Hardware-accelerated ambient occlusion computation

Mirko Sattler, Ralf Sarlette, Gabriel Zachmann, Reinhard Klein

University of Bonn
Institute for Computer Science II, Bonn, Germany
Email: {sattler,sarlette,zach,rk}@cs.uni-bonn.de

Abstract

In this paper, we present a novel, hardware-accelerated approach to compute the visibility between surface points and directional light sources. Thus, our method provides a first-order approximation of the rendering equation in graphics hardware. This is done by accumulating depth tests of vertex fragments as seen from a number of light directions. Our method does not need any preprocessing of the scene elements and introduces no memory overhead. Besides of the handling of large polygonal models, it is suitable for deformable or animated objects under time-varying high-dynamic range illumination at interactive frame rates.

Figure 1: Stanford dragon under high-dynamic range illumination including ambient occlusion, which has been computed on graphics hardware.

1 Introduction

Shadows are one of the most important visual clues about the spatial structure of an object. E.g. for virtual reality applications, such as life-sized cloth visualization, or medical surgery planning, they are important because they increase the presence of the virtual objects and the overall realism. However, the shadows need to be computed at interactive frame rates, otherwise usability and presence will break down. While local illumination models are the strength of modern graphics hardware, more advanced techniques, which include e.g. soft shadows and ambient occlusion, are notoriously hard to perform efficiently. On the other hand, recent methods like pre-computed radiance transfer [1] enable complex illumination effects at fast frame-rates, but are limited to static objects or pre-defined animations due to their long precomputation times.

In this paper, we present a method to efficiently compute the visibility for many light directions for each vertex on the GPU at interactive frame rates, even for large models. This includes self-occlusion as well as occlusion caused by other objects. This is done by using hardware occlusion query results from vertex fragments as seen from a number of light directions. All geometry in the scene can deform and move, and the illumination can change at no extra cost. Thus, we can compute and render a first-order approximation of the rendering equation [2] for opaque, polygonal objects on the graphics hardware.

Our algorithm features the following advantages:

- no precomputation, no complex data structures or special preprocessing is needed
- handles arbitrary deforming geometry
- fast hardware-accelerated occlusion calculation.
- high-dynamic range image based illumination
- easy to implement.

2 Previous work

A vast amount of work has been done on shadow algorithms. Shadow maps [3] or shadow volumes [4] and all derivates are the classical approaches for point-like light sources. A good overview of algorithms which produce soft shadows can be found in Hasenfratz et al. [5]. All these algorithms cannot efficiently be used for arbitrary lighting environments. On the other hand, environment mapping as introduced by Blinn et al. [6] is able to render reflections of incident lighting, but without shadows. Ray tracing is capable of handling globally illuminated scenes, but is naturally limited to the current camera position, Interactive rates are only achieved in a massive parallel environment with optimal acceleration structures [7, 8], which take several seconds to build. Other theoretical work [9] on GPU-based raytracing is not yet available in hardware. Very recently there have also been approaches to solve radiosity on graphics hardware [10], with interactive rates for small scenes.

To evaluate the illumination received by a point on the surface, there exist mainly two approaches. The first category gathers the incoming radiance at each surface point or vertex and scales with that count. The hemisphere defined by the vertex normal is sampled in different ways. Either rays are shot into predefined directions using standard ray tracing methods, or hemicube [11] sides are rendered using the standard pipeline [12]. With ray casting, intensive intersection tests have to be calculated and acceleration structures have to be maintained (e.g. space partitioning), whereas hemicubes cannot yet be evaluated efficiently on the graphics hardware. Other methods compute visibility cones [13], blocker maps [14], obscurance maps [15, 16], visibility maps [17] or radiance transfer [1] which also includes inter-reflections. All these methods require certain amounts of pre-computation time, and are therefore not suitable for dynamic objects. Very recent work on the topic discussed in this paper was presented by Kautz et. al [18]. It is mainly based on fast hemicube rasterization in order to detect blocker triangles. Downsampling of the visibility mask and a coarser blocker mesh are used for speed up. Interactive frame rates are achieved for small animated models. In contrast to their work, our method does not need a mesh hierarchy nor any additional graphics memory during run-time.

The second category is based on the approximation of the ambient environment by point or directional light sources, which amounts to reversing the first approach from "inside-out" to "outside-in". That is, the visibility computation is originated at the light sources. Lately, NVIDIA proposed a hardware-accelerated 2-pass method, using accumulated shadow maps [19, 20], which is also used in many shaders in commercial rendering software packages [21]. To minimize sampling artifacts, jittering of the depth maps was introduced. This approach involves common shadow mapping projection problems [22]. To achieve usable results, several seconds per frame are needed. Changing to a new viewpoint requires new shadow mapping passes or an unwrapping process to obtain an occludence texture. Superimposing images was also done by Keller et al. [23] to compute instant radiosity. This also requires space-partitioning structures.

3 Ambient occlusion calculation

In this section, we will present the core of our new method. In the following, a triangle mesh with vertex normals is given. To compute the outgoing radiance L_r at a surface point $\mathbf{x}$ into direction $\mathbf{v}$ we have to compute:

$$L_r(\mathbf{x}, \mathbf{v}) = \int_\Omega f_r(\mathbf{x}, \mathbf{v}, \mathbf{l}) L_i(\mathbf{x}, \mathbf{l}) V(\mathbf{x}, \mathbf{l}) (\mathbf{n}_x \cdot \mathbf{l}) dl$$

where Ω is the hemisphere domain over $\mathbf{x}$, f_r the BRDF, L_i the incident radiance from direction $\mathbf{l}$, $V(\mathbf{x}, \mathbf{l})$ the visibility from $\mathbf{x}$ to direction $\mathbf{l}$ and $\mathbf{n}_x$ the vertex normal at $\mathbf{x}$. We discretize the integral by k light directions $\mathbf{l}_j$, $j \in [1, \ldots, k]$, which leads to

$$L_r(\mathbf{x}, \mathbf{v}) \approx \sum_{j=1}^{k} f_r(\mathbf{x}, \mathbf{v}, \mathbf{l}_j) L_i(\mathbf{x}, \mathbf{l}_j) V(\mathbf{x}, \mathbf{l}_j) (\mathbf{n}_x \cdot \mathbf{l}_j)$$

We now concentrate on the efficient computation of the term $L_i(\mathbf{x}, \mathbf{l}_j) V(\mathbf{x}, \mathbf{l}_j) (\mathbf{n}_x \cdot \mathbf{l}_j)$.

3.1 Overview

Our method relies on depth test results from renderings of the desired object. We approach the problem of light direction visibility for each vertex by considering a set of k directional light sources $\mathbf{l}_j$ and determine the visibility of all N vertices at once as seen from each of the light source directions. Because the number of light source directions k is much smaller than the number of vertices N

for large models, the "outside-in" approach is much more efficient, than the "inside-out" approach, that is, a much smaller number of render passes, k instead of N, of the object into the depth buffer is needed.

More precisely, we render all geometry as seen from a light source direction into the depth buffer. Then, all vertices are rendered again as a point set. An individual occlusion query per vertex allows us to retrieve those vertices that passed the depth test. For these, the currently considered light source direction is marked as visible and stored in a matrix M, which we call *visibility matrix*. We repeat this process for each light source, updating the appropriate entries in M.

3.2 Visibility matrix computation

Our method makes heavy use of the OpenGL extension *ARB_OCCLUSION_QUERY* [24] . The purpose of this extension is to deliver the number of fragments that passed both depth and stencil test. In contrast to its predecessor *HP_OCCLUSION_QUERY* it is asynchronous, i.e. it does not use a "stop-and-wait" execution model for using multiple queries. This allows applications to issue many occlusion queries before asking for the result of any one. As mentioned above, in the first pass, the unlit scene is rendered into the depth buffer from one of the light source directions. We do this in orthographic projection mode. In the second pass, all vertices are rendered as *glPoints* with size 1, without updating the depth buffer. An offset (*glPolygonOffset*) with default values $(1.0, 1.0)$ is used to avoid rounding issues. Each single vertex $i \in N$ is handled by an individual occlusion query and the visibility matrix M, which stores visibility information for each vertex i to the light direction $\mathbf{l}_j$, is updated for all j. For later performance, we do not store a boolean visibility bit, but instead the dot product computed from the vertex normal $\mathbf{n}_i$ and the vector defined by the light source direction $\mathbf{l}_j$ if the vertex is visible from that direction. $\mathbf{l}_j$ is computed once for each virtual light source direction. Therefore we obtain:

$$M_{ij} = \begin{cases} \mathbf{n}_i \cdot \mathbf{l}_j & : \text{ vertex visible} \\ 0 & : \text{ vertex invisible} \end{cases}$$

Figure 2 gives an overview of the core algorithm as pseudo-code.

```
enable orthographic projection
disable framebuffer
for all light directions j do
    set camera at light direction l_j
    render object into depth buffer with polygon offset
    for all vertices i do
        begin query i
        render vertex i
        end query i
    end for
    for all vertices i do
        retrieve result from query i
        if result is "visible" then
            M_ij = n_i · l_j
        end if
    end for
end for
```

Figure 2: Outline of the core algorithm for visibility matrix calculation.

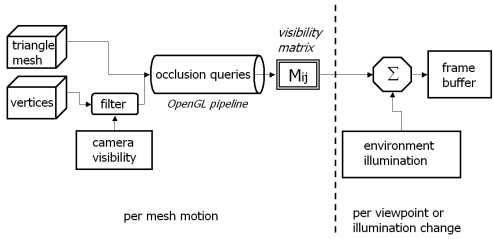

Figure 3: Simplified data flow of our approach.

3.3 Rendering

After M_{ij} has been computed, the final color c_i for each vertex i can be computed as:

$$c_i = \sum_{j=1}^{k} M_{ij} I_j$$

where I_j is the 3-component (RGB) color of the light coming from direction $\mathbf{l}_j$. Figure 3 shows a simplified data flow of our approach. Note, that neither M_{ij} nor I_j change, if the viewpoint is changed.

3.4 Creating the lightsphere

To approximate ambient occlusion with single directional light sources, we have to distribute k light

directions. Each light direction is represented by a point on the unit sphere. We seek a distribution of points on the sphere satisfying the following conditions. The visibility computation should be done only once and the illumination environment should be changeable without doing new queries (see Section 5). Furthermore, to allow an easy increase of the number of light directions, already computed parts of M should be re-usable. Therefore, the distribution should equally sample the the environment in all directions. Recent approaches like [25, 26] which do an efficient sampling of the environment map are not easily adaptable for our method. They need seconds to minutes of preprocessing time to reduce the number of light directions, therefore would not allow interactive change of the lighting environment. For equal distribution of points on a sphere, several methods exist [27–29]. We propose the following preprocessing procedure, based on subdivision of a regular solid. We start with the vertices of a unit octahedron, i.e. $k = 6$ light directions at subdivision level $s = 0$. To generate level $s + 1$, we do midpoint subdivision of the edges on level s and project the new vertices on the unit sphere, thus creating a polyhedron with $2 \cdot 4^{s+1}$ faces and $2+4^{s+1}$ vertices. This structure allows us to add new sets of well distributed points, while using the occlusion queries of all coarser subdivision levels. Figure 5 shows several increasing configurations.

4 Optimizations

Whenever the object is moved or deformed, that is when vertex positions change, we need a complete re-computation of M. In the following, we will present several methods that significantly reduce that effort, so that we can maintain interactive frame rates even under these circumstances. Even for static geometry, depending on the viewpoint, not all vertices must be evaluated.

4.1 Vertex filtering using temporal coherence

The first optimization exploits temporal coherence by observing that during a viewpoint change only a small number of polygons become visible for the first time (at most those that cross the object silhouette). Consequently, we can compute the visibility

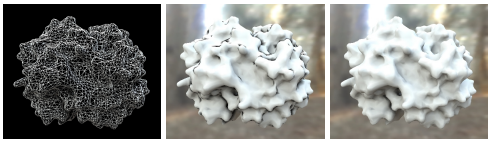

Figure 4: Vertex filtering optimization for the computation of M has to be done carefully, otherwise artifacts will occur (middle). Left: wireframe; right: correct rendering.

matrix M lazily, thereby distributing the computation effort over several frames. More precisely, we maintain two lists of triangles: a list of "unseen" triangles, $\mathcal{U}$, containing all triangles that have not yet been visible and a list $\mathcal{T}$ ("todo"), containing triangles that will be seen in the next frame for the first time. $\mathcal{U}$ is initialized with all vertices once. Vertices which belong to triangles stored in $\mathcal{U}$ have not yet been visible from the camera so far. Thus, we do not need their occlusion with respect to the light directions. Vertices which belong to triangles which are stored in $\mathcal{T}$ have to be processed for the next frame, because these triangles will then be visible. Using the two lists, in each frame we need to determine the occlusion only for a small fraction of vertices (see also Figure 3). In order to compute list $\mathcal{T}$, we perform an algorithm similar to the one in Section 3, except that here we render triangles in the second pass and use perspective projection. More precisely, in the first pass, we render the unlit mesh into the depth buffer as seen from the new camera position. In the second pass, we render only the triangles still in list $\mathcal{U}$, each with its own occlusion query (again with offset and without depth buffer update). Obviously, when $\mathcal{U}$ contains less triangles than a certain threshold, we do not gain any performance any more. In that case, we just add all triangles from $\mathcal{U}$ to $\mathcal{T}$ and skip this optimization in the remaining frames. Note that for this optimization we need to consider triangles, not vertices. Otherwise, artifacts could occur, because a triangle might be visible although one of its vertices is not, and thus its corresponding value in M has not yet been computed. This is illustrated in Figure 4. In Section 7, we discuss the performance gains introduced by this optimization.

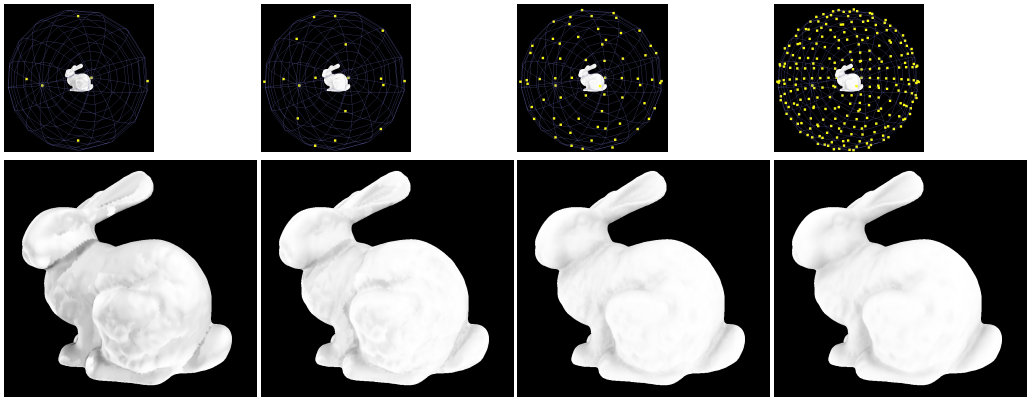

Figure 5: Object rendered with different number of light directions (k=6, 18, 66, 258). The upper row shows the lightsphere configurations, where the yellow dots represent the light directions.

4.2 Changing the Lightsphere configuration

A further optimization is to dynamically change the lightsphere configuration. While the scene is animated or objects are deformed, we can decrease the number of light directions. In idle time, in order to converge to the exact solution, we increase that count as shown in the upper row in Figure 5. Note, that if the number k is too small, under-sampling artifacts (left bunny) can occur depending on the used environment. In our experience a level of $s = 3$ and often even level $s = 2$ produces reasonable results. The algorithm allows user control through the choice of a desired frame rate or a certain configuration.

5 Image Based Illumination

In this section, the core algorithm is extended to incorporate image-based illumination [30–32], using high- dynamic range environment maps [33,34]. Similar to other approaches [35–37] we pre-filter the environment. Therefore, we use HDRshop [38] to generate a *latitude-longitude* lookup map out of a cube map and apply gaussian blur as a filter as shown in Figure 6. The black dots on the right image are the projected light directions. We now have the environment parameterized with (θ, ϕ), with the angle $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$, which is also known as Mercator projection. Thus, we can easily look-up the intensity from the light coming out of

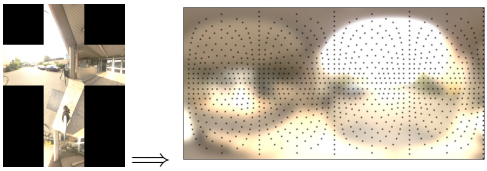

Figure 6: Illumination information stored in a high-dynamic range environment cube-map (left) and filtered version in latitude-longitude representation (right) with projected light source positions (black dots).

the light direction $\mathbf{l}_j$. When the environment map is rotated relative to the object, the light directions are multiplied by the same rotation matrix to obtain the new look-up positions. We apply standard gamma correction to all vertex color values.

6 Dynamic Geometry and Animations

It should be obvious by now, that in an environment with a single object, a rigid transformation can be handled quite efficiently: a translation can be ignored, while a rotation just amounts to an additional matrix multiplication before the look-up into the visibility matrix. If there are several objects changing positions relative to each other or an object changes shape, than the complete visibility matrix M needs to be re-computed. However, this can still be done at interactive frame rates, because our algorithm does not need any spatial accelera-

tion structures. We only need to the re-initialize list $\mathcal{U}$ (Section 4) and update the vertex arrays for the objects. Figure 7 shows sample key frames of an animation of a skeleton running at interactive frame rates, while changing the viewpoint and the lighting environment. In each frame, M is re-computed according to Figure 3.

7 Results and discussion

This section presents performance measurements of our new algorithm, which was run on an AMD Athlon64 3200+ (2.0 GHz) under Windows 2000 and an ATI Radeon 9800 XT using OpenGL 1.5. All images and videos were rendered at a resolution of 1280×960 pixels. This paper is also accompanied by a video. As for the rendering results, we have chosen to render all our objects without textures to make the visual effects of our algorithm more evident. As a matter of course, our method can be combined very easily with texturing or standard shadow mapping techniques [3, 22], in order to handle point light sources as well. The left image on Figure 8 shows, that our method is also feasible to render large objects for medical visualization. The *teeth* object consists of 116k vertices. Detailed surface structures are visible. The middle and right image in Figure 8 show several objects that occlude each other from light directions, which is handled correctly by our method. The bunny between the dragons is much darker, due to the occlusion. Figure 9 shows a comparison of our method with OpenGL Phong lighting and a ray-traced image, in a high-dynamic range illumination environment. To achieve the same visual quality 500 samples/ray were needed and it took over half an hour compared to under 1 minute using the new method. The *venus* object consists out of 134k vertices. The slight color shift is due to the slightly different exposures and environment orientations.

Table 1 gives an overview over the computation time of the visibility matrix (time_1) in milliseconds for several different objects and light directions k. If the vertex filtering (time_2) is enabled, the calculation performance is drastically increased. FPS_1 and FPS_2 show the frame rates in Hertz with and without the vertex filtering optimization, when the objects are rendered. We achieve real-time frame rates (> 30Hz) for *static* objects and interactive rates for the skeleton and pair of trousers *animation*.

object	vertices	k	time_1 (msec)	time_2 (msec)	FPS_1 (Hz)	FPS_2 (Hz)
bunny *static*	35k	6	445	339		
		18	1082	797		
		66	3606	2666		
		258	13715	10373		
		1026	60304	39765		
teeth *static*	116k	6	1495	1257		
		18	3651	3041		
		66	12243	10290		
		258	46482	38976		
skeleton *animation*	8325	6	133	113	8.08	8.85
		18	337	245	3.66	4.10
		66	937	740	1.14	1.35
		258	3301	2472	0.30	0.40
trousers *animation*	3219	6	49	34	18.85	22.75
		18	103	66	9.23	12.80
		66	330	222	3.03	4.63
		258	1217	811	0.82	1.35

Table 1: Computation times (*time*) for the visibility matrix for static objects and animations with different light-sphere configurations (k) and frame rates (*FPS*) when rendered. Times with $_1$ are without and $_2$ with vertex filtering enabled. See text for details.

8 Conclusions

We have presented a new method to calculate vertex-light direction visibility, thus providing a first-order approximation of the rendering equation. This is done by sampling the environment by several directional light sources and efficiently computing vertex visibility from these light directions using hardware-accelerated occlusion queries. In conjunction with vertex filtering optimization, we are able to handle deformable objects and animations at interactive frame rates. The proposed method also allows the usage of image based illumination stored in filtered high-dynamic environment maps. The algorithm drastically reduces visibility calculation times and might be incorporated in other visibility determination problems. Because the method is vertex based, artifacts may occur due to undersampling, which is, of course, true for all vertex-based approaches.

8.1 Future Work

As a drawback, the proposed method has to store the visibility list M on the CPU, because the query result is always sent back from the graphics card. It would be a great performance increase, if this result

would be available directly on the GPU on future hardware. Additional speed-up could be achieved by allowing parallel query updates through segmented result buffers. To achieve real-time frame rates for an animation or as a general speed-up, we want to implement our algorithm in a parallel environment.

8.2 Acknowledgements

Some of the used high-dynamic range images were made by Paul Debevec [32, 34]. The Stanford Bunny and some other models are from the Stanford Computer Graphics Laboratory and Georgia Tech [39–41]. The teeth and venus sample were provided by Cyberware [42]. The textiles shown in this work were generated within the scope of the BMBF project "Virtual Try-On" with the cloth simulation engine TüTex [43–45]. The skeleton animation was made using POSER by Curious Labs. The authors wish to thank Jutta Adelsberger and Thomas Mücken for fruitful discussions.

References

[1] P.P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 527–536. ACM Press, 2002.

[2] J. T. Kajiya. The rendering equation. In *Computer Graphics (SIGGRAPH '86 Proceedings)*, 1986.

[3] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 270–274. ACM Press, 1978.

[4] F.C. Crow. Shadow algorithms for computer graphics. *j-COMPGRAPHICS*, 11(2):242–248, July 1977.

[5] J.M. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion. A survey of real-time soft shadows algorithms. In *Eurographics*. Eurographics, Eurographics, 2003. State-of-the-Art Report.

[6] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19, 1976.

[7] Ingo Wald, Timothy J. Purcell, Joerg Schmittler, Carsten Benthin, and Philipp Slusallek. Realtime Ray Tracing and its use for Interactive Global Illumination. In *Eurographics State of the Art Reports*, 2003.

[8] I. Wald, C. Benthin, and P. Slusallek. Interactive global illumination in complex and highly occluded environments. *Proceedings Eurographics Symposium on Rendering 2003*, 2003.

[9] T.J. Purcell, I. Buck, W. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21:703–712, 2002.

[10] Greg Coombe, Mark J. Harris, and Anselmo Lastra. Radiosity on graphics hardware. In *Graphics Interface*, 2004.

[11] M. F. Cohen and D. P. Greenberg. The hemicube: A radiosity solution for complex environments. *Symposium on Computational Geometry*, 19(3):31–40, 22–26 July 1985.

[12] M. Sattler, R. Sarlette, and R. Klein. Efficient and realistic visualization of cloth. In *Eurographics Symposium on Rendering 2003*, June 2003.

[13] J. Stewart. Computing Visibility from Folded Surfaces. *Elsevier Preprint*, 1999.

[14] D. Hart, P. Dutre, and D.P. Greenberg. Direct illumination with lazy visibility evaluation. *SIGGRAPH 99 Conference Proceedings*, 1999.

[15] G. Kronin S. Zhukov, A. Iones. Ambient light illumination model. In *Proc. Eurographics Rendering Workshop '98*, 1998.

[16] A. Iones, A. Krupkin, M. Sbert, and S. Zhukov. Fast realistic lighting for video games. *IEEE Computer Graphics & Applications*, 2003.

[17] I. Neulander. Image-based diffuse lighting using visibility maps. *Siggraph Sketch 2003*, 2003.

[18] J. Kautz, J. Lehtinen, and T. Aila. Hemispherical rasterization for self-shadowing of dynamic objects. *Proceedings Eurographics Symposium on Rendering 2004*, 2004.

[19] M. Pharr. Ambient occlusion. *GDC 2004*, 2004.

[20] F. (Ed.) Randima. *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*. Addison-Wesley Professional, 2004.

[21] H. Landis. Production-ready global illumination. *'RenderMan in Production' SIGGRAPH Course Notes 2002*, 2002.

[22] Mark J. Kilgard. Shadow mapping with todays opengl hardware. *SIGGRAPH 2002 Course*, 2002.

[23] A. Keller. Instant radiosity. *Computer Graphics Proceedings SIGGRAPH 97*, 1997.

[24] http://oss.sgi.com/projects/ogl sample/registry/.

[25] S. Agarwal, R. Ramamoorthi, S. Belongie, and H.W. Jensen. Structured importance sampling of environment maps. *Transactions on Graphics*, 22:605–612, 2003.

[26] T. Kollig and A. Keller. Efficient illumination by high dynamic range images. *Proceedings Eurographics Symposium on Rendering 2003*, 2003.

[27] G. Fejes Toth. *Lagerungen in der Ebene, auf der Kugel und in Raum, 2nd ed.* Springer-Verlag, 1972.

[28] L. L. Whyte. Unique arrangement of points on a sphere. *Amer. Math. Monthly 59*, 1952.

[29] Kuijlaars A.B.J. Saff, E.B. Distributing many points on a sphere. *Mathematical Intelligencer*, 19:5–11, 1997.

[30] Gene S. Miller and C. Robert Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environments. In *SIGGRAPH '84 Advanced Computer Graphics Animation seminar notes*. July 1984.

[31] N. Greene. Environment mapping and other applications of world projection. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.

[32] P. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH 98 Conference Proceedings*, pages 189–198. ACM SIGGRAPH, 1998.

[33] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 369–378. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[34] http://www.debevec.org/Probes/.

[35] Wolfgang Heidrich and Hans-Peter Seidel. Realistic, hardware-accelerated shading and lighting. In Alyn Rockwood, editor, *Siggraph 1999, Annual Conference Proceedings*, Annual Conference Series, pages 171–178, Los Angeles, 1999. Addison Wesley Longman.

[36] J. Kautz., P.-P. Vázquez, W. Heidrich, and H.-P. Seidel. A unified approach to prefiltered environment maps. *Proc. 11th Eurographics Workshop on Rendering*, 2000.

[37] D. McAllister, A. Lastra, and W. Heidrich. Efficient rendering of spatial bi-directional reflectance distribution functions. *Graphics Hardware 2002, Eurographics / SIGGRAPH Workshop Proceedings*, 2002.

[38] http://www.debevec.org/HDRShop/.

[39] G. Turk and M. Levoy. Zippered polygon meshes from range images. *Proc. SIGGRAPH '94*, pages 311–318, 1994.

[40] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *Computer Graphics*, 30(Annual Conference Series):303–312, 1996.

[41] http://www.cc.gatech.edu/projects/large_models/.

[42] http://www.cyberware.com/.

[43] Etzmuss. O., M. Keckeisen, and W. Straßer. A Fast Finite Element Solution for Cloth Modelling. *Proceedings of Pacific Graphics*, 2003.

[44] S. Kimmerle, M. Keckeisen, J. Mezger, and M. Wacker. TüTex: A Cloth Modelling System for Virtual Humans. In *Proceedings 3D Modelling*, 2003.

[45] J. Mezger, S. Kimmerle, and O. Etzmuß. Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG*, 11(2):322–329, 2003.

Figure 7: Sample key frames of the skeleton animation running at interactive frame rates. A lot of self-shadowing occurs among the bones.

Figure 8: **Left** image: Large object *teeth* (116k vertices) rendered using our new algorithm under homogeneous white illumination. **Middle and right** images: Scene with several objects, rendered under high-dynamic range or homogenous white illumination. Note the darker bunny between the dragons.

Figure 9: Comparison between different rendering methods of the venus object (134k vertices). From left to right: wireframe, OpenGL Lighting, under HDR illumination with our new algorithm and raytraced.