Real-time Indirect Illumination and Soft Shadows in Dynamic Scenes Using Spherical Lights

Paul Guerrero[†] Stefan Jeschke[‡] Michael Wimmer[‡]

Vienna University of Technology, Austria

Abstract

We present a method for rendering approximate soft shadows and diffuse indirect illumination in dynamic scenes. The proposed method approximates the original scene geometry with a set of tightly fitting spheres. In previous work, such spheres have been used to dynamically evaluate the visibility function to render soft shadows. In this paper, each sphere also acts as a low-frequency secondary light source, thereby providing diffuse one-bounce indirect illumination.

The method is completely dynamic and proceeds in two passes: In a first pass, the light intensity distribution on each sphere is updated based on sample points on the corresponding object surface and converted into the spherical harmonics basis. In a second pass, this radiance information and the visibility is accumulated to shade final image pixels.

The sphere approximation allows us to compute visibility and diffuse reflections of an object at interactive frame rates of over 20 fps for moderately complex scenes.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: 3D Graphics and Realism

1. Introduction

Realistic illumination is one of the cornerstones of convincing computer graphics. Since a full global illumination solution is inherently complex, recent research has focused on simplifying the problem in various ways to allow real-time calculation. The earliest approach (radiosity) was to assume diffuse materials, static scene and static lighting, and precompute illumination. Precomputed radiance transfer (PRT) allows more general BRDFs and dynamic lighting, but the scene still needs to be static. PRT also made the concept of environment lighting popular, where the incoming radiance is given by an environment map instead of individual light sources.

The restriction of static scenes can be lifted by reducing the order of interreflections that are taken into account. For example, Ren et al. [RWS*06] consider only *direct* illumination by the environment lighting, which gives the effect of soft shadows due to the environment and the dynamic scene objects which act as shadow casters (blockers). However, secondary reflections are not taken into account, leading to a general over-shadowing of the scene.

In this paper, we introduce a method to compute *indirect* illumination due to diffuse reflections from dynamic objects under environment lighting. Figure 1 shows images from an animated scene rendered with the new method. Similar to Ren et al. [RWS*06], we use spheres to approximate dynamic objects, however in our method, the spheres also act as secondary light sources due to diffuse reflections, so-called *spherical lights* (SL). The light emitted in each direction is represented in the spherical harmonics basis. Contributions from multiple spheres are accumulated in log space, using SH exponentiation [RWS*06]. Like the method of Ren et al. [RWS*06], our method is best suited for diffuse reflecting geometry that can easily be approximated with spheres.

The main contribution of this paper is a method that interactively computes shadowed single-bounce diffuse indirect illumination (i.e. light is shadowed in the first bounce before being reflected) for dynamic objects in real time with a

[†] paul.guerrero@chello.at

[‡] {jeschke|wimmer}@cg.tuwien.ac.at

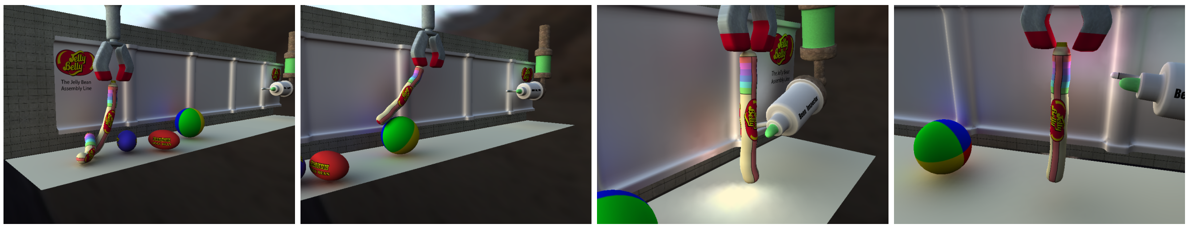

Figure 1: *Images from the jelly bean assembly plant animation, rendered at 14 fps. Our method handles local light sources and a class of object deformations which we call "soft deformations" on the fly.*

minimum of precomputed information. Object translations, uniform scaling, rotations and "soft" deformations (see Section 7.1) can be handled without affecting performance.

2. Previous Work

Static scenes: Several methods have been proposed to handle global illumination effects like soft shadows and indirect illumination in real time, most of them are based on precomputed radiance transfer (PRT) or ambient occlusion. Using the spherical harmonics basis to store a *precomputed radiance transfer* operator for each receiver point, Sloan et al. [SKS02] calculate real-time soft shadows and interreflections in static scenes under dynamically varying lighting conditions. There are many papers about extending PRT in static scenes to handle local textures, high-frequency lighting and BRDFs etc., we will however focus only on methods for dynamic scenes.

Ambient Occlusion [Lan02] describes the average visibility on the hemisphere of a receiver point. Ambient occlusion methods for soft shadows [KL05, MMAH06] and indirect lighting [Bun04] in dynamic scenes have been proposed. Since ambient occlusion methods average visibility over all directions, the resulting shadows are mostly dependent on the proximity to objects, not on actual occlusions of light sources. Our method averages visibility and radiance over the projected area of each sphere in contrast to the entire hemisphere and is therefore more accurate.

Direct illumination in dynamic scenes: Two methods that use PRT to calculate direct illumination and shadows from environment lighting in dynamic scenes are hemispherical rasterization [KLA04] and shadow fields [ZHL*05, TJCN06]. A third method by Ren et al. [RWS*06] approximates each scene object with sphere sets and efficiently accumulates the shadowing effect of multiple blocker spheres at run time using SH exponentiation. With this method, Ren et al. can handle real-time soft shadows in scenes of medium complexity. In this paper, we extend this method to handle diffuse indirect illumination.

Indirect illumination in dynamic scenes: Mei et al. [MSW04] use Spherical Radiance Transport Maps to handle glossy light interreflections and soft shadows in dynamic scenes. General indirect illumination, however, can not be handled by their method. Local, deformable PRT (LDPRT) [SLS05] can handle radiance transfer effects on deformable models, but is limited to local features, such as bumps or wrinkles, and single objects. Sun and Mukherjee [SM06] describe a method to handle all-frequency lighting at interactive frame rates in scenes with at most one moving object. Only one of viewpoint, lighting and scene geometry is allowed to change at a time. Recently, Laine et al. [LSK*07] proposed an algorithm for single-bounce indirect illumination using virtual point lights (VPLs) to approximate surface reflections. Dynamic objects can be handled, but they can only receive, not generate, indirect illumination. Dachsbacher and Stamminger [DS05, DS06] calculate indirect lighting in image-space using Reflective Shadow Maps and light splatting. Their method demonstrates impressive global illumination effects like caustics and refractive surfaces at relatively high frame rates, but is limited to directional point lights. Additionally, shadows from direct lighting are handled using traditional shadow maps and there is no occlusion of indirect lighting. In a second paper [DSDD07], Dachsbacher et al. propose a discrete finite element method for computing multi-bounce indirect illumination, using "Antiradiance" to avoid explicit visibility computations. They demonstrate their method on fairly complex scenes and glossy objects. On the downside, only bounded object motion is allowed, since the finite elements affected by the moving object have to be known in advance. Object motion also degrades performance, since the number of pairs of finite elements potentially affecting each other increases with the extent of the motion. Additionally, shadowing from direct lighting has to be handled separately, e.g. with shadow maps. In a concurrent and independent work, Sloan et al. [SGNS07] propose a method to compute soft global illumination using spherical proxies. This method is similar to our method in that it is built on the method of Ren et al. [RWS*06] and extends it to include indirect illumination. However, Sloan et al. ignore inter-proxy occlusion effects (i.e. the indirect light is not occluded) and occlusion of light incident on the proxies. Also, they demonstrate their method only on uniformly-coloured objects.

Iwasaki et al. [IDYN07] presented a PRT-based method that can handle multi-bounce interreflections and shadows in dynamic scenes, building on the shadow fields method mentioned above. Interreflections are handled by approximating the irradiance on objects with a linear combination of PCA basis functions. Pan et al. [PWL*07] present a similar method in which they precompute radiance transport operators from incident radiance at some point near the object to exit radiance at some other point in the space surrounding the object and store them in a Radiance Transfer Field. We will discuss these last two methods and compare them to our method in Section 8.

Our method is limited to single-bounce diffuse indirect illumination (although an extension to multiple bounces seems straightforward). Unlike previous methods, the SL approximation enables more dynamic effects, like soft deformations, a class of deformations we will define in Section 7.1. All objects can be translated, scaled uniformly and rotated without affecting performance. Additionally, the precomputed dataset for each object is relatively small, keeping memory requirements low.

3. Overview

The main idea of our new method is to calculate light interaction on simplified version of the scene where every object is approximated using a set of spheres. Soft shadows due to direct environment lighting are calculated by accumulating the blocking spheres using SH exponentiation as [RWS*06]. For indirect illumination, however, the spheres need to act as light sources. To make this possible, for each sphere we calculate incoming radiance at a number of given sample points (Figure 2, top right) and calculate an intensity distribution for the whole sphere from this (Figure 2, middle). During shading, we accumulate the contributions from all visible spherical lights (SLs) (Figure 2, bottom).

More specifically, our method needs three steps to render soft shadows and diffuse indirect illumination (see Figure 2), one precomputation step and two runtime passes:

- In a precomputation step, we tightly bound each scene object with a given number of spherical lights (SLs). We determine the position and radius of each SL in the local coordinate system of the bounded object. For each SL, we also define a set of sample points on the surface segment of the scene object that is bounded by the SL. These *surface sample points* are used in the first pass to compute diffuse reflections on the object's surface and update the light intensity distributions of the SLs.
- In the first pass, we approximate the diffuse reflections of each scene object with the set of SLs bounding the object. For this purpose, we compute the diffuse reflections at each surface sample point on an object's surface using environment lighting only. The light intensity distribution of the SLs are updated in each frame to best match these diffuse reflection values. Typically, the number of surface sample points in a scene is relatively small when compared to the number of receiver points used for final object shading. This allows us to efficiently update the SLs in each frame. The SLs are also used as blockers to determine the visibility of the environment and other objects, as described in [RWS*06].
- In the second pass, we accumulate the light from all SLs at each final receiver point. First, the unoccluded light from an SL incident at a receiver point is found. Then, the visibility of the SL at the receiver point is calculated using all closer SLs as blockers. The effect of multiple blocker SLs is accumulated using SH exponentiation [RWS*06]. The light of an SL incident at a receiver point is darkened according to the visibility of the SL, similar to ambient occlusion, but limited to the projected area of each SL instead of the entire receiver point hemisphere.

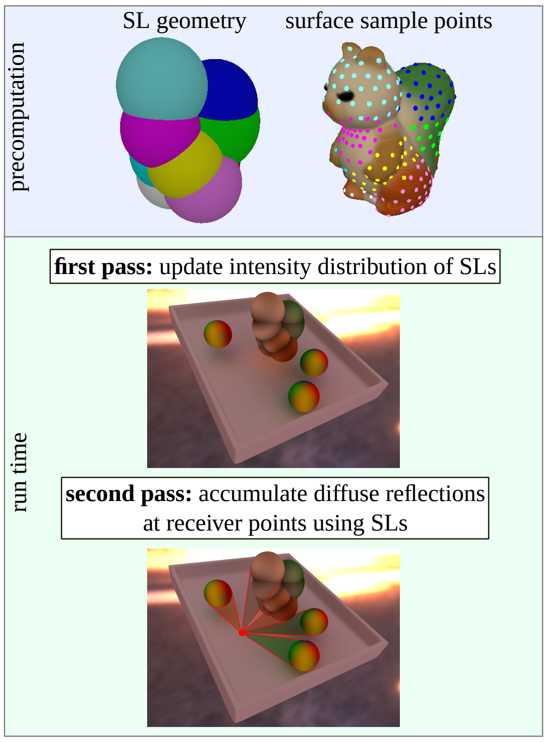
precomputation
SL geometry
surface sample points
run time
first pass: update intensity distribution of SLs
second pass: accumulate diffuse reflections at receiver points using SLs

Figure 2: *Overview of the steps in our method.*

The following sections explain these steps in more detail.

4. Precomputation of Spherical Lights

For each scene object, we precompute the geometry of its SLs (i.e. position and radius) as well as a set of surface sample points for each SL. In the following two sections, we will describe the construction of SL geometry and surface sample points for a scene object.

4.1. Spherical Light Geometry

The position and radius of SLs relative to a scene object are found using a variational method for bounding sphere set

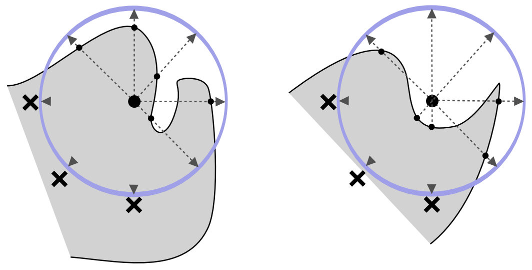

Figure 3: *Surface sampling. Black dots are surface sample points and crosses mark sample directions without surface sample point. The left figure shows surface sampling for an SL with center inside the sampled object, the right case for an SL with center outside the object.*

approximations [WZS*06]. The goal is to find a bounding sphere set $\{S_i\}$ for an object O that has minimal outside volume (see Figure 2, top left). Outside volume is the volume inside the spheres but outside the object and defined as follows:

$$T(\{S_i\}, O) = \sum_{i=0}^{n_s} V(S_i/O) \tag{1}$$

where n_s is the number of spheres in $\{S_i\}$ and $V(S_i/O)$ the volume inside the sphere S_i but outside O. The bounding property ensures that there are no 'holes' in the sphere set approximation and minimizing the outside volume gives us a tight fit on the object's geometry.

A variant of Lloyd clustering [Llo82] is used to minimize the outside volume of a sphere set with a given number of spheres. For details on the method see [WZS*06].

4.2. Surface Sample Points

Each SL is assigned a set of surface sample points at fixed positions on an object's surface (see Figure 2, top right). At run time, diffuse reflections will be computed at each sample point of an SL. The light intensity distribution of the SL is updated in each frame to best match these diffuse reflection values.

Surface sample points for a set of SLs $\{S_i\}$ and object O are constructed as follows: First, a segment of the surface of object O is assigned to each SL. An SL S_i gets the segment P_{S_i} consisting of the set of surface points with smallest distance to S_i. We use a distance measure d that takes into account the radius of S_i:

$$d(p, S_i) = \frac{||p - x_{S_i}||}{r_{S_i}} \tag{2}$$

where p is a surface point on O, x_{S_i} the center of S_i and r_{S_i} the radius. This ensures that each part of the surface is only approximated by a single SL.

Now, we shoot rays from the center of each SL S_i in a predefined set of approximately evenly spaced directions $\{\omega_j\}$. The set of directions $\{\omega_j\}$ is derived from the vertices of a geodesic sphere. Surface sample points are placed at the intersections points of these rays with the back side of the object's surface. Intersections with the front side are ignored, since these surfaces are not visible when looking at the SL.

When shooting rays from an SL S_i, the surface of object O may be hit more than once in any given direction ω. In general, there are three possible cases (see Figure 3):

- There are one or more ray intersection points with the surface segment P_{S_i}. In this case, the farthest intersection point contained in P_{S_i} is used.
- There are only intersection points outside P_{S_i}. Since the set of SLs $\{S_i\}$ bounds the object O, there must be another SL S_o of $\{S_i\}$ in direction ω that approximates the diffuse reflections in that direction. Discarding the surface sample point and letting S_i emit no light in that direction is correct, because S_i will always be occluded by S_o in direction ω. By letting the SLs emit no light in this direction, we precompute the occlusion effects of intersecting SLs. These precomputed occlusions will be useful at run time when calculating the visibility of an SL (Section 6.2).
- There are no intersections. For closed objects, which we assume here, this means that the center of S_i is outside the object. In this case, we shoot a ray in the opposite direction ω', this time allowing only the front sides of surfaces to be hit. If this ray does not hit a surface either, we discard the sample point for this direction, as in the previous case. This kind of sampling causes approximation error, since relatively small parts of the object's surface may be projected to relatively large parts on the surface of the SL, giving these small regions too much influence on the emitted light. However, this case is very rare for usual geometry as the centers of the SLs are usually inside the object.

We store surface *position* and *normal* for each surface sample point. For the surface *albedo* (i.e. the surface color), however, this kind of sampling is too sparse, as it may miss important textural detail. Instead, we sample the surface albedo by rendering the surface segment P_{S_i} of each SL S_i to a cubemap, and convert this to spherical harmonics. The following procedure is used to render a cubemap and ensures that it only contains samples that respect the sampling rules described above.

For an SL S_i and object O, we position the camera at the center of S_i. We only use the set of triangles of O that are closest to S_i, using the distance measure d defined above. For efficiency reasons, we do not split a triangle if parts of it are closer to another SL, but instead assign it to both SLs. Objects are usually tessellated finely enough so that this approximation is not noticeable in the final result. The camera is rotated to look in the direction of each cubemap face. We render the triangle set assigned to S_i normally if the sphere center is inside the object and without clipping planes such that triangles behind the camera are rendered, too, but have a lower depth, if the sphere center is outside the object (see rule 3 above). Triangles are rendered back facing with inverted depth function.

The resulting cubemaps are projected to the spherical harmonics (SH) basis, resulting in a surface albedo SH coefficient vector (SH vector) $\mathbf{c}_{S_i}$ for each SL S_i.

5. First Pass: Approximation of Diffuse Reflections with Spherical Lights

5.1. Calculation of Diffuse Reflections at Sample Points

In order to calculate the incoming radiance for the sample points on secondary light sources (SLs), we use the same method that Ren et al. [RWS*06] use to calculate soft shadows for direct illumination. SLs act as blockers and environment lighting as light source. We will review this method here briefly.

We assume diffuse BRDFs, so the exit radiance at a sample point p (i.e. the diffuse reflections) can be calculated using the triple product of visibility, BRDF and lighting:

$$L_p = \int_\Omega V_{p,env}(\omega_i)\rho_n(\omega_i)L_{env}(\omega_i)d\omega_i \tag{3}$$

where ω_i is the incoming radiance direction and Ω is the sample point hemisphere. L_{env} is the environment lighting and ρ_n the BRDF for normal n. In our case, ρ_n does not include the surface albedo, since the albedo $\mathbf{c}_{S_i}$ of an SL has been precomputed separately (see Section 4.2). The albedo is factored in at a later point (see Section 5.2). We assume that ρ_n is constant over the surface of each object. The main problem is finding the environment visibility $V_{p,env}$ since it may change in every frame, depending on the scene configuration. All three functions are represented in the spherical harmonics basis, where triple product integrals can be evaluated efficiently [NRH04].

Zhou et al. [ZHL*05] compute the environment visibility SH vector $\mathbf{V}_{p,env}$ as the product of n blockers:

$$\mathbf{V}_{p,env} = \mathbf{B}_{p,O_1} * \mathbf{B}_{p,O_2} * \ldots * \mathbf{B}_{p,O_n} \tag{4}$$

where $*$ denotes the SH product [NRH04] and $\mathbf{B}_{p,O_i}$ is the SH projection of the blocker function B_{p,O_i} of object O_i at sample point p:

$$B_{p,O_i}(\omega) = \begin{cases} 0 & \text{if } O_i \text{ blocks in direction } \omega \\ 1 & \text{otherwise.} \end{cases} \tag{5}$$

Ren et al. [RWS*06] accelerate this method by accumulating the log of the visibility of spherical blockers, avoiding expensive SH products:

$$\mathbf{V}_{p,env} = \exp\left(\mathbf{G}_{p,S_1} + \mathbf{G}_{p,S_2} + \ldots + \mathbf{G}_{p,S_n}\right) \tag{6}$$

$$\mathbf{G}_{p,S_i} = \log(\mathbf{B}_{p,S_i}) \tag{7}$$

where exp denotes the SH exponential [RWS*06], log the SH logarithm [RWS*06] and $\mathbf{B}_{p,S_i}$ denotes the blocker SH vector of the spherical blocker S_i at sample point p. The SH exponential can be calculated efficiently at run time. The log SH vector $\mathbf{G}_{p,S_i}$ of spherical blockers is circularly symmetric. It can be parametrized by angular radius θ of the spherical blocker at a sample point p and direction ω of the spherical blocker on the hemisphere of p. Ren et al. precompute the log $\mathbf{G}_\theta$ for spherical blockers of any angular radius θ and $\omega = (0,0)$. Since the precomputed blockers are circularly symmetric, they can be represented in a subset of the spherical harmonics basis, the zonal harmonics (ZH) basis [SLS05]. At run time, the precomputed $\mathbf{G}_\theta$ are looked up by angular radius θ and then rotated to the correct direction ω on the hemisphere of the sample point using the simple rotation rules for ZH coefficient vectors [SLS05].

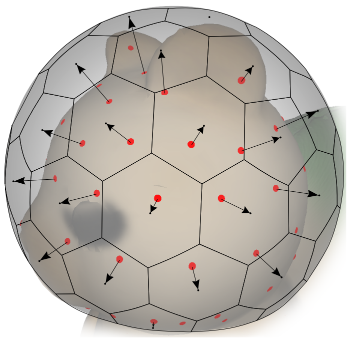
Figure 4: *The light intensity distribution of an SL is updated by projecting the diffuse reflection values (red) to the surface of the SL. Nearest neighbour interpolation is used between these values, resulting in a function consisting of the Voronoi cells of the sample directions.*

If we assume fixed environment lighting and BRDF, we can precompute the product of BRDF and lighting. Equation 3 then reduces to a simple dot product between the environment visibility SH vector $\mathbf{V}_{p,env}$ and the precomputed product of BRDF and lighting $\mathbf{L}_{env,\rho_n}$:

$$L_p = \mathbf{V}_{p,env} \cdot \mathbf{L}_{env,\rho_n} \tag{8}$$

5.2. Updating the Light Intensity Distributions

After we have calculated the diffuse reflections at each sample point, we update the light intensity distribution of each SL to approximate the diffuse reflections computed at its set of sample points. Again, the spherical harmonics basis is used to represent the light intensity distribution of each SL, so we have to project the diffuse reflections to the SH basis.

We use Monte Carlo integration with constant probability distribution [Lep78] to project the diffuse reflection values L_{p_j} of the m surface sample points p_j of SL S_i to the SH basis $\mathbf{y}$:

$$\mathbf{L}_{S_i} = \frac{4\pi}{N}\sum_{k=1}^{N} L(\hat{\omega}_k)\mathbf{y}(\hat{\omega}_k) \tag{9}$$

where $\{\hat{\omega}_k\}$ is a set of N evenly distributed sample directions, generated using jittered stratification [Lep78]. Note that we now have N sample directions $\{\hat{\omega}_k\}$ and m surface sample points with directions $\{\omega_j\}$ for sphere S_i ($N \gg m$). The function $L(\hat{\omega}_k)$ uses nearest neighbour interpolation between the diffuse reflection values of each surface sample point, i.e. it is a function consisting of the Voronoi cells of the surface sample point directions $\{\omega_j\}$ (see Figure 4).

Since the sample directions $\{\omega_j\}$ are fixed, we can speed up the SH projection by precomputing the SH basis function sums for each Voronoi cell. At run time, each sum is multiplied with the corresponding diffuse reflection value L_{p_j}:

$$\mathbf{L}_{S_i} = \frac{4\pi}{N}\sum_{j=0}^{m} L_{p_j}\mathbf{Y}_j, \quad \mathbf{Y}_j = \sum_{k=1}^{N_j}\mathbf{y}(\hat{\omega}_k^j) \tag{10}$$

where $\mathbf{Y}_j$ is the precomputed SH basis function sum for sample direction ω_j of surface sample point p_j. N_j is the number of sample directions $\hat{\omega}_k^j$ in the Voronoi cell of receiver point p_j and $\sum_{j=0}^{m} N_j = N$. By precomputing the SH basis function sums, the computational complexity of the SH projection is reduced from $O(N)$ to $O(m)$.

Finally, the light intensity distribution SH vector $\mathbf{L}_{S_i}$ is multiplied with the surface albedo SH vector $\mathbf{c}_{S_i}$ (see Section 4.2) to get the final light intensity distribution $\mathbf{L}^c_{S_i}$ of the SL S_i, including surface color:

$$\mathbf{L}^c_{S_i} = \mathbf{L}_{S_i} * \mathbf{c}_{S_i} \tag{11}$$

6. Second Pass: Accumulating Diffuse Reflections at Receiver Points

In the second pass, we compute the final exit radiance at each receiver (i.e., final image) point. The direct component is computed using the method by Ren et al. (see Section 4.2). To calculate the indirect component, we accumulate diffuse reflections from all visible SLs, using the updated light intensity distributions $\mathbf{L}^c_{S_i}$. The total exit radiance at a receiver point p is the sum of direct exit radiance L_p and indirect exit radiance L^I_p.

$$L^T_p = L^I_p + L_p \tag{12}$$

We use the light emitted by the SLs to approximate incident diffuse reflections:

$$L^I_p = \sum_{i=0}^{n_S}\int_\Omega L_{p,S_i}(\omega_i)V_{p,S_i}(\omega_i)\rho_n(\omega_i)d\omega_i \tag{13}$$

where n_S is the number of SLs, L_{p,S_i} is the unoccluded light from SL S_i incident at receiver point p and V_{p,S_i} is the visibility of S_i at p. ρ_n is the BRDF for normal n. Note that since we assume diffuse BRDFs in our implementation, $\rho_n(\omega_i) = n\cdot\omega_i$. Other types of view-independent low-frequency BRDFs are also possible in the second pass. View-dependent BRDFs are left for future work (see Section 9).

Equation 13 would require evaluating relatively expensive triple product integrals for each SL. We use a more approximate approach instead: First, we decompose Equation 13 as follows:

$$\begin{aligned} L^I_p &= \textstyle\sum_{i=0}^{n_S}\overline{L}_{p,S_i}\widetilde{V}_{p,S_i} \\ \overline{L}_{p,S_i} &= \frac{\int_\Omega L_{p,S_i}(\omega_i)V_{p,S_i}(\omega_i)\rho_n(\omega_i)d\omega_i}{\int_\Omega V_{p,S_i}(\omega_i)\ \rho_n(\omega_i)d\omega_i} \\ \widetilde{V}_{p,S_i} &= \textstyle\int_\Omega V_{p,S_i}(\omega_i)\rho_n(\omega_i)\ d\omega_i \end{aligned} \tag{14}$$

We call $\overline{L}_{p,S_i}$ the average radiance and $\widetilde{V}_{p,S_i}$ the average visibility of an SL at p. $\widetilde{V}_{p,S_i}$ can be determined with a simple dot product of the BRDF SH vector ρ_n with the visibility SH vector $\mathbf{V}_{p,S_i}$. For the purpose of finding the average radiance of an SL, we approximate the BRDF term ρ_n to be constant over the area of the SL on the receiver point hemisphere. Since this area is typically small and both the BRDF and the intensity distribution of the SL are low-frequency, the approximation error is minimal. This removes the dependence of $\overline{L}_{p,S_i}$ on the BRDF:

$$\overline{L}_{p,S_i} \approx \widetilde{L}_{p,S_i} = \frac{\int_\Omega L_{p,S_i}(\omega_i)V'_{p,S_i}(\omega_i)}{\int_\Omega V'_{p,S_i}} \tag{15}$$

We use the unoccluded visibility V'_{p,S_i} of S_i at p instead of V_{p,S_i}, effectively averaging the incident radiance over the whole facing sphere area, instead of only the visible area. We then 'darken' this area by multiplying with the average visibility value $\widetilde{V}_{p,S_i}$. This can be thought of as per-SL ambient occlusion. The effects of this approximation error depend mainly on the resolution of the SL approximation (i.e. higher resolution introduces less approximation error) and the amount of occlusion in a scene (i.e. there is no approximation error if there are no occlusions between SLs). This approximation does not produce objectionable artifacts and since the light intensity distribution of SLs is low-frequency, the approximation error is noticeable only in extreme cases.

$\widetilde{L}_{p,S_i}$ is the average radiance of the SL S_i in direction to p. In the next section, we show how to find the average radiance $\widetilde{L}_{p,S_i}$ using a simple geometric term. In Section 6.2 we describe how to calculate the average visibility value $\widetilde{V}_{p,S_i}$.

6.1. Average Radiance of a Spherical Light

The average radiance $\widetilde{L}_{p,S_i}$ of an SL S_i in direction of a receiver point p is found by multiplying the light intensity distribution of S_i with the cosine of the surface inclination as seen from p (the angle γ in Figure 5, right):

$$\widetilde{L}_{p,S_i} = \frac{\int_S L_{S_i}(\omega_S)\max(0,\cos\gamma(\omega_S))d\omega_S}{\int_S \max(0,\cos\gamma(\omega_S))d\omega_S} \tag{16}$$

where S is the sphere of the SL. The term $\cos\gamma$ is circularly symmetric and can be parametrized by the angular distance α from the direction SL center to p:

$$\cos\gamma(\alpha) = \max\left[0, \sin\left(\operatorname{atan}\left(\frac{1-\sin\theta\ \cos\alpha}{\sin\theta\ \sin\alpha}\right) - \alpha\right)\right] \tag{17}$$

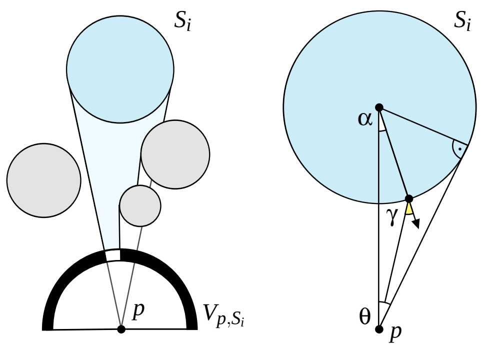

Figure 5: *The left diagram shows the visibility function V_{p,S_i} for the SL S_i on the hemisphere of the receiver point p. Zero values are black, values of 1 are white. The right diagram shows the geometry of the average radiance of S_i to p.*

where θ is the angular radius of the SL S_i at the receiver point p. In a precomputation step, this function is normalized and tabulated as a ZH vector $\mathbf{w}_\theta$ by angular radius θ. At run time, $\mathbf{w}_\theta$ is looked up by angular radius and rotated to the direction of the receiver point using the simple rotation rules for ZH vectors [SLS05]. The average radiance from S_i in direction to p can then be computed with a simple dot product of the rotated ZH vector $\mathbf{w}'_\theta$ and the light intensity distribution SH vector $\mathbf{L}^c_{S_i}$ of the SL S_i:

$$\widetilde{L}_{p,S_i} = \mathbf{w}'_\theta \cdot \mathbf{L}^c_{S_i} \tag{18}$$

6.2. Visibility of a Spherical Light

The visibility value $\widetilde{V}_{p,S_i}$ of an SL S_i at a receiver point p is calculated using all SLs closer to p as blockers. The blocker function of an SL S_i at p is defined as follows:

$$B_{p,S_i}(\omega) = \begin{cases} 0 & \text{if } S_i \text{ blocks in direction } \omega \\ 1 & \text{otherwise.} \end{cases} \tag{19}$$

The visibility function V_{p,S_i} of S_i at p is given by:

$$V_{p,S_i}(\omega) = (1 - B_{p,S_i}(\omega))\, B_{p,S_{i-1}}(\omega)\, \ldots\, B_{p,S_1}(\omega) \tag{20}$$

where all SLs S_j with $j < i$ are closer to p than S_i (see Figure 5, left). The product of these blocker functions is evaluated in the SH basis using SH exponentiation:

$$\mathbf{V}_{p,S_i} = \exp\left(\log(\widehat{\mathbf{B}}_{p,S_i}) + \mathbf{G}_{p,S_{i-1}} + \ldots + \mathbf{G}_{p,S_1}\right) \tag{21}$$
$$\mathbf{G}_{p,S_j} = \log(\mathbf{B}_{p,S_j}) \tag{22}$$

where $\widehat{\mathbf{B}}_{p,S_i}$ is the SH projection of $1 - B_{p,S_i}(\omega)$. The $\mathbf{B}_{p,S_j}$ are the same circularly symmetric blocker functions used for calculating the environment visibility (see Section 4.2). Their SH logarithms are precomputed as ZH vectors and tabulated by angular radius θ. At run time, they are looked up by angular radius and rotated to the direction of S_j.

The SH vector $\widehat{\mathbf{B}}_{p,S_i}$ however, is not well suited for the SH logarithm, since $1 - B_{p,S_i}(\omega)$ is zero in most parts of its domain. $\widehat{\mathbf{B}}_{p,S_i}$ would have to be clamped, increasing the approximation error. Additionally, the SH logarithm of $\widehat{\mathbf{B}}_{p,S_i}$ has a large vector magnitude, which is difficult to handle with SH exponentiation. Instead, we use this equivalent formulation to determine the visibility SH vector $\mathbf{V}_{p,S_i}$:

$$\begin{aligned} \mathbf{V}_{p,S_i} = {} & \exp\left(\mathbf{G}_{p,S_{i-1}} + \mathbf{G}_{p,S_{i-2}} + \ldots + \mathbf{G}_{p,S_1}\right) \\ & - \exp\left(\mathbf{G}_{p,S_i} + \mathbf{G}_{p,S_{i-1}} + \ldots + \mathbf{G}_{p,S_1}\right) \end{aligned} \tag{23}$$

Using this formulation, we avoid using $\widehat{\mathbf{B}}_{p,S_i}$, although at the cost of an additional SH exponential.

At each receiver point, the visibility SH vector $\mathbf{V}_{p,S_i}$ is then dotted with the precomputed BRDF SH vector ρ_n for normal n to get the visibility value $\widetilde{V}_{p,S_i}$ of S_i:

$$\widetilde{V}_{p,S_i} = \mathbf{V}_{p,S_i} \cdot \rho_n \tag{24}$$

Intersections of Spherical Lights are difficult to handle when computing the visibility of an SL S_i. The blocker function of an SL intersecting S_i may not be circularly symmetric and can therefore not be precomputed in the ZH basis, like the functions for non-intersecting blockers. To prevent intersection of SLs of different objects, we do not allow objects to intersect. Intersections between SLs of the same object remain constant. To precompute their blocking effect on S_i, we remove the surface sample points of S_i corresponding to directions of intersecting SLs during precomputation (the second rule in Section 4.2). Therefore, an SL does not emit light in these directions. When calculating the visibility of an SL S_i at run time, we ignore intersecting SLs, since their occlusions have already been accounted for in the light intensity distribution of S_i.

Avoiding Problems with Receiver Points Inside SLs Since SLs bound each object, each receiver point is inside of at least one SL. To avoid incorrect shadows and diffuse reflections from these SLs, we adjust SLs individually at each receiver point, using the rules described by Ren et al. [RWS*06]. SLs that contain a receiver point p are either reduced in size, so they no longer contain p, or they are completely culled. SLs on the horizon of receiver point hemispheres that intersect the tangent plane of the receiver point are also scaled down and translated until they are completely above the tangent plane. This is done to avoid discontinuities as receiver points move from inside an SL to outside. For details see [RWS*06]. These rules ensure that a receiver point is not contained in any SL that is used to compute indirect illumination and shadows at the receiver point.

7. Application and Implementation

7.1. Object Transformations

Translation and uniform scaling are trivial. Surface sample points have to be translated/scaled along with the object and for uniform scaling, the center and radius of SLs have to be scaled accordingly. Rotations of SLs are handled by rotating

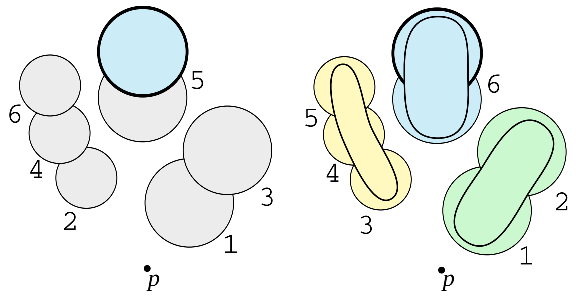

Figure 6: *Per-sphere and per-object blocker accumulation strategies. Numbers indicate the order of blocker accumulation. Left: when using per-sphere blocking, blocker SLs are accumulated in the order of distance from the receiver point p. Note that SL number* 5 *cannot be used to occlude the blue SL, since it is intersecting. Right: when using per-object blocking, the SLs are accumulated in the order of object distance.*

the weight SH vector $\mathbf{w}_\theta$ (see Section 6.1) with the inverse rotation of the SL, resulting in a correct average radiance at each receiver point. By rotating the weight vector instead of the SL, we avoid expensive rotations of surface sample points and surface albedo SH vector (see Section 4.2).

To handle object deformations for articulated motion with deformable joints, we would need to re-sample the object's surface in each frame (i.e. find surface sample points and surface albedo SH vector for each SL). Finding methods to re-sample an object's surface at run time is left for future work. However, if the object's surface deformation relative to each individual SL can be kept minimal (i.e. when bending a tall object), the error caused be keeping the original surface samples is not noticeable. We call this class of deformations *soft deformations*. By rotating and translating each SL with the average rotation and translation of all its associated vertices, we can handle soft deformations. Articulated motion that does not deform the object's surface (e.g. rigid joints) can also be handled by our method.

7.2. Per-Sphere and Per-Object Blocking

Finding the visibility of each SL as described in Equation 23 requires $\frac{n(n+1)}{2}$ SH vector additions per receiver point, resulting in a quadratic runtime complexity. To reduce the number of additions, we could start computing the visibility at the closest SL and then incrementally add blocker SLs to get the visibility at farther SLs. However, this is generally not possible, since we have to ignore intersecting SLs.

When using per-object blocking, we group SLs by objects and compute the visibility of all SLs in one object before moving to the next object. Since SLs of different objects are not allowed to intersect, we can re-use the accumulated blocking effect of one object for all following objects. In this manner, we can incrementally build up the blocking effect of all objects closer than the current object. For a particular SL, we then only have to add the blocking effect of all non-intersecting SLs in the same object to the accumulated blocking effect of all closer objects, which has already been computed. This reduces the runtime complexity to a linear complexity in the number of objects. Figure 6 illustrates the difference between per-sphere (left) and per-object blocking (right).

In our implementation and our test scenes we use both per-sphere and per-object blocking. See Section 8 for results and a more thorough discussion of both blocking methods.

7.3. Hierarchies of Spherical Lights and Local Lights

In order to speed up computations, hierarchies of SLs and receiver point clusters are constructed and used as in [RWS*06]. Starting from leaf SLs, parent SLs are constructed to bound a given number of child SLs. Receiver points far away from an object use a coarser hierarchy level to calculate shadows and diffuse reflections. A SL hierarchy cut is calculated once per frame for all receiver points in a cluster. Analogous to the ratio defined in [RWS*06], we define a *reflection ratio* to eliminate artifacts caused by using different SL hierarchy cuts in adjacent clusters.

A reflection ratio t_{S_p} is calculated once per frame at the center x_C of each receiver point cluster. It represents the best scaling of the diffuse reflection value $(\widetilde{L}_{x_C,S_p}\widetilde{V}_{x_C,S_p})$ of a parent SL S_p (see Equation 14) to match the sum of the diffuse reflection values of all child SLs S_j. It is calculated via:

$$t_{S_p} = \frac{\sum_{j=0}^{n} \widetilde{L}_{x_C,S_j}\widetilde{V}_{x_C,S_j}}{\widetilde{L}_{x_C,S_p}\widetilde{V}_{x_C,S_p}} \tag{25}$$

At run time, all receiver points in a cluster multiply the diffuse reflection values of each parent SL S_i with the reflection ratio t_{S_i} calculated at the cluster center, thus Equation 14 becomes:

$$L_p^I \approx \sum_{i=0}^{n_s} \widetilde{L}_{p,S_i}\widetilde{V}_{p,S_i} t_{S_i}^D \tag{26}$$

Local lights can be treated like SLs with user-defined intensity distributions. In the first pass, we do the same calculation for local light sources, as we do in the second pass for normal SLs.

7.4. GPU Implementation

Since we handle only low-frequency light transport effects, dense sampling is usually not required. In our implementation, we sample at each vertex and increase the geometry tessellation if necessary. We work with $n = 4$ SH bands, which is usually a good tradeoff between quality and performance.

In a precomputation step, we generate all lookup tables that are independent of the scene configuration and store them in floating-point textures. Two 256×256 pixel floating-point textures are easily enough to hold all tables.

In the first pass, we update the light intensity distribution of the SLs. First, we prepare floating-point textures on the CPU, containing information on the SLs, including position, radius and ratio vectors, and the surface albedo SH vector (see Section 4.2). In a first GPU computation step, we use these textures to calculate diffuse reflections at the surface sample points. The resulting sampled values are rendered directly to texture using an OpenGL frame buffer object. In a second GPU computation step, we project the sampled values to the SH basis and multiply with the surface albedo SH vector. The resulting updated light intensity SH vectors are rendered to textures using a frame buffer object.

The second pass calculates the final exit radiance at each vertex on the GPU, using the output from the first pass and the SL information textures. In this step, we iterate over each vertex and first sort spheres (per-sphere blocking) or objects (per-object blocking) by distance from the vertex. Then we accumulate incident radiance and blocking effect as described in Equations 12, 14 and 8. We have implemented the second pass both as vertex- and pixel shader. The vertex shader implementation directly calculates each vertex color. The pixel shader implementation is less direct, but has better performance (see Section 8.5). We first render the vertex colors to the frame buffer, then we read the result to a vertex buffer using the OpenGL function `ReadPixels`, as described in [RWS*06]. The vertex colors in the buffer are used in a standard render pass without lighting to render the final image.

8. Results and Discussion

Frame rates were measured on a PC with a Core2Quad Q6600 CPU, 2048 MB ram and a GeForce 8800 GTX. Rendering was done at a resolution of 1024×1024 using OpenGL. In our test scenes, large flat surfaces like walls and the ground plane are shaded properly, but do not contribute to indirect illumination, since this type of geometry is not well suited for sphere approximation.

8.1. Sources of Approximation Error

Apart from the sources of approximation error described by Ren et al. [RWS*06], the following are the main sources of approximation error in our method:

First, the accuracy of indirect lighting depends on the resolution of the SL approximation (i.e. how many SLs are used). The number of SH bands used for the intensity distribution of each SL is also important, because it determines how much surface detail a single SL can approximate.

Second, size adjustments of SLs at the horizon of the receiver point hemisphere (see Section 6.2) are also a source of approximation error. Since the size of SLs at the horizon of receiver point hemispheres is reduced, soft shadows and diffuse reflections incident at shallow angles (i.e. large angles with the surface normal) may be too weak. However,

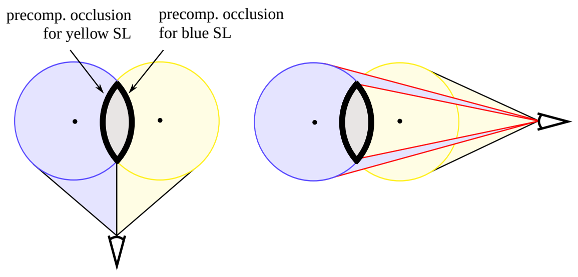
precomp. occlusion for yellow SL
precomp. occlusion for blue SL

Figure 7: *Approximation error due to precomputed occlusions of intersecting SLs. The left diagram shows a setup without approximation error. Both spheres have correct visibility. In the right diagram, too little of the blue SL is occluded, resulting in diffuse reflections that are too bright.*

this error is very small when compared to the other sources of approximation error and not noticeable in typical scenes.

Third, the precomputed occlusions for intersecting SLs (see Section 6.2) cause approximation error. Normally, these occlusions would be view-dependent, but by precomputing them, we make them less dependent on the viewpoint (see Figure 7). The result is a brightening of diffuse reflections for some configurations of intersecting SLs.

In our tests, approximation of diffuse reflections in the SL sets and the precomputation of occlusions for intersecting spheres were the largest sources of approximation error. The former can be alleviated by increasing the number of SLs per object at the cost of speed. This is a decision of quality versus performance. The latter could be solved by calculating the visibility of intersecting SLs at run time. This could be done by precomputing the visibility for possible configurations of intersecting SLs, but this is left for future work.

8.2. Validation

We present four sample scenes of increasing complexity (see Figure 8) to illustrate the overall effect of our approximation on the accuracy of the resulting images. For each scene, we show our solution, a reference image, the absolute difference to the reference image and the SLs used in the scene. In Figure 9, we show two model deformations with corresponding reference images. Reference images were rendered with 3D Studio Max [3ds] and VRay [VRa], using irradiance maps and path tracing. The absolute difference to reference images was multiplied by a factor of two to enhance visibility.

The first scene in Figure 8 consists of a simple sphere, which is the optimal case for our method. Approximation error can only arise from the approximated light intensity distribution of the SL and the SH basis approximation of incident light. In the second scene we have added multiple spheres. In this scene, additional approximation error arises from SH blocker accumulation. The third image shows a squirrel model approximated with multiple SLs. In this scene, we can also observe approximation error from

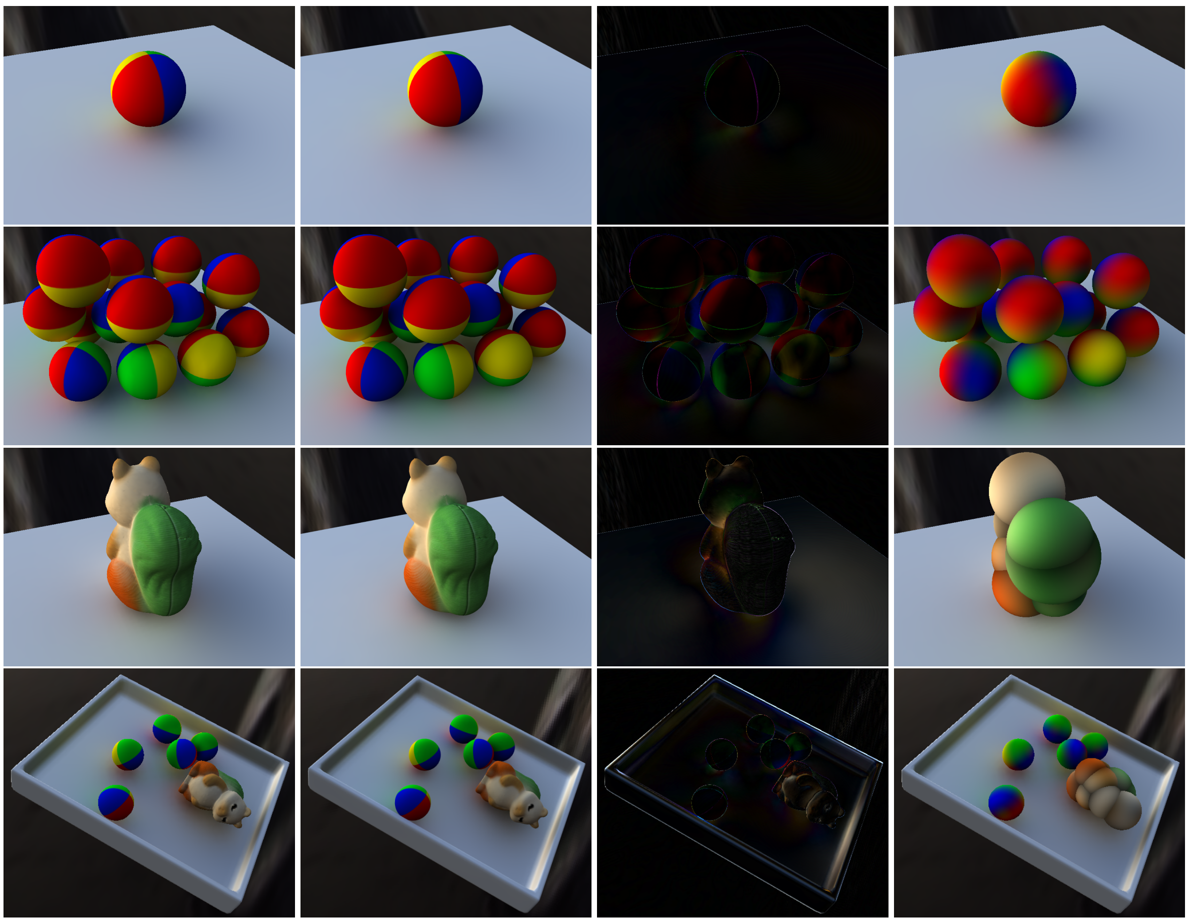

Figure 8: *Four scenes used to illustrate approximation error. Columns from left to right: (a) Our solution. (b) Reference computed using path-tracing and irradiance maps. (c) absolute difference between our solution and the reference solution, multiplied by a factor of two to enhance visibility (large absolute difference at edges mainly due to different anti-aliasing and texture filtering). (d) SL approximation of the scene.*

precomputed occlusions and from the geometry of the SL approximation, which has a larger volume than the original geometry. The difference between SL approximation and original geometry is most evident from the viewpoint of vertices on the model which are facing SLs of the same model. This leads to the darkening of crevices we can observe on the squirrel model. The last row shows one of the benchmarking scenes. Approximation error is mainly due to the larger volume of the approximating SL set.

Figure 9 shows a model in two different deformation states. We chose a view of the scene that is most likely to show approximation error due to model deformations. Approximation errors occur because a single SL can only be transformed with the average transformation of all its associated vertices. The strength of the approximation error is determined by the deviation of individual vertices from this average. When bending a cylinder, we can observe this approximation error near the outward-facing part of the cylinder, where surface stretching is strongest. At a 90 degree bend (first row of Figure 9), the approximation error is still relatively small. At a 180 degree bend (second row of Figure 9), we can observe the approximation error in the form of slightly darker bands on the ground below the cylinder, perpendicular to the cylinders main axis (note that the difference to the reference on the cylinder itself is caused by a different bone animation technique in the reference renderer). This banding is caused by black parts of the SLs (i.e. parts without emission) that become visible because of the deformation, as highlighted on the SL visualization in the second row of Figure 9. However, the strength of the approximation error for simple deformations, such as articulated motion or the deformation demonstrated in this scene, is very small when compared to other sources of approximation error.

Flat objects and objects with sharp edges are difficult to approximate with spheres and need a higher resolution SL approximation (i.e. more SLs) to keep the approximation er-

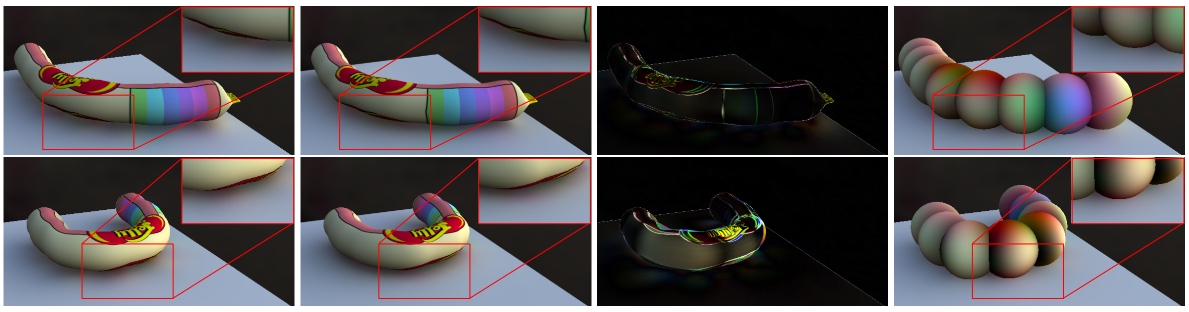

Figure 9: *Approximation error caused by deformations. Columns from left to right: (a) Our solution. (b) Reference computed using path-tracing and irradiance maps. (c) absolute difference between our solution and the reference solution, multiplied by a factor of two to enhance visibility. (The large absolute difference on the cylinder in the second row is mainly caused by a different bone animation technique in 3D Studio Max.) (d) SL approximation of the scene.*

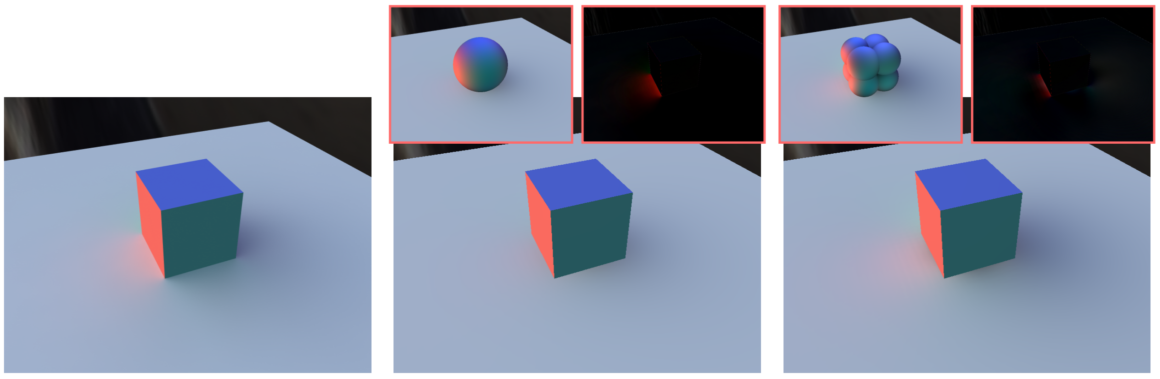

Figure 10: *Different SL resolutions on an object with sharp edges. Images from left to right: (a) Reference computed using path-tracing and irradiance maps. (b) Our solution using one SL and (c) using 8 SLs. At the top left of (b) and (c) we show the SL approximation of the cube, at the top right the absolute difference between our solution and the reference solution, multiplied by a factor of two to enhance visibility.*

ror low. In Figure 10 we show a box with SL approximations of different resolutions. The left image shows the reference rendered offline. Note that there are high-frequency brightness changes at the edges of the cube, particularly the brightness at the bottom side of the cube is much lower than on the other sides. When using a single SL to approximate the cube (center image), the light intensity distribution of the SL blurs these high frequencies, resulting in indirect lighting that is too weak near the edges of the cube. Higher SL resolutions (right image) result in less blurring and therefore more accurate indirect lighting.

8.3. Performance

Blocker accumulation is the most expensive step in our method. When using standard per-sphere blocking (see Section 7.2), the performance of this step depends quadratically on the number n of SLs in the scene ($O(\frac{n(n+1)}{2})$), since every SL has to be occluded by every closer SL. This could be reduced to $O(mn)$, where m is the average number of intersections per SL, if results were re-used during the calculation of the SL visibility value. Finding efficient ways to store and re-use such results on the GPU is left for future work.

When using per-object blocking, we can reduce the complexity of the blocker accumulation step to $O(n_l^2 N)$ where n_l is the average number of SLs per object (closely related to the geometrical complexity of objects) and N the number of objects in the scene. If we place an upper bound on the geometrical complexity of objects in the scene, blocker accumulation has a complexity that is linear in the number of scene objects, with a constant factor representing object complexity. (A corresponding factor can be found in the methods of Pan et al. [PWL*07] and Iwasaki et al. [IDYN07]. We will make a more thorough comparison in Section 8.6.) Sorting objects by distance from a receiver point has a runtime complexity of $O(N \log N)$, although in practice, the computational cost of this step is low when compared to blocker accumulation.

Memory requirements for our method are low, as the precomputed dataset for each scene object is relatively small, typically in the range of 32 - 150 Kbyte. The dataset for precomputed lookup tables is also relatively small, 775 Kbyte in our implementation.

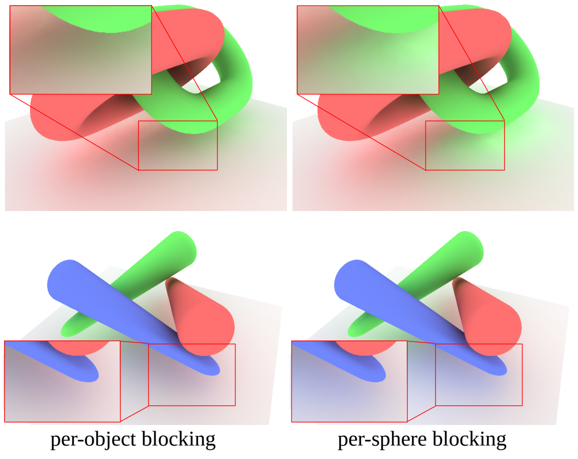

Figure 11: *Situations where per-object blocking is inappropriate. First row: two interlocking chain links occlude each other. Second row: three convex objects cannot be sorted unambiguously.*

8.4. Per-Sphere vs Per-Object Blocking

In practice, per-object blocking, as also used by Pan et al. [PWL*07] and Iwasaki at al. [IDYN07], gives a considerable performance improvement over per-sphere blocking. On the downside, we have to establish a definite order for each pair of objects (i.e. objects can only be completely in front or behind other objects), since we cannot split the occlusion effect of one object. This raises some problems with concave objects, as illustrated in Figure 11, top left. Here, both objects occlude each other partially, as seen from a receiver point directly below the objects. Since we cannot split up the occlusion effect of either object, we have to treat one of the objects as being completely in front of the other one. Per-sphere blocking (Figure 11, top right) can naturally handle interlocking concave objects.

Sorting of both convex and concave objects also raises problems when using per-object blocking. Consider Figure 11, bottom left. Although it is possible to determine a definite order for each pair of objects, there is no single correct order for all the objects as seen from a receiver point below the objects. Thus, exact sorting would require $O(N^2)$ comparisons, one for each object pair. One way to guarantee unambiguous sorting is to disallow object bounding spheres to intersect. However, for typical scenes, we did not observe large error when using standard object-center distance sorting. In our test scenes, we used object-center distance sorting when using object blocking.

8.5. Example Scenes

We demonstrate our method on four example scenes. All objects in the scenes, except the ground and walls, are source of soft shadows and diffuse indirect illumination. The squirrel scene is an animated sequence where a squirrel and a few balls slide down an inclined box. It contains 21k vertices, 14 leaf SLs and 25 vertex clusters. The robot scene shows an animated robot. It contains 25k vertices, 26 leaf SLs and 68 vertex clusters. The packages scene shows two deforming packages of jelly beans. It contains 10k vertices, 16 leaf SLs and 18 vertex clusters. The assembly scene shows various moving and deforming objects in a jelly bean assembly plant. It contains 21k vertices, 38 leaf SLs and one SL that acts as a local light source. All scenes are illuminated by distant environment lighting, the assembly scene is additionally illuminated by a local light source in part of its animation. All geometry in the scenes is dynamic. Figure 12 shows images from each of the scenes. The measured frame rates are summed up in the following table:

	shadows & ind. ill.				shadows only	
	per-sphere bl.		per-object bl.			
	p.s.	v.s.	p.s.	v.s.	p.s.	v.s.
squirrel	26	13	35	16	174	77
robot	12	5	20	9	97	25
packages	30	13	43	18	190	112
assembly	8	5	14	7	90	31

The first two columns show timing with indirect illumination and per-sphere blocking, the next two columns show timing with per-object blocking and the last two columns show timing for rendering with soft shadows only, using the method by Ren et al. [RWS*06]. p.s. is the pixel shader implementation, v.s. the vertex shader implementation. All timings are in frames per second. Memory requirements were 1.09 MB for the squirrel scene, 1.18 MB for the robot scene, 0.92 MB for the packages scene and 1.12 MB for the assembly scene.

Rendering speed is relatively slow when compared to rendering direct illumination and soft shadows only, but the first bounce of indirect lighting has a strong impact on scene lighting (see also the accompanying video), already giving a good approximation to low-frequency global illumination.

8.6. Comparison to Related Methods

In the following, we compare our method to the basis irradiance method [IDYN07] and the radiance transfer field (RTF) method [PWL*07], as these algorithms also handle indirect illumination for dynamic scenes. Both methods can only use per-object blocking, since occlusion effect and lighting response is precomputed per object. In our method, we can use per-sphere blocking where per-object blocking is inappropriate (see Section 8.4), although per-object blocking has better performance. When using per-object blocking, the computational complexity of our method is $O(n_l^2 N)$, where N is the number of objects in a scene and n_l^2 a constant factor representing the geometrical complexity of objects. Both the basis irradiance method and the RTF method have a similar computational complexity as ours, although with a different constant factor. The constant factor in the RTF method is the

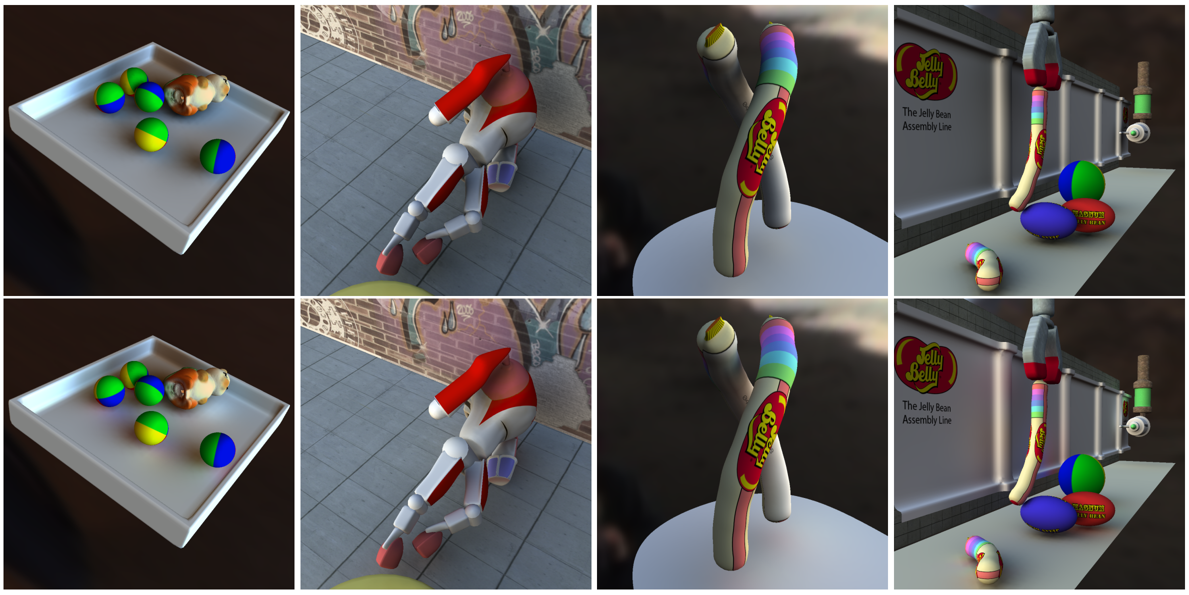

Figure 12: *Four scenes used for benchmarking. The top row was rendered with soft shadows only, using the method by Ren et al. [RWS*06]. The bottom row was rendered with one bounce of diffuse indirect illumination using our new method (per-object blocking was used for all images). All objects are dynamically animated.*

number of sample points for incident radiance, which ideally depends on the variation of the incident light over the surface of an object. The constant factor in the basis irradiance method is the number of PCA basis functions used to represent diffuse reflections on objects, which depends on the geometrical complexity of objects. In the basis irradiance method, the per-object cost for updating PCA basis functions is relatively high, limiting the amount of objects that can be dynamically updated in a single frame. The rendering speed of our method in scenes of similar complexity is slightly faster than the RTF method and comparable to the basis irradiance method if only few objects are moving in that method. Our method is likely to show better performance on scenes with many objects of low geometrical complexity (e.g. the squirrel scene), since the computation time needed for each SL is low when compared to the computation time needed for each object in the basis irradiance method or the RTF method.
Our precomputed dataset is also much smaller, only 1-2 MB of memory in our test scenes, whereas both other methods need 30-70 MB for comparable scenes.

As explained earlier in this section, the approximation of diffuse reflections with SLs is a main source of approximation error in our method. This approximation is loosely comparable to the approximation of diffuse reflections with PCA basis functions in the basis irradiance method, however a direct comparison is difficult. In the RTF method, the full radiance transfer from incident to exit radiance is precomputed, therefore no intermediate representation of diffuse reflections is needed. However, the authors assume low variation of the incident light over the surface of objects, allowing them to use a relatively small number of sample points for incident radiance per object, typically 8-24 sample points. This assumption may not hold if primary or secondary light sources are close to the surface of objects, leading to a degradation in quality. We use 92 sample points per SL, allowing for a higher degree of variation in the incident radiance.

The main drawback of our method is its limitation to single-bounce diffuse indirect illumination (multi-bounce indirect illumination is considered in future work). In contrast to the basis irradiance method and the RTF method, SL approximation provides a less static approach to indirect illumination. The main advantages provided by our method are interactive deformations, the possibility of dynamic objects with intersecting convex hulls, faster performance for scenes with many simple objects, and a significantly lower memory footprint.

9. Conclusion and Future Work

We have presented a method for soft shadows and diffuse indirect illumination in dynamic scenes, based on the method by Ren et al. [RWS*06]. We approximate diffuse reflections from scene objects with sets of SLs and update their light intensity distributions in each frame using sample points on the surface of objects. At each receiver point, we use the SLs to efficiently accumulate incident radiance from diffuse reflections. Our method achieves interactive performance for scenes of medium complexity, has a low memory footprint and allows for interactive deformation.

In future work, we would like to implement multi-bounce indirect illumination, extend the method to use glossy BRDFs in the final light bounce, handle intersections of spherical lights at run time, eliminating the need for precomputed occlusions, and implement more general object deformations for articulated motion. To enhance performance, we could use screen-space SL splatting, as detailed in the paper by Sloan et al. [SGNS07].

References

[3ds] Autodesk 3D Studio Max. http://www.autodesk.com.

[Bun04] BUNNELL M.: Dynamic ambient occlusion and indirect lighting. In *GPU Gems2* (2004), pp. 223–233.

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (2005), pp. 203–231.

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (2006), pp. 93–100.

[DSDD07] DACHSBACHER C., STAMMINGER M., DRETTAKIS G., DURAND F.: Implicit visibility and antiradiance for interactive global illumination. *ACM Trans. Graph. 26*, 3 (2007), 61.

[IDYN07] IWASAKI K., DOBASHI Y., YOSHIMOTO F., NISHITA T.: Precomputed radiance transfer for dynamic scenes taking into account light interreflection. In *Proc. of Eurographics Symposium on Rendering* (2007), pp. 35–44.

[KL05] KONTKANEN J., LAINE S.: Ambient occlusion fields. In *SI3D '05: Proc. of the 2005 Symposium on Interactive 3D graphics and games* (2005), pp. 41–48.

[KLA04] KAUTZ J., LEHTINEN J., AILA T.: Hemispherical rasterization for self-shadowing of dynamic objects. In *Proc. of Eurographics Symposium on Rendering 2004* (2004), pp. 179–184.

[Lan02] LANDIS H.: Renderman in production. *ACM SIGGRAPH 2002 Course 16* (2002).

[Lep78] LEPAGE G. P.: A new algorithm for adaptive multidimensional integration. *J. Comput. Phys. 27* (1978), 192.

[Llo82] LLOYD S.: Least squares quantization in pcm. *IEEE Transactions on Information Theory 28* (1982), 129–137.

[LSK*07] LAINE S., SARANSAARI H., KONTKANEN J., LEHTINEN J., AILA T.: Incremental instant radiosity for real-time indirect illumination. In *Proc. of Eurographics Symposium on Rendering* (2007), pp. 277–286.

[MMAH06] MALMER M., MALMER F., ASSARSON U., HOLZSCHUCH N.: Fast precomputed ambient occlusion for proximity shadows. *Journal of Graphics Tools* (2006).

[MSW04] MEI C., SHI J., WU F.: Rendering with spherical radiance transport maps. *Computer Graphics Forum 23*, 3 (2004), 281–290.

[NRH04] NG R., RAMAMOORTHI R., HANRAHAN P.: Triple product wavelet integrals for all-frequency relighting. *ACM Trans. Graph. 23*, 3 (2004), 477–487.

[PWL*07] PAN M., WANG R., LIU X., PENG Q., BAO H.: Precomputed radiance transfer field for rendering interreflections in dynamic scenes. *Proc. of Eurographics 26*, 3 (2007), 485–493.

[RWS*06] REN Z., WANG R., SNYDER J., ZHOU K., LIU X., SUN B., SLOAN P.-P., BAO H., PENG Q., GUO B.: Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM Trans. Gr. 25*, 3 (2006), 977–986.

[SGNS07] SLOAN P.-P., GOVINDARAJU N. K., NOWROUZEZAHRAI D., SNYDER J.: Image-based proxy accumulation for real-time soft global illumination. In *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications* (2007), pp. 97–105.

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph. 21*, 3 (2002), 527–536.

[SLS05] SLOAN P.-P., LUNA B., SNYDER J.: Local, deformable precomputed radiance transfer. *ACM Trans. Gr. 24*, 3 (2005), 1216–1224.

[SM06] SUN W., MUKHERJEE A.: Generalized wavelet product integral for rendering dynamic glossy objects. *ACM Trans. Graph. 25*, 3 (2006), 955–966.

[TJCN06] TAMURA N., JONAH H., CHEN B.-Y., NISHITA T.: A practical and fast rendering algorithm for dynamic scenes using adaptive shadow fields. *The Visual Computer 22*, 9–11 (2006), 702–712.

[VRa] Chaosgroup VRay. http://www.chaosgroup.com.

[WZS*06] WANG R., ZHOU K., SNYDER J., LIU X., BAO H., PENG Q., GUO B.: Variational sphere set approximation for solid objects. *The Visual Computer 22*, 9 (2006), 612–621.

[ZHL*05] ZHOU K., HU Y., LIN S., GUO B., SHUM H.-Y.: Precomputed shadow fields for dynamic scenes. *ACM Trans. Gr. 24*, 3 (2005), 1196–1201.