Eurographics Symposium on Rendering - Experimental Ideas & Implementations (2018)
T. Hachisuka and W. Jakob (Editors)

Screen Space Approximate Gaussian Hulls

J. Meder[1,2] and B. Brüderlin[1]

[1]TU Ilmenau, Germany
[2]3DInteractive GmbH, Ilmenau, Germany

Abstract

The Screen Space Approximate Gaussian Hull method presented in this paper is based on an output sensitive, adaptive approach, which addresses the challenge of high quality rendering even for high resolution displays and large numbers of light sources or indirect lighting. Our approach uses dynamically sparse sampling of the light information on a low-resolution mesh approximated from screen space and applying these samples in a deferred shading stage to the full resolution image. This preserves geometric detail unlike common approaches using lower resolution rendering combined with upsampling strategies. The light samples are expressed by spherical Gaussian distribution functions, for which we found a more precise closed form integration compared to existing approaches. Thus, our method does not exhibit the quality degradation shown by previously proposed approaches and we show that the implementation is very efficient. Moreover, being an output sensitive approach, it can be used for massive scene rendering without additional cost.

CCS Concepts

•***Computing methodologies*** → ***Rasterization;*** *Reflectance modeling; Virtual reality; Image processing;*

1. Introduction

High-quality lighting is still a significant challenge in interactive rasterization applications, even more so with screen resolutions moving to 4k and beyond. In VR applications this requirement is exacerbated further by high frame rates demanded. But even for consoles and mobile platforms resolution and frame rate demands are steadily growing. A common shortcut we have seen thus far is to carry out some or all per-pixel work on lower resolutions than the true target and using upscaling or splatting, sometimes combined with interpolation or extrapolation to give the impression of the higher target resolution. These approaches can however lose geometric detail which should be visible at the target resolution.

In this paper we present an experimental method and implementation using new means of approximation to allow resolution independency for hemispherical light gathering. Our method does not rely on precalculations or additional a priori information and purely works like a deferred shading approach on a G-Buffer. The basic idea is to approximate the screen space as a low-detail triangle mesh, only gathering light by means of mathematical distributions at the mesh's vertex positions and integrate over these distributions after rasterization of this mesh. In detail, our contributions include

- a fast image segmentation approximation based on connected components
- a quick depth based tessellation of the screen space
- a novel closed form approximation convolving a spherical Gaussian with a cosine factor.

We first cover related work in section 2, while the rest of this paper is organized as follows: In section 3 we detail our screen space tessellation while section 4 concentrates on our work with and application of spherical Gaussians. Finally section 5 gives some empirical data and section 6 concludes this paper.

2. Related Work

Screen space lighting. The method we discuss in this paper does not shade in screen space directly, but uses geometric simplifications of the screen space as part of shading cost reduction. With similar ideas, Nichols et al. proposed a recursive subdivision of the screen space combined with splatting to reduce high rendering costs of many virtual point lights (VPLs) [NW09, NW10]. As inherent to splatting approaches, surface details get lost in the process, which they hide by manipulating the output in dependence of the surface normal. Ritschel et al. extend the popular screen space ambient occlusion method by lighting propagation to approximate global illumination in screen space [RGS09]. As an extension to the single layered conventional screen space, Nalbach et al. introduce the concept of deep screen space [NRS14] as a collection of microsurfaces generated on-the-fly from visible geometry and splat calculated surfel lighting via an approach similar to aforementioned method of Nichols et al.

Tessellating the image space. Image space tessellation like the one we use and further explain in section 3 has mostly seen applications in depth based reprojection or Depth-Image-Based-Rendering

© 2018 The Author(s)
Eurographics Proceedings © 2018 The Eurographics Association.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

DOI: 10.2312/sre.20181177

(DIBR) as it is called by the stereoscopic and free viewpoint synthesis community [Feh03]. One of the earliest work using image space triangle meshes can be traced back to Mark et al., who simply used a full-resolution regular tessellation of the image space [MMB97]. Later works have entered the realm of interactive processing speeds, with Didyk et al. generating pseudo-stereoscopic image pairs by hierarchically subdividing the image space in a multi-pass geometry shader implementation using a measure of pixel displacement and reprojecting each resulting quad [DRE*10]. In contrast, Meder et al. employ single pass hardware tessellation to subdivide an initially coarse mesh in regions of large depth variance to build impostor geometry, allowing for interactive reprojection to arbitrary views [MB16]. Outside the use case of reprojection, Liktor et al. use a regular triangle mesh placed onto objects in light image space and reflected in the local reflection direction to create volumetric caustics [LD10]. Müller et al. apply triangulation to animated fluid point clouds by creating Marching Squares based tessellations over image space renderings of these point clouds to yield a conventionally renderable fluid surface [MSD07].

Spherical Gaussian based methods. Spherical Gaussians have been used by Green et al. to approximate the transport function as a weighted mix [GKMD06]. In other variants, Wang et al. use them as approximations of complex normal distributions [WRG*09] and Xu et al. represent the whole sphere of environmental light during interactive hair rendering [XMR*11] with these distributions. Similar to VPLs, virtual spherical Gaussian lights (VSGLs) are another form of light approximation, which Yan et al. investigate to render translucent objects accurately and efficiently [YZXW12] in their presence. Tokuyoshi converted Reflective Shadow Maps (RSMs) [DS05] to a small set of VSGLs and used them to interactively render convincing diffuse and glossy reflections, even supporting caustics [Tok15]. This work was later extended for more meaningful filtering of the RSM via adaptive kernels [Tok16]. As a near-interactive approach, Xu et al. use spherical Gaussians as representations of bi-directional reflectance distribution functions (BRDFs) paired with a triangle based light integration to model all-frequency interreflections between geometry [XCM*14].

Lighting integration cost reduction. There are numerous ways to reduce the cost of shading in real time computer graphics of which we will focus on two main areas, the first being optimizations at the light level. Advanced culling of lights is explored by Olsson et al. with their propositions on clustered shading [OBA12, OPB15]. Another option is importance sampling, for example used by Dachsbacher et al. in their indirect illumination splatting [DS06] and Ritschel et al. with their bidirectional RSM [REH*11]. Other methods use clustering of present lights like RSM clustering by Prutkin et al. [PKD12] or the work of Nichols et al. with a similar, hierarchical image space clustering [NSW09]. Using a voxel grid as a scene approximation, Crassin et al. first accumulate present scene lighting into this structure and use voxel cone tracing on it with a constant number of cones to compute final lighting [CNS*11]. A comparable voxel based approach is used by Sans et al. [SR13].

In contrast, lighting can also be approximated at the geometry level, with Lensing et al. gathering the light at specific precomputed points on the object geometry in their Lightskin framework [LB13]. They then compute proxy point lights from the gathered shading and interpolate them over the geometry, yielding plausible diffuse but only low-frequency specular lighting in fully dynamic scenes. Jendersie et al. also use such a cache based approach but assume a mostly static scene allowing the precomputation of radiosity values and surface interdependence regarding interreflections for these scene parts [JKG16]. This allows for much faster dynamic lighting of the static scene, but has severe limitations for dynamic objects. Our algorithm mainly fits into this last category of lighting approximation at the geometry level and, like Lensing et al., assumes a fully dynamic scene. Still, combinations with aforementioned complexity reductions at the light level are possible because we make no assumptions on the form of scene light representations.

3. Screen space approximate hulls

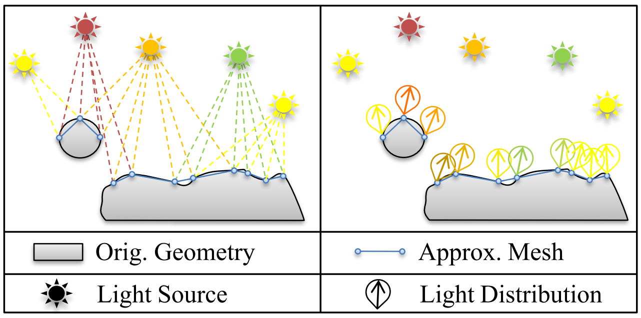


Figure 1: *We approximate scene geometry using a low-detail mesh, where incident lighting from scene lights is only captured at this mesh's vertices by distribution functions.*

As outlined in section 1 we want to model the screen space as a low-poly mesh and only gather hemispherical lighting at this mesh's vertices (see figure 1). No assumptions about the underlying nature of the lighting situation is made and thus our method can be used in conjunction with both direct and indirect lighting. This section will mostly cover the approximate hull construction while the next section 4 will treat the theoretical basis for building and applying the spherical Gaussian distributions we use.

Among others, Lensing et al. [LB13] and Jendersie et al. [JKG16] already established the viability of sparsely computing lighting at discrete caches on the scene geometry with subsequent interpolation. While they placed these caches statically on the object level, we will dynamically place them using the currently visible screen space. We therefore do not need to precalculate and store these caches with the geometry and can use the complete cache budget in the visible space, allowing more detailed light capture. This is particularly important in the use case of massive scene rendering, where precomputations in object space may not be applicable due to time or storage space constraints. Also in contrast to the approach of Lensing et al., we do not use virtual lights as a representation of gathered light but rather mathematical approximations of the incident radiance $L_i(\omega)$ (see equation (6) in section 4), more akin to the method of Jendersie et al.. The main algorithm has three distinct steps:

1. Render a G-Buffer of the scene and downsample screen space depth to 1024x1024

© 2018 The Author(s)
Eurographics Proceedings © 2018 The Eurographics Association.

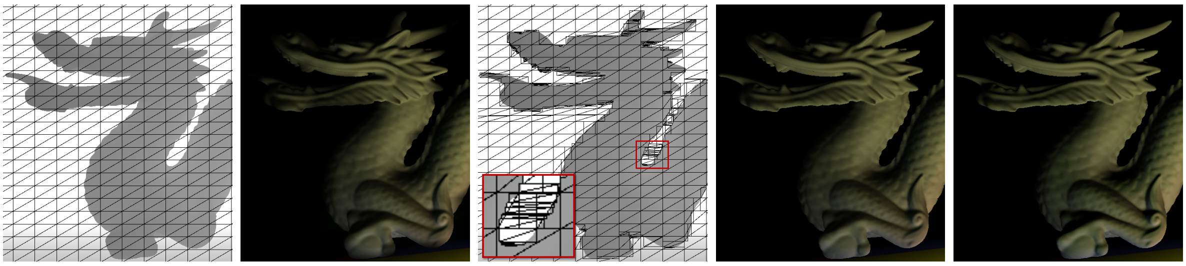


Figure 2: *(a) Our starting tessellation superimposed on the scene depth image. (b) This tessellation leads to wrong shading at depth boundaries (observe dragon muzzle, horns). (c) We identify local depth segments and approximate them by bounding rectangles which can overlap (red: enlarged section). (d) This tessellation allows shading comparable to the (e) ground truth.*

2. Segment the downsampled depth into different regions of similar surfaces separated by depth edges
3. For each segment of the downsampled depth draw an xy-bounding rectangle for light gathering and integration

We will explain the reasoning behind these steps in the following sections in detail.

Approximating the screen space mesh. Our algorithm starts with a low-detail mesh of 32x32 quads of which the vertices are placed regularly in x-y direction of the screen space. For lighting its vertices are assigned a world space position read from the G-Buffer at the corresponding x-y position. As Figure 2b shows this does not suffice to create plausible lighting. Positions inside the triangles and G-Buffer pixel positions can differ substantially in depth edge regions. This is a similar problem Liktor et al. report [LD10]. Their solution however to simply exclude these triangles from rendering is not applicable here, as we want to cover every G-Buffer pixel with a fitting triangle. We therefore opt for mesh subdivision.

We experimented with different subdivision and triangulation strategies. Using the hardware tessellation based depth subdivision of Meder et al. [MB16] yielded good performance in terms of the tessellation itself but unfortunately creates too many vertices in depth edge regions which in turn induce a large overhead for the light gathering. Trying a Delaunay triangulation over depth edge points on the GPU remedied this problem but unfortunately was too slow on our target hardware. We also investigated the approaches of Müller et al. [MSD07] and Didyk et al. [DRE*10] but finally came to the conclusion that we actually don't need the exact triangulations these methods provide: only a mesh covering each G-Buffer pixel, with the respective triangles residing closely to the pixels' surfaces, is needed. The individual mesh parts may even overlap as long as we can choose the correct surface for each G-Buffer pixel in the end.

Local depth segmentation. With this knowledge, we do not really subdivide the individual 32x32 quads. Instead, we use a compute shader to apply local image segmentation to each quad's corresponding depth region of the downsampled depth and create a new quad for each identified segment (see figure 2c). Corresponding to the local pixel count we use 32x32 as the work group size. As a base for the segmentation we use the connected component labeling pro-

Algorithm 1 Local region segmentation of one thread

function SWAPLABEL($x, y, d_c, l, t, depth, label$)
  $d_n \leftarrow depth(x, y)$
  if $|d_c - d_n| > t(1 - d_c)$ **then**
    $l \leftarrow label(x, y)$
  else
    $label(x, y) \leftarrow l$
  end if
  $d_c \leftarrow d_n$
end function

function PROPAGATE($x_s, y_s, x_e, y_e, t, depth, label$)
  $d_c \leftarrow depth(x_s, y_s)$
  $l \leftarrow label(x_s, y_s)$
  for $x' = x_s ... x_e, y' = y_s ... y_e$ **do**
    swapLabel($x', y', d_c, l, t, depth, label$)
  end for
end function

function SEGMENT($x, y, t, depth, label$)
  propagate($0, y, 31, y, t, depth, label$)
  propagate($x, 0, x, 31, t, depth, label$)
  propagate($31, y, 0, y, t, depth, label$)
  propagate($x, 31, x, 0, t, depth, label$)
end function

posed by He et al. [HCS08]. Connected components are identified using a simple depth difference compared to an adaptive threshold (comparison in function "swapLabel" in algorithm 1). The union find structure necessary for label equivalency resolves presented us with problems concerning thread contention and control flow which is why we currently omit this step. We instead propagate the component labels as given with function "segment" in algorithm 1 in right-down-left-up direction over the 32x32 pixel region each using 32 threads running line-wise or column-wise without contention. Of course, this heuristic potentially creates more segments than the original algorithm, which apart from generating additional light gathering overhead does not pose any problems, and we found it to work well in practice.

© 2018 The Author(s)
Eurographics Proceedings © 2018 The Eurographics Association.

Finally, the screen space axis aligned bounding box of each identified segment is calculated: Each of the 32x32 threads takes the final label of its corresponding local pixel as an index in the local bounding box list and applies minimum/maximum atomic operations between its (x, y, z) screen space coordinates and this bounding box's minimum/maximum corners. Due to a limitation of atomic operations the coordinates have to be converted to integer, so we use integer image coordinates for x and y while for the pixel depth we simply use a direct bit conversion. Comparisons will still work under the IEEE 754 floating point definition, assuming all depth values being positive. The resulting bounding boxes are written back to GPU memory for rendering in the final lighting pass. Figure 2d shows a result of this approach.

Light gathering, interpolation and integration. The created mesh is subsequently rendered into the full viewport using the standard rasterization pipeline. Light gathering occurs in the vertex stage, where we accumulate the contributions of all present lights into a spherical Gaussian distribution to approximate the incident radiance $L_i(\omega)$ (see section 4). How the individual light radiance values and directions are obtained depends on the type of light. For the polygonal area lights we use in section 5 we use the radiant exitance of the polygon times its projected area on the unit sphere around the vertex's world position and accumulate all normalized directions from the world position to the polygon's vertices to simulate the directional distribution in the spherical Gaussian.

Additionally in the vertex stage, we fetch the actual world space position of a quad vertex from the G-Buffer at the vertex's x-y position, project it to screen space and compare the result to the quad's screen space bounding box. If it lies outside, meaning the vertex overlaps into another depth region, we correct the world position: we take the intersections of the camera's view direction with the quad's segment bounding box planes and use their mean reprojected to world space.

Perspective-correct interpolation of the distributions' parameters automatically occurs during rasterization of the mesh yielding a distribution per quad fragment. Green et al. already showed such an interpolation of the distribution's parameters over object geometry to be sensible [GKMD06]. Final lighting is calculated by integrating the distribution (see section 4) with respect to the original viewport pixel's surface hemisphere and parameters in the G-Buffer. To avoid applying fragments residing on a quad overlapping from another depth region we again use a test of the G-Buffer pixel's depth against the region bounds of the current fragment, discarding all fragments failing the test.

4. Gaussian lighting integration

Spherical Gaussians. Generally, spherical Gaussians are defined for all normalized directions ω in the whole sphere Ω as

$$G(\mu,\lambda,\phi,\omega) = \mu e^{\lambda(\phi\circ\omega-1)}, \tag{1}$$

with μ being its mean coefficient, λ its sharpness and ϕ its main direction. We use $\circ$ as the dot product in this paper. This distribution has a simple analytical integral over Ω [Tok16]:

$$A(\mu,\lambda) = \int_\Omega G(\mu,\lambda,\phi,\omega)d\omega = \mu\frac{2\pi}{\lambda}(1-e^{-2\lambda}). \tag{2}$$

In section 4 we rely on the multiplication of two spherical Gaussians $G(\mu_1,\lambda_1,\phi_1,\omega)$ and $G(\mu_2,\lambda_2,\phi_2,\omega)$ which yields a new spherical Gaussian $G(\mu_3,\lambda_3,\phi_3,\omega)$ with [XCM*14]

$$\begin{aligned} \lambda_3 &= \|\lambda_1\phi_1+\lambda_2\phi_2\| \\ \phi_3 &= \frac{\lambda_1\phi_1+\lambda_2\phi_2}{\lambda_3} \\ \mu_3 &= \mu_1\mu_2 e^{\lambda_3-\lambda_2-\lambda_1}. \end{aligned} \tag{3}$$

Tokuyoshi proposed to use Toksvig's filtering [Tok05] to approximate a spherical Gaussian's parameters [Tok15]. Due to the inexpensive operations needed we also use this method to accumulate a set of incoming light rays L with light directions ω_l and color intensity values μ_l at a screen space mesh's vertices during hemispherical sampling (see section 3). Direction ϕ and sharpness λ of the spherical Gaussian are estimated as follows:

$$\begin{aligned} \phi' &= \frac{\sum_{l\in L}\mu_l\omega_l}{\sum_{l\in L}\mu_l}, \\ \phi &= \frac{\phi'}{\|\phi'\|}, \\ \lambda &= \frac{\|\phi'\|}{1-\|\phi'\|}. \end{aligned} \tag{4}$$

We use separate distributions for each color channel so μ_l denotes the respective channel value and followingly the corresponding mean coefficient μ is calculated for each channel as

$$\mu = \frac{\sum_{l\in L}\mu_l}{A(\lambda)}, \tag{5}$$

i.e. normalizing the spherical integral (2) by $A(\lambda) = \frac{2\pi}{\lambda}(1-e^{-2\lambda})$.

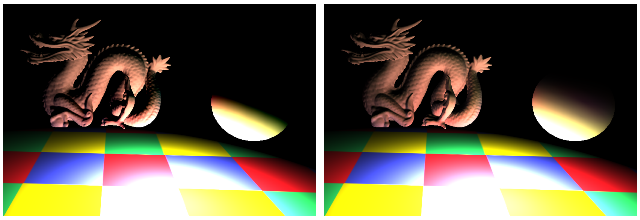
The rendering equation. When the approximate mesh of section 3 is rasterized, the spherical Gaussian distributions associated with its vertices will be automatically interpolated by the GPU to yield a single distribution per generated pixel. We now want to use these to calculate the final lighting of each pixel. In general, to calculate the reflected light of a surface point the rendering equation [Kaj86] has to be solved:

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{\Omega_n^+} L_i(\omega)f_r(x,\omega,\omega_o)(\omega\circ n)d\omega \tag{6}$$

Ignoring self-emittance L_e for the sake of simplicity, we denote the hemisphere above pixel surface point x in direction of the surface normal n, i.e. where $(\omega\circ n)$ is positive, as Ω_n^+. Thus far, we approximated $L_i(\omega)$ which is given by the spherical Gaussian distribution. For the bidirectional reflectance distribution function (BRDF) f_r, we assume, that we can approximate it as another spherical Gaussian, which enables usage of the well-defined multiplication. We can omit this step for the commonly used Lambert diffuse BRDF, as this BRDF is just a constant $\frac{c}{\pi}$, c being the surface's albedo. For other BRDFs meaningful spherical Gaussian approximations have been proposed for Phong distributions and well-known microfacet BRDFs by Wang et al. [WRG*09]. We will not focus any further on this topic as it exceeds the scope of this paper.

Cosine convolution. The remaining problem is to convolve the combined spherical Gaussian with the cosine factor $\omega\circ n$. We found two solutions for this in literature, one of which assumes the cosine

© 2018 The Author(s)
Eurographics Proceedings © 2018 The Eurographics Association.



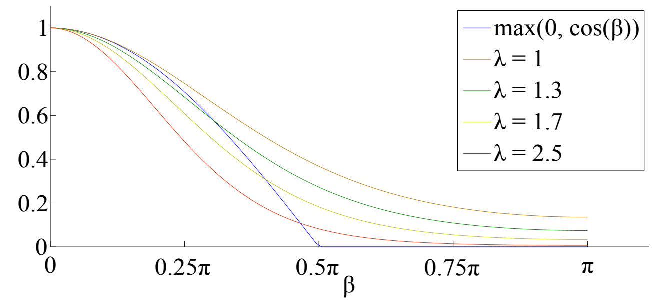
(a) (b)

Figure 3: *(a) Assuming the cosine factor of the rendering integral to be constant leads to noticeably wrong shading compared to (b) the ground truth when this assumption is violated.*

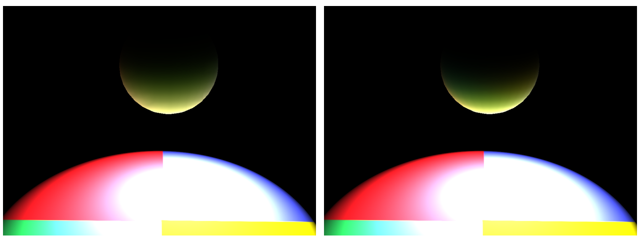
factor to be quasi-constant, which allows its extraction from the integral [Tok16, XCM*14, YZXW12]:

$$\begin{aligned} &\int_{\Omega_n^+} G(\mu,\lambda,\phi,\omega)(\omega\circ n)d\omega \\ &\approx \max(0,\phi\circ n)\int_{\Omega} G(\mu,\lambda,\phi,\omega)d\omega \\ &= \max(0,\phi\circ n)A(\mu,\lambda) \end{aligned} \tag{7}$$

By its underlying assumption, this solution only works for spherical Gaussians of sufficient sharpness. In other words, the spherical Gaussian is interpreted as a directional light with direction ϕ and flux $A(\mu,\lambda)$. A failure case we quickly encountered was the representation of broad area lights which yields noticeably wrong shading when using equation (7) as figure 3 shows.



(a)



(b) (c)

Figure 4: *(a) Accurately approximating a clamped cosine is difficult with a standard spherical Gaussian. (b) The resulting shading has longer tails compared to the (c) ground truth.*

The other solution we encountered is to approximate the cosine factor itself as a spherical Gaussian, which allows to use both the spherical Gaussian multiplication (3) and analytical integral (2):

$$\begin{aligned} &\int_{\Omega_n^+} G(\mu,\lambda,\phi,\omega)(\omega\circ n)d\omega \\ &\approx \int_{\Omega} G(\mu,\lambda,\phi,\omega)G(1,2,n,\omega)d\omega \\ &= \int_{\Omega} G(\mu',\lambda',\phi',\omega)d\omega = A(\mu',\lambda') \end{aligned} \tag{8}$$

Tokuyoshi uses this solution to calculate diffuse lighting under a Lambert BRDF [Tok16]. However, it is impossible to completely reflect the distribution of a clamped cosine factor using approximation (8). As Figure 4 illustrates, trade-offs have to be made leading to overly bright or dark shading depending on the surface normal. Because of these shortcomings, we investigated new ways of calculating the convolution directly.

Direct integration. One observation when trying to solve equation (6) is the necessity to integrate in Ω_n^+ rather than Ω. The previous solutions circumvented this by either clamping the cosine factor or using a cosine-like distribution which always yields positive values. Approaching the cosine convolution directly, we need a formalization for the hemispherical convolution of a spherical Gaussian with a cosine in the cosine's domain of positive values.

We found that two specific hemispherical integrals have easy analytical solutions when integrating in spherical coordinates: the integral in the upper hemisphere, i.e. when n equals ϕ is given by

$$\begin{aligned} A_u(\lambda) &= \int_0^{2\pi}\int_0^{\frac{\pi}{2}} G(\lambda,n,\omega)\cos(\theta)\sin(\theta)d\theta d\varphi \\ &= \frac{2\pi}{\lambda^2}(e^{-\lambda}-1+\lambda), \end{aligned} \tag{9}$$

and the lower hemispherical integral when ϕ equals -n

$$\begin{aligned} A_b(\lambda) &= \int_0^{2\pi}\int_0^{\frac{\pi}{2}} G(\lambda,-n,\omega)\cos(\theta)\sin(\theta)d\theta d\varphi \\ &= \frac{2\pi}{\lambda^2}e^{-2\lambda}(e^{\lambda}-1-\lambda). \end{aligned} \tag{10}$$

where $\omega = \left(\sin(\theta)\cos(\varphi),\sin(\theta)\sin(\varphi),\cos(\theta)\right)^T$. Proof can be found in appendix A. We leave out μ as it is a constant which can be pulled out of the integral. We express the general form of these integrals, when the angle $\beta = \arccos(n\circ\phi)$ is between 0 and π, by rotating the input direction ω away from the original upper hemisphere by this angle. As spherical Gaussians are isotropic we can w.l.o.g. assume a rotation around the local x-axis and thus use a standard rotation matrix

$$M(\beta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & \sin(\beta) \\ 0 & -\sin(\beta) & \cos(\beta) \end{pmatrix}. \tag{11}$$

Assuming further w.l.o.g. that locally ϕ is the z-axis $(0,0,1)^T$ leaves us with the integral

$$\begin{aligned} A_h(\lambda,\beta) &= \int_0^{2\pi}\int_0^{\frac{\pi}{2}} G(\lambda,(0,0,1)^T,M\omega)\cos(\theta)\sin(\theta)d\theta d\varphi \\ &= \int_0^{2\pi}\int_0^{\frac{\pi}{2}} e^{\lambda(\cos(\beta)\cos(\theta)-\sin(\beta)\sin(\phi)\sin(\theta)-1)}\cos(\theta)\sin(\theta)d\theta d\varphi. \end{aligned} \tag{12}$$

© 2018 The Author(s)
Eurographics Proceedings © 2018 The Eurographics Association.

Unfortunately, we did not find an analytical solution for A_h and previous work suggests that indeed none exists [XMR*11]. We therefore investigated the integral numerically.

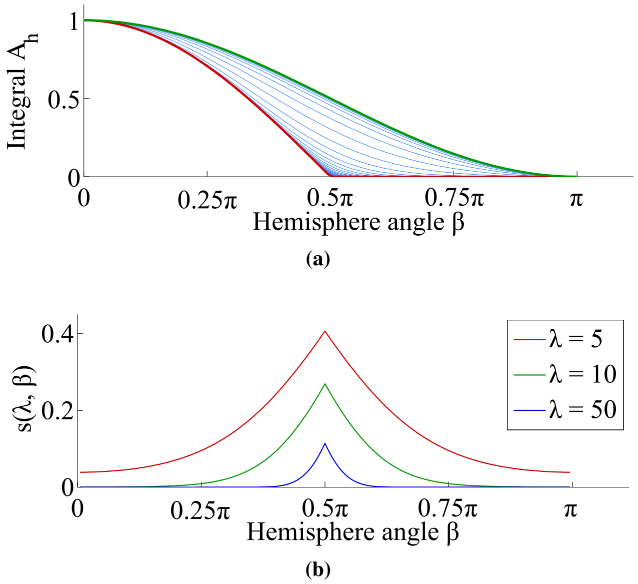


Figure 5: *(a) Normalized curves of A_h for $\lambda \in [10^{-5};700]$ progressively change from $0.5\cos(\beta)+0.5$ to $\max(0,\cos(\beta))$. (b) Curves $s(\lambda,\beta)$ directly calculated from numerically integrated A_h.*

Inspecting the absolute values depending on β for different sharpnesses yielded no valuable hints for us at first, but a quite different picture presented itself when normalizing the curves to an image of $[0;1]$. Observing the curve shapes in 5a, the function becomes equivalent to the normalized cosine

$$\widehat{\cos}(\beta) = 0.5\cos(\beta) + 0.5 \tag{13}$$

when λ approaches zero and progressively turns into the clamped cosine

$$\langle\cos(\beta)\rangle = \max(0, \cos(\beta)) \tag{14}$$

for λ approaching infinity. Given the known analytical values $A_h(\lambda, 0) = A_u(\lambda)$ and $A_h(\lambda,\pi) = A_b(\lambda)$ we approximate A_h generally using a basic interpolation of the form

$$A_h(\lambda,\beta) \approx c(\lambda,\beta)A_u(\lambda) + (1-c(\lambda,\beta))A_b(\lambda) \tag{15}$$

with c being the appropriate normalized curve dependent on λ as seen in figure 5a. For a closed form of c, from our observations we can again use an interpolation, this time of the known cosine curves

$$c(\lambda,\beta) \approx s(\lambda,\beta)\widehat{\cos}(\beta) + (1-s(\lambda,\beta))\langle\cos(\beta)\rangle. \tag{16}$$

Because we actually know the precise curves we can solve (16) for $s(\lambda,\beta)$ using the numerical ground truth of c. Some examples are given in figure 5b.

Now we need a closed form solution for s. We experimented with a range of distributions and came to the conclusion that s could be related to $|\cos(\beta)|$. Additionally, Wang et al. suggested a close relation between the general hemispherical integral of a spherical Gaussian and sigmoid-shaped functions [WRG*09]. We therefore tried a logistic curve [GQ38] of the form

$$s(\lambda,\beta) = \frac{u(\lambda)}{e^{|\cos(\beta)|t(\lambda)}+1}. \tag{17}$$

Using non-linear automatic curve fitting via Matlab this function allowed good fits to the numerical ground truths for s, with root-mean-square errors (RMSEs) ranging from $4.38 \cdot 10^{-11}$ to $3.8 \cdot 10^{-3}$. As for the remaining values of u and t, we first noticed we can use $\frac{1}{u} \approx 0.2958t + 0.5033$. Secondly, the values of t in figure 6 imply some variant of a square root over λ, and we ultimately used

$$t(\lambda) = 2.1007\sqrt{\lambda}\frac{\lambda}{\lambda+4.4653}, \tag{18}$$

with the numerical constants obtained from non-linear curve-fitting via Matlab. With this, we can now closely approximate the hemispherical convolution of a cosine factor with a spherical Gaussian and indeed, compared to the other two methods we discussed in the previous section, our method shows significantly better error values (see figure 7). Also, our new approximation is not necessarily more expensive, as it does not need a spherical Gaussian multiplication, mainly uses linear interpolations and integrals A_u and A_b, largely consisting of the same factors, can be optimized to avoid unnecessary exponentials and divisions.

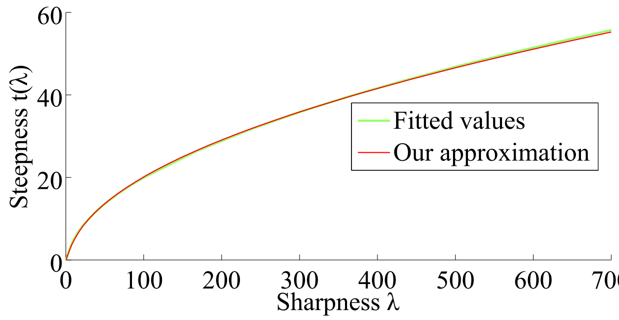


Figure 6: *Values for steepness $t(\lambda)$ for the used logistic curve.*

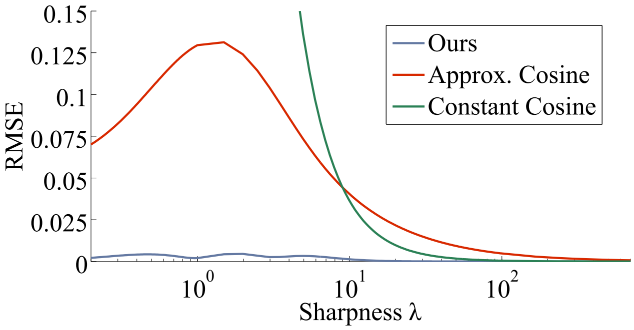


Figure 7: *Our closed form convolution approximation generally shows good results compared to existing methods. The assumption of a constant cosine yields RMSE values up to 4 (curve cropped).*

© 2018 The Author(s)
Eurographics Proceedings © 2018 The Eurographics Association.

Table 1: *Frame rate averages*

Scene		Resolution			
	width	1280	1920	2560	3840
	height	720	1080	1600	2400
Bunny/Sphere/Dragon					
Ground Truth		30.45	11.37	6.27	2.71
Ours		179.34	172.51	130.16	80.48
Crytek Sponza					
Ground Truth		28.38	10.59	6.10	2.44
Ours		142.26	100.36	72.71	40.98
Conference Room					
Ground Truth		28.55	10.67	6.22	2.52
Ours		170.42	145.11	103.10	63.23

5. Evaluation

Setup. To create a use case of complex hemispherical lighting we employ gathering from a 16x16 Reflective Shadow Map (RSM) [DS05]. Each RSM pixel is interpreted as a triangular surface and the polygon-to-point form factor [BRW89] is used for shading. The ground truth shades each G-Buffer pixel using all 256 lights directly while our algorithm applies the same approach to the screen space mesh, with subsequent per-pixel integration. We implemented both renderers using the Vulkan API and run them on an Intel Core i7 860 CPU and a Nvidia GeForce GTX 970 GPU. Frame rates are given in table 1 and rendering examples in figure 8. Supplemental videos show the method running on the respective scenes.

Frame rate. Following the results in table 1 we see good speedups compared to the full resolution gathering. True high frame rate 4k is in reach under such workload. Further Optimizations are possible using existing techniques like bidirectional RSM [REH*11], RSM clustering [PKD12], combinations with Tokuyoshi's VSGL methods [Tok15, Tok16] or clustered shading [OBA12].

Temporal coherence. We obtain mostly stable lighting results under animation, camera and light movement for diffuse lighting. Using the same distributions for specular BRDFs, we obtain results comparable to Lightskin [LB13], i.e. temporally coherent but only low frequency reflections due to the single, broad spherical Gaussian. Using a separate narrow specular spherical Gaussian allowed for sharper reflections than Lightskin due to our more fine grained cache placement. However, this can induce strong temporal flickering on surfaces with high normal variance. Finally, typical depth inaccuracies negatively impacting the position correction are visible as varying lighting, for example on the Sponza scene's curtains.

Energy conservation. Because we ignore pixel positions for the final lighting integration, we will not see a problem that Lensing et al. faced with their method [LB13], where light interpolation could cause the proxy-light to get too close to the surface and thus wrongly create brighter lighting. This means we have to ascertain that G-Buffer pixel positions reside closely to the approximate mesh's triangles, which is implied by our depth segmentation.

Artifacts. Due to the depth downsampling, entire segments may get lost in the process, noticeable as strong, mostly one pixel wide differences in figure 8. Affected pixels mainly receive incorrect lighting from their surrounding neighbors, but sometimes none at all. In the latter case, all quad-fragments are discarded by the segment bounds test in the fragment stage. Furthermore, screen space depth inaccuracies can cause the vertex stage bounding test to fail, wrongfully applying position correction to non-overlapping vertices and in turn causing suboptimal shading. This case is visible in figure 8 with small/medium differences plus noticeable difference edges.

Applicability. Any source providing hemispherical lighting may be used to gather, such as global illumination focused approaches beside RSM like voxel cone tracing [CNS*11]. Accelerating direct sources like image based lighting or discrete sets of analytical lights is also possible. Even ray tracing or approximations like screen space reflections are imaginable for glossy lighting. Highly specular reflections pose a limitation of our approach. The screen space mesh would need a tessellation to the G-Buffer pixel level in the worst case. Depending on the respective application's accuracy requirements, more individual distributions are needed in general.

6. Conclusion and future work

In this paper, we have investigated a new variant of resolution decoupled deferred shading retaining the original resolution and have found a higher quality use of spherical Gaussians. The technique is well-suited for alleviating traditionally pixel-bound operations without a priori knowledge of the scene and is applicable both to direct and indirect lighting methods. Its output-sensitivity and non-reliance on precalculations make it particularly suitable for massive scene rendering. Our resulting experimental implementation shows promising results concerning speed and quality.

To solve the artifacts of our segmentation heuristic we plan to incorporate a more sophisticated depth downsampling, either preserving all local segments or giving depth bounds approximations, and to modify our segmentation heuristic to give a minimal local segmentation. Moving the algorithm entirely to world space and replacing the bounding box with a plane approximation promises better accuracy. Observing the improvements when using available RSM optimizations is also interesting. We further like to evaluate our method with other techniques for light gathering.

Finding precise closed form solutions or approximations using other BRDFs or replacing our distributions with anisotropic spherical Gaussians [XSD*13] present additional research opportunities.

Acknowledgments

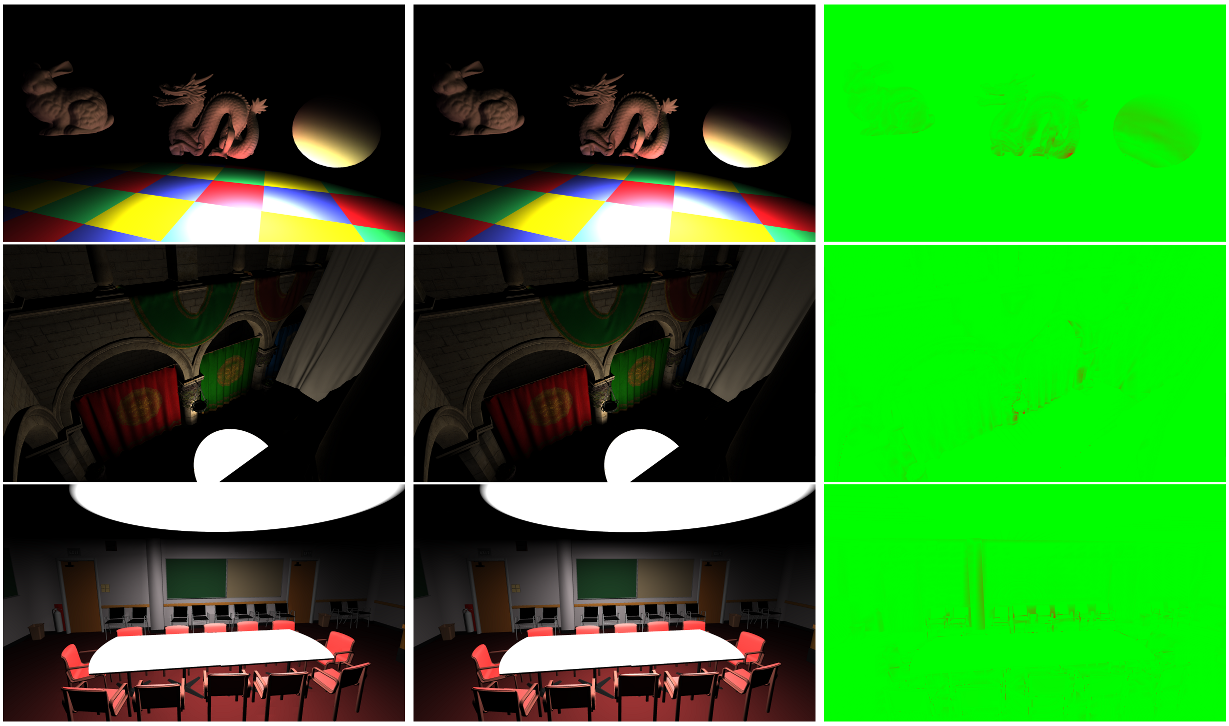
Used scenes are courtesy of the Standford 3D Scanning Repository (Bunny, Dragon, `http:graphics.stanford.edu/data/3Dscanrep/`), Crytek (Sponza, `http:crytek.com/cryengine/cryengine3/downloads/`) and A. Grynberg and G. Ward. (Conference Room, `http:radsite.lbl.gov/radiance/pub/models/conf.tar.Z`). We thank the anonymous reviewers for their valuable feedback in improving this paper.

Appendix A: Hemispherical convolution

We start with expression (9) given in section 4 and can immediately solve the outer integral and expand G:

$$A_u(\lambda) = 2\pi \int_0^{\frac{\pi}{2}} e^{\lambda(\cos(\theta)-1)} \cos(\theta)\sin(\theta)d\theta \qquad (19)$$

© 2018 The Author(s)
Eurographics Proceedings © 2018 The Eurographics Association.



(a) **(b)** **(c)**

Figure 8: *Rendering results for the Bunny/Sphere/Dragon, the Crytek Sponza and the Conference Room scene: (a) our results and (b) the ground truth. (c) Differences artificially increased by a factor of 4 and plotted as false color images (green: low, red: high).*

Substituting $t = \cos(\theta)$ yields

$$A_u(\lambda) = 2\pi \int_0^{\frac{\pi}{2}} -e^{\lambda(t-1)} t dt. \tag{20}$$

Pulling out $\frac{1}{\lambda^2}$ and expanding the sum by $e^{\lambda(t-1)}\lambda$ gives

$$A_u(\lambda) = \frac{2\pi}{\lambda^2}\left(\int_0^{\frac{\pi}{2}} e^{\lambda(t-1)}\lambda dt - \int_0^{\frac{\pi}{2}} e^{\lambda(t-1)}\lambda^2 t + e^{\lambda(t-1)}\lambda dt\right), \tag{21}$$

of which we can integrate the first integral directly, the second via the inverse product rule and finally resubstitute $\cos(\theta)$:

$$\begin{aligned} A_u(\lambda) &= \frac{2\pi}{\lambda^2}[e^{\lambda(t-1)} - e^{\lambda(t-1)}\lambda t]_0^{\frac{\pi}{2}} \\ &= \frac{2\pi}{\lambda^2}[e^{\lambda(\cos(\theta)-1)}(1-\lambda\cos(\theta))]_0^{\frac{\pi}{2}}. \end{aligned} \tag{22}$$

Thus, the integral value resolves to

$$A_u(\lambda) = \frac{2\pi}{\lambda^2}(e^{-\lambda} - 1 + \lambda) \tag{23}$$

A_b from equation (10) can be integrated analogously.

References

[BRW89] Baum D. R., Rushmeire H. E., Winget J.: Improving radiosity solutions through the use of analytically determined form-factors. In *ACM Siggraph Computer Graphics* (07 1989), vol. 23, pp. 325–334. 7

[CNS*11] Crassin C., Neyret F., Sainz M., Green S., Eisemann E.: Interactive indirect illumination using voxel cone tracing: A preview. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, ACM, pp. 207–207. URL: `http://doi.acm.org/10.1145/1944745.1944787`, `doi:10.1145/1944745.1944787`. 2, 7

[DRE*10] Didyk P., Ritschel T., Eisemann E., Myszkowski K., Seidel H.-P.: Adaptive image-space stereo view synthesis. In *Vision, Modeling and Visualization Workshop* (Siegen, Germany, 2010), pp. 299–306. 2, 3

[DS05] Dachsbacher C., Stamminger M.: Reflective shadow maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2005), I3D '05, ACM, pp. 203–231. URL: `http://doi.acm.org/10.1145/1053427.1053460`, `doi:10.1145/1053427.1053460`. 2, 7

[DS06] Dachsbacher C., Stamminger M.: Splatting indirect illumination. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2006), I3D '06, ACM, pp. 93–100. URL: `http://doi.acm.org/10.1145/1111411.1111428`, `doi:10.1145/1111411.1111428`. 2

[Feh03] Fehn C.: A 3d-tv system based on video plus depth information. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on* (2003), vol. 2, pp. 1529–1533. `doi:10.1109/ACSSC.2003.1292241`. 2

[GKMD06] Green P., Kautz J., Matusik W., Durand F.: View-dependent precomputed light transport using nonlinear gaussian function approximations. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2006), I3D '06, ACM, pp. 7–14. URL: `http://doi.acm.org/10.1145/1111411.1111413`, `doi:10.1145/1111411.1111413`. 2, 4

[GQ38] GARNIER J., QUETELET A.: *Correspondance mathématique et physique*. No. 10. Impr. d'H. Vandekerckhove, 1838. URL: https://books.google.de/books?id=8GsEAAAAYAAJ. 6

[HCS08] HE L., CHAO Y., SUZUKI K.: A run-based two-scan labeling algorithm. *IEEE Transactions on Image Processing 17*, 5 (May 2008), 749–756. doi:10.1109/TIP.2008.919369. 3

[JKG16] JENDERSIE J., KURI D., GROSCH T.: Precomputed illuminance composition for real-time global illumination. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2016), I3D '16, ACM, pp. 129–137. URL: http://doi.acm.org/10.1145/2856400.2856407, doi:10.1145/2856400.2856407. 2

[Kaj86] KAJIYA J. T.: The rendering equation. *SIGGRAPH Comput. Graph. 20*, 4 (Aug. 1986), 143–150. URL: http://doi.acm.org/10.1145/15886.15902, doi:10.1145/15886.15902. 4

[LB13] LENSING P., BROLL W.: Lightskin: Real-time global illumination for virtual and mixed reality. In *Proceedings of the 5th Joint Virtual Reality Conference* (Aire-la-Ville, Switzerland, Switzerland, 2013), JVRC '13, Eurographics Association, pp. 17–24. URL: http://dx.doi.org/10.2312/EGVE.JVRC13.017-024, doi:10.2312/EGVE.JVRC13.017-024. 2, 7

[LD10] LIKTOR G., DACHSBACHER C.: Real-time volumetric caustics with projected light beams. In *Graphics and Geometry* (2010), pp. 151–158. 2, 3

[MB16] MEDER J., BRÜDERLIN B.: Decoupling rendering and display using fast depth image based rendering on the gpu. In *Computer Vision and Graphics: International Conference, ICCVG 2016, Warsaw, Poland, September 19-21, 2016, Proceedings* (2016), pp. 61–72. URL: https://doi.org/10.1007/978-3-319-46418-3_6, doi:10.1007/978-3-319-46418-3_6. 2, 3

[MMB97] MARK W. R., MCMILLAN L., BISHOP G.: Post-rendering 3d warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics* (New York, NY, USA, 1997), I3D '97, ACM, pp. 7–ff. doi:10.1145/253284.253292. 2

[MSD07] MÜLLER M., SCHIRM S., DUTHALER S.: Screen space meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), SCA '07, Eurographics Association, pp. 9–15. URL: http://dl.acm.org/citation.cfm?id=1272690.1272692. 2, 3

[NRS14] NALBACH O., RITSCHEL T., SEIDEL H.-P.: Deep screen space. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2014), I3D '14, ACM, pp. 79–86. URL: http://doi.acm.org/10.1145/2556700.2556708, doi:10.1145/2556700.2556708. 1

[NSW09] NICHOLS G., SHOPF J., WYMAN C.: Hierarchical image-space radiosity for interactive global illumination. In *Proceedings of the Twentieth Eurographics Conference on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2009), EGSR'09, Eurographics Association, pp. 1141–1149. URL: http://dx.doi.org/10.1111/j.1467-8659.2009.01491.x, doi:10.1111/j.1467-8659.2009.01491.x. 2

[NW09] NICHOLS G., WYMAN C.: Multiresolution splatting for indirect illumination. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 83–90. URL: http://doi.acm.org/10.1145/1507149.1507162, doi:10.1145/1507149.1507162. 1

[NW10] NICHOLS G., WYMAN C.: Interactive indirect illumination using adaptive multiresolution splatting. *IEEE Transactions on Visualization and Computer Graphics 16*, 5 (Sept 2010), 729–741. doi:10.1109/TVCG.2009.97. 1

[OBA12] OLSSON O., BILLETER M., ASSARSSON U.: Clustered deferred and forward shading. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics* (Goslar Germany, Germany, 2012), EGGH-HPG'12, Eurographics Association, pp. 87–96. URL: https://doi.org/10.2312/EGGH/HPG12/087-096, doi:10.2312/EGGH/HPG12/087-096. 2, 7

[OPB15] OLSSON O., PERSSON E., BILLETER M.: Real-time many-light management and shadows with clustered shading. In *ACM SIGGRAPH 2015 Courses* (New York, NY, USA, 2015), SIGGRAPH '15, ACM, pp. 12:1–12:398. URL: http://doi.acm.org/10.1145/2776880.2792712, doi:10.1145/2776880.2792712. 2

[PKD12] PRUTKIN R., KAPLANYAN A., DACHSBACHER C.: Reflective Shadow Map Clustering for Real-Time Global Illumination. In *Eurographics 2012 - Short Papers* (2012), Andujar C., Puppo E., (Eds.), The Eurographics Association. doi:10.2312/conf/EG2012/short/009-012. 2, 7

[REH*11] RITSCHEL T., EISEMANN E., HA I., DOKYOON KIM J., SEIDEL H.-P.: Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes. 2258–2269. 2, 7

[RGS09] RITSCHEL T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 75–82. URL: http://doi.acm.org/10.1145/1507149.1507161, doi:10.1145/1507149.1507161. 1

[SR13] SANS F., RAMÍREZ J. E.: Real-time diffuse global illumination based on voxelization. In *2013 XXXIX Latin American Computing Conference (CLEI)* (Oct 2013), pp. 1–12. doi:10.1109/CLEI.2013.6670656. 2

[Tok05] TOKSVIG M.: Mipmapping normal maps. *Journal of Graphics Tools 10*, 3 (2005), 65–71. URL: https://doi.org/10.1080/2151237X.2005.10129203, arXiv:https://doi.org/10.1080/2151237X.2005.10129203, doi:10.1080/2151237X.2005.10129203. 4

[Tok15] TOKUYOSHI Y.: Virtual spherical gaussian lights for real-time glossy indirect illumination. *Computer Graphics Forum 34*, 7 (2015), 89–98. URL: http://dx.doi.org/10.1111/cgf.12748, doi:10.1111/cgf.12748. 2, 4, 7

[Tok16] TOKUYOSHI Y.: Modified filtered importance sampling for virtual spherical gaussian lights. *Computational Visual Media 2*, 4 (Dec 2016), 343–355. URL: https://doi.org/10.1007/s41095-016-0063-3, doi:10.1007/s41095-016-0063-3. 2, 4, 5, 7

[WRG*09] WANG J., REN P., GONG M., SNYDER J., GUO B.: All-frequency rendering of dynamic, spatially-varying reflectance. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 133:1–133:10. URL: http://doi.acm.org/10.1145/1618452.1618479, doi:10.1145/1618452.1618479. 2, 4, 6

[XCM*14] XU K., CAO Y.-P., MA L.-Q., DONG Z., WANG R., HU S.-M.: A practical algorithm for rendering interreflections with all-frequency brdfs. *ACM Trans. Graph. 33*, 1 (Feb. 2014), 10:1–10:16. URL: http://doi.acm.org/10.1145/2533687, doi:10.1145/2533687. 2, 4, 5

[XMR*11] XU K., MA L.-Q., REN B., WANG R., HU S.-M.: Interactive hair rendering and appearance editing under environment lighting. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 173:1–173:10. URL: http://doi.acm.org/10.1145/2070781.2024207, doi:10.1145/2070781.2024207. 2, 6

[XSD*13] XU K., SUN W.-L., DONG Z., ZHAO D.-Y., WU R.-D., HU S.-M.: Anisotropic spherical gaussians. *ACM Trans. Graph. 32*, 6 (Nov. 2013), 209:1–209:11. URL: http://doi.acm.org/10.1145/2508363.2508386, doi:10.1145/2508363.2508386. 7

[YZXW12] YAN L.-Q., ZHOU Y., XU K., WANG R.: Accurate translucent material rendering under spherical gaussian lights. *Comput. Graph. Forum 31*, 7pt2 (Sept. 2012), 2267–2276. URL: http://dx.doi.org/10.1111/j.1467-8659.2012.03220.x, doi:10.1111/j.1467-8659.2012.03220.x. 2, 5

© 2018 The Author(s)
Eurographics Proceedings © 2018 The Eurographics Association.