Eurographics Symposium on Rendering (2004)
H. W. Jensen, A. Keller (Editors)

Hemispherical Rasterization for Self-Shadowing of Dynamic Objects

Jan Kautz[†] Jaakko Lehtinen[*‡] Timo Aila[*§]

† MIT, ∗ Helsinki University of Technology / TML, ‡ Remedy Entertainment, Ltd., § Hybrid Graphics, Ltd.

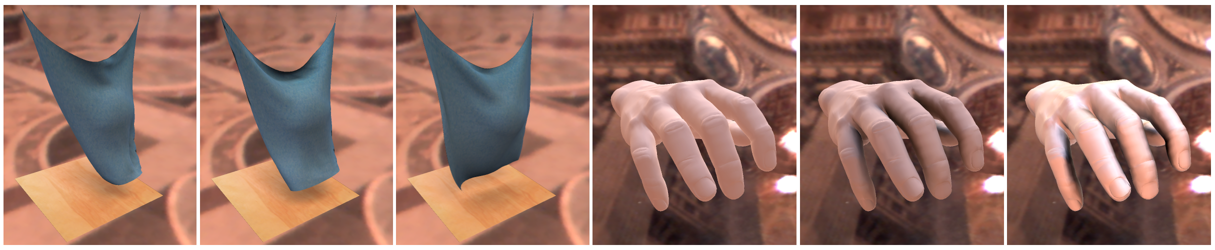

Figure 1: *A simulated piece of cloth is rendered at 15 FPS with soft shadows cast by a dynamic lighting environment. Images of an animated hand are shown without shadows, with shadows using a diffuse BRDF (6.1 FPS), and using a glossy BRDF (6.0 FPS).*

Abstract

We present a method for interactive rendering of dynamic models with self-shadows due to time-varying, low-frequency lighting environments. In contrast to previous techniques, the method is not limited to static or pre-animated models. Our main contribution is a hemispherical rasterizer, which rapidly computes visibility by rendering blocker geometry into a 2D occlusion mask with correct occluder fusion. The response of an object to the lighting is found by integrating the visibility function at each of the vertices against the spherical harmonic functions and the BRDF. This yields transfer coefficients that are then multiplied by the lighting coefficients to obtain the final, shadowed exitant radiance. No precomputation is necessary and memory requirements are modest. The method supports both diffuse and glossy BRDFs.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Bitmap and frame buffer operations I.3.7 [Computer Graphics]: Color, Shading, Shadowing and Texture

1. Introduction

Lighting from area sources or from environment maps generate complex, soft shadows that are important for realistic image synthesis. Shadows convey important information of the configuration of the scene and contribute much to the perceived quality of synthetic images. Unfortunately, most shadow techniques [HLHS03] are limited to point-like or small area light sources. In contrast, recent methods of precomputed radiance transfer (e.g., [SKS02]) are able to render complex lighting effects due to dynamic lighting environments at interactive or even real-time rates. The techniques are applicable only to rigid objects or pre-stored animations due to long precomputation times.

Contributions. We present a method for interactive rendering of self-shadowed, animated objects lit by dynamic, low-frequency lighting environments. Our method requires no precomputation, has modest memory requirements, and is capable of rendering both diffuse and other BRDFs. In particular, no prior knowledge of the animation is required. Lighting environments are specified by environment maps, and lighting is evaluated at the vertices of the model.

We introduce a hemispherical rasterizer for efficient computation of visibility. We rasterize occluding geometry in a small bitmap that contains binary visibility information. Our technique rasterizes spherical blocker triangles, i.e., their edges are curves in the planar visibility mask. Rasterized vis-

© The Eurographics Association 2004.

ibility information is then used for computing transfer vectors [SKS02] that are used to determine final shading values for vertices. While our method does not support interreflections, our results show that it is possible to render self-shadowed, animated models of a few thousand polygons at interactive rates.

2. Related Work

There is a vast amount of literature on shadowing algorithms. Most techniques are restricted to point-like or area lights and cannot efficiently render shadows cast by arbitrary lighting environments. See the survey by Hasenfratz et al. [HLHS03] and the references therein for an introduction.

Several techniques to speed up visibility computation in off-line rendering have been proposed, e.g. [HDG99, ARBJ03]. These do not easily lend to an interactive implementation.

Environment mapping [BN76] and its derivatives render reflections of dynamic, spherical incident lighting, but do not account for shadowing. Modern methods can handle arbitrary BRDFs.

Gibson et al. [GCHH03] present a method to place new objects interactively into an acquired scene so that the new objects correctly cast shadows onto the existing scene. The incident lighting can be arbitrary as long as it remains static.

Wald et al. [WBS03] demonstrate a parallel ray-tracer that is capable of rendering globally illuminated scenes at interactive rates, running on a cluster of computers. The key aspects are parallelism and optimal acceleration data structures for ray-tracing. As building such structures takes up to a few seconds even for small models, the method is too expensive to be applied to interactively animated geometry.

2.1. Radiance Transfer

Here we briefly review *precomputed radiance transfer*, which is the basis for our work. Assume that an object is illuminated by distant illumination $L_{in}(\mathbf{s})$, represented by an environment map. The exitant radiance $L_{out,p}(\mathbf{v})$ into direction $\mathbf{v}$ from a point p on the object is computed by

$$\begin{aligned} L_{out,p}(\mathbf{v}) &= \int_\Omega L_{in}(\mathbf{s})\, V_p(\mathbf{s})\, f_r(\mathbf{s}\to\mathbf{v})\, \max(0,\mathbf{n}_p\cdot\mathbf{s})\, ds, \\ &= \int_\Omega L_{in}(\mathbf{s})\, V_p^*(\mathbf{s},\mathbf{v})\, ds, \qquad \text{with} \qquad (1)\\ V_p^*(\mathbf{s},\mathbf{v}) &:= V_p(\mathbf{s})\, f_r(\mathbf{s}\to\mathbf{v})\, \max(0,\mathbf{n}_p\cdot\mathbf{s}), \end{aligned}$$

where $L_{in}(\mathbf{s})$ is the incident radiance, $V_p(\mathbf{s})$ is the visibility function that has zero value for directions where the environment cannot be seen due to self-shadowing and one otherwise, $f_r(\mathbf{s}\to\mathbf{v})$ is the BRDF, and $\mathbf{n}_p$ is the surface normal. $V_p^*(\mathbf{s},\mathbf{v})$ is the *transfer function*. Note that since the lighting is distant, it is assumed to be independent of p.

Now we project the incident lighting and the transfer function into the orthonormal basis $\mathcal{B}$ with basis functions $b_i(\mathbf{s})$, which yields coefficient vectors $\mathbf{L}_{in}$ and $\mathbf{V}_p^*(\mathbf{v})$. Sloan et al. [SKS02] showed that computing the exitant radiance then simplifies to the inner product of the two coefficient vectors:

$$L_{out,p}(\mathbf{v}) = \mathbf{L}_{in}\cdot\mathbf{V}_p^*(\mathbf{v}). \qquad (2)$$

We call $\mathbf{V}_p^*$ the *transfer vector.* Assuming static models and a diffuse BRDF the transfer vectors are constants that can be precomputed. The lighting is projected into the basis $\mathcal{B}$ at run-time and exitant radiances are evaluated as above. This enables shading static objects by time-varying, spherical lighting environments, with soft shadows and even interreflections using a more elaborate preprocess. Also non-diffuse BRDFs are possible, but then transfer needs to be captured by a transfer matrix instead of a vector. This necessitates offline data reduction [SHHS03] for achieving real-time framerates. A special case that avoids these problems is rendering with a fixed viewpoint [NRH03].

Sloan et al. used the spherical harmonics (SH) as the basis $\mathcal{B}$, while Ng et al. used Haar wavelets. Spherical harmonics are good for representing low-frequency lighting environments, since only a few coefficients are needed. See Sloan et al. [SKS02] for an introduction. Wavelets are preferable for higher-frequency lighting, since most of the energy of ordinary environment maps is captured by a small number of coefficients. However, which coefficients carry most energy is unknown at preprocessing time and thus the offline transfer simulation needs to account for all basis functions.

James and Fatahalian [JF03] compute transfer vectors for a sparse set of frames from a given set of animations. They perform principal component analysis (PCA) on the data, and interpolate solutions for intermediate frames in the low-dimensional PCA basis. While the algorithm is able to render preanimated models in dynamic lighting conditions in real-time, the precomputation times are prohibitively long.

3. Dynamic Radiance Transfer

We propose to compute the transfer coefficients on the fly. In order to determine the coefficients $\mathbf{V}_p^*(\mathbf{v})$ for vertex p, the hemispherical integral

$$V_{p,i}^*(\mathbf{v}) = \int_\Omega b_i(\mathbf{s})\, \left(V_p(\mathbf{s})\, f_r(\mathbf{s}\to\mathbf{v})\, \max(0,\mathbf{n}_p\cdot\mathbf{s})\right)\, ds \qquad (3)$$

needs to be evaluated. Evaluating the visibility function is challenging, while the other terms are straightforward. Our approach is to represent $V_p(\mathbf{s})$ as a discrete binary image, the *visibility mask*, where blocking geometry is rendered using a spherical rasterizer. After the rasterization is complete, evaluation of $\mathbf{V}_p^*(\mathbf{v})$ is simple. We use 4th-order (25-term) spherical harmonics as the basis $\mathcal{B}$ for all results in this paper.

Our visibility rasterizer can also be used to integrate the lighting directly, instead of integrating against the SH basis functions. Direct integration requires a higher-resolution visibility mask to suppress aliasing artifacts, and our tests indicate that more computation is required to achieve comparable quality. Another argument in favor of using SH transfer coefficients is that if a model that has a diffuse BRDF is static in some frames, the coefficients from previous frames can be reused.

Rendering. For each vertex, we clear the visibility mask and rasterize all blocking triangles into it. Then the transfer coefficients for the vertex are computed according to Equation 3, with the visibility term $V_p(\mathbf{s})$ fetched from the visibility mask. After transfer coefficients have been evaluated for all vertices that belong to front-facing triangles, exitant radiances are determined according to Equation 2 and then passed on to the GPU for rendering the image. The following sections describe this process in detail.

3.1. Visibility Rasterization

Our visibility mask is a regular grid inside the unit disk. The unit hemisphere, the domain of the visibility function, is projected down onto the disk by dropping the z coordinate. We work in a local tangent frame of the vertex, so that the vertex normal coincides with the z axis. This parameterization amounts to a simple change of variables in the hemispherical integral in Equation 3, and it is easy to show that it perfectly importance samples the hemisphere according to the cosine distribution. This parameterization directly implements the well-known Nusselt analog.

Our method bears close resemblance to the hemicube algorithm [CG85] for computing form factors. In fact, the hemicube could be used for computation of the visibility masks. The downside is that all blocker geometry would need to be rendered five times (once for each face of the hemicube), whereas our method produces a perfectly importance-sampled result in a single pass. The single-plane method for form factor computation [SP89] also requires only one pass over the blocking geometry. However, uniform sampling of the projection plane would lead to poor sampling on the hemisphere, since much of the area projects to near the equator and occluders near the horizon will be missed. Sillion and Puech circumvent these problems by using a more complicated hidden-surface removal algorithm.

Since the visibility function is binary and we do not need knowledge of which surface is closest, a 1-bit frame buffer is sufficient, and no depth sorting is required. We may thus iterate through all blocker triangles in any order.

3.1.1. Rasterization of Blocker Triangles

When a blocker triangle has been determined to lie above the tangent plane, the image of its spherical projection is rasterized into the visibility mask as follows (see Figure 2 for illustration).

To determine which pixels belong to the image, we first note that each edge of the triangle, together with the origin, defines a plane. Now, a pixel of the visibility mask is inside the image of the triangle if the point on the hemisphere that corresponds to the pixel is above all three planes defined by the edges.

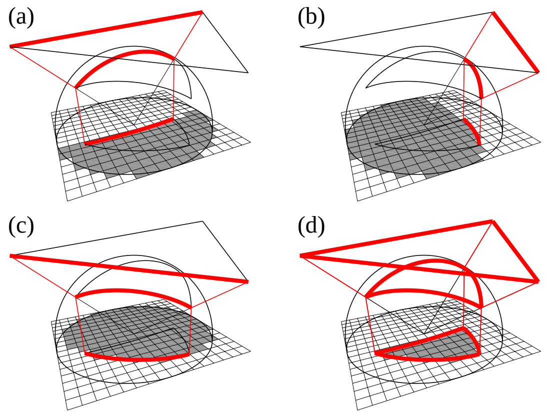

Figure 2: *Principle of spherical rasterization. (a)-(c): Each edge of the triangle defines a plane with the origin. For a discrete set of planes, the pixels in the visibility mask that are above the plane have been stored in a look-up table. (d): The final image of the spherically projected triangle is obtained from a bit-wise AND of the three masks from the look-up table that most closely match the planes defined by the edges.*

Because of the low resolution of the visibility mask (32×32), a lookup table of bitmasks can be precomputed for a discrete set of planes. Each set bit in these masks indicates that the corresponding point on the hemisphere is above the plane. To determine the rasterized image of the triangle, bitmasks that correspond to the three planes defined by the edges are fetched from the table, and a bit-wise AND is performed between them. The result is a coverage mask that has set bits for the pixels inside the triangle. See Figure 2(d) for an example. A similar algorithm using coverage masks for conventional rasterization with straight edges has been described earlier [FFR83]. Multiple occluder triangles are rasterized correctly by ORing the resulting masks of each triangle to the visibility mask. This results in proper occluder fusion.

For indexing the look-up table, we use a cube map parameterization of the normal vectors that uniquely define the planes. With a 32×32 bitmask and a $6 \times 128 \times 128$ cube map, the table consumes 12 MB of memory. Quantizing the planes naturally decreases the accuracy of the rasterization. We have found that these values give results that are visually correct when used for computing transfer coefficients with the 25-term SH basis.

3.2. Optimizations

This section describes a method for optimizing transfer coefficient evaluation and several techniques that we use for

© The Eurographics Association 2004.

reducing the amount of geometry that has to be processed by the rasterizer.

Downsampling the Visibility Mask. To reduce the number of samples when evaluating the transfer coefficients by Equation 3 we first downsample the visibility masks in 4×4 blocks, so that we obtain a grayscale occlusion term from the binary values.

Mesh Hierarchy. Naïve application of the visibility rasterizer results in complexity proportional to the number of vertices times the number of triangles, i.e. using all triangles of the full resolution mesh is expensive. Fortunately, this is not necessary since in smooth lighting environments blockers cast soft shadows unless they are close to the receiver. Hence, more distant blockers do not need to be represented accurately.

We use a simple two-level mesh hierarchy. In the vicinity of a vertex we use triangles from the high-resolution mesh, usually the one- or two-ring. For the remaining parts of the model we use a coarser mesh. See Figure 3 for an illustration. Triangles of the two meshes will often overlap, which poses no problem, as the rasterization fully supports occluder fusion. Mismatches between the hierarchies may produce shading artifacts in regions where small features undergo such deformations that the lower-resolution mesh cannot faithfully represent the surface. The use of more complex mesh hierarchies would help to avoid these rare cases.

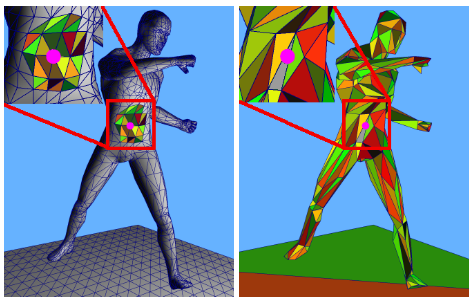
Figure 3: *A simple two-level mesh hierarchy. In the local neighborhood of a vertex we use the high-resolution mesh. More distant blockers are approximated by a low-resolution version.*

Hierarchical Culling. As an optional method for speeding up the culling of triangles that cannot contribute to the visibility masks, we cluster the geometry into a small number of groups that consist of static sets of triangles. The bounding boxes of the groups are updated as the animation progresses. When rendering the visibility masks, the boxes are tested against the horizon plane, and if a box is below the horizon, none of the triangles in the group need to be rasterized.

In typical applications, e.g., characters in computer games, the depth complexity of the animated models is so low that using more sophisticated visibility culling methods [COCSD03] is not expected to provide significant performance improvements.

PVS. If prior knowledge of the animation is available, it is possible to construct a potentially visible set (PVS) for each vertex. This is done by analyzing the geometry visible to a vertex in the animation frames in an offline preprocess. Such PVSs, which may be static or time-dependent, are easily incorporated into the visibility rasterizer. As an example, we have computed static PVSs for some test scenes. See Section 4 for a comparison of rendering performance with and without PVSs.

4. Results

First we present images of animated models rendered with our technique. All tests were run on a dual 3.06GHz Intel P4-Xeon. In all cases, changing the incident lighting does not incur performance loss. Table 1 summarizes performance figures and the triangle and vertex counts of the models. In the table, the average number of triangles denotes the number of triangles that have to be rasterized when computing visibility for a single vertex. This number is greater than the number of low-resolution triangles, since we also render a small neighborhood of the high-resolution mesh.

Model	Cloth	Fight	Chess	Hand
#Vertices	1378	2978	1897	8636
#Triangles	2560	4806	3704	15854
#Lowres-Triangles	160	444	382	378
Avg. #Tris	236.4	448.8	378.8	414.1
Avg. #Tris (with PVS)	–	99.5	73.9	49.7
FPS	15.5	4.9	7.8	2.6
FPS with PVS	–	9.5	15.1	6.1
Average L_2 error in $\mathbf{V}^*$	2.47%	7.93%	2.89%	3.82%

Table 1: *Comparison of all models, timings, and errors in the transfer coefficients.*

Figure 1 shows three frames of a realtime cloth simulation. Note how the wrinkles cause soft self-shadowing, and how the cloth casts a soft shadow on the floor. The figure also compares three renderings of a hand, the first without shadowing [RH01], the second with a purely diffuse BRDF and the third with a slightly glossy Phong BRDF. Performance loss due to the more complicated BRDF is negligible, since evaluation of the SH functions dominates BRDF evaluation in the integration.

A frame from the Fighting Man animation is shown in Figure 4(a)-(b). The image produced by our method is compared to a rendering where transfer coefficients are evaluated by Monte-Carlo integration. Visible differences between our

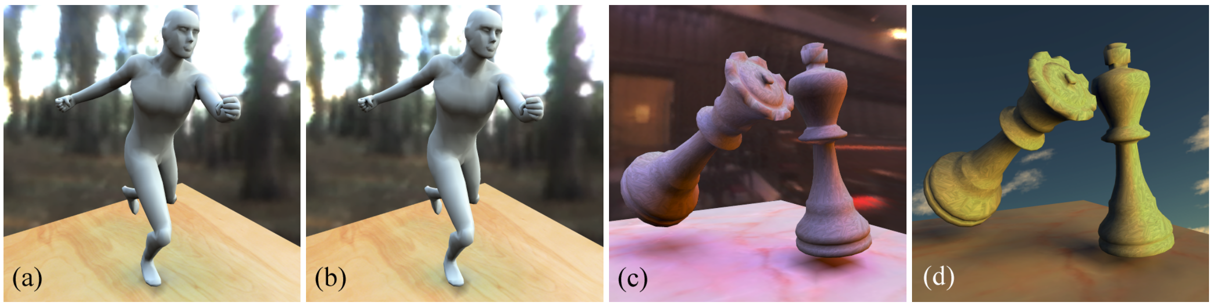

Figure 4: *Images rendered with our method. (a) A frame rendered using our method. (b) The same model rendered using ray-traced per-vertex visibility. (c)-(d) A chess scene in two different lighting environments.*

	Rasterization	Integration	Transform	Other
Cloth	48.0%	23.5%	26.5%	2.0%
Fight	54.4%	19.4%	24.4%	1.8%
Chess	52.0%	23.8%	21.9%	2.3%
Hand	42.3%	35.2%	19.4%	3.1%

Table 2: *Distribution of CPU time inside the shadow loop (transform includes backface culling and view transformation of blockers).*

technique and the accurate rendering are negligible. The error in the transfer coefficients is fairly small, as shown in Table 1. The error is measured by average relative L_2 error over all vertices and many animation frames. This example also shows that utilizing prior information of the animation yields better framerates. Here, the number of blocker triangles processed by the visibility rasterizer was reduced by roughly 75% by using a PVS computed from the animation.

Finally, in Figure 4(c)-(d), a simple scene with moving chess pieces is illuminated with different environment maps. In this scene the error in the coefficients is low and there is no noticeable visual difference to the accurate rendering.

4.1. Discussion

In the following, we will discuss some properties and issues of our method.

Scalability. The complexity of our algorithm is always proportional to the average number of blocker triangles $\times$ number of vertices (see Table 1), i.e., assuming our low-resolution mesh stays fixed, we can increase the resolution of the high-resolution mesh at linear cost.

Our algorithm is mainly targeted for self-shadowing. If multiple objects are required to cast shadows onto each other, the complexity is increased. This quadratic increase of computation can be avoided by processing one object at a time, and by casting approximate inter-object shadows by projecting coarse approximations of the other shadow-casting objects onto the lighting environment used by the object receiving the shadow. This leaves per-object computational complexity unchanged.

Per-Vertex Sampling. As we sample visibility on a per-vertex basis, artifacts may occur as a result of undersampling, e.g., contact shadows will “creep” from underneath the areas in contact (see Figure 5). We point out that this is an inherent limitation associated with sampling visibility only at the vertices; the method of Sloan et al. [SKS02] and per-vertex Monte-Carlo integration suffer from exactly the same problem. By only computing lighting at the vertices, we have deliberately chosen to sacrifice quality for speed.

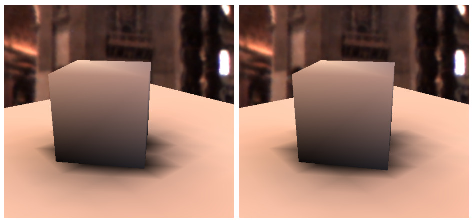

Figure 5: *Two frames of an animation showing undersampling artifacts in a coarse mesh.*

Distribution of CPU Time. As can be seen in Table 2, roughly half of the CPU time is spent on rasterization. Integration against the basis functions takes up about 30% of the time. In all cases, the shadow rendering loop consumes approximately 95% of total frame time. The rest of the time is spent on dotting the transfer vectors against the lighting, updating the vertex arrays and on GPU communication.

Granularity of Coverage Mask Table. The discretization granularity of the plane equations depends on the resolution of the visibility mask – the better the resolution, the more densely we have to sample the plane space. Given that we use a 32×32 mask, our $128 \times 128 \times 6$ parameterization has been chosen specifically as one that still gives results that are still visually indistinguishable from a finer granularity. Too low a granularity results in noticeable artifacts. The granularity has to be kept as low as possible, since increasing the size of the plane LUT will slow execution down because of more cache trashing.

Accuracy of Local Geometry. Theoretically, shadow fidelity may be compromised in configurations where the 3D distance between triangles is small, but the corresponding distance along the surface of the mesh is larger than the one- or two-ring used for the vicinity of a vertex. However, the choice of one- or two-ring seems to be a good and fast heuristic, at least for the types of models and animations in our tests. For more complex models, it should be possible to avoid this problem altogether by using more complicated spatial data structures. This is an interesting line of future work.

Comparison to Hardware Rendering. To compare the speed of our method against a possible GPU implementation of visibility rasterization, we rendered 32×32 visibility masks as seen from each of the 2978 vertices of the Fighting Man scene on an ATI Radeon 9800XT with depth buffering disabled. Using a vertex shader, the vertices were projected onto the unit disk in a fashion similar to our rasterizer. Since GPUs do not support hemispherical rasterization, we approximate the curved edges by subdividing the geometry to contain 16 times as many triangles as in the original low-resolution geometry. Performance with rasterization only (no integration) was just 0.76 FPS. Even though the result of this performance test does not seem to support attempting a GPU implementation of our method, integrating the transfer coefficients on the GPU would certainly be possible, if visibility information was readily available.

5. Conclusions and Future Work

We have presented a method for rendering dynamic objects with self-shadows in time-varying lighting environments. The method has interactive or even real-time performance on models with some thousands of polygons, and supports both diffuse and glossy BRDFs. No prior information of the animation of the model is required, although such information, if available, can be used for optimization. No previous method has demonstrated such performance without prior information on the animation. A new visibility rasterizer is the key to interactive framerates.

Future work includes looking into ways of distributing the computation between the CPU and the GPU, so that the GPU could be utilized more efficiently. Of course, in scenarios such as computer games the GPU would be used for rendering background geometry while the CPU computes self-shadowing using our method.

6. Acknowledgments

We would like to thank Frédo Durand, Petri Häkkinen, Janne Kontkanen, Jussi Räsänen and Lauri Savioja for their valuable comments and Paul Debevec for the light probe images. This work has been sponsored in part by the German Academic Exchange Service (DAAD), the National Technology Agency of Finland, Bitboys, Hybrid Graphics, Nokia and Remedy Entertainment.

References

[ARBJ03] Agarwal S., Ramamoorthi R., Belongie S., Jensen H. W.: Structured Importance Sampling of Environment Maps. *ACM Transactions on Graphics 22*, 3 (July 2003), 605–612.

[BN76] Blinn J., Newell M.: Texture and Reflection in Computer Generated Images. *Communications of the ACM 19* (1976), 542–546.

[CG85] Cohen M. F., Greenberg D. P.: The hemi-cube: a radiosity solution for complex environments. In *Computer Graphics (Proceedings of ACM SIGGRAPH 85)* (1985), ACM, pp. 31–40.

[COCSD03] Cohen-Or D., Chrysanthou Y., Silva C., Durand F.: A Survey of Visibility for Walkthrough Applications. *IEEE Transactions on Visualization and Computer Graphics 9*, 3 (2003), 412–431.

[FFR83] Fiume E., Fournier A., Rudolph L.: A parallel scan conversion algorithm with anti-aliasing for a general-purpose ultracomputer. In *Computer Graphics (Proceedings of ACM SIGGRAPH 83)* (1983), ACM, pp. 141–150.

[GCHH03] Gibson S., Cook J., Howard T., Hubbold R.: Rapid Shadow Generation in Real-World Lighting Environments. In *Eurographics Symposium on Rendering* (June 2003), pp. 219–229.

[HDG99] Hart D., Dutré P., Greenberg D.: Direct Illumination With Lazy Visibility Evaluation. In *Proceedings of ACM SIGGRAPH 99* (Aug. 1999), Computer Graphics Proceedings, Annual Conference Series, pp. 147–154.

[HLHS03] Hasenfratz J.-M., Lapierre M., Holzschuch N., Sillion F.: A Survey of Real-Time Soft Shadows Algorithms. *Computer Graphics Forum, 22*, 4 (2003). State-of-the-Art Report.

[JF03] James D., Fatahalian K.: Precomputing Interactive Dynamic Deformable Scenes. *ACM Transactions on Graphics 22*, 3 (July 2003), 879–887.

[NRH03] Ng R., Ramamoorthi R., Hanrahan P.: All-Frequency Shadows Using Non-linear Wavelet Lighting Approximation. *ACM Transactions on Graphics 22*, 3 (2003), 376–381.

[RH01] Ramamoorthi R., Hanrahan P.: An Efficient Representation for Irradiance Environment Maps. In *Proceedings of ACM SIGGRAPH 2001* (August 2001), ACM Press, pp. 497–500.

[SHHS03] Sloan P.-P., Hall J., Hart J., Snyder J.: Clustered Principal Components for Precomputed Radiance Transfer. *ACM Transactions on Graphics 22*, 3 (2003), 382–391.

[SKS02] Sloan P.-P., Kautz J., Snyder J.: Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *ACM Transactions on Graphics 21*, 3 (2002), 527–536.

[SP89] Sillion F., Puech C.: A General Two-Pass Method Integrating Specular and Diffuse Reflection. In *Computer Graphics (Proceedings of ACM SIGGRAPH 89)* (July 1989), ACM, pp. 335–344.

[WBS03] Wald I., Benthin C., Slusallek P.: Interactive Global Illumination in Complex and Highly Occluded Environments. In *Eurographics Symposium on Rendering* (June 2003).