Precomputed Illuminance Composition for Real-Time Global Illumination

Johannes Jendersie[1] David Kuri[1,2] Thorsten Grosch[3]
[1]Otto-von-Guericke University Magdeburg [2]Volkswagen AG [3]TU Clausthal

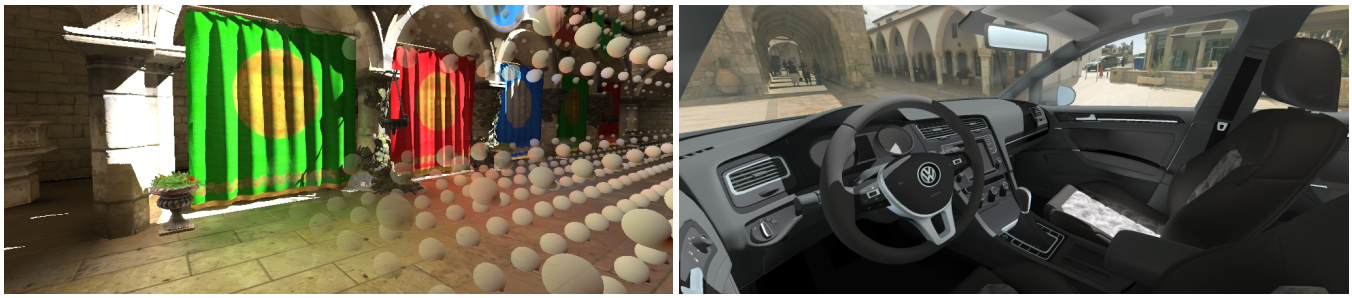


Figure 1: *The Crytek Sponza[1] (5.7ms, 245k triangles, 10k caches) with a visualization of the caches and a Volkswagen test data set using an environment map from Persson[2] (7.5ms, 1.35M triangles, 11k caches) with multiple bounce indirect illumination and slightly glossy surfaces.*

Abstract

In this paper we present a new real-time approach for indirect global illumination under dynamic lighting conditions. We use surfels to gather a sampling of the local illumination and propagate the light through the scene using a hierarchy and a set of precomputed light transport paths. The light is then aggregated into caches for lighting of static and dynamic geometry. By using a spherical harmonics representation, caches preserve incident light directions to allow both diffuse and slightly glossy BRDFs for indirect lighting.

The sparse sampling of direct light also enables indirect lighting from many light sources and efficient progressive multi-bounce. Furthermore, any existing pipeline can be used for surfel lighting, enabling the use of all kinds of light sources. This also includes area lights which can be computed together with the first bounce indirect illumination of other light sources at no additional cost.

Keywords: global illumination, real-time, spherical harmonics, caches, surfels

Concepts: •**Computing methodologies** → *Reflectance modeling;*

1 Introduction

The simulation of light, including the effects of bounced light known as global illumination, is crucial for the generation of photorealistic imagery. The concepts of light transport for the purpose of rendering are well understood but expensive to calculate. Especially for interactive applications the computational cost is prohibitive. Both precomputation and approximation are viable methods to achieve considerable speedups.

Current illumination algorithms often solve this problem at the expense of computation time and memory consumption. We propose a new algorithm which achieves high real-time frame rates at comparably low memory consumption and supports all kinds of dynamic light sources. Except for the comparably fast per pixel evaluation, the approach is resolution independent.

To achieve competitive performance, our algorithm relies on the precomputation of light transport factors in the form of spherical harmonics (SH). The lighting in the scene is sampled using surfels, a finite set of discs representing the scene's static surfaces. Then light is conveyed through precomputed links and accumulated in caches placed in a grid or light map. Dynamic and highly detailed objects can be shaded using the grid of light caches. Using a light map for static geometry increases the quality on planar surfaces and saves memory in empty regions.

DOI: http://dx.doi.org/10.1145/2856400.2856407.

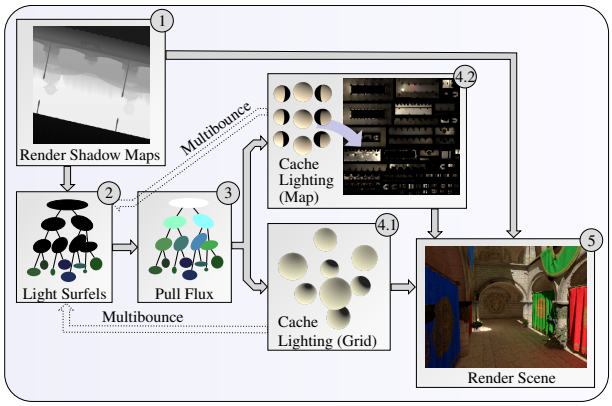
2 Related Work

Global Illumination and Radiosity Approaches: The problem of global illumination can be dated back to the formulation of the rendering equation [Kajiya 1986]. Radiosity [Goral et al. 1984] is a finite element method that reduces the complexity of global illumination calculation for environments containing only Lambertian diffuse surfaces. It has been extended to account for glossy and mirror-like reflections at little additional cost, but with artifacts due to discretization of directions [Immel et al. 1986]. Hierarchical radiosity [Hanrahan et al. 1991] allows the illumination of large scenes and is closely related to the proposed algorithm. Instant radiosity [Keller 1997] is a GPU-friendly technique that approximates radiosity using only point light sources, yielding interactive frame rates. [Laine et al. 2007] describe a technique to reuse point lights and incrementally maintain a good distribution. They achieve real-time frame rates for single-bounce indirect illumination.

Precomputed radiance transfer: PRT [Sloan et al. 2002] can be used to simulate diffuse and glossy global illumination from a light environment at infinite distance. Similar to the proposed algorithm, PRT uses a precomputation step, allowing for lighting updates in real-time. Local light sources can be handled by using unstructured light clouds [Kristensen et al. 2005].

Cache-based Approaches: Because Lambertian diffuse surfaces exhibit isotropic luminance, indirect diffuse illumination from static light sources can be precalculated and stored in textures known as light maps. The technique was extended to account for small-scale surface details from normal maps [Mitchell et al. 2006]. Lighting can also be precalculated for discrete points in 3D space. [Greger et al. 1998] store spherical irradiance information in a so-called irradiance volume. Compressed Radiance Caching (CRC) reduces the memory overhead due to sparse volume allocation and chrominance compression in YCoCg color space [Vardis

[1]Meinl, F. Crytek Sponza. On *www.crytek.com*
[2]Persson, E. Saint Lazarus Church. CC BY 3.0, On *www.humus.name*



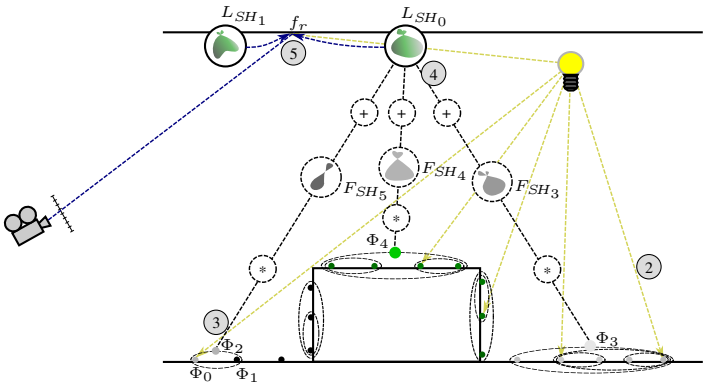


Figure 2: ***Pipeline and light transport overview***. *After creating standard shadow maps (1), surfels are lit directly (2). The flux per sender Φ_s is collected from the clustered surfels (3). Then the light is transported to the caches by precomputed links (4). There are caches within a map and a grid for different geometry. Finally, neighbored caches are interpolated to sample indirect illumination at a surface point (5) using its BRDF f_r. The same sampling is possible at surfel locations to add a progressive multibounce. The schematic overview shows a selection of direct light paths and links with "form factors" F_{SH}. Two caches at the ceiling containing incident radiance L_{SH} are lit through their links.*

et al. 2014]. [Lensing and Broll 2013] use a sparse cache distribution on surfaces to reduce memory and computational overhead in their LightSkin framework.

Real-time Approaches: Recently, various dynamic real-time global illumination techniques have been published. [Dong et al. 2009] suggested using virtual area lights for fast rendering of global illumination using the graphics hardware. Cascaded light propagation volumes (C-LPV) [Kaplanyan and Dachsbacher 2010] is a method for (mainly) single-bounce indirect illumination in real-time. Using C-LPV, light is discretized in a voxel volume and exchanged between adjacent cells iteratively. Geomerics developed a real-time global illumination middleware called Enlighten [Martin and Einarsson 2010] that presumably works similar to the technique proposed in this paper. Unfortunately, the publicly available details of Enlighten's inner workings are sparse. [Thiedemann et al. 2011] suggested a technique using a voxel representation of the scene for fast calculation of single-bounce indirect illumination. Voxel cone tracing (VCT) [Crassin et al. 2011] is based on a similar idea and enables both diffuse and glossy light bounces in dynamic scenes using a sparse voxelization of geometry. Latest real-time implementations can be found in the VXGI framework [Nvidia 2014] and in [McLaren 2014], where the sparse voxelization is replaced by a cascaded volume. A comprehensive overview of interactive global illumination techniques can be found in [Ritschel et al. 2012].

Spherical bases: Many of the mentioned techniques rely on spherical basis functions for the compact representation of lighting information. Spherical harmonics (SH) [Green 2003], orthographic basis functions defined on the sphere, are most commonly used. SH allow for the compact storage of low-frequency spherical functions with only a few (e.g. 9-25 in the case of PRT) coefficients and have desirable mathematical properties that greatly accelerate lighting-related computations [Sloan 2008]. While SH are defined over a full sphere, hemispherical harmonics (HSH) [Gautron et al. 2004] allow for an even more compact representation when only hemispherical information is needed. The $\mathcal{H}$-basis [Habel and Wimmer 2010] is another hemispherical basis that preserves the desirable properties of SH.

We propose a new algorithm which is a combination of different techniques mentioned. Our contributions are:

- A hybrid method for multiple bounce indirect illumination that combines hierarchical radiosity and (ir)radiance volumes
- Using surfels for direct light gathering allowing any kind of dynamic light source in real-time
- A fast GPU implementation with low memory consumption
- High temporal coherency due to a view independent surfel hierarchy and SH caches

3 Algorithm Overview

The algorithm heavily relies on precomputation of important light transport paths. The steps that are executed at runtime are shown in Figure 2. In step (1), shadow maps used for all direct lighting calculations are generated. Then, surfels with known material properties are lit and the outgoing flux is stored in the leaves of a binary tree (2). In step (3), the flux is propagated up along the hierarchy of sender patches. The cache lighting steps (4) accumulate the flux, multiplied with light transport factors, from a fixed size set of linked nodes in the tree. We compute caches in a 3D grid and a 2D light map to increase the quality on static surfaces and overcome light bleeding artifacts. At this point, caches contain a compressed representation (SH) of the incoming spherical radiance for the respective positions. Finally, indirect illumination is interpolated per pixel either from four associated light map caches or from the surrounding eight grid caches (5). Simultaneously, direct lighting is computed as usual, using the shadow maps from (1).

The precomputation consists of four tasks and provides the surfel hierarchy, the cache placement and the linking between sender surfels, i.e. tree nodes, and caches. The tasks are:

- Surfel placement (ideally Poisson-disc sampled)
- Hierarchy building (kd-tree or hierarchical clustering)
- Cache placement
- Link generation (creating SH transport factors)

In the early stages of the runtime pipeline an existing renderer can be used to illuminate the surfels like a set of diffuse pixels. Using the existing renderer allows any implemented type of light source to also affect indirect lighting. This includes further progressive light bounces. To this end, cache values of the last frame are used to add another indirection to the surfels' illumination. Additionally, area lights can be implemented by sampling an emissive and optionally animated texture for each surfel. The first-bounce indirect lighting computation will then include the direct lighting of the area lights.

The light transport itself is simulated by the accumulation of flux from the illuminated surfel hierarchy. We tested two variants for the summation process. In the first case, the transport factors are projected into the SH basis during preprocessing. Hence, the cache

illumination only needs to sum up the stored coefficients from each of its n_L links multiplied with the actual flux. The second solution is to compute the projection at runtime. This requires the direction from cache to sender and a scalar transport factor to be provided. It then takes more ALU instructions but fewer memory accesses to compute the final SH. The second solution is slightly faster and has a low memory cost which is invariant to the number of final SH bands l. On the downside, it is a bit less accurate because the projection during precomputation preserves more detail.

3.1 Light Transport Solution

To illuminate a point on a surface we need to solve the following modified lighting equation. The integration scope Ω is 2πsr for surface points and hemispherical caches or 4πsr for fully spherical caches. Without loss of generality we assume $\Omega = 4\pi$sr and SH caches in the following equations.

$$L_r(\omega_o) = \sum_{s \in \mathcal{N}} \int_\Omega f_r(\omega, \omega_o) V_s(\omega) L_s(\omega) \langle \mathtt{n}_r, \omega \rangle^+ d\omega \quad (1)$$

where $\mathcal{N}$ with $|\mathcal{N}| = n_L$ denotes the set of linked sender patches for the receiving cache r where $V_s(\omega)$ is the sender visibility at the receiver position and L_s is its radiance. The term $\langle \mathtt{n}_r, \omega \rangle^+$ is the clamped (positive) cosine lobe at the receiver's normal $\mathtt{n}_r$ and $f_r(\omega, \omega_o)$ is the scattering function (BSDF or BRDF dependent on the domain Ω) of the material at the receiver. The visibility $V_s(\omega) \mapsto \{0, 1\}$ is defined on the whole sphere and yields 1 for all directions in which the patch s is visible, and 0 otherwise. Because of the Lambertian property and the assumption of a constant radiance over the entire sender, L_s is constant and does not depend on the integration over ω. Consequently, the radiosity B_s is equal to $L_s \cdot \pi$ again using the Lambertian property. Furthermore, the receiver-dependent quantities f_r and $\langle \mathtt{n}_r, \omega \rangle^+$ do not involve the patch s. This allows the following reordering:

$$L_r(\omega_o) = \int_\Omega f_r(\omega, \omega_o) \left(\sum_{s \in \mathcal{N}} \frac{B_s}{\pi} V_s(\omega) \right) \langle \mathtt{n}_r, \omega \rangle^+ d\omega \quad (2)$$

Now, $V_s(\omega)/\pi$ is projected into an SH "form factor" F_{SH_s} with coefficients $c_{s,i}$ and basis functions $y_i(\omega)$, where $i \in [0, \ldots, l^2-1]$. Thus, a link is represented as $F_{SH_s}(\omega) = \sum_i c_{s,i} y_i(\omega)$. The first value $c_{s,0}$ roughly corresponds to the form factor in radiosity, all $c_{s,i}$ with $i > 0$ additionally preserve directional information.

$$\begin{aligned} c_{s,i} &= \frac{1}{\pi} \int_\Omega y_i(\omega) V_s(\omega) d\omega \\ &= \frac{A_s}{N'\pi} \sum_{j=1}^{N'} y_i(\omega_j) V_s(\omega_j) \frac{\langle \mathtt{n}_s, \omega_j \rangle^+}{d_j^2} \end{aligned} \quad (3)$$

Equation 3 is numerically integrated by Monte Carlo integration as shown in appendix A. Thereby, N' is the number of rays used to sample the visibility of the sender, where all ω_j are chosen to point to some position on the sender. A_s is the surface area of the sender and d_j is the distance from the cache to the sampling position.

Using F_{SH_s}, the incident radiance L_{SH} at a cache can be computed as:

$$L_{SH} = \sum_{s \in \mathcal{N}} B_s F_{SH_s} \quad (4)$$

Eventually, in the cache lighting step (4) of Figure 2, this sum is computed for all caches. For reasons of performance we store flux Φ_s instead of B_s (see Section 4.1).

3.2 Per Pixel Lighting

Once the caches are computed, the illumination can be applied to pixels in an arbitrary renderer. In case of dynamic geometry and objects without light maps the 3D radiance grid is interpolated trilinearly and the resulting SH function is evaluated. For static light-mapped geometry four caches are interpolated bilinearly instead. To guarantee that all four texels are filled, we apply a dilation over the eight neighbors to prevent artifacts at texture seams.

Given Equations 2 and 4, solving the indirect illumination at runtime can be reduced to a dot product of SH coefficients. Let the term $f_r(\omega, \omega_o) \langle \mathtt{n}_r, \omega \rangle^+$ be given as X_{SH} with SH coefficients c_X.

$$\begin{aligned} L_r(\omega_o) &= \int_\Omega f_r(\omega_i, \omega) L_{SH} \langle \mathtt{n}_r, \omega \rangle^+ d\omega \\ &= \langle L_{SH}, X_{SH} \rangle \end{aligned} \quad (5)$$

It is invariant whether SH coefficients are interpolated first and then evaluated or vice versa. Let w_i be the interpolation weights. The following holds true because of the distributive property:

$$\sum_i w_i \left(\sum_j c_{L_i,j} c_{X,j} \right) = \sum_j \left(\sum_i w_i c_{L_i,j} \right) c_{X,j} \quad (6)$$

Since we can use the hardware to interpolate the coefficients efficiently we interpolate first and then perform the multiplication.

For the evaluation, X_{SH} must be provided at runtime because it depends on the incident angle ω_i. Our solution is to split the BRDF into a diffuse and a specular term and solve the integral for two different SH projections X_{SH}. In the diffuse case this is $(\frac{\rho}{\pi} \cos\theta_r) \mapsto D_{SH}$ where D_{SH} is the projection of a scaled cosine lobe which is aligned in direction of the surface normal $\mathtt{n}$. The factor ρ/π is the reflectance defined by the material. Similarly, the function for the specular part is defined as $(\frac{n+1}{2\pi} \cos^n \theta) \mapsto S_{SH}$ where θ is the angle between incident light and the half vector instead of $\mathtt{n}$ and $(n+1)/(2\pi)$ is used to normalize the lobe to a volume of one.

To compute the SH coefficients for D_{SH} and S_{SH} we need to integrate over a directed half space which is possible but complex. It is also possible to integrate over an upward-oriented cosine lobe and rotate the result later. Since the lobe is rotationally invariant around the up axis, only projections on zonal harmonics y_l^0 are non-zero. For S_{SH} these are:

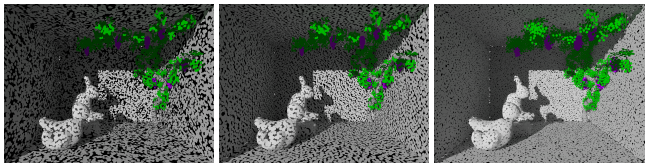
$$c_l^0 = \frac{n+1}{2\pi} \int_0^{2\pi} \int_0^{\pi/2} y_l^0(\theta, \phi) \cos^n \theta \sin\theta d\theta d\phi \quad (7)$$

The rotation of the zonal harmonic into some direction $\mathtt{d}$ is obtained as (see zonal harmonics in [Sloan 2008]):

$$c_l^m = \sqrt{\frac{4\pi}{2l+1}} c_l^0 y_l^m(\mathtt{d}) \quad (8)$$

Thus, the cosine lobe evaluation results in a single constant factor per band which is multiplied before the SH evaluation in direction $\mathtt{d}$. Those factors are given in appendix B for S_{SH}. For D_{SH} the same can be done by removing n and replacing the normalization factor from Equation 7 with ρ/π.

We also experimented with HSH representation for the caches in the light map. Unfortunately, directly integrating the cos^n lobe oriented in an arbitrary direction is difficult for HSH. We found no closed solutions for $n \neq 1$ or more than 3 bands. Also, rotations are not feasible. In the original formulation of the HSH basis, [Gautron et al. 2004] suggest the rotation through an intermediate SH representation. However, this is impractical in our case since the rotation has to be done for each pixel during shading. [Elhabian et al. 2011] used numerical integration at this point, which also is unfeasible at runtime. Hence, we had to discard the HSH basis for light maps.



(a) *Xorshift RNG* **(b)** *Halton-sequence* **(c)** *Halton-sequence with additional relaxation*

Figure 3: *Differently generated distributions of 30000 surfels.*

4 Implementation Details

While the previous section provided the overall idea of the algorithm we now provide an insight to details and optimizations.

4.1 Flux in Surfels and Hierarchy

In the derivation of lighting formulas in Section 3.1 the radiosity B_s is computed from the radiance L_s and stored for the surfels. However, using B_s leads to an overhead in the pull step, because radiosity is defined as flux per area and an area weighted average is necessary. Thus, the area must be fetched repeatedly during pulling.

Instead of radiosity, flux Φ is cumulated directly without averaging. It turned out that propagating flux through the hierarchy is $1{-}2.5$ ms faster, because the pulling is limited in bandwidth. Therefor, B_s must be converted to flux Φ_s by multiplying with the surfel's area during illumination of the leaves. Since leaf surfels are of the same area this is a constant factor. During cache lighting, Φ_s must be converted back to radiosity by a division by the sender's area which is the sum of all surfel areas in the respective subtree. We incorporate this area by scaling F_{SH} accordingly during precomputation and hence do not add any costs at runtime. The scaling of F_{SH} is allowed due to distributivity of the sum in the SH evaluation.

4.2 Surfel Placement

The surfels are the primary samples which are illuminated during runtime. Optimally, they have a Poisson disc distribution over all meshes. Gaps between surfels are allowed, but it must be assured that the area of all surfels equals the area of the geometry. The quality of this step has a noticeable impact on the final results.

To distribute surfels each triangle is sampled proportional to its area. The area of too small triangles is accumulated until their combined area is sufficient to spawn another surfel. Using a low-discrepancy Halton-sequence [Halton and Smith 1964] improves the distribution compared to a Xorshift random number generator [Marsaglia 2003] as shown in Figure 3. To further increase the quality we subsequently relax the distances. For that the surfels are repelled from their nearest neighbors iteratively, allowing only movements along the tangential plane.

There are other approaches to target this problem as the Dart Throwing in [Cline et al. 2009]. Most methods require a geodesic distance to achieve the Poisson disc distribution over the surfaces. These methods could be evaluated in future work. Our approach is simpler and faster and we do not expect a qualitative gain through more complex algorithms compared to the results in Figure 3c.

4.3 Surfel Clustering and Tree Metric

The next task in the precomputation is to build a hierarchy of the samples. This is required for the hierarchical light transport simulation, where each cache is illuminated only by a cut of limited size through that hierarchy.

It is possible to quickly build a kd-tree or a similar space partitioning tree over the surfels. Nevertheless, we considered hierarchical agglomerative clustering (HAC) as higher quality option to achieve more meaningful sender patches. We implemented the generic approach from [Müllner 2011] with a custom distance metric:

$$d(a,b) = \frac{\|\mathbb{x}_a - \mathbb{x}_b\|^2}{\|\mathbb{x}_{max} - \mathbb{x}_{min}\|^2} + C_n(1 - \langle \mathbb{n}_a, \mathbb{n}_b \rangle) + C_b \frac{|A_a - A_b|}{\max(A_a, A_b)}$$

The most important term is the squared distance between the cluster centers $\mathbb{x}$ to create compact clusters. It is normalized by the maximum scene extent. The next term includes the deviation of normals $\mathbb{n}$ to avoid the early clustering of opposite faces (e.g. two sides of a wall). Further, we added a penalty on the tree balancing, where the area A is a direct measure of the number of leaves since all surfels have the same size. The balancing is important for the performance of the pull step during runtime. The two factors C_n and C_b weight the importance of the normal and the balancing term. Values of $C_n{=}0.1$ and $C_b{=}0.02$ worked well for all our scenes.

4.4 Cache Placement

For dynamic objects caches are placed in a uniform grid with manually set resolution. Placing caches for the light map is more involved. Therefor, we iterate over all light mapped triangles in the scene and locate the texel centers. For a triangle the bounding rectangle is determined in texture space and each of the texels is tested if it is within the triangle. The position of the cache is computed by the texel's position within the triangle. Texels in empty areas of the light map are not filled by this process. As mentioned, we duplicate probes along texture seams by dilation to allow linear interpolation.

4.5 Link Generation

The target of link generation is to determine the n_L most important sender patches with respect to a chosen cache. We use the summed solid angle of all clustered surfels as metric for the importance, since the lighting conditions are not known in advance. First, the term c_{s_0} (equation 3 for the constant basis function) is computed per surfel and summed up along the hierarchy. Note that this term includes visibility and invisible parts of the scene become unimportant. Afterwards, links are chosen top down by refining the cluster with the largest value iteratively until a cut of n_L elements is found. This cut covers the entire visible part of the scene.

Table 1: *Comparison to other real-time global illumination algorithms. (* a progressive multibounce is possible.)*

Technique	Ind. diffuse	Ind. specular	Dyn. lights	Dyn. objects	Performance
Ours	n (progressive)	★☆☆	full	receive only	★★★
VCT [Crassin et al. 2011]	1*	★★★	good (rsm)	full	★☆☆
C-LPV [Kaplanyan and Dachsbacher 2010]	1*	☆☆☆	good (rsm)	full (coarse)	★★☆
CRC [Vardis et al. 2014]	n (coarse)	★☆☆	good (rsm)	full (medium)	★★★
PRT [Sloan et al. 2002]	n	★☆☆	only environmental	no	★★★
LightSkin [Lensing and Broll 2013]	1	★★☆	good (rsm+)	full (coarse)	★★☆

All links for a cache are stored sequentially such that the cache index times n_L gives an offset to the first link. A link contains the target cluster index and either the form factor F_{SH} or a direction vector plus a scalar form factor (see Section 4.6).

4.6 Alternative Link Precomputation

In Section 3.1 F_{SH} was precomputed as a set of SH coefficients. Consequently, a lot of values must be stored and fetched during cache lighting. As stated, it is also possible to store a direction and a scalar transport factor instead and to compute the projection at runtime. This requires a different projection technique opposed to Monte Carlo sampling. Now, a larger solid angle must be projected directly. A solution for that can be achieved by integration and rotation again.

This optimization is a bit less precise, because the projection is simplified and preserves less details in theory. However, it still yields results very similar to those before. Also, it reduces memory consumption considerably and slightly increases performance, too.

4.7 Coefficient Textures

During runtime all SH coefficients need to be accessed efficiently. In our tests with up to 4 bands and trichromatic values this leads up to 48 coefficients per cache and 16 per link.

The coefficients for the links are stored sequentially in a Shader Storage Buffer. Since bandwidth is the bottleneck of the cache lighting we encoded the coefficients in 16-bit floats.

The coefficients of L_{SH} are stored in 2D/3D textures respectively to utilize hardware interpolation. We packed all coefficients into a single texture, such that the two textures have the sizes $\mathrm{X} \times (\mathrm{Y} \cdot l^2)$ and $\mathrm{X} \times \mathrm{Y} \times (\mathrm{Z} \cdot l^2)$. Due to the necessity of write accesses only RGBA formats are available, where one channel remains unused. Again we used 16-bit floats per value to decrease the bandwidth.

5 Evaluation

First, we will compare our method to other algorithms and the unbiased solution. Afterwards, the performance is benchmarked and time-quality tradeoffs are shown.

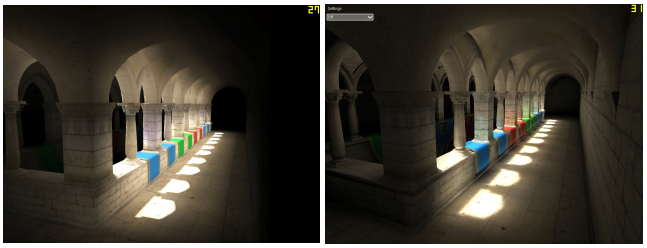
5.1 Comparison to other Techniques

Table 1 summarizes the most important properties of related illumination algorithms. Methods which rely on precomputations are naturally faster but more limited for dynamic content. Here, only CRC has a similar performance, because of their advanced cache allocation optimization. C-LPV, VCT and LightSkin can perform at real-time frame rates in medium quality. However, the original VCT implementation using a sparse octree is only interactive, but also offers the best quality of all compared methods.

Certainly, all of the techniques support at least one indirection of diffuse lighting. The qualitative comparison of Figure 4 shows that our technique is much closer to the ground truth than LPV and VCT. Our approach does not suffer from light bleeding because of the light map caches which are fully aware of the geometry.

Only CRC and PRT already include solutions for multiple bounces. Thereby, CRC is progressive, as ours, but uses a coarse approximation for subsequent bounces. Contrary, our technique uses the same approximation quality for all bounces. In C-LPV and VCT it is theoretically possible to induce indirect illuminated geometry into the voxel volume to extend the approaches towards multiple bounces.

Almost all other algorithms use reflective shadow maps to sample direct illumination. Therefore, dynamic lights are supported,



(a) *Light Propagation Volumes* **(b)** *Voxel Cone Tracing*

(c) *Ours* **(d)** *Ground Truth*

Figure 4: *Comparison of indirect diffuse lighting (multiple bounces). Image (a) taken from [Kaplanyan and Dachsbacher 2010], (b) and (d) taken from [Crassin et al. 2011].*

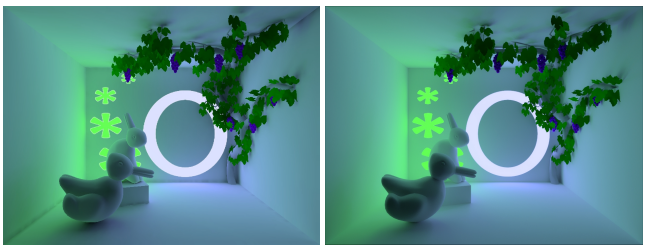
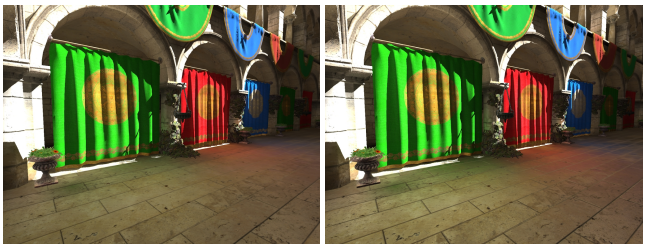
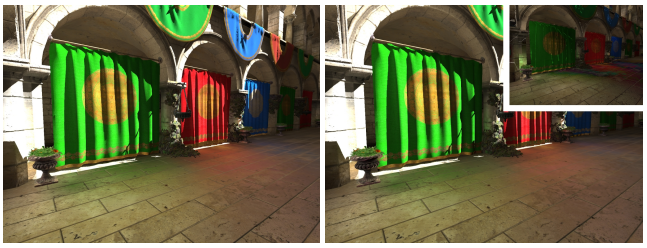


Figure 5: *Comparison of a textured area light (four asterisks and a ring). Left: Ours with 40k surfels, 13785 caches and many bounces at 195 fps. Right: path-traced 62k rays/pixel, 16 bounces, >1h. Artifacts are visible in the shadow details, on the two objects illuminated by the grid and at edges of the box.*



(a) *Specular 2 SH bands* **(b)** *Specular 3 SH bands*



(c) *Specular 4 SH bands* **(d)** *Runtime Projected 4 SH bands*

Figure 6: *Glossy reflections with a Blinn-Phong-exponent of 20. The indirect lighting is amplified by a factor of 2 to emphasize the differences. The alternative implementation is similar, but not equal as the inlay of the 32×difference image shows.*

but computation time increases rapidly with the number of light sources. In our solution a fixed number of surfels, n_S, is lit using a standard rendering pipeline. Hence, we are able to support arbitrary lights at the native performance of the rendering engine.

The ability to use arbitrary light sources also includes area lights which can have color variations and animations using textures. The first bounce indirect light includes the direct light of area light sources including their shadow. Figure 5 compares our algorithm to a path-traced ground truth. It shows a high accordance at first. However, the cache interpolation leads to some artifacts. Most of them are missing shadow details (ceiling) and artifacts along edges, where the placement of caches is a problem. Also, the *Bunnyduck* has a perceptible different shading, because of the SH compression and cache interpolation.

We also experimented with specular materials. Figure 6 shows the results using SH representation with two to four bands. Most other techniques could be extended in the same way, so this is not a specific advantage of our algorithm. The only requirement for specular sampling is to store incident radiance to be able to sample into arbitrary outgoing directions afterward. Here, VCT achieves reflections for higher exponents at the costs of a highly detailed octree and a fine grained cone marching. LightSkin also achieves slightly higher coefficients by the use of local virtual lights which are created for known reflection directions. However, the model of known reflection directions does not apply to normal mapped geometry.

The accompanying video shows the illumination of dynamic objects. We are able to light those objects without including their indirect shadows or reflections. VCT, LPV and CRC use voxelized blocker volumes which can be generated in real-time. This could be included in our technique, since for each link the direction as well as the solid angle are known and cones can be traced, too. An advantage is, that only the dynamic objects need to be voxelized. However, since voxelization is not a native part of our technique it is considered as a plugin-extension and therefore not implemented. Contrary, LightSkin explicitly projects blocker proxies to generate indirect shadows. This could also be implemented by a cone-sphere test in our algorithm, but we consider providing the proxies more complicated and less performant than the voxelization approach.

5.2 Failure Cases

First, there are common artifacts of light maps and interpolations in grids. Most noticeable are misplaced caches as in Figure 7a. This can be avoided by better light mapping of the geometry or back-face culling during the ray casted visibility tests.

(a) *Light map caches (black ones) are placed behind other geometry.*

(b) *A small spotlight (top right) shows discontinuities in linkage on the floor.*

(c) *Light bleeding (amplified by $\times 4096$): the upper corridor should be black*

Figure 7: *Possible artifacts of the algorithm.*

Also, there are algorithm specific problems, which are hardly visible under usual lighting conditions. In Figure 7b only a few surfels are lit directly. Neighbored caches use different approximating links due to local decisions in the link generation step.

Light bleeding occurs in two cases. Besides interpolation in the grid the clustering of senders can introduce this artifact (Figure 7c). Caches in the upper corridor are linked to clusters which partially contain surfels from below. However, this effect is small and not perceptible if multibounce is enabled or more lights are in the scene.

5.3 Performance

Table 2 lists the costs of the individual steps as well as the whole frame time on different machines for a high and a moderate quality setup. We also included a CPU-reference implementation, because we expected that some steps scale better on CPU than on GPU. The timings for the upload are not listed, but lead to an additional overhead which is included in the total frame time of CPU experiments.

Table 2: *Performance breakdown in ms for two different settings at a resolution of* 1920×1080 *for the sponza scene.*
[1] *settings of Figure 6 (d):* n_C=10585, l=4, n_L=254, n_S=128k.
[2] *moderate settings:* $n_C = 10585$, $l = 3$, $n_L = 128$, $n_S = 32k$.

	Shadow Maps	Surfel Lighting	Pull (Sender Hierarchy)	Cache Lighting	Shading	Total Frame
GTX 980[1]	0.21	0.09	1.72	2.27	2.59	7.17
GTX 850M[1]	0.85	0.47	8.88	11.1	9.96	33.3
K5100M[1]	0.58	0.25	11.0	15.7	6.76	34.9
GTX 980[2]	0.21	0.04	0.38	0.58	1.99	3.65
GTX 850M[2]	0.85	0.21	1.27	3.09	7.93	15.3
K5100M[2]	0.58	0.10	1.26	2.84	5.32	11.26
i7-4790S[1]	–	8.12	5.00	116	–	131
i7-4510U[1]	–	7.06	8.04	148	–	198
i7-4790S[2]	–	4.87	1.13	11.5	–	21.1
i7-4510U[2]	–	2.57	1.34	28.0	–	33.8

The most expensive steps are *Pull*, *Cache Lighting* and *Shading*. In almost no case is the CPU faster than the GPU implementations. Due to its recursive nature, only the pull step achieves similar or better CPU timings on some hardware configurations. However, the overhead for up- and download of the surfel hierarchy make a hybrid solution pointless. The cache lighting costs are moderate and can be controlled by parameters which are explained in detail later. Shading is comparable expensive, but rasterization and direct lighting add costs beside the global illumination, too.

Our technique scales well with larger resolutions. We experimented with resolutions of 960×540, 1920×1080 and 3840×2160 with setup one. On GTX 980, the shading with indirect illumination takes 0.99ms, 2.59ms and 8.84ms respectively and 0.67ms, 1.41ms, 3.51ms without. All other steps are independent of the resolution.

In total, four parameters influence the performance and quality. Those are the number of SH bands l, the number of caches n_C, their number of links to the senders n_L and the number of surfels n_S. All of them are investigated deeper in the following.

The two parameters n_S and n_L have similar effects on the quality as they both reduce the detail of the light transport itself. In Figure 8 the influence of both parameters is visualized. As expected they have an almost proportional impact on the respective pipeline step. Thereby the surfel illumination scales well with an increasing n_S, whereas the pull gets more expensive faster.

The first column of images in Figure 8 shows, that 32 links are not sufficient. In general, reducing n_S is more stable than reducing n_L. More links make the technique robust against variances in the surfel density. More surfels increase details close to the cache, but lead to a degradation of far links when n_L is not increased simultaneously,

Figure 8: *Influence of the two parameters* surfel-count n_S *and* links per cache n_L *on performance and quality. A larger* n_S *takes more time for surfel relighting as well as pull and reduces cache efficiency of the cache relighting (see overlayed numbers). More links increase time for cache relighting proportionally.*
In general fewer links result in a less stable light transport (top rows) while fewer surfels reduce these artifacts at the costs of details and energy preservation (left column).
The images in the last row show the surfel distribution for the respective column.

because more links are used for the close details. Since details are blurred through the SH compression anyway it is better to use a smaller surfel density in general.

The parameter n_C is set by the grid and the light map resolution. Using more caches reduces the artifacts due to interpolation (e.g. shadow bleeding) and increases the details in indirect shadows. On the other hand, cache lighting becomes the most expensive step in some configurations (e.g. see Table 2 K5100M[1]). Still, cache density should be kept as high as possible. Here, a view dependent adaptive choice of the cache density would increase quality and performance at the same time. However, caches outside the view frustum cannot be rejected if multiple bounces of light are desired.

Reducing the number of SH bands l reduces the storeable light details in the caches as shown in Figure 6. For diffuse lighting, 3 bands usually suffice. However, reducing l decreases the memory consumption and costs of the cache illumination and the shading stage, because fewer coefficients need to be fetched/written. Table 3 shows that the cache lighting scales well with the number of

Table 3: *Influence of the number of SH bands on performance (ms) for the configuration in Figure 6.*

	Cache Lighting			Shading		
SH bands	2	3	4	2	3	4
GTX 980	0.92	1.35	2.27	1.68	1.98	2.59
GTX 850M	3.84	7.19	11.1	7.04	7.93	9.96

coefficients. Although 4 bands have four times the number of coefficients than 2 bands, the lighting only takes $2.5\times$ and $2.8\times$ as much time on the two tested GPUs. During shading the influence of more bands is even less noticeable ($1.5\times$ and $1.4\times$ from 2 to 4 bands) as direct lighting and other rendering costs hide the work. All other steps are invariant to the cache representation.

Since cache lighting is bandwidth bound, the proposed runtime projection from Section 4.6 improves performance. This alternative replaces fetches by more ALU-instructions, which are faster at this point. Thus, the cache lighting for four bands takes 0.93 ms and 0.37 ms on GTX 980 for the setups [1] and [2] respectively. Compared to 2.27 ms and 0.58 ms from table 2 this is a noticeable performance gain at similar quality (see Figure 6).

5.3.1 Memory Consumption

The memory consumption of our algorithm is determined by three factors: surfels, caches and links. Table 4 gives an overview of the individual costs. For each surfel, area, position, normal, texture coordinates (light map) and albedo need to be stored. We manually compress the normal to 32 bit as well as UV+RGB+Emissive into a single RGBA16UI texture and use an RGBA32F texture for surfel position and area. Since we use a binary tree, $2n_S$ nodes with a surfel ID, a child ID and flux have to be stored. Thereby, the flux is encoded into a shared exponent format with 3x9 bit unsigned mantissa and 5 bit exponent (32 bit in total). For the caches, the trichromatic radiance is stored in RGBA16F textures (see, 4.7), where each coefficient takes 2 bytes, but the alpha channels are unused. Coefficients in links are also stored as 16 bit floats together with a 32 bit surfel ID. Their total count is $n_L \cdot n_C$.

The total memory consumption for the high quality setup in the Crytek Sponza scene is 94.77 MiB and 29.51 MiB for the moderate setup. This is comparable to other techniques as VCT and C-LPV. If SHs are not stored and link-projection is done at runtime the costs for the two setups become 36.64 MiB and 17.30 MiB instead which is less than the requirements of comparable techniques.

Most of the costs result from the links. The costs can be reduced by decreasing one of the parameters n_L, l, n_C or by projecting at runtime. The first three options all reduce quality significantly, whereas the runtime projection is visually similar (see Figure 6 and supplied video) and allows more bands without higher memory costs.

Table 4: *Memory consumption of individual components with numbers for the high quality settings [1]. Using the runtime projection for links (Section 4.6) requires the same memory as 2-band SH but supports a dynamic band number.*

	Per Instance	Count[1]	Total
Surfels	28 B	128k	3.42 MiB
Surfel Nodes	12 B	256k	2.93 MiB
Caches SH2	32 B	10k	0.31 MiB
Caches SH3	72 B	10k	0.69 MiB
Caches SH4	128 B	10k	1.22 MiB
Link SH2/RTP	12 B	2540k	29.07 MiB
Link SH3	22 B	2540k	53.29 MiB
Link SH4	36 B	2540k	87.20 MiB

6 Conclusions and Future Work

We proposed a new algorithm of cache based indirect lighting which relies on precomputations. Compared to similar methods this one is faster and requires less memory. It natively supports any type of dynamic lighting including multiple diffuse indirections at very low costs. However, due to the precomputations dynamic objects can only be shaded and do not cast indirect shadows or reflect light back to the scene.

As mentioned before the missing shadows and reflections of dynamic objects could be included by a voxel cone tracing approach. This is possible with a voxelization of the dynamic parts only instead of the whole scene.

Another possible optimization is to reduce the number of caches view dependent. One option is the dynamic cache allocation from [Vardis et al. 2014] and another would be the use of a cascaded volume. In any case all caches must have precomputed linkage, but it is not necessary to illuminate and use all of them.

Besides sparse cache selection a dynamic link selection is interesting. Currently only geometric visibility is considered. The albedo or the current light situation are not included. It is thinkable to reduce the number of links based on the current flux at runtime.

To improve specular reflections it would be interesting to test the SH chrominance compression, also proposed by [Vardis et al. 2014]. It is possible to use more bands at the same costs by allowing a higher error for chrominance than for luminance.

7 Acknowledgements

This work is partially supported by the German Research Foundation (DFG), Grant Nr. GR 3833/3-1.

References

CLINE, D., JESCHKE, S., WHITE, K., RAZDAN, A., AND WONKA, P. 2009. Dart Throwing on Surfaces. *Computer Graphics Forum* vol. 28(4), 1217–1226.

CRASSIN, C., NEYRET, F., SAINZ, M., GREEN, S., AND EISEMANN, E. 2011. Interactive Indirect Illumination Using Voxel Cone Tracing. *Computer Graphics Forum* vol. 30(7), 1921–1930.

DONG, Z., GROSCH, T., RITSCHEL, T., KAUTZ, J., AND SEIDEL, H.-P. 2009. Real-Time Indirect Illumination with Clustered Visibility. In *Proc. Vision, Modeling and Visualization*, 187–196.

ELHABIAN, S., RARA, H., AND FARAG, A. 2011. On the Use of Hemispherical Harmonics for Modeling Images of Objects Under Unknown Distant Illumination. In *Proc. IEEE Int. Conference on Image Processing*, 1109–1112.

GAUTRON, P., KRIVANEK, J., PATTANAIK, S. N., AND BOUATOUCH, K. 2004. A Novel Hemispherical Basis for Accurate and Efficient Rendering. In *Proc. 15th Eurographics Conference on Rendering Techniques*, 321–330.

GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. 1984. Modeling the Interaction of Light Between Diffuse Surfaces. In *Computer Graphics*, vol. 18(3), 213–222.

GREEN, R. 2003. Spherical Harmonic Lighting: The Gritty Details. In *Archives of the Game Developers Conference*.

GREGER, G., SHIRLEY, P., HUBBARD, P. M., AND GREENBERG, D. P. 1998. The Irradiance Volume. *IEEE Computer Graphics and Applications* vol. 18(2), 32–43.

HABEL, R., AND WIMMER, M. 2010. Efficient Irradiance Normal Mapping. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 189–195.

HALTON, J. H., AND SMITH, G. B. 1964. Algorithm 247: Radical-inverse Quasi-random Point Sequence. *Communications of the ACM* vol. 7(12), 701–702.

HANRAHAN, P., SALZMAN, D., AND AUPPERLE, L. 1991. A Rapid Hierarchical Radiosity Algorithm. In *Computer Graphics*, vol. 25, 197–206.

IMMEL, D. S., COHEN, M. F., AND GREENBERG, D. P. 1986. A Radiosity Method for Non-diffuse Environments. In *Computer Graphics*, vol. 20(4), 133–142.

KAJIYA, J. T. 1986. The Rendering Equation. In *Computer Graphics*, vol. 20(4), 143–150.

KAPLANYAN, A., AND DACHSBACHER, C. 2010. Cascaded Light Propagation Volumes for Real-time Indirect Illumination. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 99–107.

KELLER, A. 1997. Instant Radiosity. In *Proc. SIGGRAPH 97,* Annual Conference Series, 49–56.

KRISTENSEN, A. W., AKENINE-MÖLLER, T., AND JENSEN, H. W. 2005. Precomputed Local Radiance Transfer for Real-time Lighting Design. *ACM Trans. Graph* vol. 24(3), 1208–1215.

LAINE, S., SARANSAARI, H., KONTKANEN, J., LEHTINEN, J., AND AILA, T. 2007. Incremental Instant Radiosity for Real-time Indirect Illumination. In *Proc. 18th Eurographics Conference on Rendering Techniques*, 277–286.

LENSING, P., AND BROLL, W. 2013. LightSkin: Real-Time Global Illumination for Virtual and Mixed Reality. In *Proc. 5th Joint Virtual Reality Conference*, 17–24.

MARSAGLIA, G. 2003. Xorshift RNGs. *Journal of Statistical Software* vol. 8(14), 1–6.

MARTIN, S., AND EINARSSON, P. 2010. A Real-Time Radiosity Architecture for Video Games. In *SIGGRAPH 2010 Courses*.

MCLAREN, J. 2014. Cascaded Voxel Cone Tracing in The Tomorrow Children. In *Computer Entertainment Developers Conference*, CEDEC.

MITCHELL, J., MCTAGGART, G., AND GREEN, C. 2006. Shading in Valve's Source Engine. In *SIGGRAPH 2006 Courses*.

MÜLLNER, D. 2011. Modern Hierarchical, Agglomerative Clustering Algorithms. *arXiv:1109.2378*.

NVIDIA, 2014. Voxel Global Illumination.

RITSCHEL, T., DACHSBACHER, C., GROSCH, T., AND KAUTZ, J. 2012. The State of the Art in Interactive Global Illumination. *Computer Graphics Forum* vol. 31(1), 160–188.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed Radiance Transfer for Real-time Rendering in Dynamic, Low-frequency Lighting Environments. *ACM Trans. Graph* vol. 21(3), 527–536.

SLOAN, P.-P. 2008. Stupid Spherical Harmonics (SH) Tricks. In *Archives of the Game Developers Conference*.

THIEDEMANN, S., HENRICH, N., GROSCH, T., AND MÜLLER, S. 2011. Voxel-based Global Illumination. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 103–110.

VARDIS, K., PAPAIOANNOU, G., AND GKARAVELIS, A. 2014. Real-Time Radiance Caching using Chrominance Compression. *Journal of Computer Graphics Techniques* vol. 3(4), 111–131.

A Projecting the Transport Factor to SH

In Section 3.1 we formulated the light transport in dependence of an SH factor which primarily contains the visibility of a sender cluster.

$$c_{s,i} = \frac{1}{\pi} \int_\Omega y_i(\omega) V_s(\omega) d\omega$$

Now, the integral can be solved through Monte Carlo sampling. Therefor the integral is replaced by a sum over N samples and each sample is weighted by its inverse sampling probability.

$$c_{s,i} = \frac{1}{N\pi} \sum_{j=1}^{N} \frac{1}{p(\omega_j)} y_i(\omega_j) V_s(\omega_j)$$

For a uniform sampling on a sphere the probability is $p = 1/4\pi$.

$$c_{s,i} = \frac{4}{N} \sum_{j=1}^{N} y_i(\omega_j) V_s(\omega_j) \tag{9}$$

Equation 9 computes the correct solution, but casting N rays is computational expensive. Especially, if the target sender only covers a small fraction of the sphere many rays yield zero. We are able to decrease this overhead by casting only rays into the direction of the sender patch. To incorporate this into the Monte Carlo integration we can weight the result with the probability to hit the patch if it is not occluded. I.e. we multiply the result with the solid angle of the sender divided by the full spherical angle. Since we know each other ray would simply add zero to the sum this has the same effect as using a larger N.

$$\mathrm{w}(\omega) = \frac{A_s \langle \mathbb{n}_s, \omega \rangle^+}{d^2} \cdot \frac{1}{4\pi} \tag{10}$$

Note that the weighting factor w depends on the sampling direction, as the distance d and the angle to surface θ_s depend on it. Inserting this into equation 9 and replacing the samples by the subset N' into the direction of the sender yields the final result.

$$c_{s,i} = \frac{A_s}{N'\pi} \sum_{j=1}^{N'} y_i(\omega_j) V_s(\omega_j) \frac{\langle \mathbb{n}_s, \omega_j \rangle^+}{d_j^2} \tag{3}$$

B Cosine Lobe Integration on SH

As explained in section 3.2 the integration of a camped cosine lobe and a function in SH representation can be done over projection to zonal harmonics (Equation 7) and an SH rotation (Equation 8). Equation 8 has the structure of a usual sampling $y_l^m(\mathbb{d})$ multiplied by a factor s which depends on l and n only.

$$s_l(n) = \sqrt{\frac{4\pi}{2l+1}} c_l^0 \tag{11}$$

Hence, there is a single factor per band depending on the exponent n which can be computed at runtime and inserted to the normal SH lookup. Solving the integral for c_l^0 and inserting the result to Equation 11 gives:

$$\begin{aligned} s_0(n) &= 1 \\ s_1(n) &= \frac{n+1}{n+2} \\ s_l(n) &= s_{l-2}(n) \frac{n+2-l}{n+1+l} \end{aligned}$$

Hence, integrating the specular cosine lobe which is aligned in direction $\mathbb{d}$ is computed as:

$$c_l^m = s_l(n) y_l^m(\mathbb{d}) \tag{12}$$