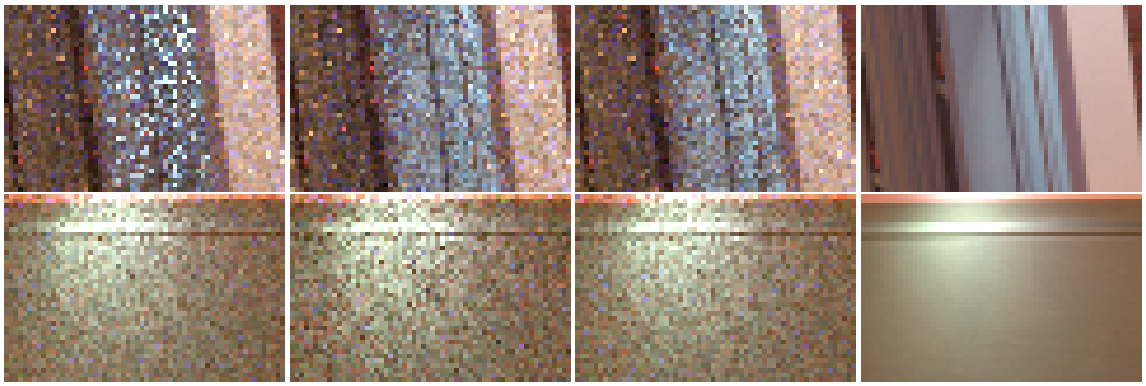
$$\sigma^2 = \sigma_s^2(1 - c) + 0.5r^2c, \tag{6}$$

where $c = \max(\mathbf{n} \cdot (\mathbf{x} - \boldsymbol{\mu})/\|\mathbf{x} - \boldsymbol{\mu}\|, 0)$. With this hybrid approach, we reduce noise for visible outliers (Fig. 2c).




(a) Outlier light above the surface (b) Bounding sphere

Fig. 1. Outliers in Gaussian-based light cluster representation. (a) When only a few outlier lights are visible, the Gaussian-based representation can underestimate the contribution of the outlier lights. (b) To mitigate this underestimation error, we use a conservative spatial variance based on the bounding sphere when the cluster center is below the surface and invisible.



(a) Original (RMSPE: 41.4%) (b) Bound-based (RMSPE: 42.2%) (c) Hybrid (RMSPE: 39.6%) (d) Reference

Fig. 2. Closeups of the Bistro scene (64 spp) rendered using our method with different spatial variances for light clusters. (a) Original spatial variance can induce noise for outlier lights in a cluster (upper row). (b) Bound-based variance is conservative for outliers, while slightly increasing noise in most cases (lower row). (c) Our hybrid approach (Eq. 6) is robust for both cases. For full-size images, please refer to the supplementary images.

## 7 Real-time Demo for Our SG Lighting Visualization

To visualize our SG lighting method, we implemented our SG lighting approximation on top of an existing open-source virtual SG light demo [Tokuyoshi 2024]. As shown in this visualization demo (Fig. 3), our glossy SG lighting takes into account elongated highlights (i.e., anisotropic reflection lobes) created by microfacet BRDFs at grazing angles. We have attached the source code of this visualization demo in the supplementary code. For details, please refer to our example HLSL implementation in the source code.

Fig. 3. Visualization of our SG lighting approximation. Unlike isotropic reflection lobe approximations (e.g., using SPTDs [Dupuy et al. 2017; Liu et al. 2019] or isotropic SGs [Wang et al. 2009]), our glossy SG lighting takes into account elongated highlights created by microfacet BRDFs at grazing angles.

## 8 Additional Experimental Results

Here we show additional experimental results rendered on an AMD Radeon™ RX 7900 XTX GPU. For all the methods, we build light trees using a high-quality sweep surface area orientation heuristic (SAOH) [Conty and Kulla 2018] in preprocessing on an AMD Ryzen™ 9 7950X CPU without performance optimization. For this naïve CPU implementation, the SAOH construction times are 25.1 ms for the Hangarship scene, 92.5 ms for the Museum scene, 1145.2 ms for the Toyshop scene, and 121.431 ms for the Bistro scene. Improving the SAOH construction time is beyond the scope of this paper.

### 8.1 Glossy Scenes

Fig. 4 shows plots of root mean square percentage error (RMSPE) and mean absolute percentage error (MAPE) for additional glossy scenes. For these scenes, our method is more efficient than previous methods.

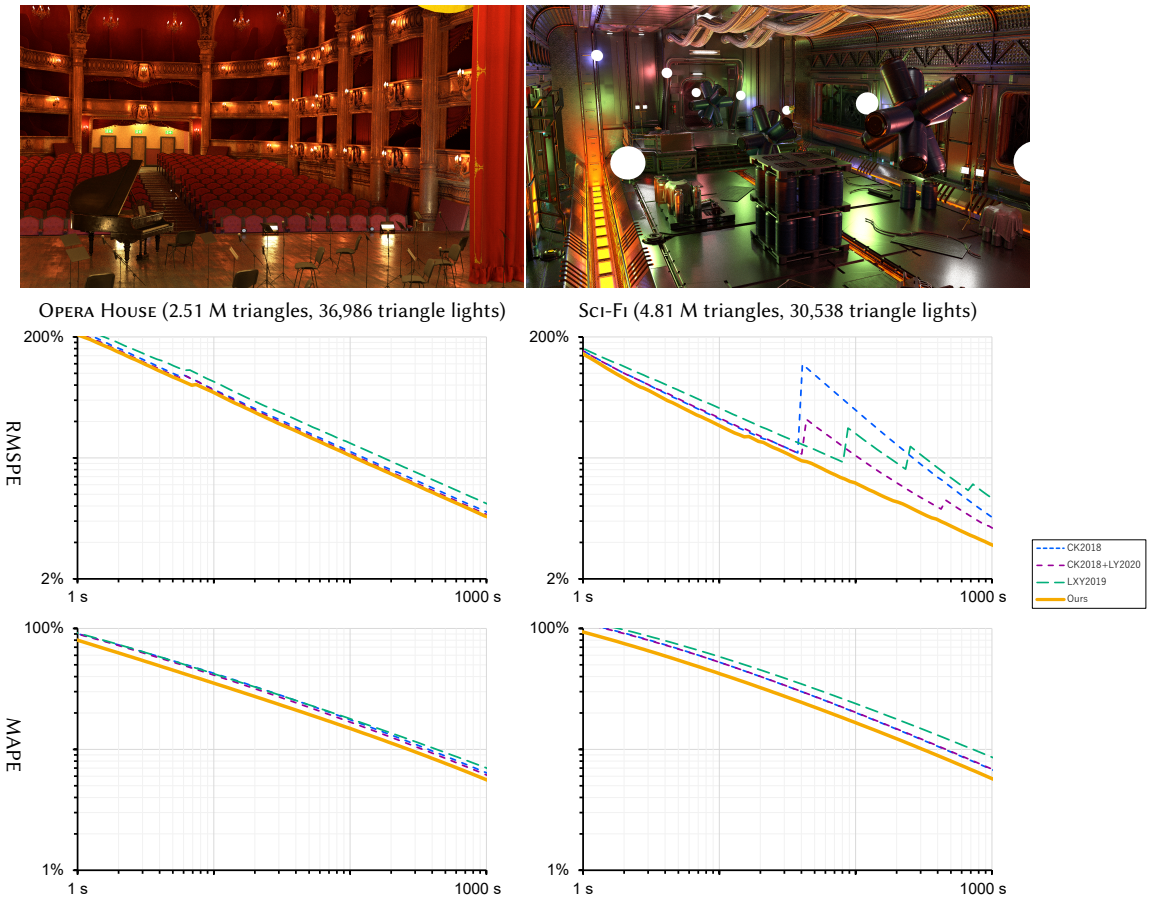



Fig. 4. Path tracing using one light sample per tree-traversal query for glossy scenes (3840×2160 pixels, AMD Radeon™ RX 7900 XTX GPU). For these glossy scenes, our method improves the error convergence speed in the RMSPE and MAPE metrics.

### 8.2 Performance of the Sampling Routines

Table 1 shows the computation times of the sampling routines evaluated by separating the compute kernel. By separating the kernel, our computational overhead is reduced compared to Table 3 in the main document. This is because kernel separation reduces the register pressure on the GPU. Thus, our method can be efficient for GPU rendering algorithms that use multiple kernels [Laine et al. 2013].

Table 1. Computation times of light sampling routines for direct illumination (3840×2160 pixels, 1 spp)

| | CK2018 | CK2018+LY2020 | LXY2019 | Ours |
|---|---|---|---|---|
| Hangarship | 7.3 ms | 8.1 ms | 40.4 ms | 11.3 ms |
| Museum | 9.4 ms | 9.1 ms | 46.2 ms | 13.9 ms |
| Toyshop | 10.5 ms | 11.7 ms | 53.9 ms | 16.0 ms |
| Bistro | 8.6 ms | 9.1 ms | 47.6 ms | 14.5 ms |

### 8.3 Denoising

Fig. 5 shows path tracing using Intel® Open Image Denoise (OIDN) [Áfra 2024] in post-processing. As shown in Fig. 10 in the main document, the previous methods can produce abrupt changes in the amount of noise between pixels. Denoisers such as OIDN can produce visual artifacts for such noise changes, even if input images are unbiased. Our method mitigates the visual artifacts by reducing the abrupt noise changes as well as the noise. To evaluate the quality of denoised images, we use the symmetric mean absolute percentage error (SMAPE) metric in addition to the RMSPE and MAPE metrics. This is because, unlike unbiased rendering, denoisers introduce a brightening bias in dark pixels by blurring pixels. RMSPE and MAPE are sensitive to this brightening bias, while SMAPE is not. For all of these metrics, our method produces smaller errors than the previous methods.

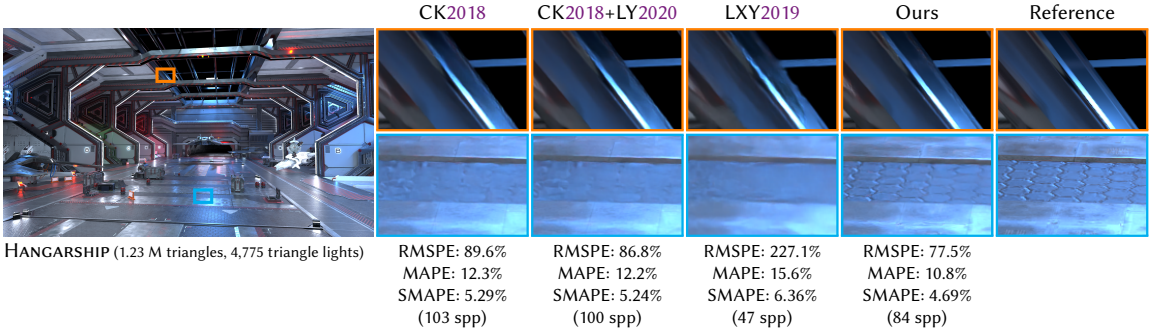



Fig. 5. Denoised images using Intel® Open Image Denoise [Áfra 2024] for path tracing (10 s). The number of path samples per pixel (spp) is the same as Fig. 10 of the main document. Our method not only reduces error, but also reduces visual artifacts (shown in orange boxed closeups) caused by abrupt changes in the amount of noise between pixels.

### 8.4 Adaptive Tree Splitting on the GPU

Fig. 6 shows the rendering results using adaptive tree splitting [Conty and Kulla 2018] on the GPU. For full-size images, please refer to the supplementary images. In this experiment, we use the same splitting threshold as Conty and

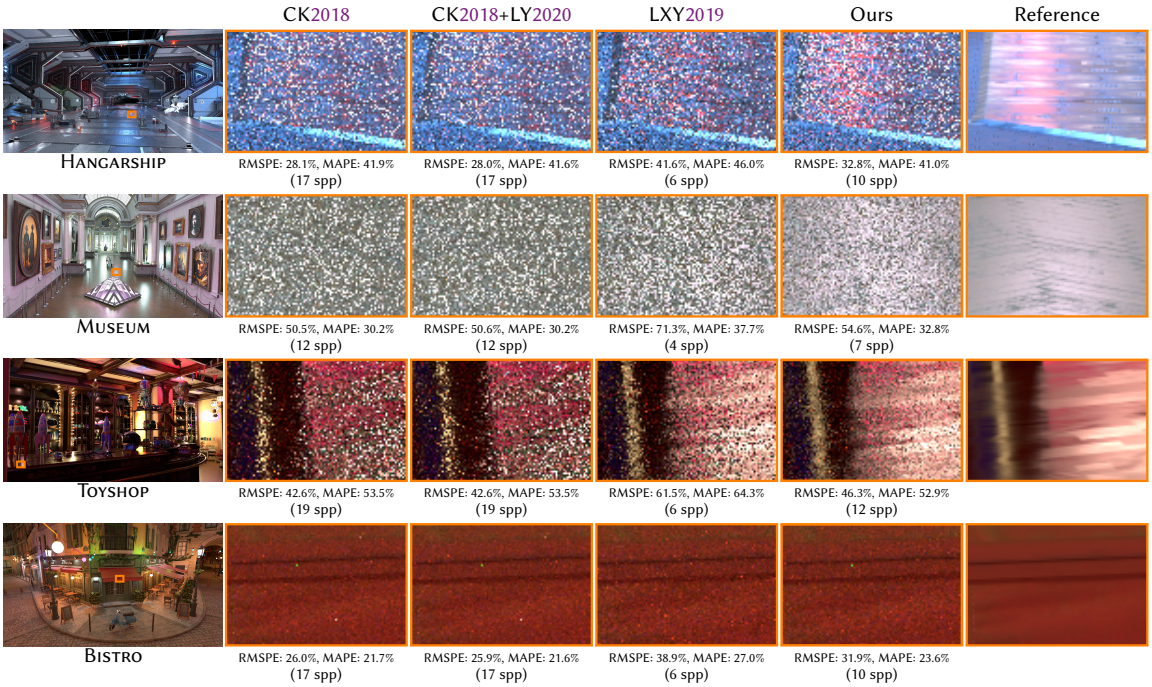



Fig. 6. Equal-time (10 s) comparison of our method and previous methods with adaptive tree splitting on the GPU. Although adaptive tree splitting reduces the effectiveness of our method for glossy surfaces, our method can still reduce the variance for glossy highlights. On the other hand, adaptive tree splitting degrades the performance on the GPU and thus reduces the sampling efficiency. Compared to Fig. 1 and Fig. 10 in the main document, adaptive tree splitting reduces the MAPEs for a diffuse scene (Bistro), while it increases the MAPEs for glossy scenes (Hangarship and Toyshop). The increase in RMSPE is due to firefly noise caused by decreasing indirect illumination path samples.

Kulla [2018]. Unlike the previous CPU implementation, which used an unbounded light list, we limit the light list size to 32 and use reservoir sampling [Vitter 1985] to perform adaptive tree splitting on the GPU. A more sophisticated adaptive tree splitting method that uses two-pass reservoir resampling [Conty et al. 2024] (published after our submission) is a future work. Compared to Fig. 1 and Fig. 10 in the main document, naïve adaptive tree splitting degrades the sampling efficiency on the GPU for glossy scenes. This is because, unlike the CPU implementation, adaptive tree splitting introduces code path divergence on the GPU and increases register pressure or memory overhead due to the light list and a stack for tree traversal. In addition, the splitting method ignores BSDFs and produces abrupt changes in the amount of noise. Adaptive tree splitting prevents our importance from being used in the upper levels of the light tree. Thus, it reduces the effectiveness of our method. Although our method can still reduce the variance for glossy highlights, the performance is degraded due to the increase in register pressure. To mitigate these problems, we should use error bounds that account for glossy BSDFs [Huo et al. 2020; Nabata et al. 2016], and reduce the register pressure (e.g., by separating the compute kernel). In our experimental results, our method without adaptive tree splitting is the most efficient on the GPU for glossy scenes.

## Acknowledgments

We would like to thank ArtcoreStudios for the Museum scene [2022a] and the Opera House scene [2022b], J. Frederick for the Sci-Fi scene [2018], and the Amazon Lumberyard team for the Bistro scene [2017].

## References

Attila T. Áfra. 2024. Intel® Open Image Denoise. https://www.openimagedenoise.org

ArtcoreStudios. 2022a. Museum Environment Kit. https://www.unrealengine.com/marketplace/en-US/product/museum-environment-kit

ArtcoreStudios. 2022b. Opera House Kit. https://www.unrealengine.com/marketplace/en-US/product/opera-house-kit

Alejandro Conty, Pascal Lecocq, and Chris Hellmuth. 2024. A Resampled Tree for Many Lights Rendering. In *SIGGRAPH '24 Talks*. Article 35, 2 pages. https://doi.org/10.1145/3641233.3664352

Estevez Alejandro Conty and Christopher Kulla. 2018. Importance Sampling of Many Lights with Adaptive Tree Splitting. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 2 (2018), 25:1–25:17. https://doi.org/10.1145/3233305

Robert L. Cook and Kenneth E. Torrance. 1982. A Reflectance Model for Computer Graphics. *ACM Trans. Graph.* 1, 1 (1982), 7–24. https://doi.org/10.1145/357290.357293

Jonathan Dupuy, Eric Heitz, and Laurent Belcour. 2017. A spherical cap preserving parameterization for spherical distributions. *ACM Trans. Graph.* 36, 4, Article 139 (2017), 12 pages. https://doi.org/10.1145/3072959.3073694

Jonathon Frederick. 2018. Modular Scifi Season 2 Starter Bundle. https://www.unrealengine.com/marketplace/en-US/product/modular-scifi-season-2-starter-bundle

Eric Heitz. 2014. Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs. *JCGT* 3, 2 (2014), 48–107. http://jcgt.org/published/0003/02/03/

Nicholas J. Higham. 2002. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics.

Yuchi Huo, Shihao Jin, Tao Liu, Wei Hua, Rui Wang, and Hujun Bao. 2020. Spherical Gaussian-based Lightcuts for Glossy Interreflections. *Comput. Graph. Forum* 39, 6 (2020), 192–203. https://doi.org/10.1111/cgf.14011

Samuli Laine, Tero Karras, and Timo Aila. 2013. Megakernels Considered Harmful: Wavefront Path Tracing on GPUs. In *HPG '13*. 137–143. https://doi.org/10.1145/2492045.2492060

Daqi Lin and Cem Yuksel. 2020. Real-Time Stochastic Lightcuts. *Proc. ACM Comput. Graph. Interact. Tech.* 3, 1, Article 5 (2020), 18 pages. https://doi.org/10.1145/3384543

Yifan Liu, Kun Xu, and Ling-Qi Yan. 2019. Adaptive BRDF-Oriented Multiple Importance Sampling of Many Lights. *Comput. Graph. Forum* 38, 4 (2019), 123–133. https://doi.org/10.1111/cgf.13776

Amazon Lumberyard. 2017. Amazon Lumberyard Bistro, Open Research Content Archive (ORCA). http://developer.nvidia.com/orca/amazon-lumberyard-bistro

Julian Meder and Beat Brüderlin. 2018. Hemispherical Gaussians for Accurate Light Integration. In *ICCVG' 18*. 3–15. https://doi.org/10.1007/978-3-030-00692-1_1

Kosuke Nabata, Kei Iwasaki, Yoshinori Dobashi, and Tomoyuki Nishita. 2016. An Error Estimation Framework for Many-Light Rendering. *Comput. Graph. Forum* 35, 7 (2016), 431–439. https://doi.org/10.1111/cgf.13040

Bruce G. Smith. 1967. Geometrical shadowing of a random rough surface. *IEEE Trans. Antennas Propag.* 15, 5 (1967), 668–671. https://doi.org/10.1109/TAP.1967.1138991

Jos Stam. 2001. An Illumination Model for a Skin Layer Bounded by Rough Surfaces. In *EGWR '01*. 39–52. https://doi.org/10.1007/978-3-7091-6242-2_4

Yusuke Tokuyoshi. 2022. Accurate Diffuse Lighting from Spherical Gaussian Lights. In *SIGGRAPH '22 Posters*. Article 35, 2 pages. https://doi.org/10.1145/3532719.3543209

Yusuke Tokuyoshi. 2024. Fast Indirect Illumination Using Two Virtual Spherical Gaussian Lights. https://github.com/yusuketokuyoshi/VSGL

Jeffrey S. Vitter. 1985. Random sampling with a reservoir. *ACM Trans. Math. Softw.* 11, 1 (1985), 37–57. https://doi.org/10.1145/3147.3165

Jiaping Wang, Peiran Ren, Minmin Gong, John Snyder, and Baining Guo. 2009. All-Frequency Rendering of Dynamic, Spatially-Varying Reflectance. *ACM Trans. Graph.* 28, 5 (2009), 133:1–133:10. https://doi.org/10.1145/1618452.1618479

Kun Xu, Wei-Lun Sun, Zhao Dong, Dan-Yong Zhao, Run-Dong Wu, and Shi-Min Hu. 2013. Anisotropic Spherical Gaussians. *ACM Trans. Graph.* 32, 6 (2013), 209:1–209:11. https://doi.org/10.1145/2508363.2508386