Real-time Soft Shadows in Dynamic Scenes using Spherical Harmonic Exponentiation

Zhong Ren[1*] Rui Wang[1*] John Snyder[2] Kun Zhou[3] Xinguo Liu[3]
Bo Sun[4†] Peter-Pike Sloan[5] Hujun Bao[1] Qunsheng Peng[1] Baining Guo[3]

[1] Zhejiang Univ. [2] Microsoft Research [3] Microsoft Research Asia [4] Columbia Univ. [5] Microsoft Corporation

Abstract

Previous methods for soft shadows numerically integrate over many light directions at each receiver point, testing blocker visibility in each direction. We introduce a method for real-time soft shadows in dynamic scenes illuminated by large, low-frequency light sources where such integration is impractical. Our method operates on vectors representing low-frequency visibility of blockers in the spherical harmonic basis. Blocking geometry is modeled as a set of spheres; relatively few spheres capture the low-frequency blocking effect of complicated geometry. At each receiver point, we compute the product of visibility vectors for these blocker spheres as seen from the point. Instead of computing an expensive SH product per blocker as in previous work, we perform inexpensive vector sums to accumulate the *log* of blocker visibility. SH exponentiation then yields the product visibility vector over all blockers. We show how the SH exponentiation required can be approximated accurately and efficiently for low-order SH, accelerating previous CPU-based methods by a factor of 10 or more, depending on blocker complexity, and allowing real-time GPU implementation.

Keywords: ambient occlusion, lighting environment, Volterra series.

1 Introduction

Soft shadows are critical for realistic image synthesis. While many real-time techniques generate hard shadows from point or directional light sources, they can't be practically extended to large area light sources. Too many passes are required to properly sample points on the large sources.

Precomputed radiance transfer (PRT) solves the light integration problem using an offline tabulation of an object's response to low-frequency lighting [Sloan et al. 2002]. Ignoring higher frequencies of lighting and transfer (in contrast to "all frequency" PRT [Ng et al. 2003]) greatly reduces the dimensionality of transfer vectors or matrices and their spatial sampling rate, making the idea much more practical. Our method has the same limitation of low-frequency lighting and relatively matte BRDFs. Unlike our method, PRT is limited to static objects, precomputed sequences [James and Fatahalian 2003], or local textural features [Sloan et al. 2005].

Our goal is accurate generation of soft shadows in general, dynamic scenes where tabulation is impractical. Our method operates directly on vectors representing the low-frequency visibility function of a blocker as seen from a receiver point, expressed using the spherical harmonic (SH) basis. A previous method [Zhou et al. 2005] has also taken this approach by rotating each blocker visibility function into the local coordinate frame and computing the SH product over all blockers. SH rotations and products are very expensive, precluding GPU implementation and restricting real-time CPU implementation to a few precomputed blockers.

Our solution approximates blocker geometry as a collection of spheres. This simple model is easy to animate. It is also easy to compute SH visibility: the projection of a sphere as seen from any receiver point is circularly symmetric and thus can exploit a simple rotation rule [Sloan et al. 2005]. Rather than computing blocker products directly, we represent blocker visibility in log space. The total log blocking vector can then be accumulated as a simple sum of log vectors over all blockers. For order-n SH vectors, this reduces per-blocker computation from $O(n^5)$ to $O(n^2)$; for order-4 SH vectors, it is less than $1/20$ the per-blocker cost. The reduced per-blocker cost allows us to handle complicated scenes with many more blockers than in previous work. We call the method SHEXP.

Given the total log of blocker visibility at a receiver point, we perform SH exponentiation to yield the total blocker visibility. The result is a low-frequency, spherical visibility function that accounts for blocker overlap and can be used to modulate lighting before applying it to any BRDF. Our SH exponentiation approximation is fast enough to provide large speedups over previous soft shadowing approaches, and simple enough to map to the GPU.

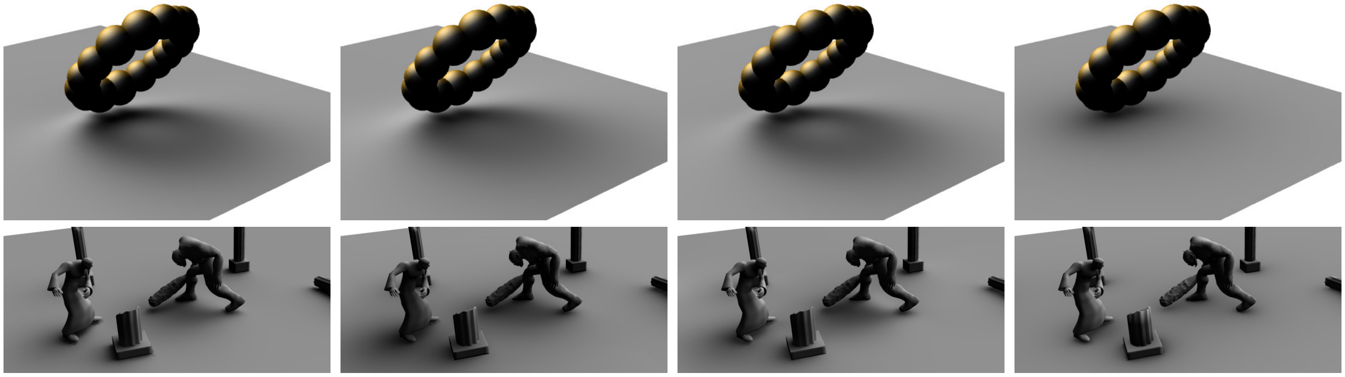
Ours is the first real-time method for rendering soft shadows from low-frequency environmental lighting in dynamic scenes with many blockers. In particular, we handle deformable objects like articulated characters including their self-shadows. We introduce two technical contributions to do this. One is to accumulate low-frequency blocker visibility in log space. The Volterra series, long used for circuit and signal analysis, forms the mathematical foundation for SH exponentiation. We also describe a series inversion method for taking SH log to improve accuracy. The second is to efficiently approximate low-frequency blocker visibility using sets of bounding spheres. Our novel construction minimizes volume outside the blocker geometry. To prevent complete self-shadowing, we formulate replacement rules for blocker spheres near the receiver point. We also apply a sphere hierarchy to adapt complexity of the blocker approximation to its receiver distance.

2 Previous Work

Using triple products of lighting, reflectance, and visibility, [Ng et al. 2004] presents a method for fast relighting and view changes in static scenes. Shadow fields [Zhou et al. 2005] extend this method to account for dynamic visibility changes and form the basis for our approach of directly manipulating low-frequency visibility vectors. Using the spherical harmonic basis, this method demonstrates real-time performance for at most six rigidly-moving blocker objects because of the expense of SH rotations and products. These operations are even more expensive with the "all-frequency" basis (wavelets over cube maps), which typically has an order of magnitude more basis components than low-order SH, and a comparable

[*] This work was done while Zhong Ren and Rui Wang were interns at Microsoft Research Asia.

[†] This work was done while Bo Sun was an intern at Microsoft Research.



(a) ray traced

(b) SHEXP, n=4

(c) SHEXP, n=6

(d) AO [Bunnell 2004]

Figure 1: *Rendering method comparison. The middle two columns use our method, accumulating using SH order n (n=4 in (c), n=6 in (d)) visibility vectors and applying order n exponentiation. SH lighting order applied is n=4. Column (d) uses software available at http://download.developer.nvidia.com. In the bottom row, columns (a) and (d) shadow from the actual geometry while (b) and (c) use blocker spheres. SHEXP casts much more realistic shadows than AO.*

increase in spatial sampling required at the receiver. Precomputed shadow fields for rigidly-moving objects can be combined with our approach by tabulating object visibility in SH log space.

As noted in the introduction, most PRT methods rely on static or precomputed objects. Local, deformable PRT (LDPRT) [Sloan et al. 2005] precomputes transfer on local features like bumps or wrinkles, but can reorient them on a deforming object without knowing the object's motion in advance. We apply its idea of zonal harmonics (ZH) to simplify SH rotation, using spherical blockers whose logs require only a single-lobe ZH model. Our method handles global shadowing such as from one character to another or a character's arm onto his body rather than reorienting fine-scale details as in LDPRT.

Ambient occlusion (AO) [Bunnell 2004; Kontkanen and Laine 2005; Malmer et al. 2005] defines a simple cone by the average visibility direction and its total spherical area. Shadows are produced, but maximally soft ones determined mostly by occluder proximity rather than lighting direction. Our method generalizes AO by computing true low-frequency blocker visibility rather than simple averages, and so is able to generate recognizable shadows that trail behind bright lighting directions (see Figure 1). While other AO approaches tabulate over rigid components, [Bunnell 2004] handles deformable geometry. It organizes simple approximating elements for blockers into a hierarchy. We borrow this idea but distribute spheres over the blocker's volume rather than discs over its surface.

Several methods [Kautz et al. 2004; Bunnell 2004; Zhou et al. 2005] including ours may be termed *blocker accumulation* methods because they process a list of blocker geometry at each receiver point. Other methods employ multiple shadow buffers [Segal et al. 1992; Agarwala et al. 2000; Mei et al. 2004] and so entail slow integration over lighting directions when rendering large lights. In fact, [Kautz et al. 2004] is a blocker accumulation method that also uses the directional lighting basis by rasterizing blockers into hemispherical bitmaps. Real-time rendering is limited only to very simple scenes. On the other hand, soft shadow volumes [Assarsson and Akenine-Möller 2003] can handle greater scene complexity but only small area light sources. Convolution can also be used to shadow [Soler and Sillion 1998], but it produces a scalar modulation rather than true hemispherical radiance and is difficult to apply in general scenes with non-planar receivers and large-depth blockers. We solve the problem of light integration over large sources by accumulating over coarsely-approximated, low-frequency blockers representing large subtended angles rather than individual directions. Our approach is efficient for moderate numbers of blocker primitives (we can handle several hundred) which are made more effective by our use of a blocker hierarchy.

Homomorphic factorization [McCool et al. 2001] uses log space to factor high-dimensional BRDFs into a sum of positive, lower-dimensional functions. Only scalar exponentiation is required at run-time; our method "pushes" entire SH vectors through nonlinear log or exp operators based on the Volterra series.

3 Overview and Terminology

We use math italic for scalars and 3d points or vectors (e.g., x, s), boldface italic for SH vectors (e.g., $\boldsymbol{f}$, $\boldsymbol{g}$), and sans serif for matrices and higher-order tensors (e.g., $\mathsf{\Gamma}$, M, D).

Spherical Harmonics are useful to represent low-frequency spherical functions such as radiance incident at a point, as well as blocker visibility functions which modulate distant radiance. Given a spherical function $f(s)$, we can *project* this function to determine a vector $\boldsymbol{f}$ representing its low-frequency behavior via

$$\boldsymbol{f} = \int_S f(s)\,\boldsymbol{y}(s)\,ds \qquad (1)$$

where $\boldsymbol{y}(s)$ is the vector of SH basis functions. The SH basis functions are orthogonal polynomials in $s = (x, y, z)$ restricted to the sphere $s \in S$. An order n SH projection has n^2 vector coefficients. Conversely, given an SH vector $\boldsymbol{f}$ we can *reconstruct* a continuous, low-frequency spherical function $\tilde{f}(s)$ approximating $f(s)$ via

$$\tilde{f}(s) = \sum_{i=0}^{n^2-1} \boldsymbol{f}_i\, y_i(s) = \boldsymbol{f} \cdot \boldsymbol{y}(s). \qquad (2)$$

SH Products and the Triple Product Tensor are useful for computing the combined shadowing effect of multiple block-
ers directly in the SH basis, without resorting to numerical inte- -07-
gration over directions [Kautz et al. 2004; Mei et al. 2004] or per- -10-
forming complicated geometric clipping operations [Assarsson and Akenine-Möller 2003; Laine et al. 2005]. The *SH product*, denoted $\boldsymbol{f} * \boldsymbol{g}$, represents the order-$n$ projected result of multiplying the reconstruction of two order-n vectors, $\boldsymbol{f}$ times $\boldsymbol{g}$, or

$$\boldsymbol{f} * \boldsymbol{g} = \int_S f(s)\, g(s)\, \boldsymbol{y}(s)\, ds \;\Rightarrow\; (\boldsymbol{f} * \boldsymbol{g})_i = \sum_{jk} \mathsf{\Gamma}_{ijk}\, \boldsymbol{f}_j\, \boldsymbol{g}_k \qquad (3)$$

where the *SH triple product tensor*, $\mathsf{\Gamma}_{ijk}$, is defined by

$$\mathsf{\Gamma}_{ijk} = \int_S \boldsymbol{y}_i(s)\, \boldsymbol{y}_j(s)\, \boldsymbol{y}_k(s)\, ds. \qquad (4)$$

$\mathsf{\Gamma}$ is a symmetric, sparse, order-3 tensor. This definition incurs truncation error because the product of two order-n vectors is actually order $2n-1$.

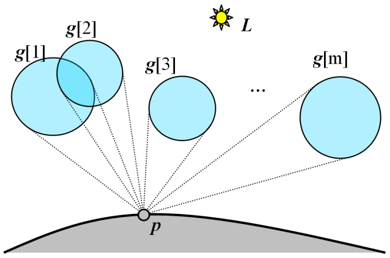


Figure 2: *Shadowing from multiple blockers at receiver point p.*

Order-n SH products are $O(n^5)$ [Ng et al. 2004]. The following table records the number of nonzero entries in Γ as function of n:

n	1	2	3	4	5	6	7	8
# coefs	1	10	83	369	1164	2961	6586	13018

Even at low orders, SH product is an expensive operation. Efficient evaluation uses code generation and factoring [Snyder 2006].

We can also define the *SH product matrix*, M_f, given an SH vector $\boldsymbol{f}$. The product matrix is a symmetric matrix which encapsulates SH product with $\boldsymbol{f}$; in other words, $\boldsymbol{f} * \boldsymbol{g} = \mathsf{M}_f \, \boldsymbol{g}$ for an arbitrary vector $\boldsymbol{g}$. M_f is defined by

$$(\mathsf{M}_f)_{ij} = \sum_k \Gamma_{ijk} \boldsymbol{f}_k. \tag{5}$$

Shadowing using SH Products [Zhou et al. 2005] computes the product of a collection of m blockers $\boldsymbol{g}[1], \boldsymbol{g}[2], \ldots, \boldsymbol{g}[m]$, via

$$\boldsymbol{g} = \boldsymbol{g}[1] * \boldsymbol{g}[2] * \cdots * \boldsymbol{g}[m] \tag{6}$$

where each $\boldsymbol{g}[i]$ is the SH projection of the corresponding blocker visibility function (Figure 2):

$$g[i](s) = \begin{cases} 0, & \text{if object } i \text{ blocks in direction } s; \\ 1, & \text{otherwise.} \end{cases} \tag{7}$$

交换律 结合律

Although SH product is commutative, it is not associative, so the ordering in which the above products are performed matters.

Shadowing in Log Space instead accumulates the log of blocker visibilities, denoted by $\boldsymbol{f}[1], \boldsymbol{f}[2], \ldots, \boldsymbol{f}[m]$ where $\boldsymbol{f}[i] = \log(\boldsymbol{g}[i])$. Thus

$$\boldsymbol{g} = \exp(\boldsymbol{f}) = \exp\left(\boldsymbol{f}[1] + \boldsymbol{f}[2] + \cdots + \boldsymbol{f}[m]\right). \tag{8}$$

Accumulating the log now involves vector sums which are independent of the blocker ordering and much cheaper than SH products. Section 4 discusses how the SH exponential is computed while SH log is discussed in Section 5.

Shading then makes use of the total visibilty vector $\boldsymbol{g}$. For diffuse surfaces, the computation is $(\boldsymbol{H}(N), \boldsymbol{L}, \boldsymbol{g})$ where $\boldsymbol{L}$ is the light vector, $\boldsymbol{g}$ is the total blocker visibility vector, and $\boldsymbol{H}(N)$ is the irradiance weighting function given the surface normal N [Ramamoorthi and Hanrahan 2001]:

$$\boldsymbol{H}(N) = \frac{1}{\pi} \int_s \max(s \cdot N, 0)\, \boldsymbol{y}(s)\, ds \tag{9}$$

$(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ for three SH vectors $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ denotes the integral of the product of the three reconstructed functions and is given by

$$(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = (\boldsymbol{a} * \boldsymbol{b}) \cdot \boldsymbol{c} = (\boldsymbol{b} * \boldsymbol{c}) \cdot \boldsymbol{a} = (\boldsymbol{c} * \boldsymbol{a}) \cdot \boldsymbol{b} = \sum_{ijk} \Gamma_{ijk} \boldsymbol{a}_i \boldsymbol{b}_j \boldsymbol{c}_k$$

The total visibility vector can also be used to shade other types of BRDFs [Kautz et al. 2002] or textural detail [Sloan et al. 2003; Sloan et al. 2005]: the vector $\boldsymbol{g} * \boldsymbol{L}$ represents shadowed incident radiance to apply to the receiver.

4 SH Exponential

Let $\boldsymbol{f}$ be an SH vector to be exponentiated, and $\boldsymbol{g}$ be the result of this exponentiation. The *Volterra series* [Schetzen 1980] allows any analytic, univariate, scalar function $h(x)$ (e.g. $h(x) = \exp(x)$) to be applied to an SH vector, or indeed any discrete function basis. We begin with the integral formulation

$$h(\boldsymbol{f}) = \int_S h(f(s))\, \boldsymbol{y}(s)\, ds = \int_S h\left(\sum_i \boldsymbol{f}_i y_i(s)\right) \boldsymbol{y}(s)\, ds \tag{10}$$

which applies h to the reconstructed function $f(s)$ at each spherical point s and then projects the result to the vector $h(\boldsymbol{f})$. Substituting the Taylor expansion of the function $h(x)$

$$h(x) = h_0 + h_1 x + h_2 x^2 + h_3 x^3 + \cdots, \tag{11}$$

we obtain the *SH power series*

$$h(\boldsymbol{f}) = h_0 \mathbf{1} + h_1 \boldsymbol{f}^1 + h_2 \boldsymbol{f}^2 + h_3 \boldsymbol{f}^3 + \cdots \tag{12}$$

where

$$\begin{aligned} \mathbf{1} &= (\sqrt{4\pi}, 0, 0, \ldots, 0) \\ \boldsymbol{f}^p &= \int f^p(s)\, \boldsymbol{y}(s) ds = \int \left(\sum_i \boldsymbol{f}_i \boldsymbol{y}_i(s)\right)^p \boldsymbol{y}(s)\, ds. \end{aligned}$$

The vector $\mathbf{1}$ corresponds to a constant value of 1 over the sphere, and satisfies $\mathbf{1} * \boldsymbol{f} = \boldsymbol{f}$ for any $\boldsymbol{f}$. Degree p powers of $\boldsymbol{f}$ can be written in terms of order-(p+1) tensors Γ, via

$$(\boldsymbol{f}^p)_i = \sum_{i_1, i_2, \ldots, i_p} \Gamma_{i, i_1, i_2, \ldots, i_p} \boldsymbol{f}_{i_1} \boldsymbol{f}_{i_2} \cdots \boldsymbol{f}_{i_p} \tag{13}$$

where the tensor Γ represents the *Volterra kernel* (when scaled by h_i) and generalizes the triple product tensor we encountered before:

$$\Gamma_{i, i_1, i_2, \ldots, i_p} = \int_S \boldsymbol{y}_i(s) \boldsymbol{y}_{i_1}(s) \boldsymbol{y}_{i_2}(s) \ldots \boldsymbol{y}_{i_p}(s)\, ds \tag{14}$$

Numerical integration (10) or high-order tensors (12) are too expensive to evaluate on-the-fly. The result can be approximated by substituting repeated SH products for true SH powers in the series. This incurs approximation error because it truncates after each binary product. For example, $\boldsymbol{f}^3 \approx (\boldsymbol{f} * \boldsymbol{f}) * \boldsymbol{f}$ because the result of the first square $\boldsymbol{f} * \boldsymbol{f}$ is truncated back to an order-n SH vector before multiplying by $\boldsymbol{f}$ again. Errors are typically small for bandlimited visibility, since the original functions are bounded in [0,1], and can be further reduced by accumulating products at higher order than the input vectors. Using repeated SH products, we obtain the following approximation called the *SH product series*, more practical for real-time evaluation:

$$\begin{aligned} h_*(\boldsymbol{f}) &= h_0 \mathbf{1} + h_1 \boldsymbol{f} + h_2 \boldsymbol{f} * \boldsymbol{f} + h_3 \boldsymbol{f} * \boldsymbol{f} * \boldsymbol{f} + \cdots \\ &= h_0 \mathbf{1} + h_1 \boldsymbol{f} + h_2 \boldsymbol{f}^{2*} + h_3 \boldsymbol{f}^{3*} + \cdots \end{aligned} \tag{15}$$

We use the notation

$$\boldsymbol{f}^{p*} = \underbrace{\boldsymbol{f} * \boldsymbol{f} * \cdots * \boldsymbol{f}}_{\text{repeated } p \text{ times}}$$

and note that $\boldsymbol{f}^p \approx \boldsymbol{f}^{p*}$. For $p > 3$, product order matters; we assume the product is amassed from left to right.

Now applying the Volterra series using the Taylor expansion for $h(x) = \exp(x)$ in (15), we obtain the product series

$$\exp(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots \tag{16}$$

$$\exp_*(\boldsymbol{f}) = \mathbf{1} + \boldsymbol{f} + \frac{\boldsymbol{f}^{2*}}{2} + \frac{\boldsymbol{f}^{3*}}{3!} + \cdots \tag{17}$$

4.1 Product Series Approximation

For a finite number of terms in (17), approximation error increases as $\|\boldsymbol{f}\|$ increases, just as it does in (16) as $|x|$ increases. For this reason, evaluation techniques try to reduce the magnitude of the input vector $\boldsymbol{f}$ and thereby increase accuracy for a fixed number of terms. Another technique factors the series to reduce the number of SH products. These techniques are analogous to ones used for the matrix exponential [Higham 2005]. In fact, computing the exponential of an SH vector and a matrix are related since

$$\mathsf{M}_{f^{p*}} \approx (\mathsf{M}_f)^p \;\Rightarrow\; \mathsf{M}_{\exp_*(f)} \approx \exp\left(\mathsf{M}_f\right). \tag{18}$$

The left relation in (18) is only an approximate equality because of the non-associativity of SH product: $\mathsf{M}_{f^{2*}}\,\boldsymbol{g} = (\boldsymbol{f} * \boldsymbol{f}) * \boldsymbol{g} \neq \boldsymbol{f} * (\boldsymbol{f} * \boldsymbol{g}) = \mathsf{M}_f\,\mathsf{M}_f\,\boldsymbol{g} = (\mathsf{M}_f)^2\,\boldsymbol{g}$.

DC Isolation We express $\boldsymbol{f}$ as the sum of its DC component ($\boldsymbol{f}_0$, representing the average value of $f(s)$) plus its remaining components, or $\boldsymbol{f} = \hat{\boldsymbol{f}} + \frac{\boldsymbol{f}_0}{\sqrt{4\pi}}\mathbf{1}$. $\hat{\boldsymbol{f}}$ simply zeroes out the DC component of $\boldsymbol{f}$, i.e.,

$$\hat{\boldsymbol{f}} = (0, \boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_{n^2-1}). \tag{19}$$

Then (17) becomes

$$\exp_*(\boldsymbol{f}) = \exp\left(\frac{\boldsymbol{f}_0}{\sqrt{4\pi}}\right)\exp_*(\hat{\boldsymbol{f}}) \tag{20}$$

which is easily derived since $\boldsymbol{f} * \mathbf{1} = \boldsymbol{f}$.

Eq. (20) *analytically* computes the exponential of the DC component, reducing the magnitude of the residual vector $\hat{\boldsymbol{f}}$. This vector is then exponentiated using the series method augmented by additional techniques described in the following.

Scaling/Squaring We also make use of a technique used in evaluating scalar (and matrix) exponentials, which observes that

$$\exp(x) = \left(\exp\left(\frac{x}{2^p}\right)\right)^{2^p} \;\Rightarrow\; \exp_*(\boldsymbol{f}) \approx \left(\exp_*\left(\frac{\boldsymbol{f}}{2^p}\right)\right)^{2^p *} \tag{21}$$

where p is a positive integer. In other words, to compute $\exp(x)$ we first divide the input x by a power of 2, compute the exponential of this scaled input, and finally repeatedly square the result p times. The same idea can be applied to SH exponentiation using p repeated squarings via the recurrence $\boldsymbol{f}^{2^p *} = \boldsymbol{f}^{2^{p-1} *} * \boldsymbol{f}^{2^{p-1} *}$.

Eq. (21) only approximates the product series in (17), but typically reduces error relative to the power series in (12). SH squares are also cheaper than general SH products, making this approximation more useful. We choose p as a function of $\|\boldsymbol{f}\|$ using $p = \max(0, \lfloor \log_2 \|\boldsymbol{f}\| + 3 \rfloor)$. At most p=3 squarings are needed for low-order ($n \leq 6$) SH vectors in our examples.

Factoring Eq. (17) can be evaluated by accumulating successively higher powers of $\boldsymbol{f}$ via $\boldsymbol{f}^{(p+1)*} = \boldsymbol{f}^{p*} * \boldsymbol{f}$. This requires p-1 SH products for a degree p expansion. The number of SH products can be reduced by segregating even and odd powers. (15) becomes

$$\begin{aligned} h_*(\boldsymbol{f}) \;\; \approx \;\; & \left(h_0\,\mathbf{1} + h_2\,\boldsymbol{f}^{2*} + h_4\,\boldsymbol{f}^{4*} + \cdots\right) + \\ & \boldsymbol{f} * \left(h_1\,\mathbf{1} + h_3\,\boldsymbol{f}^{2*} + h_5\,\boldsymbol{f}^{4*} + \cdots\right) \end{aligned} \tag{22}$$

improving to only (p+1)/2 products for degree p expansion. Furthermore, fewer products implies smaller truncation error and thus a better approximation to (12). Powers of $\boldsymbol{f}$ should be computed so as to minimize the number of products in each term: $\boldsymbol{f}^{2*} = \boldsymbol{f} * \boldsymbol{f}$, $\boldsymbol{f}^{4*} = \boldsymbol{f}^{2*} * \boldsymbol{f}^{2*}$, $\boldsymbol{f}^{6*} = \boldsymbol{f}^{4*} * \boldsymbol{f}^{2*}$, and so on. Even better factorings can be obtained for series degree $p > 12$ [Higham 2005].

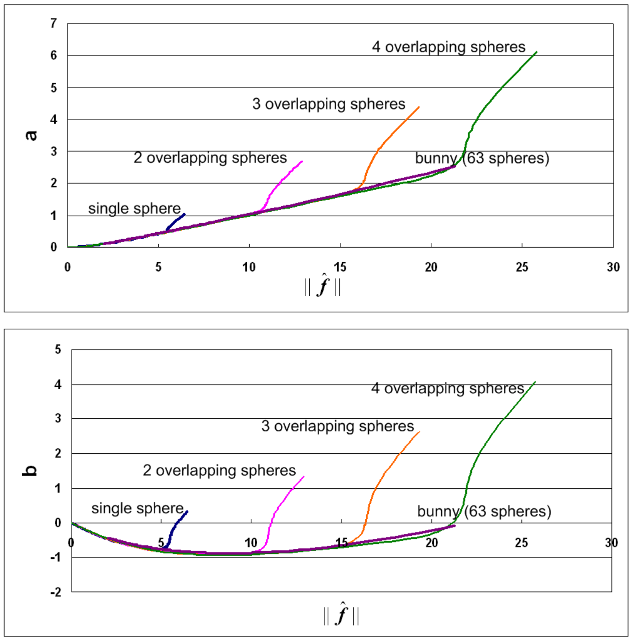


Figure 3: *Optimal linear coefficients for 4-th order SH. The curves match, and thus accuracy is obtained, only for* $\|\hat{\boldsymbol{f}}\| < 4.8$. *For bigger input vectors, scaling/squaring can be used to divide the input by a power of 2, and so reduce its magnitude to lie within this range.*

4.2 Optimal Linear Approximation

For SH order-4 or lower, an extension of the simple two-term series $\exp_*(\boldsymbol{f}) \approx \mathbf{1} + \boldsymbol{f}$ from (17) provides good accuracy without the need for even a single SH product. Given an input vector $\boldsymbol{f}$ to be exponentiated, we first apply DC isolation (20) to obtain $\hat{\boldsymbol{f}}$ and compute its magnitude $\|\hat{\boldsymbol{f}}\|$, followed by

$$\exp_*(\boldsymbol{f}) \approx \exp\left(\frac{\boldsymbol{f}_0}{\sqrt{4\pi}}\right)\left(a(\|\hat{\boldsymbol{f}}\|)\,\mathbf{1} + b(\|\hat{\boldsymbol{f}}\|)\,\hat{\boldsymbol{f}}\right). \tag{23}$$

The coefficients a and b are determined using the following preprocess. We generate a set of SH vector pairs representing circles of increasing angular radius; one vector in the pair is the visibility function $\boldsymbol{g}$ and the other is its corresponding log vector $\boldsymbol{f}$. To account for DC isolation we then zero out the DC component of $\boldsymbol{f}$ to obtain $\hat{\boldsymbol{f}}$ and correspondingly scale $\boldsymbol{g}$ to obtain $\hat{\boldsymbol{g}} = \exp(-\boldsymbol{f}_0/\sqrt{4\pi})\,\boldsymbol{g}$. Finally, we find the least-squares best projection of $\hat{\boldsymbol{g}}$ onto the orthogonal vectors $\mathbf{1}$ and $\hat{\boldsymbol{f}}$ via

$$a = \frac{\hat{\boldsymbol{g}} \cdot \mathbf{1}}{\mathbf{1} \cdot \mathbf{1}} = \frac{\hat{\boldsymbol{g}}_0}{\sqrt{4\pi}}, \qquad b = \frac{\hat{\boldsymbol{g}} \cdot \hat{\boldsymbol{f}}}{\hat{\boldsymbol{f}} \cdot \hat{\boldsymbol{f}}} \tag{24}$$

providing the minimum error $\|\hat{\boldsymbol{g}} - (a\mathbf{1} + b\hat{\boldsymbol{f}})\|$. Least-squares projection is performed for each circle of a different angular radius, and the resulting a and b coefficients tabulated as a function of $\|\hat{\boldsymbol{f}}\|$, which increases with angular radius.

Remarkably, for order-4 SH we have found that models agree on their a and b curves over a substantial part of the domain: roughly $\|\hat{\boldsymbol{f}}\| < 4.8$, corresponding to a blocker of angular radius less than 50° (see Figure 3). The approximation is also much more accurate than the simple two-term series within this domain. For bigger $\|\hat{\boldsymbol{f}}\|$ there is "baseline" agreement: the curves follow an initial baseline curve until they suddenly diverge, allowing us to derive plausible asymptotic behavior for a and b though it is not accurate for all geometry. Scaling/squaring from the previous section can optionally be applied to reduce the magnitude of the input vector and so extend the domain over which we obtain accurate results.

4.3 Combining Techniques into Algorithms

We define and compare a number of algorithms for evaluating the SH exponential, using the techniques discussed above. PS-p uses the simple product series evaluation of degree p from (17). PS*-p uses DC isolation (20) and scaling/squaring (21) applied to a factored degree-p product series from (22). OL applies the optimal linear method from Section 4.2. We extend the accuracy of this method via a hybrid method, called HYB, which applies scaling/squaring in (21) to OL. HYB simply divides the input by a power of 2, applies OL, and then repeatedly squares the result. The different algorithms for SHEXP are compared in Figure 9.

5 SH Logarithm

A naive method for SH log applies (10):

$$\boldsymbol{f} = \log(\boldsymbol{g}) = \int_S \log\left(\max(g(s),\varepsilon)\right) \boldsymbol{y}(s)\, ds \qquad (25)$$

where we clip evaluations of $g(s)$ that are close to 0 or negative using a small threshold ε. This method works poorly for two reasons. First, it neglects how truncation error from taking log affects the subsequent exponential. Substantial error $\|\exp_*(\log(\boldsymbol{g})) - \boldsymbol{g}\|$ can be produced, which typically attenuates frequencies near the Nyquist band. Second, clipping to a constant introduces artificial high frequencies and so suboptimally picks a signal that's close to the original, $g(s)$, but avoids places where log is undefined.

We address both these problems by approximately inverting the exponential using an eigenanalysis of the product matrix M_g. Inverting the $\exp_*$ operator takes truncation into account, reducing high-frequency attenuation after the final exponential. Clipping in the space of eigenvalues of M_g rather than sampled spherical values $g(s)$ eliminates artificially-introduced high frequencies.

Diagonalizing SH Exp Rewriting (17) in terms of the product matrix M_f, we obtain

$$\boldsymbol{g} = \exp_*(\boldsymbol{f}) = \mathbf{1} + \boldsymbol{f} + \frac{\mathsf{M}_f \boldsymbol{f}}{2!} + \frac{\mathsf{M}_f^2 \boldsymbol{f}}{3!} + \cdots. \qquad (26)$$

Then eigenanalysis on the symmetric product matrix yields

$$\mathsf{M}_f = \mathsf{R}_f^T \, \mathsf{D}_f \, \mathsf{R}_f \;\Rightarrow\; (\mathsf{M}_f)^p = \mathsf{R}_f^T \, \mathsf{D}_f^p \, \mathsf{R}_f \qquad (27)$$

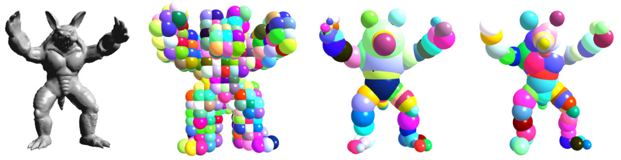
where R_f is a rotation matrix, D_f is a diagonal matrix, and powers of D are taken with respect to each of its diagonal components. Substituting these powers, (26) becomes

$$\begin{aligned} \boldsymbol{g} &= \exp_*(\boldsymbol{f}) = \mathbf{1} + \mathsf{R}_f^T \, q(\mathsf{D}_f)\, \mathsf{R}_f \, \boldsymbol{f} && (28) \\ q(x) &= 1 + \frac{x}{2!} + \frac{x^2}{3!} + \cdots = \frac{\exp(x) - 1}{x} && (29) \end{aligned}$$

where q is applied to each diagonal element of D_f. Note that (28) represents an alternative method of evaluating SH exponential which avoids error from truncating to a finite series. However, it is not practical to compute the required eigenanalysis on-the-fly, and we use this formulation only to derive a method for SH log, which is computed as a preprocess.

Inverting SH Exp We begin with an eigenanalysis of the product matrix of $\boldsymbol{g}$, $\mathsf{M}_g = \mathsf{R}_g^T \, \mathsf{D}_g \, \mathsf{R}_g$. (18) then implies that $\mathsf{M}_{\log(g)} \approx \mathsf{R}_g^T \log(\mathsf{D}_g)\, \mathsf{R}_g$ for positive definite product matrices M_g. (28) can therefore be approximately inverted using

$$\begin{aligned} \log(\boldsymbol{g}) &= \mathsf{R}_g^T \, q'(\mathsf{D}_g)\, \mathsf{R}_g \, (\boldsymbol{g} - \mathbf{1}) && (30) \\ q'(x) &= 1/q(\log(x)) = \log(x)/(x-1) && (31) \end{aligned}$$



original

E=4.1, n_S=575
octree

E=0.52, n_S=64
medial axis

E=0.35, n_S=64
our method

Figure 4: *Sphere set approximation. E is outside volume divided by the object's total volume.*

where the function q' is applied to each diagonal component.

To avoid applying log to values that are negative or close to 0, we clip the eigenvalues of M_g via

$$\tilde{\mathsf{D}}_g = \max(\mathsf{D}_g, \varepsilon), \quad \tilde{\mathsf{M}}_g = \mathsf{R}_g^T \, \tilde{\mathsf{D}}_g \, \mathsf{R}_g \qquad (32)$$

and apply (30) to $\tilde{\mathsf{D}}_g$ rather than D_g. Eigenvalue clipping yields smaller error $\|\mathsf{M}_g - \tilde{\mathsf{M}}_g\|$ compared with clipping values of $g(s)$ over the sphere as in (25). In practice, we have found that setting the threshold ε to 0.02 times the largest eigenvalue works well for low-order SH vectors.

6 Approximating Blocker Geometry

6.1 Single Sphere Visibility

As a preprocess, we tabulate the SH visibility function for circles of angular radius θ centered around the z axis, given by

$$g(s,\theta) = \begin{cases} 0, & \text{if } s \cdot (0,0,1) \geq \cos(\theta); \\ 1, & \text{otherwise.} \end{cases} \qquad (33)$$

Projecting g yields many zero components because of its circular symmetry around z. Using doubly-indexed SH notation, $\boldsymbol{g}_{lm}$ where $l = 0, 1, \ldots, n-1$, is the *band index* and m indexes the $2l+1$ components of band l, only the $\boldsymbol{g}_{l0}$ are non-zero. So g's projection can be represented with n rather than n^2 components, via the vector $\boldsymbol{g}_l(\theta)$, which we compute using numerical integration in (1).

Define a scaled version of this vector's components, via $\boldsymbol{g}_l^*(\theta) = \boldsymbol{g}_l(\theta)\sqrt{\frac{4\pi}{2l+1}}$ and assemble its components into a diagonal matrix $\mathsf{G}(\theta)$ in which $\boldsymbol{g}_l^*$ is repeated $2l+1$ times in band l. Using these definitions, the visibility function can be rotated from z to an arbitrary axis z' via the rotation rule [Sloan et al. 2005]

$$\boldsymbol{g}_{z'}(\theta) = \mathsf{G}(\theta)\, \boldsymbol{y}(z') = \operatorname{diag}(\boldsymbol{g}_0^*(\theta), \boldsymbol{g}_1^*(\theta), \boldsymbol{g}_1^*(\theta), \boldsymbol{g}_1^*(\theta), \ldots)\, \boldsymbol{y}(z').$$

Visibility vectors for circles of any angular radius and around any axis can therefore be defined using a 1D table of n projection coefficients, $\boldsymbol{g}^*(\theta)$, and a 2D table of n^2 SH basis functions, $\boldsymbol{y}(s)$. In fact, we tabulate the *logs* of circle visibility vectors $\boldsymbol{f} = \log(\boldsymbol{g})$ rather than their direct projections, using (30) in Section 5. But the logs are also circularly symmetric and so obey the same rotation rule. Only the projection coefficients change, which we denote by the vector $\boldsymbol{f}^*(\theta)$ and its corresponding diagonal matrix $\mathsf{F}(\theta)$.

At run-time, assume receiver point p is shadowed by a sphere of radius r centered at P. We can now define the log visibility vector, $\boldsymbol{f}(p,P,r)$, for this single sphere blocker used in (8):

$$\begin{aligned} \theta(p,P,r) &= \sin^{-1}(r/\|P-p\|) \\ s(p,P) &= (P-p)/\|P-p\| \\ \boldsymbol{f}(p,P,r) &= \mathsf{F}\left(\theta(p,P,r)\right) \boldsymbol{y}\left(s(p,P)\right) && (34) \end{aligned}$$

6.2 Sphere Sets

Construction We bound geometry within a set of spheres using a variational approach [Wang et al. 2006] which we briefly review here. Specifically, the algorithm seeks a set of n_S spheres $S_i = (P_i, r_i)$ bounding the shadowing geometry T but having minimal *outside volume* E, defined as

$$E(\{S_i\}, T) = \sum_{i=1}^{n_S} V(S_i - T) \tag{35}$$

where V is volume and subtraction represents set difference. Volume within T's interior is neglected regardless of how many spheres overlap there. This is appropriate for shadowing since it does not matter how many times light is blocked by a solid object. The sphere set's bounding property eliminates gaps that would otherwise let light leak through solid objects. Figure 4 compares our approach to octrees [Hubbard 1995] and the medial axis [Bradshaw and O'Sullivan 2004].

Our algorithm applies a variant of the Lloyd clustering algorithm [Lloyd 1982]. We first discretize T into a set of points including points on the surface (triangle midpoints) and points in the interior (grid corners from a mesh voxelization). The spheres S_i are initialized using a randomly-picked center point, P_i, and radius r_i=0. We then iteratively apply three steps: point clustering, cluster sphere update, and cluster teleportation, until error converges.

Point clustering is performed using a "flood fill" (stack-based) order away from the sphere centers. The next point is repeatedly popped from the stack and inserted into the cluster having minimal error. Its neighbors are then inserted onto the stack if not already there. Error from adding a point is measured by extending the cluster sphere's radius r_i so as to include the new point and then measuring the outside volume of the new sphere with respect to T.

After all points have been clustered, the set of cluster spheres bounds the set of points. We then independently update each cluster sphere center P_i in order to minimize outside volume $V(S_i - T)$ while constraining r_i to continue bounding the cluster's points. The minimal P_i is found using Powell's method [Press et al. 1992].

Cluster teleportation [Cohen-Steiner et al. 2004] is applied when iterations of the above two steps fail to significantly reduce E. The idea is to subdivide the cluster of maximal error into two by finding its farthest-apart pair of points. The cluster of maximal overlap, defined as volume it shares with other cluster spheres divided by its own volume, is chosen for deletion. After teleporting the cluster centers we again perform clustering and cluster sphere updates. If E is reduced we accept the teleported perturbation and otherwise revert to the previous state.

Outside volume of a sphere S, $V(S - T)$, is computed via a sum over each triangle of T of its signed outside volume with respect to S. See [Wang et al. 2006] for more details.

Animation Our models are animated using "skinning" which applies a weighted combination of transformations, attached to bones in an articulated skeleton, to the mesh vertices. As the model moves, we must also update its bounding spheres. We do this by finding the mean value coordinates (MVCs) [Tao et al. 2005] of each sphere center P_i with respect to the rest pose of the mesh, which expresses P_i as a weighted combination of the vertices. Applying the same weights to vertices in a deformed pose yields the corresponding deformed sphere center P_i'. We keep the original sphere radius r_i, which remains a bound assuming typical articulated motion. To speed up the weighting calculations, we premuliply the vector of vertex MVCs by a matrix of bone weights per vertex to express the sphere center transformation as a weighted combination over a few bones rather than many vertices.

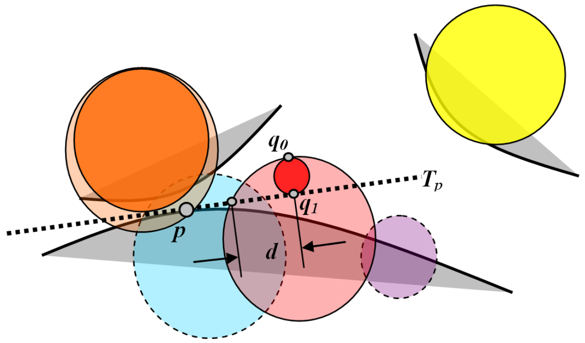


Figure 5: *Self-shadowing with blocker spheres. Spheres (purple) behind the tangent plane T_p at receiver p are eliminated. Spheres straddling T_p are eliminated if p is inside but the sphere center is behind T_p (blue), or are replaced with spheres tangent to T_p otherwise (red, orange). Spheres (yellow) that are entirely in front of T_p are unaltered.*

Avoiding Problems with Self-Shadowing A bounding sphere set implies that every point on an object is completely self-shadowed. We therefore need to distinguish a blocker representing the same local geometry as the receiver which casts incorrect self-shadows, e.g., the blue sphere in Figure 5, from a blocker representing non-local geometry which casts shadows we want to preserve, e.g., the orange sphere in Figure 5. Denote a receiver point as p and its outward-facing normal as N_p, together defining the tangent plane T_p. If a blocker sphere S contains p, we use the relationship of S's center to T_p as the local/non-local proxy. We eliminate S if its center is behind T_p and reduce its radius until it becomes tangent to T_p if its center is in front.

When p is outside S, it is simplest to accumulate the blocker regardless of its position relative to T_p. This method produces objectionable banding because we obtain different shadows depending on whether the receiver is inside one or more local blockers. On the other hand, entirely eliminating local spheres fails to capture important local self-shadowing effects.

We instead remove a blocking sphere S outside p only if it is entirely behind T_p. If it partially pokes through, we replace it with a sphere S' tangent to and in front of T_p. S' is determined by the point of maximal distance of S in front of T_p, q_0, as well as the projection of q_0 onto T_p, q_1. q_0 and q_1 form a diameter of S' and their midpoint forms its center.

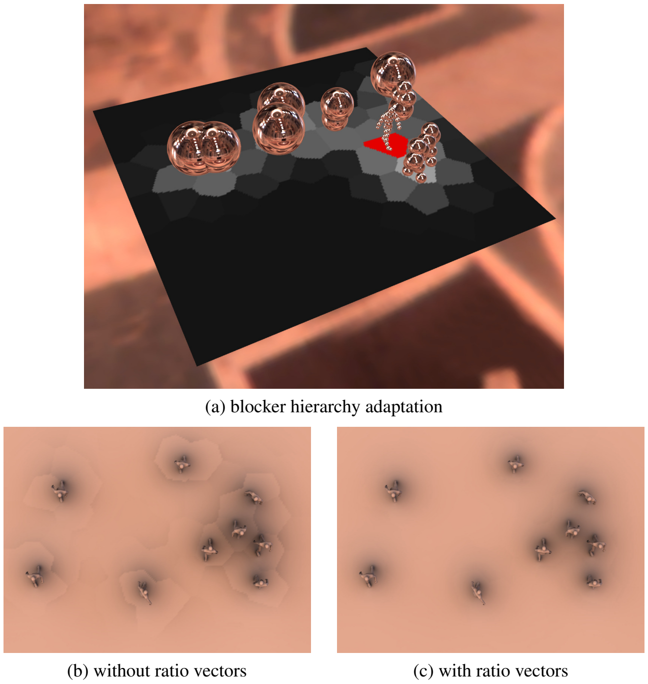
By itself, this replacement rule causes a spatial discontinuity as p moves from inside S to outside. We solve this problem by gradually scaling up the radius of S' as a function of p's distance to S along the tangent plane, using the scale factor α

$$\alpha = \max\left(1, (\|p - q_1\| - d)/d\right), \;\; d = \sqrt{r^2 - (r - \|q_1 - q_0\|)^2}$$

where r is the radius of S and d represents the distance of q_1 to the outside of S along the tangent plane.

6.3 Sphere Hierarchies

As a blocker gets close to a receiver point, accurate shadows require detailed knowledge of the blocker's shape. But as the blocker recedes, it can be approximated more coarsely. We exploit this observation using a blocker sphere hierarchy and clustering over receiver points (Figure 6a). For each receiver point cluster, we assemble a *cut* or list of blocker spheres from appropriate levels of the hierarchy. They are selected conservatively based on the angle they subtend over the cluster, using an angular radius threshold $\theta_{\max}$ between 20 and 30 degrees. We then accumulate the log visibility vector at each receiver vertex p in the cluster using a simple approximation that exploits spatial coherence and based on detailed information computed at a single point p^* centered in the cluster. This eliminates artifacts from inconsistent blocker approximations used in different clusters (Figure 6bc).



(a) blocker hierarchy adaptation

(b) without ratio vectors

(c) with ratio vectors

Figure 6: *Blocker hierarchy adaptation and receiver clustering. In (a), the blocker approximation adapts to the receiver cluster in red. Brightness of clusters on the ground corresponds to the number of spheres processed in its cut. Artifacts due to inconsistencies in the blocker approximation from one cluster to the next (b) are eliminated using ratio vectors based on detailed blocker information at a single cluster point (c).*

Construction and Animation The sphere set from Section 6.2 forms the leaf nodes in a blocker hierarchy. Hierarchy levels are constructed one at a time from the leaves up to the root, using a clustering technique based on [Lloyd 1982]. Each cluster stores its current bounding sphere. Clustering iteratively assigns spheres to the closest cluster, based on the distance from the sphere center to the cluster's. The cluster's bounding sphere is then updated. Its center is initially taken as the average center over all spheres assigned to it, and then optimized using Powell's method to minimize the bounding sphere's radius. After convergence, each cluster is made a parent node in the hierarchy; the spheres assigned to it become its children. The number of clusters is chosen so as to make the average branching ratio in the hierarchy equal to 4.

During the animation, bounding spheres at each parent node are updated bottom-up in the hierarchy by applying a sphere pair bound to successively merge in each child node.

Receiver point clusters are computed using simple Lloyd clustering into a manually-specified number of clusters. Our examples typically contain several hundred vertices per receiver cluster.

Per-Cluster Computation We first assemble bounding sphere nodes, S_i, with centers at P_i and radii r_i, which are appropriate for shadowing the cluster. To do this, assume the receiver cluster is bounded by a sphere with center p_R and radius r_R. Then the blocker sphere subtends an angle less than $\theta_{\max}$ if

$$r_i < \sin(\theta_{\max})\,(||P_i - p_R|| - r_R). \qquad (36)$$

If $S_i = (P_i, r_i)$ satisfies the above test, it is inserted into the sphere blocker list; otherwise, we recurse to the node's children.

For each S_i in the list just assembled, we compute two visibility vectors at a central cluster point p^*. The *bounding* log visibility vector $\boldsymbol{f}_b[i](p^*, P_i, r_i)$ applies (34) to the bounding sphere S_i. The *detailed* log visibility vector $\boldsymbol{f}_d[i]$ sums log visibility over all leaf node spheres below S_i. To accelerate this computation, we can prune detailed spheres using a minimum angular radius $\theta_{\min}$.

We then compute a *ratio vector*, $\boldsymbol{w}[i]$, representing the least-squares best per-band scaling of $\boldsymbol{f}_b[i]$ to match $\boldsymbol{f}_d[i]$:

$$\boldsymbol{w}_l[i] = \left(\sum_{m=-l}^{+l} (\boldsymbol{f}_b[i])_{lm} (\boldsymbol{f}_d[i])_{lm} \right) \Big/ \left(\sum_{m=-l}^{+l} (\boldsymbol{f}_b[i])^2_{lm} \right) \qquad (37)$$

A diagonal matrix $\mathsf{W}[i]$ is derived from $\boldsymbol{w}[i]$ by repeating its component $2l+1$ times along the diagonal, as for F in Section 6.1.

Per-Point Computation For each receiver point p in the cluster, we use the information computed previously at the cluster center p^* to accumulate a log visibility vector at p. We apply a modified version of (34) to each sphere i in the cluster's blocker list, which multiplies by the ratio vector $\mathsf{W}[i]$:

$$\boldsymbol{f}[i](p, P_i, r_i) = (\mathsf{W}[\mathrm{i}]\;\mathsf{F}(\theta(p, P_i, r_i)))\;\boldsymbol{y}(s(p, P_i)). \qquad (38)$$

Finally, we sum vectors over i and apply $\exp_*$ using (8).

Rationale Cluster artifacts arise because different cuts are used in different clusters to approximate the same leaf node blocker spheres. These artifacts are greatly reduced by ratio vectors, which modulate the "overblocking" visibility function of a bounding sphere to provide a better approximation to the visibility of its actual leaf nodes. Cost is minimal: just an additional scaling by a diagonal matrix. In fact, the cost is even less since the n per-band weights from F and W can be multiplied before they are repeated and used to scale $\boldsymbol{y}$. The approximation works because the blocker has been restricted to a small solid angle. Intuitively, as we get further from a blocker its log visibility function approaches a circularly symmetric delta function. When the receiver point moves from p^* to p, we account for visibility change based on how the bounding sphere's projection rotates (from $s(p^*, P_i)$ to $s(p, P_i)$) and scales (from $\theta(p^*, P_i, r_i)$ to $\theta(p, P_i, r_i)$). This is a good approximation assuming there is little parallax between individual leaf spheres. But this should be true because we have limited the solid angle subtended by a bounding sphere on the group, which therefore limits the group's depth extent.

7 Shading

For diffuse surfaces in lighting environments, we tabulate $\boldsymbol{L}_H(N) = \boldsymbol{L} * \boldsymbol{H}(N)$ as a cube map, where $\boldsymbol{L}$ represents the lighting and $\boldsymbol{H}(N)$ is defined in (9). At run-time we index $\boldsymbol{L}_H$ at the receiver normal N_p to obtain cosine-weighted incident radiance at p. The result is then dotted with the exponentiated blocker vector $\boldsymbol{g}$ from (8) to produce the shadowed result, $\boldsymbol{L}_H(N_p) \cdot \boldsymbol{g}$.

If the lighting changes every frame, tabulating $\boldsymbol{L}_H$ is difficult. Shading can instead be calculated by forming the light's product matrix M_L and computing $(\mathsf{M}_L\, g) \cdot \boldsymbol{H}(N_p)$ on-the-fly. Our examples and video results use rotations of fixed lighting and so use the faster method of tabulating $\boldsymbol{L}_H$.

For static receiver points, local shadowing effects can be "baked in" by dotting with the precomputed vector $\boldsymbol{H}(N_p) * \boldsymbol{g}_p$ where $\boldsymbol{g}_p$ represents local visibility due to static occluders. Blocker accumulation in (9) then needs to take place only over blockers from the dynamic geometry. $\boldsymbol{L}$ can be multiplied into $\boldsymbol{g}_p$ as well if the lighting is static. For receiver points on dynamic geometry, both static and dynamic blockers must be accumulated every frame.

Analytic Lights We can also handle circular/spherical light sources, which we define in terms of $\boldsymbol{L}(\theta, d)$ where θ is the angular radius of the light circle, and d is its central direction. To handle local light sources, we allow θ and d to vary as a function of the receiver point p. We support this by tabulating $\boldsymbol{L}_H(\theta, \phi) =$

$\boldsymbol{H}((0,0,1)) * \boldsymbol{L}(\theta,\phi)$ where θ is the light's angular radius and ϕ is the angle the central light direction makes with the normal N. This 2D table uses a canonical orientation aligning the normal N with z and the light direction in the xz plane, making an angle of ϕ with z. This canonical configuration must then be rotated into its actual orientation at each receiver point before computing the dot product with $\boldsymbol{g}$. We accelerate this rotation by fitting a single-lobe ZH model [Sloan et al. 2005] to the 2-parameter family of vectors $\boldsymbol{L}_H(\theta,\phi)$. Rotation then uses the same rule discussed in Section 6.1.

Windowing Frequency bases like SH induce "ringing" artifacts (Gibbs oscillation) when reconstructed. The standard remedy is to attenuate the higher frequencies via a windowing function. We window lighting and visibility functions via a cosine filter in frequency space, which scales SH coefficents in band l by $\alpha_l = \cos(\pi/2\,(l/h))$. For order-$n$ SH, a window size $h = 2n$ works well. Some HDR lighting environments may need greater windowing (smaller h), depending on their frequency content.

8 GPU Implementation

Soft shadows from low-frequency lighting do not require dense spatial sampling – sampling at vertex rather than pixel rates typically suffices. Our GPU implementation takes advantage of this fact to accelerate the rendering. Unfortunately, current graphics hardware such as the Nvidia 6800/7800 makes available much less computational power in the vertex compared to the pixel processors.[1] We therefore resort to a stopgap device: using pixel processors to do per-vertex shading. Future hardware, such as Xbox360 and Direct3D 10 [Blythe 2006], will better balance processing power and so enable the straightforward use of vertex shaders.

The whole rendering process consists of three passes. The first pass generates vertex and blocker information on the CPU, the second pass does the shadowing and shading on the GPU, and the final pass renders the resulting shaded vertices to the screen on the GPU.

As a preprocess (computed offline or at program load time), the log visibility coefficients $\boldsymbol{f}_l(\theta)$ and a,b tables of the OL method are prepared and stored as texture maps. To handle local light sources, another 2D table of ZH coefficients, $\boldsymbol{L}_H(\theta,\phi)$, is also prepared. At runtime, the first pass prepares vertex info: position, normal, and receiver cluster id. For each receiver cluster, it also assembles the blocker information: center, radius, and weight vector $\boldsymbol{w}_l[i]$ for each blocker sphere from the sphere tree cut (see Section 6.3).

The second pass reads from a texture called the *vertex info map* which stores the position, normal and receiver cluster id for an array of pixels, each representing a vertex. Since each vertex in the same cluster shares the same sphere cut, the shader uses the cluster id to locate the corresponding row in the *sphere info map*, where the center, radius and weight vector of cut spheres are stored. In a while loop, each sphere i in the cut list is processed via (38) and the log visibility vector is accumulated. Dynamic branching is used here to avoid self-shadow problems, following the rules in Section 6.2. After accumulating all the blockers in the cut list, the shader evaluates the HYB algorithm for exponentiation (Section 4.3), yielding the visibility vector. A dot product with $\boldsymbol{L}_H(N)$ then yields the color for the pixel (really, the vertex).

To indicate when the while loop above should terminate, we store a terminator sphere at the end of each cut list having a radius of 0. The row size of the sphere info map specifies the maximum number of blocker spheres we can store in any cut list. We use space for 128 or 256 blocker spheresa in our examples. Circle log vectors, $\boldsymbol{f}^*(\theta)$ from Section 6.1, use 256 samples in θ.

We perform per-vertex shading using the OpenGL pixel buffer object (PBO) and vertex buffer object (VBO) extensions. These routines form a video memory management API which in combination eliminate a read-back from video memory to host memory between the second and final pass. The result of the second pass is transferred from the frame buffer to a PBO via a *glReadPixels* call. This performs a video-memory to video-memory copy which can be done very fast, over 300Hz for a 512 × 512 RGBA frame buffer on a NVidia 6800GT. The PBO then becomes a VBO to make the shaded colors available as vertex information.

More implementation details are included in [Ren et al. 2006].

9 Results

Timings in this section were performed on a 3.2Ghz PC with 1GB of memory and an NVidia 7800GTX graphics card. Screen/image generation was done at 1280×1024 resolution.

Figure 7 shows three progressive sources of approximation error in our approach: using a sphere set rather than the actual blocker geometry (b), truncating to a fixed order after each SH product rather than performing a numerical integration that samples the actual (binary) blocker product (c), and accumulating in log space rather than product space (d). This comparison uses SH order n=4 and PS*-2 for SH exponentiation; all rendering is done on the CPU. The main source of visual error is not from our log space approximation. In fact our method is almost visually identical to the slower product accumulation method [Zhou et al. 2005]. Most error arises from truncating visibility after each SH product (causing a general darkening of shadows), and to a lesser extent, approximating blockers as sphere sets (causing slightly enlarged shadows). The second type of error is unsurprising given our bounding blocker approximation, subtle in our view, and easily reduced by using more blocker spheres, though obviously at increased rendering cost. The first type of error can be reduced by accumulating blockers at higher SH order. In (e), we accumulate using n=6 blocker vectors, exponentiate, and then truncate to n=4 before shading, resulting in a more accurate rendering than (c) or (d). The log space method makes such higher-order accumulation much more tractable, and thus addresses a previously unidentified problem in [Zhou et al. 2005].

Figure 8 compares shadowing at various SH orders n. In order to accentuate differences, the lighting applied is as close to a delta function as is realizable at n; no windowing is used. The factor below each log space image represents total rendering speedup observed in a CPU implementation, when accumulating blockers in log space and applying PS*-2 rather than accumulating products. Approximation error from doing the computation in log space is difficult to see. The figure includes only the order 6 product result for conciseness but product images for other orders are available in [Ren et al. 2006]. Higher-order SH vectors produce "peakier" lights and sharper shadows, as well as bigger speedups for log compared to product accumulation.

Figure 9 compares error from various SH exponentiation methods to the "gold standard" rendering using product accumulation in (a). SH order n=4 is used without windowing to accentuate differences. HYB and OL are computed on the GPU, all other methods on the CPU. The figure records the number of SH operations (squares and products) and rendering rate for each method, using a ground plane shadow receiver with 128×128 vertices. A simple product series (b), even as high as degree 21, has an incorrect bright spot in the shadow. (Actually, odd and even degree p exhibit incorrect bright and dark spots alternately and converge in this example only after p=30.) HYB (d) and PS*-2 (e) match very accurately. Even OL (c) provides a good approximation, though it blurs more in the heavily shadowed regions. More such comparisons are available in [Ren et al. 2006].

Figure 10 presents images rendered on the GPU from two, more

[1] The NVidia 6800/7800 has 16 fragment processors, compared with 6 vertex processors. Texture fetching in the pixel shader is also much more efficient than in the vertex shader.

complex scenarios. SH order n=4 is used, and lighting and visibility vectors windowed to reduce ringing. HYB is used for SH exponentiation. Both scenes use a blocker hierarchy, receiver point clustering, and per-cluster ratio vectors from Section 6.3. The entire sequences, captured in real time, are available in the video results. We obtain good performance (10-30Hz) and high-quality soft shadows, including self-shadows, from the moving characters. Additional snapshots, including comparisons with shadowing turned on and off, are available in [Ren et al. 2006].

The battle scene involves two characters (troll and wizard) and contains 65k vertices (41k static and 24k dynamic) and 244 leaf node sphere blockers. 120 receiver clusters were used: 100 for the ground plane, 8 each for the wizard and troll, 4 for the troll's club, and 1 each for each of the other objects (rock and columns). On average, 47 spheres were accumulated per receiver point; a maximum of 117 blocker spheres was observed. The scene is illuminated by the Uffizi HDR environment, bandlimited to SH order n=4, as well as two local spherical light sources. Measured frame rate was 15.6-44.5Hz and averaged 26.2Hz.

The dino scene contains a sequence of different clips. The most complicated, running at 10.6-13.7Hz, contains 8 moving dinosaurs, 120k vertices (75k static and 45k dynamic), 500 blocker leaf node spheres and 256 receiver clusters (192 for the ground surface and 8 for each dinosaur). An average of 42 sphere blockers per receiver were accumulated, up to a maximum of 160. Frame rate over all clips was 10.6-33.2Hz and averaged 14.1Hz.

10 Conclusion and Future Work

Accumulating low-frequency blocker visibility in the spherical harmonic basis provides a direct method for rendering soft shadows without integrating over a huge number of lighting directions. We acclerate this approach by accumulating in log space rather than product space, and then computing the SH exponential required using new methods (HYB and PS*-p). Per-blocker computation is greatly reduced, allowing us to handle more blockers and to map the computation to the GPU in a single shading pass. Our algorithm is applicable to general, dynamic geometry including deforming characters whose motion need not be known in advance.

In future work we are considering anisotropic blocker models, handling diffuse inter-reflection, and experimenting with alternative models for spatial shading variation.

Acknowledgements

The authors would like to thank Becky Sundling for her help in video production and Mingdong Xie for modeling and animation of the battle and dinosaur scenes. We are grateful to the anonymous reviewers for their helpful suggestions and comments. The ZJU authors were partially supported by NSFC (No. 60021201), 973 Program of China (No. 2002CB312102) and the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China (No.705027).

References

Agarwala, M., Ramamoorthi, R., Heirich, A., and Moll, L. 2000. Efficient image-based methods for rendering soft shadows. In *Proc. of ACM SIGGRAPH 2000*, 375–384.

Assarsson, U., and Akenine-Möller, T. 2003. A geometry-based soft shadow algorithm using graphics hardware. *ACM Trans. Gr. 22*, 3, 511–520.

Blythe, D. 2006. The Direct3D 10 system. *to appear in Proc. ACM SIGGRAPH 2006 (ACM Trans. Gr.)*.

Bradshaw, G., and O'Sullivan, C. 2004. Adaptive medial-axis approximation for sphere-tree construction. *ACM Trans. Gr. 23*, 1 (Jan.), 1–26.

Bunnell, M. 2004. Dynamic ambient occlusion and indirect lighting. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Weseley Professional, 223–233.

Cohen-Steiner, D., Alliez, P., and Desbrun, M. 2004. Variational shape approximation. *ACM Trans. Gr. 23*, 3, 905–914.

Higham, N. 2005. The scaling and squaring method for the matrix exponential revisited. In *SIAM Journal of Matrix Analysis Applications*, no. 4, 1179–1193.

Hubbard, P. 1995. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 3, 218–230.

James, D., and Fatahalian, K. 2003. Precomputing interactive dynamic deformable scenes. *ACM Trans. Gr. 22*, 3, 879–887.

Kautz, J., Sloan, P., and Snyder, J. 2002. Fast, arbitrary BRDF shading for low-frequency lighting using spherical harmonics. In *Proc. of the 13th Eurographics Workshop on Rendering*, 291–296.

Kautz, J., Lehtinen, J., and Aila, T. 2004. Hemispherical rasterization for self-shadowing of dynamic objects. In *Proc. of Eurographics Symposium on Rendering 2004*, 179–184.

Kontkanen, J., and Laine, S. 2005. Ambient occlusion fields. In *Proc. of 2005 Symposium on Interactive 3D Graphics, SI3D 2005*, 41–48.

Laine, S., Aila, T., Assarsson, U., Lehtinen, J., and Akenine-Möller, T. 2005. Soft shadow volumes for ray tracing. *ACM Trans. Gr. 24*, 3, 1156–1165.

Lloyd, S. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory IT-28*, 2 (Mar.), 129–137.

Malmer, M., Malmer, F., Assarsson, U., and Holzschuch, N. 2005. Fast precomputed ambient occlusion for proximty shadows. Tech. Rep. 5779, INRIA.

McCool, M., Ang, J., and Ahmad, A. 2001. Homomorphic factorization of BRDFs for high-performance rendering. In *Proc. of ACM SIGGRAPH 2001*, 171–178.

Mei, C., Shi, J., and Wu, F. 2004. Rendering with spherical radiance transport maps. *Eurographics 2004 (Computer Graphics Forum) 23*, 3, 281–290.

Ng, R., Ramamoorthi, R., and Hanrahan, P. 2003. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Trans. Gr. 22*, 3, 376–381.

Ng, R., Ramamoorthi, R., and Hanrahan, P. 2004. Triple product integrals for all-frequency relighting. *ACM Trans. Gr. 23*, 3, 477–487.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. 1992. *Numerical Recipes in C, Second Edition*. Cambridge University Press, Cambridge, England.

Ramamoorthi, R., and Hanrahan, P. 2001. An efficient representation for irradiance environment maps. In *Proc. of ACM SIGGRAPH 2001*, 497–500.

Ren, Z., Wang, R., Snyder, J., Zhou, K., Liu, X., Sun, B., Sloan, P., Bao, H., Peng, Q., and Guo, B. 2006. Supplement for real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. Tech. rep., Microsoft Corporation. available on the SIGGRAPH 2006 Conference DVD.

Schetzen, M. 1980. *The Volterra and Wiener Theories of Nonlinear Systems*. John Wiley and Sons.

Segal, M., Korobkin, C., Van Widenfelt, R., Foran, J., and Haeberli, P. 1992. Fast shadows and lighting effects using texture mapping. In *Proc. of SIGGRAPH 92*, 249–252.

Sloan, P., Kautz, J., and Snyder, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Gr. 21*, 3, 527–536.

Sloan, P., Hall, J., Hart, J., and Snyder, J. 2003. Clustered principal components for precomputed radiance transfer. *ACM Trans. Gr. 22*, 3, 382–391.

Sloan, P., Luna, B., and Snyder, J. 2005. Local, deformable precomputed radiance transfer. *ACM Trans. Gr. 24*, 3, 1216–1224.

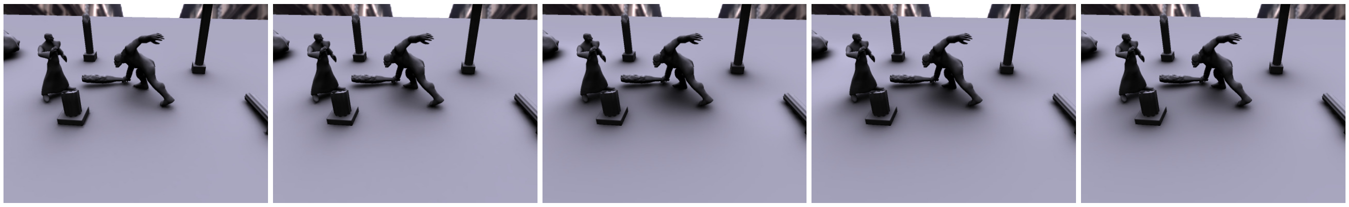
Snyder, J. 2006. Code generation and factoring for fast evaluation of low-order spherical harmonic products and squares. Tech. Rep. MSR-TR-2006-53, Microsoft Corporation.

Soler, C., and Sillion, F. 1998. Fast calculation of soft shadow textures using convolution. In *Proc. of ACM SIGGRAPH 1998*, 321–332.

Tao, J., Schaefer, S., and J., W. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Gr. 24*, 3, 561–566.

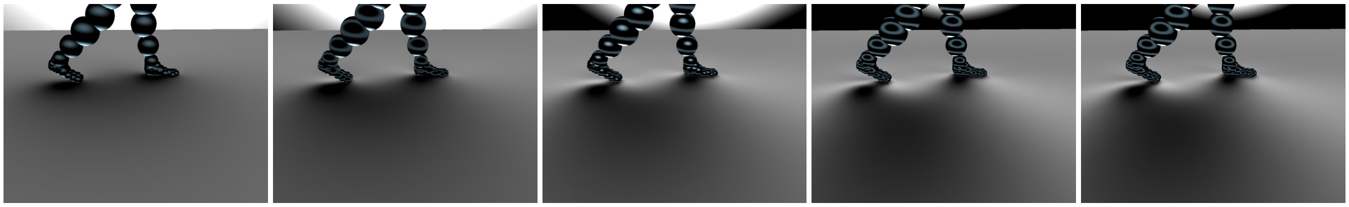
Wang, R., Liu, X., Zhou, K., and Snyder, J. 2006. Variational sphere set approximation for solid objects. *Submitted to Pacific Graphics*.

Zhou, K., Hu, Y., Lin, S., Guo, B., and Shum, H. 2005. Precomputed shadow fields for dynamic scenes. *ACM Trans. Gr. 24*, 3, 1196–1201.



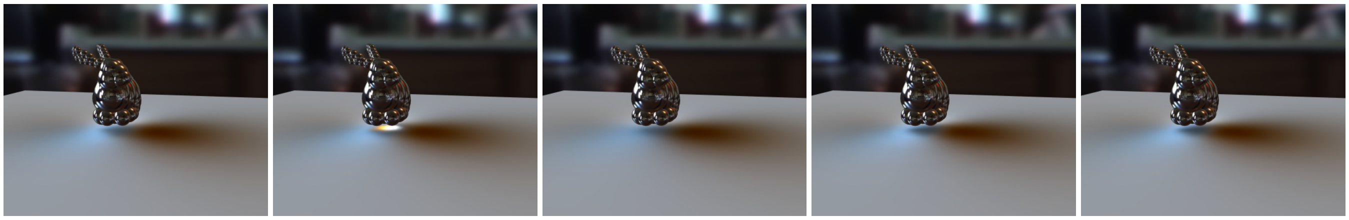
(a) ray traced, actual blockers (b) ray traced, sphere blockers (c) SH product, *n*=4 (d) SHEXP, *n*=4 (e) SHEXP, *n*=6

Figure 7: *Approximation error progression. Lighting is SH order n=4.*



(a) *n*=3, SHEXP, 1.9x, 7.1Hz (b) *n*=4, SHEXP, 3.4x, 5.5Hz (c) *n*=5, SHEXP, 6.5x, 4.3Hz (d) *n*=6, SHEXP, 11.0x, 3.3Hz (e) *n*=6, SH prod., 0.3Hz

Figure 8: *SH order comparison. Factors like "1.9x" in (a) are the increase in speed of the SHEXP method over the SH product method of the same order. Timings were done on a CPU-only rendering using 200×200 shaded vertices on the ground plane and n_S=60 shadowing spheres (ho hierarchy).*



(a) product space (CPU)
#prod=494,790; #squares=0
2.47Hz

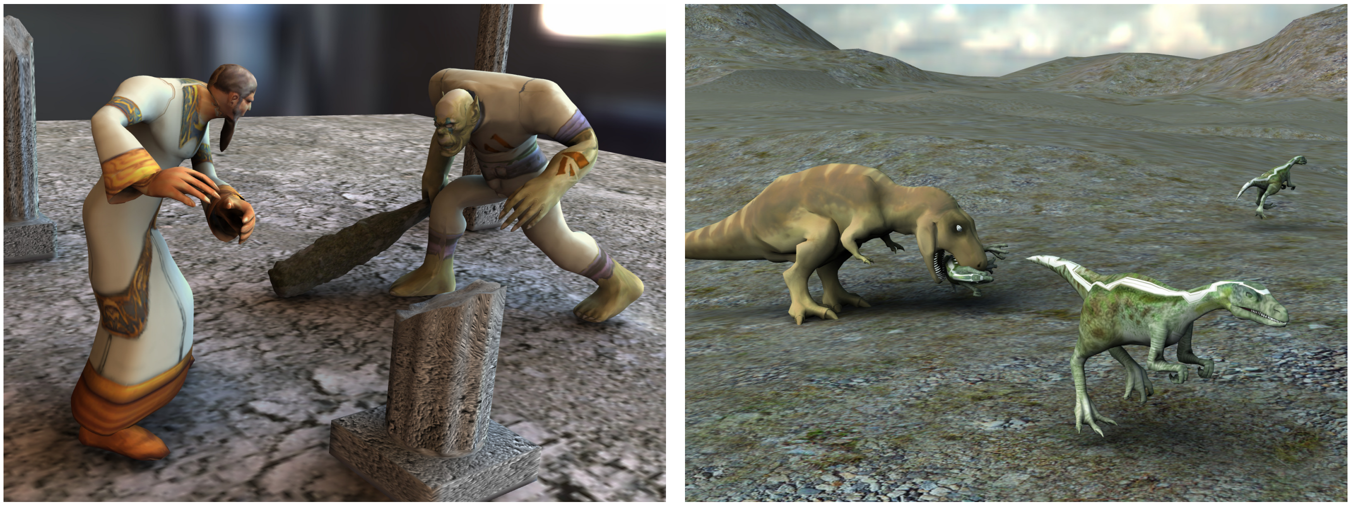
(b) PS-21 (CPU)
#prod=327,680; #squares=0
3.27Hz

(c) OL (GPU)
#prod=0; #squares=0
90.4Hz

(d) HYB (GPU)
#prod=0; #squares=30,784
82.6Hz

(e) PS*-2 (CPU)
#prod=0; #squares=45,518
7.53Hz

Figure 9: *SH exponential method comparison. The bunny model contains n_S=63 spheres.*



(a) battle scene, 23.2Hz (b) dino scene, 22.4Hz

Figure 10: *Images from GPU rendering. Frame rate for the particular image is shown.*