

Real-Time Global Illumination Using Precomputed Illuminance Composition with Chrominance Compression

Johannes Jendersie¹ David Kuri² Thorsten Grosch¹
¹TU Clausthal ²Volkswagen AG

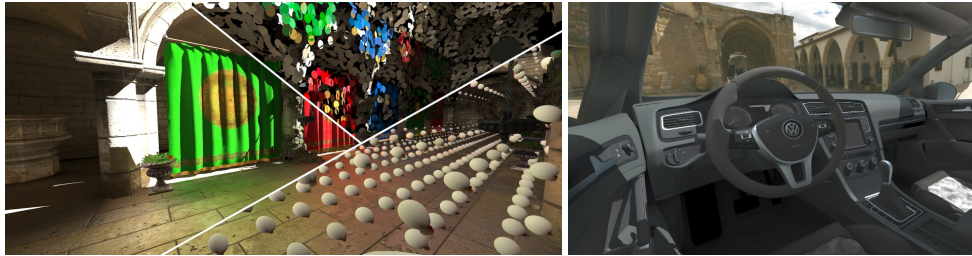


Figure 1. The Crytek Sponza¹ (5.7ms, 245k triangles, 10k caches, 4 band SHs) with a visualization of the caches (left) and a Volkswagen test data set using an environment map from Persson² (10.5ms, 1.35M triangles, 10k caches, 6 band SHs) with multiple bounce indirect illumination and slightly glossy surfaces (right).

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 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. We provide experimental results for up to eight bands of spherical harmonics to stress the limits of specular reflections. In addition, we apply a chrominance downsampling to reduce the memory overhead of the caches.

The sparse sampling of illumination in surfels also enables indirect lighting from many light sources and an efficient progressive multi-bounce implementation. Furthermore, any existing pipeline can be used for surfel lighting, facilitating the use of all kinds of light sources, including sky lights, without a large implementation effort. In addition to the general initial

¹Meinl, F. Crytek Sponza (www.crytek.com).

²Persson, E. Saint Lazarus Church. CC BY 3.0 (www.humus.name).

lighting, our method adds a simple way to incorporate area lights. If using an emissive term in the surfels, area lights are included in the precomputed light transport. Thus, precomputation takes care of the proper shadowing.

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. The computational cost is prohibitive especially for interactive applications. 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 that 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.

2. Related Work

2.1. Global Illumination and Radiosity Approaches

The problem of determining the global illumination of a scene can be traced 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 the 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.

2.2. Precomputed Radiance Transfer

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].

2.3. 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 et al. 2014]. We apply this color space compression to our SH-caches. Lensing and Broll [2013] use a sparse cache distribution on surfaces to reduce memory and computational overhead in their LightSkin framework.

2.4. 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 have developed a real-time global illumination middleware called Enlighten [Martin and Einarsson 2010] that presumably works similarly to the technique proposed in this paper. Unfortunately, publicly available details on 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. The 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].

2.5. Spherical Bases

Many of the aforementioned 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;
- the use of 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;
- an evaluation of SH for the purpose of specular lighting for up to eight bands.

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

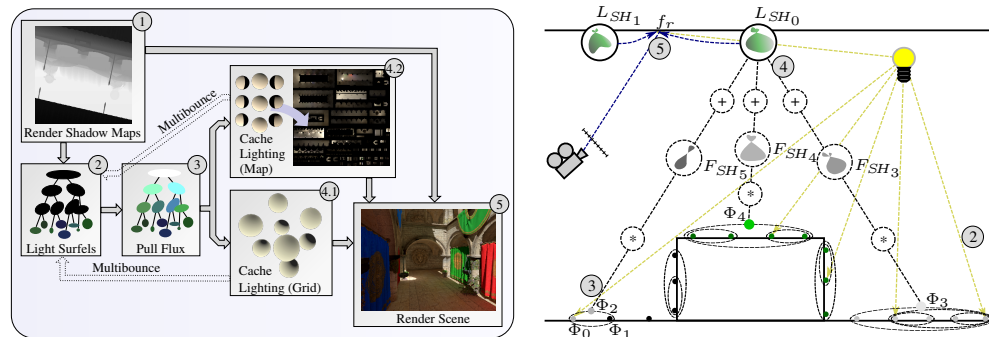


Figure 2. Pipeline and light transport overview.

material properties are lit, and the outgoing flux is stored in the leaves of a binary tree (step 2). In step (3), the flux is propagated up along the hierarchy of sender patches. The cache lighting steps ((4.1) and (4.2)) 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 (step (5)). Simultaneously, direct lighting is computed as usual, using the shadow maps from step (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 in 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 $2\pi sr$ for surface points and hemispherical caches

or $4\pi sr$ for fully spherical caches. Without loss of generality, we assume $\Omega = 4\pi sr$ and the use of SH caches in the following equations:

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

where \mathcal{N} with $|\mathcal{N}| = n_L$ denotes the set of linked sender patches for the receiving cache r , $V_s(\omega)$ is the sender visibility at the receiver position, and L_s is its radiance. The term $\langle \mathfrak{m}_r, \omega \rangle^+$ is the clamped (positive) cosine lobe at the receiver's normal \mathfrak{m}_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 $\langle \mathfrak{m}_r, \omega \rangle^+$, do not involve the patch s . This allows the following reordering:

$$L_r(\omega_o) = \int_{\Omega} f_r(\omega, \omega_o) \left(\sum_{s \in \mathcal{N}} \frac{B_s}{\pi} V_s(\omega) \right) \langle \mathfrak{m}_r, \omega \rangle^+ d\omega. \quad (1)$$

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 $i \in [0, \dots, l^2-1]$. 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:

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

Equation (2) is numerically integrated by Monte Carlo integration as shown in Appendix A. There, 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. The value 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

$$L_{SH} = \sum_{s \in \mathcal{N}} B_s F_{SH_s}. \quad (3)$$

Eventually, in the cache lighting step (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) and need to divide by the area.

3.2. Per-pixel Lighting

Once the caches are computed, the illumination can be applied to pixels in an arbitrary renderer. In the 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 (1) and (3), solving the indirect illumination at runtime can be reduced to a dot product of SH coefficients. Let the term $f_r(\omega, \omega_o)\langle \mathfrak{m}_r, \omega \rangle^+$ be given as X_{SH} with SH coefficients c_X . Then,

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

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:

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

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 to 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 the direction of the surface normal \mathfrak{m} . 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 \mathfrak{m} , 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

$$c_l^0 = \frac{n+1}{2\pi} \int_0^{2\pi} \int_0^{\pi/2} y_l^0(\theta, \phi) \cos^n \theta \sin \theta d\theta d\phi. \quad (4)$$

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

$$c_l^m = \sqrt{\frac{4\pi}{2l+1}} c_l^0 y_l^m(\mathbf{d}). \quad (5)$$

Thus, the cosine lobe evaluation results in a single constant factor per band which is multiplied before the SH evaluation in direction \mathbf{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 (4) with ρ/π .

We also experimented with the 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 three 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.

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 accumulated 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. Therefore, 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

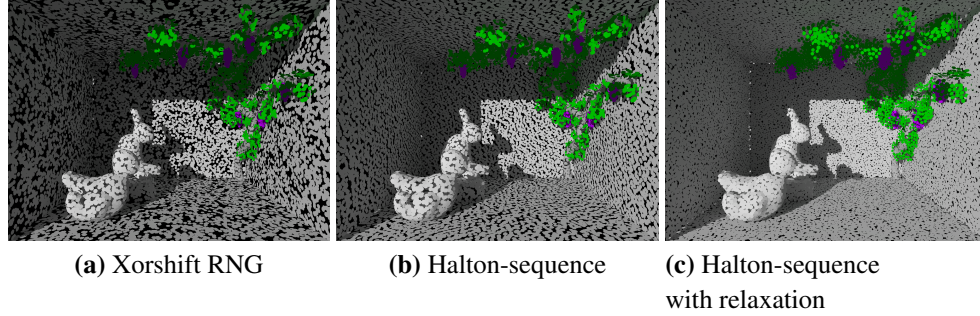


Figure 3. Differently generated distributions of 30000 surfels.

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, such 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 3(c).

4.3. Surfel Clustering and the 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 a higher quality option to achieve more meaningful sender patches. We implemented the generic approach from [Müllner 2011] with a custom distance metric:

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

The most important term is the squared distance between the cluster centers \mathbf{x} to create compact clusters. It is normalized by the maximum scene extent. The next term includes the deviation of normals \mathbf{m} 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. All texel centers must be found on the geometry to be placed in 3D. Therefore, we iterate over all light-mapped triangles in the scene once 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. If so, 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. Figure 4 visualizes the results for a single probe. Note that surfels from the right wall are excluded since they are invisible to the probe.

We use the summed solid angle of all clustered surfels as the metric for the importance, since the lighting conditions are not known in advance. First, the term c_{s_0} (Equation (2) 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

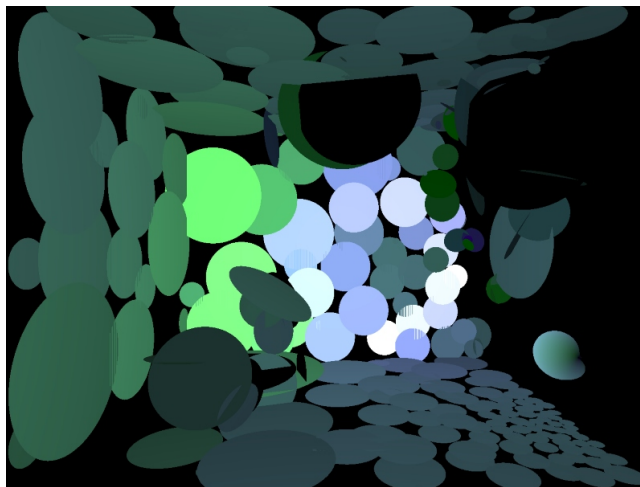


Figure 4. Linked surfels for a single probe on the right wall.

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.

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. Then, the projection can be computed at runtime. This requires a different projection technique from Monte Carlo sampling. With a single projection, a larger solid angle must be projected directly.

For a surfel, we approximate its solid angle projection to the zonal harmonics with an integration over a cone with an half opening angle of α . Afterwards, the SH is rotated into the direction of the surfel using Equation (5) again.

$$c_l^0(\alpha) = \int_0^{2\pi} \int_0^\alpha y_l^0(\theta, \phi) \sin(\theta) d\theta d\phi \quad (6)$$

The results of that integral for up to 8 SH bands are given in Appendix C. All other base functions are symmetric to the up axis and yield 0 if plugged into the equation. All terms depend on α in the form of $\cos^x \alpha$ terms, so we precompute $\cos \alpha$ and store these values per link.

The runtime projection optimization is a bit less precise, because the sampling integration preserves more 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. Memory Layouts

There are three categories of data: the hierarchy of surfels, the linkage between surfels and caches, and the caches themselves. All need to be optimized for the access in different pipeline steps. To allow caching for all read accesses, we use textures and texture buffers, respectively. A texture buffer allows 1D texture access for up to 128M texels and write accesses if bound as shader-storage buffer. Figure 5 shows the details of our memory management. The shaded yellow boxes also contain the used texture formats and dimensions of the texture buffers.

The buffers (1), (2), (3), and (4) all encode different parts of the surfel data. The number of leaf surfels is n_S . Since only leaves are illuminated, buffer (3) is smaller than the other three, which store information for hierarchical cluster surfels as well.

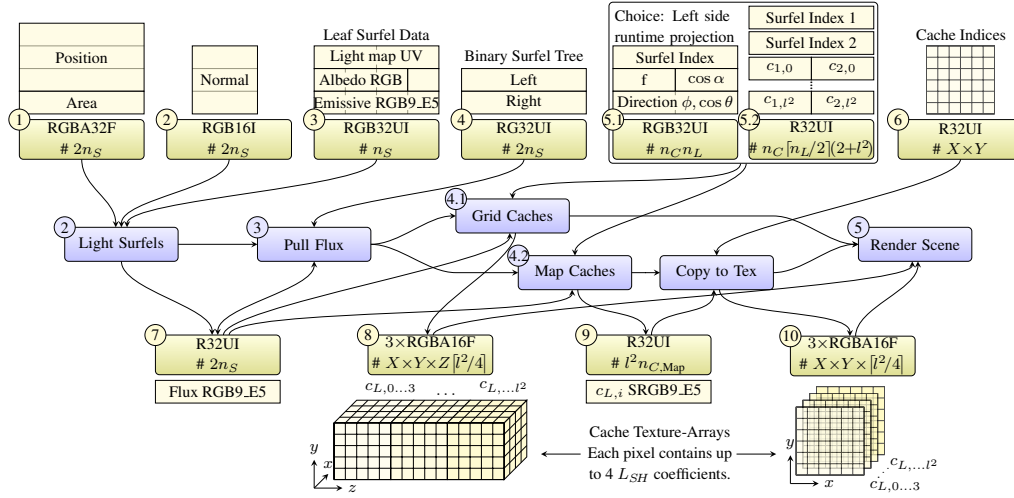


Figure 5. Buffer layouts and accesses in the pipeline. Yellow blocks depict data and blue ones depict the pipeline steps with numbers from Figure 2. The data in the top row is stored in read-only texture buffers whereas all other buffers are written as shader-storage buffer and read as texture buffer/texture. The shaded yellow boxes show texture format and size. The interpretation of texels is given above/below.

The HDR colors in buffers (3) and (7) are manually encoded in 32 bits with a shared exponent of 5 bits and 9 bits per channel.

To illuminate caches, either buffer (5.1) or buffer (5.2) is consumed together with the flux in buffer (7). The size of the buffers in (5.x) depends on the number of caches n_C times the number of links per cache n_L . In the case of runtime projection, the surfel index, the light transport factor f , the cone opening angle $\cos \alpha$ (see Section 4.6), and the direction to the surfel (compressed in polar coordinates) are required per link. Otherwise, all l^2 SH coefficients must be provided per link. To reduce memory requirements and bandwidth in general, we store all SH coefficients as 16-bit floats (half). However, this results in a bad alignment for an odd number of coefficients. We solved this by storing two links in a single sequence (see buffer (5.2)), requiring an even numbers of links.

The final cache coefficients of L_{SH} are stored into three 2D and 3D textures for fast hardware interpolation between caches in the illumination step. Since there are no 3D texture arrays, one of the dimensions must be used to address different sets of coefficients manually. Each of the textures is monochromatic to allow a different number of SH coefficients per channel. For details see the next section.

In the case of the light map, caches must be duplicated among gaps in the map (UV islands). One option would be to redundantly compute all those caches. Instead, it is also possible to store the results in a temporary buffer (9) and do an additional

copy step, which we found to be faster. For that reason, texture (6) contains an index to the temporary buffer (9) for each texel of the light map.

4.8. Chrominance Compression

$YCoCg$ is a color space by [Malvar and Sullivan 2003]. It has good decorrelation properties and is fairly easy to transform from and to RGB:

$$\begin{aligned} Co &= (R - B)/2 & t &= Y - Cg \\ t &= B + Co & G &= Y + Cg \\ Cg &= (G - t)/2 & B &= t - Co \\ Y &= t + Cg & R &= t + Co \end{aligned} \quad (7)$$

Chrominance compression can be used for the final caches as in [Vardis et al. 2014]. In Figure 5, this corresponds to textures (8) and (10). For those, two chrominance textures ($CoCg$) can be stored with fewer coefficients than the third with luma Y . This reduces the number of writes during cache lighting and the number of fetches during shading. Since the link buffer is monochromatic and it does not matter if surfel lighting is stored in RGB or $YCoCg$, the other textures are not compressed in this way.

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. VCT offers the best quality of all compared methods

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)	★★★

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

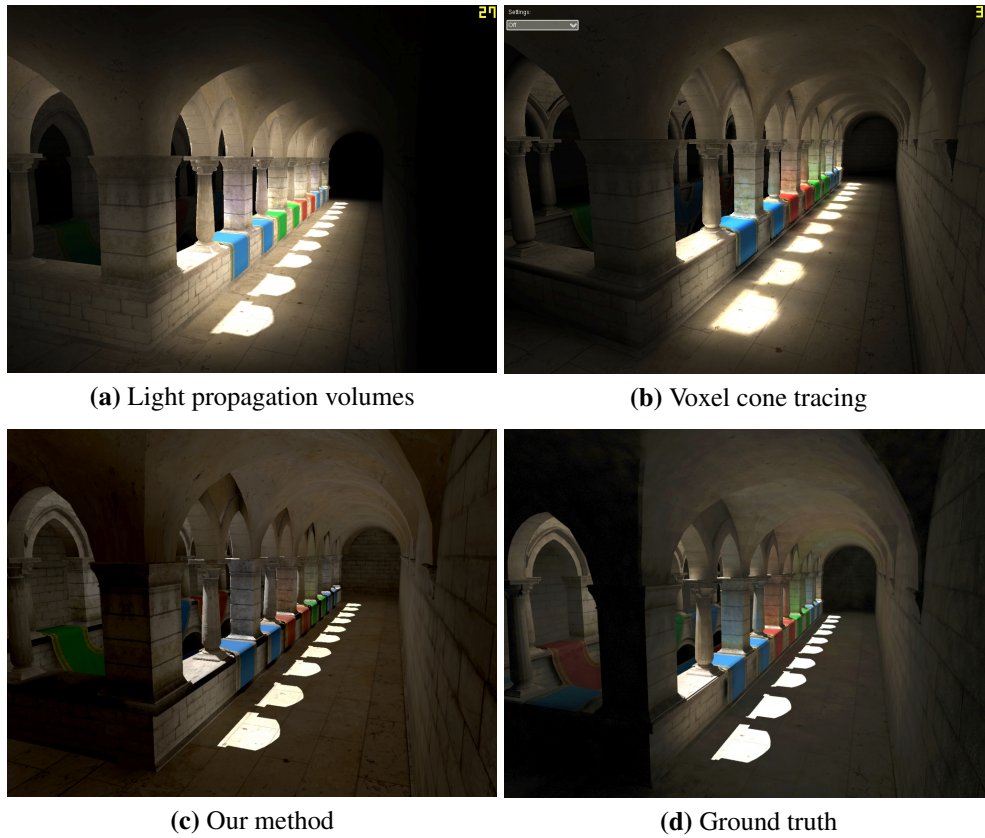


Figure 6. Comparison of indirect diffuse lighting (multiple bounces). Image (a) taken from [Kaplanyan and Dachsbacher 2010], (b) and (d) taken from [Crassin et al. 2011].

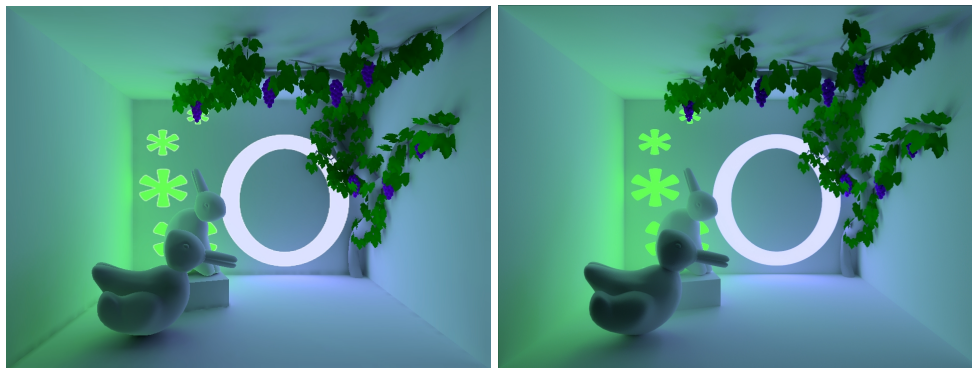


Figure 7. Comparison of a textured area light (four asterisks and a ring). Left: Ours with 40k surfels, 13785 caches, and many bounces at 195 fps. Right: path-traced 62k samples/pixel, 16 bounces, >1h. Artifacts are visible in the shadow details, on the two objects illuminated by the grid and at edges of the box.

if using a large sparse octree as originally done. However, the implementation then becomes slower and achieves only interactive frame rates.

Certainly, all of the techniques support at least one bounce of diffuse lighting. The qualitative comparison of Figure 6 shows that our technique is much closer to the ground truth than LPV and VCT. Our approach does not suffer from light bleeding due to the light map caches which are fully aware of the geometry.

Only CRC and PRT already include solutions for multiple bounces. CRC is progressive, as is our algorithm, but uses a coarse approximation for subsequent bounces. In contrast, our technique uses the same approximation quality for all bounces. In C-LPV and VCT it is theoretically possible to introduce indirectly 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, 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 7 compares our algorithm to a path-traced ground truth. It shows a high level of agreement 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 perceptibly different shading due to the SH compression and cache interpolation.

We also experimented with specular materials. Figure 8 shows the results using SH representation with two to eight bands including performance values on GTX980. The runtime projection allows more bands and is faster, but it also results in more errors as image (m) emphasizes. With the setting used, it is not possible to have more than seven bands with the full SH link buffer, because the number of texels then exceeds the 128M limit. The figure also shows the time-quality trade-off of chrominance compression. When using half as many bands for *CoCg*, the difference to full shading is barely visible but $\approx 31\%$ faster. The performance gain is smaller if using fewer bands in general, because everything up to the cache illumination is performing the same operations. For details see Section 5.3.3.

Most other techniques could be extended to specular lighting at a similar quality, 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 afterwards. Here, VCT achieves reflections for higher exponents at the cost of a highly detailed octree and fine-grained cone marching. LightSkin also achieves slightly higher coefficients by the use of local virtual lights which are created for



Figure 8. Glossy reflections with a Blinn-Phong-exponent of 30 at a resolution of 1920×1080 with $n_C = 10585$, $n_L = 254$ and $n_S = 128k$. The indirect lighting is amplified by a factor of two to emphasize the differences. The alternative implementation is similar, but not equal as the difference image (m) shows. All other difference images look very similar and are therefore not shown. Image (n) shows the results for 8 band SHs using runtime projection. Without the optimization only seven bands are possible due to the link-buffer size. The rightmost column shows the gain of chrominance compression for full eight-band runtime projection.

known reflection directions. However, the model of known reflection directions does not apply to normal-mapped geometry.

The accompanying video (see the supplementary materials at <http://jcgt.org/published/0005/04/02/supplemental.mp4>) 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 an extension and therefore not implemented. In contrast, 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 effective than the voxelization approach.

5.2. Failure Cases

First of all, there are common artifacts of light maps and interpolations in grids. Most noticeable are misplaced caches as in Figure 9(a). This can be avoided by better light mapping of the geometry or back-face culling during the ray-cast visibility tests.

Also, there are algorithm specific problems, which are hardly visible under usual lighting conditions. In Figure 9(b) 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 9(c)). Caches in the upper corridor are linked to clusters which partially contain surfels from below. However, this effect is



Figure 9. Possible artifacts of the algorithm. (a) Light map caches (black ones) are placed behind other geometry. This can be fixed by enabling backface culling during cache precomputation; (b) A small spotlight (top right) reveals discontinuities in linkage on the floor; (c) Single bounce light bleeding (amplified by $\times 4096$): the upper corridor should be black. The cause is that the hierarchy can merge clusters which cannot see each other.

small and not perceptible if multibounce is enabled or if there are more lights in the scene.

5.3. Performance

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

The time for *Surfel Lighting* is rather small compared to all other steps. In almost no case is the CPU implementation 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 upload and download of the surfel hierarchy makes a hybrid solution pointless. The cache lighting costs are moderate and can be controlled by parameters which are explained in detail later. Shading is comparably expensive, but the timings in Table 2 also include rasterization and direct lighting. Hence, global illumination is not responsible for all of the costs.

	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

Table 2. Performance breakdown in ms for two different settings at a resolution of 1920×1080 for the sponza scene.

¹ settings of Figure 8 (g): $n_C=10585$, $l=4$, $n_L=254$, $n_S=128k$.

² moderate settings: $n_C = 10585$, $l = 3$, $n_L = 128$, $n_S = 32k$.

5.3.1. Resolution Dependency

Our technique scales well with larger resolutions. We experimented with resolutions of 960×540 , 1920×1080 , and 3840×2160 with setup one. On the GTX 980, the

shading with indirect illumination takes 0.99ms, 2.59ms, and 8.84ms, respectively, and 0.67ms, 1.41ms, and 3.51ms without. All other steps are independent of the resolution.

5.3.2. Parameter Space

In total, four parameters influence the performance and quality. Those parameters 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 these parameters are analyzed in more depth here.

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 10, the influence of both parameters is visualized. As expected, they have an almost proportional impact on the respective pipeline step. Thus, the surfel illumination scales well with an increasing n_S , whereas the pull gets more expensive more quickly.

The first column of images in Figure 10 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 variance 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, 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 K5100M¹). 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 decreases the memory consumption and costs of the cache illumination and the shading stage. Higher number of bands preserve more details as is visible in Figure 8. Figure 11 shows the performance impact of parameter l on different passes with and without optimization. Most steps are invariant to the number of SH-bands as the blue bars show. The shading costs grow quadratically as the number of SH coefficients l^2 . For fewer bands, this is barely noticeable, because direct lighting and other rendering costs hide the work. The cache lighting is the only step that performs differently if runtime projection is enabled. Without the optimization, its costs grow faster than that of the shading. With RTP it changes much slower from 0.78 ms for two bands to 1.95 ms for eight bands. Surprisingly, three and four bands (0.99/1.02 ms) as well as five and six bands (1.29/1.31 ms) have nearly the same timings. This is caused by register pressure from the l^2 temporary


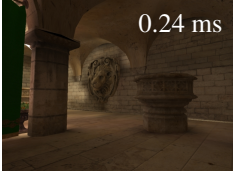






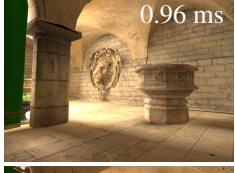




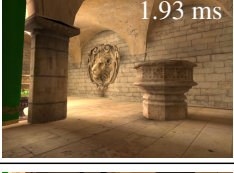

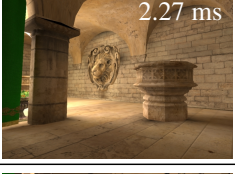
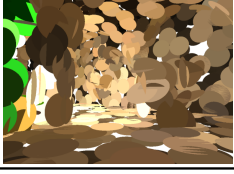
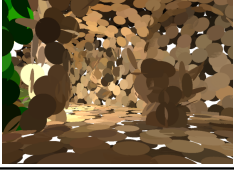
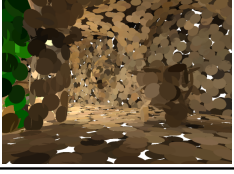
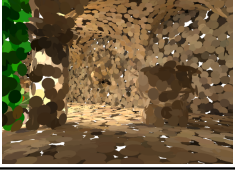
		Number of Surfels (Timings for Surfel Relighting+Pull)			
		16k (0.03 + 0.20 ms)	32k (0.04 + 0.38 ms)	64k (0.06 + 0.76 ms)	128k (0.08 + 1.72 ms)
Links per Cache (Timings for Cache Relighting)	32	 0.22 ms	 0.24 ms	 0.25 ms	 0.25 ms
	64	 0.41 ms	 0.43 ms	 0.45 ms	 0.49 ms
	128	 0.96 ms	 0.99 ms	 1.08 ms	 1.17 ms
	254	 1.81 ms	 1.93 ms	 2.04 ms	 2.27 ms
Surfels					

Figure 10. 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 overlaid 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.



RGB coefficient vectors.

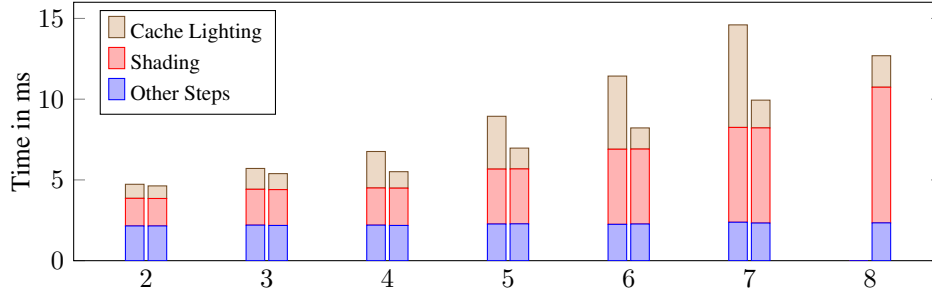


Figure 11. Influence of the number of SH bands l on the performance for the configuration in Figure 8. The right bars show results for runtime projection.

5.3.3. Chrominance Compression

The influence of chrominance compression on the memory requirements is detailed in Section 5.3.4. A qualitative comparison can be found in Figure 8. Table 3 compares the performance gain of different compression settings. In most cases there is a continuous small time-quality tradeoff. However, in two configurations (3:4 and 7:8), the compression overhead crosses a boundary with a severe performance penalty leading to even slower results. We found that two bands for chrominance are usually too few and that the difference between three and four bands is still perceptible. Hence, using the compression with our algorithm should only be considered for higher number of bands (6+). Otherwise, the quality loss is not worth the small performance gain.

		SH-bands Y						
		2	3	4	5	6	7	8
SH-bands $CoCg$	2	2.48	2.69	2.95	3.59	3.78	4.50	5.35
	3	-	3.21	3.41	4.01	4.07	5.22	5.83
	4	-	-	3.33	4.14	4.39	5.35	6.31
	5	-	-	-	4.69	5.67	6.26	6.78
	6	-	-	-	-	5.95	6.94	7.80
	7	-	-	-	-	-	7.61	11.89
	8	-	-	-	-	-	-	10.35

Table 3. Influence of chrominance compression on the performance (GTX980) with runtime projection enabled. Times are given for Cache-Lighting + Shading in ms. The main diagonal shows times without any compression (RGB mode).

5.3.4. Memory Consumption

The memory consumption of our algorithm is determined by three factors: surfels, caches, and links. Figure 5 gives an overview of all stored buffers and Table 4 extends this by an example.

	Instance	Count ¹	Total
Surfels (3)	12 B	128k	1.46 MiB
Hierarchy (1) (2) (4) (7)	34 B	256k	8.30 MiB
Caches (8) (10)	96 B	18.7k	1.71 MiB
Link SH2/RTP (5.1)	12 B	2689k	30.08 MiB
Link SH4 (5.2)	36 B	2689k	92.31 MiB
Light map Indices (6)	4 B	16k	0.06 MiB
LM Intermediate (9)	4 B	132k	0.51 MiB

Table 4. Memory consumption of individual components for the high quality settings.¹ The numbers in column one reference the textures of Figure 5. Using the runtime projection for links (Section 4.6) requires the same memory as two-band SH but supports a dynamic band number.

Textures (1), (2), (4), and (7) all scale with $2n_S$ and (3) with n_S . Hence, the memory consumptions dependent on surfel count is $n_S \cdot 80$ bytes. For $128k$ surfels, this total is 9.77 MiB.

The link buffer has different sizes for the two proposed implementations. Without runtime projection it takes $4n_C \lceil n_L/2 \rceil (2+l^2)$ bytes and $12n_C n_L$ bytes with the optimization. For Sponza setting 1 these totals are 92.31 MiB and 30.8 MiB, respectively. Chrominance compression does not reduce this footprint as links are monochromatic anyway.

The final cache textures (8) and (10) have 8 bytes per texel and $\lceil l^2/4 \rceil$ layers which leads to $24R(\lceil l_y^2/4 \rceil + \lceil l_{CoG}^2/4 \rceil)$ bytes in total where R (resolution) is the total number of texels. For our Sponza setting¹ R is $24 \times 8 \times 12 + 128 \times 128 = 18688$ resulting in 1.71 MiB memory consumption. The number is different from $n_C = 10585$ because of redundancy in the light map after copying. I.e., n_C is the number of unique caches which need to be computed. In addition, the intermediate buffers (6) and (9) for the light-map generation take 64 KiB and 517.6 KiB. Therefore, the size of buffer (9) depends on the number of caches in the light map which is scene dependent. If all 128×128 texels have their own cache, 1 MiB is required, but in that case copying is useless and both buffers (6) and (9) could be removed.

In Figure 8, the total memory consumption is given in the captions of all shown settings. Without changing n_S, n_C , or n_L (only changing SH bands l), the requirements range from 41.2 MiB to 209.7 MiB where values around 45 MiB are necessary for practical settings.

The total memory consumption for the high-quality setup in the Crytek Sponza scene is 104.3 MiB and 32.2 MiB for the moderate setup. This is comparable to other techniques like VCT and C-LPV. If SHs are not stored and link projection is done at runtime, the costs for the two setups become instead 42.8 MiB and 19.3 MiB, 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 , or 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 8 and supplied video in the supplementary materials) and allows more bands without higher memory costs.

6. Conclusions and Future Work

We proposed a new algorithm of cache-based indirect lighting that relies on pre-computations. Compared to similar methods, this algorithm is faster and requires less memory. It natively supports any type of dynamic lighting, including multiple diffuse indirections at very low costs. In particular, the runtime projection implementation requires few memory and computation costs and should always be preferred.

We also experimented with higher-order SHs for better glossy reflections. Compared to the necessary computation cost, the visual gain above six bands is small. Using chrominance compression can help to lower the costs for a higher number of bands. However, for the usual numbers of three or four bands, it does not improve the performance much.

Due to the precomputations, dynamic objects can only be shaded and do not cast indirect shadows or reflect light back into the scene. 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 because shadow and reflections of everything else are already included.

Another possible optimization is to reduce the number of caches dependent on the view. One option is the dynamic cache allocation from [Vardis et al. 2014] and another option 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.

In addition to sparse cache selection, a dynamic link selection is interesting. Currently only geometric visibility is considered. The albedo or the current lighting is not included. It is conceivable that the number of links could be reduced based on the current flux at runtime.

Acknowledgements

This work is partially supported by the German Research Foundation (DFG), Grant Nr. GR 3833/3-1. This work is based on an earlier work: Precomputed Illuminance Composition for Real-Time Global Illumination, in *I3D '16 Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pp. 129–137, ACM, 2016, <http://dx.doi.org/10.1145/2856400.2856407>.

A. Projecting the Transport Factor to SH

In Section 3.1, we formulated the light transport depending on 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. Therefore, 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 $p = 1/4\pi$:

$$c_{s,i} = \frac{4}{N} \sum_{j=1}^N y_i(\omega_j) V_s(\omega_j). \quad (8)$$

Equation (8) computes the correct solution, but casting N rays is computationally 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 :

$$w(\omega) = \frac{A_s \langle \mathbf{m}_s, \omega \rangle^+}{d^2} \cdot \frac{1}{4\pi}.$$

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 (8) and replacing the samples by the subset N' into the direction of the sender yields the final result (Equation (2)):

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

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 (4))

and an SH rotation (Equation (5)). Equation (5) has the structure of a usual sampling $y_l^m(\mathbf{d})$ multiplied by a factor s which depends on l and n only:

$$s_l(n) = \sqrt{\frac{4\pi}{2l+1}} c_l^0. \quad (9)$$

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 into Equation (9) gives

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

Finally, integrating the specular cosine lobe which is aligned in direction \mathbf{d} is computed as

$$c_l^m = s_l(n) y_l^m(\mathbf{d}).$$

C. Zonal Harmonics for Runtime Projection

In Section 4.6 small area segments are projected to the zonal harmonics by Equation (6). Each cone is parameterized by the half-open angle α . The results for the first 8 bands are

$$\begin{aligned} &- 2\pi(\cos \alpha + 1); \\ &- \pi(\cos^2 \alpha + 1); \\ &- \pi(\cos^3 \alpha + \cos \alpha); \\ &- \frac{\pi}{4}(5 \cos^4 \alpha - 6 \cos^2 \alpha + 1); \\ &- \frac{\pi}{4}(7 \cos^5 \alpha - 10 \cos^3 \alpha + 3 \cos \alpha); \\ &- \frac{\pi}{8}(21 \cos^6 \alpha - 35 \cos^4 \alpha + 15 \cos^2 \alpha - 1); \\ &- \frac{\pi}{8}(33 \cos^7 \alpha - 63 \cos^5 \alpha + 35 \cos^3 \alpha - 5 \cos \alpha); \\ &- \frac{\pi}{64}(429 \cos^8 \alpha - 924 \cos^6 \alpha + 630 \cos^4 \alpha - 140 \cos^2 \alpha + 5). \end{aligned}$$

Given that $\cos \alpha$ is a precomputed constant, all terms are simple polynomials which can be evaluated quickly. Then, the resulting values are rotated into the correct direction using Equation (5).

References

- CLINE, D., JESCHKE, S., WHITE, K., RAZDAN, A., AND WONKA, P. 2009. Dart Throwing on Surfaces. *Computer Graphics Forum* 28, 4, 1217–1226. URL: <http://dx.doi.org/10.1111/j.1467-8659.2009.01499.x>. 16
- CRASSIN, C., NEYRET, F., SAINZ, M., GREEN, S., AND EISEMANN, E. 2011. Interactive Indirect Illumination Using Voxel Cone Tracing. *Computer Graphics Forum* 30, 7, 1921–1930. URL: <http://dx.doi.org/10.1111/j.1467-8659.2011.02063.x>. 10, 20, 21
- DONG, Z., GROSCHE, 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. URL: <https://www.mpi-inf.mpg.de/departments/computer-graphics/publications/>. 10
- 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*, IEEE, Los Alamitos, CA, 1109–1112. URL: <http://dx.doi.org/10.1109/ICIP.2011.6115621>. 15
- 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*, Eurographics Association, Aire-la-Ville, Switzerland, 321–330. URL: <https://diglib.eg.org/handle/10.2312/EGWR.EGSR04.321-330>. 11, 15
- GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. 1984. Modeling the Interaction of Light Between Diffuse Surfaces. *Computer Graphics* 18, 3, 213–222. URL: <http://doi.acm.org/10.1145/964965.808601>. 9
- GREEN, R. 2003. Spherical Harmonic Lighting: The Gritty Details. In *Archives of the Game Developers Conference*. 11
- GREGER, G., SHIRLEY, P., HUBBARD, P. M., AND GREENBERG, D. P. 1998. The Irradiance Volume. *IEEE Computer Graphics and Applications* 18, 2, 32–43. URL: <http://dx.doi.org/10.1109/38.656788>. 10
- HABEL, R., AND WIMMER, M. 2010. Efficient Irradiance Normal Mapping. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, 189–195. URL: <http://www.cg.tuwien.ac.at/research/publications/2010/Habel-2010-EIN/>. 11
- HALTON, J. H., AND SMITH, G. B. 1964. Algorithm 247: Radical-inverse Quasi-random Point Sequence. *Communications of the ACM* 7, 12, 701–702. URL: <http://doi.acm.org/10.1145/355588.365104>. 16
- HANRAHAN, P., SALZMAN, D., AND AUPPERLE, L. 1991. A Rapid Hierarchical Radiosity Algorithm. *Computer Graphics* 25, 4, 197–206. URL: <http://doi.acm.org/10.1145/127719.122740>. 9

- IMMEL, D. S., COHEN, M. F., AND GREENBERG, D. P. 1986. A Radiosity Method for Non-diffuse Environments. *Computer Graphics* 20, 4, 133–142. URL: <http://doi.acm.org/10.1145/15886.15901>. 9
- KAJIYA, J. T. 1986. The Rendering Equation. *Computer Graphics* 20, 4, 143–150. URL: <http://doi.acm.org/10.1145/15886.15902>. 9
- 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*, ACM, New York, NY, 99–107. URL: <http://doi.acm.org/10.1145/1730804.1730821>. 10, 20, 21
- KELLER, A. 1997. Instant Radiosity. In *Proc. SIGGRAPH 97, Annual Conference Series*, ACM, New York, NY, 49–56. URL: <http://dx.doi.org/10.1145/258734.258769>. 9
- KRISTENSEN, A. W., AKENINE-MÖLLER, T., AND JENSEN, H. W. 2005. Precomputed Local Radiance Transfer for Real-time Lighting Design. *ACM Trans. Graph* 24, 3, 1208–1215. URL: <http://doi.acm.org/10.1145/1073204.1073334>. 10
- 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*, Eurographics Association, Aire-la-Ville, Switzerland, 277–286. URL: <http://dx.doi.org/10.2312/EGWR/EGSR07/277-286>. 9
- LENSING, P., AND BROLL, W. 2013. LightSkin: Real-Time Global Illumination for Virtual and Mixed Reality. In *Proc. 5th Joint Virtual Reality Conference*, Eurographics Association, Aire-la-Ville, Switzerland, 17–24. URL: <http://dx.doi.org/10.2312/EGVE.JVRC13.017-024>. 10, 20
- MALVAR, H., AND SULLIVAN, G. 2003. YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range. Tech. rep., ITU-T & ISO/IEC MPEG Joint Video Team. Doc. JVT-I014r3. URL: <https://www.microsoft.com/en-us/research/publication/ycocg-r-a-color-space-with-rgb-reversibility-and-low-dynamic-range/>. 20
- MARSAGLIA, G. 2003. Xorshift RNGs. *Journal of Statistical Software* 8, 14, 1–6. URL: <https://www.jstatsoft.org/article/view/v008i14/>. 16
- MARTIN, S., AND EINARSSON, P. 2010. A Real-Time Radiosity Architecture for Video Games. In *SIGGRAPH 2010 Courses*, ACM, New York, NY. URL: <http://www.geomerics.com/wp-content/uploads/2014/03/radiosityarchitecture.pdf>. 10
- MCLAREN, J. 2014. Cascaded Voxel Cone Tracing in The Tomorrow Children. In *Computer Entertainment Developers Conference*, CEDEC. URL: <http://fumufumu.q-games.com/archives/000934.php>. 10
- MITCHELL, J., MCTAGGART, G., AND GREEN, C. 2006. Shading in Valve’s Source Engine. In *SIGGRAPH 2006 Courses*, ACM, New York, NY. URL: <http://doi.acm.org/10.1145/1185657.1185832>. 10

- MÜLLNER, D. 2011. Modern Hierarchical, Agglomerative Clustering Algorithms. *arXiv:1109.2378*. URL: <https://arxiv.org/abs/1109.2378>. 16
- NVIDIA, 2014. Voxel Global Illumination. URL: <http://www.geforce.com/hardware/technology/vxgi>. 10
- RITSCHER, T., DACHSBACHER, C., GROSCH, T., AND KAUTZ, J. 2012. The State of the Art in Interactive Global Illumination. *Computer Graphics Forum* 31, 1, 160–188. URL: <http://dx.doi.org/10.1111/j.1467-8659.2012.02093.x>. 10
- 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* 21, 3, 527–536. URL: <http://doi.acm.org/10.1145/566654.566612>. 10, 20
- SLOAN, P.-P. 2008. Stupid Spherical Harmonics (SH) Tricks. In *Archives of the Game Developers Conference*. URL: [http://www.gdcvault.com/play/273/Stupid-Spherical-Harmonics-\(SH\)](http://www.gdcvault.com/play/273/Stupid-Spherical-Harmonics-(SH)). 11, 15
- 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*, ACM, New York, NY, 103–110. URL: <http://doi.acm.org/10.1145/1944745.1944763>. 10
- VARDIS, K., PAPAIOANNOU, G., AND GKARAVELIS, A. 2014. Real-Time Radiance Caching using Chrominance Compression. *Journal of Computer Graphics Techniques* 3, 4, 111–131. URL: <http://jcgt.org/published/0003/04/06/>. 10, 20, 30

Jendersie, Johannes, Kuri, David, and Grosch, Thorsten, Precomputed Illuminance Composition for Real-Time Global Illumination, *Journal of Computer Graphics Techniques (JCGT)*, vol. 5, no. 4, 8–35, 2016
<http://jcgt.org/published/0005/04/02/>

Received: 2016-06-21
Recommended: 2016-09-16
Published: 2016-11-17
Corresponding Editor: Andrew Willmot
Editor-in-Chief: Marc Olano

© 2016 Jendersie, Johannes, Kuri, David, and Grosch, Thorsten (the Authors).
The Authors provide this document (the Work) under the Creative Commons CC BY-ND 3.0 license available online at <http://creativecommons.org/licenses/by-nd/3.0/>. The Authors further grant permission for reuse of images and text from the first page of the Work, provided that the reuse is for the purpose of promoting and/or summarizing the Work in scholarly venues and that any reuse is accompanied by a scientific citation to the Work.

