Eurographics Symposium on Rendering 2012
Fredo Durand and Diego Gutierrez
(Guest Editors)

Volume 31 (2012), Number 4

Tessellation-Independent Smooth Shadow Boundaries

Oliver Mattausch[1] Daniel Scherzer[2] Michael Wimmer[3] Takeo Igarashi[4]

[1] University of Zurich [2] MPI [3] Vienna University of Technology [4]University of Tokyo

Abstract

We propose an efficient and light-weight solution for rendering smooth shadow boundaries that do not reveal the tessellation of the shadow-casting geometry. Our algorithm reconstructs the smooth contours of the underlying mesh and then extrudes shadow volumes from the smooth silhouettes to render the shadows. For this purpose we propose an improved silhouette reconstruction using the vertex normals of the underlying smooth mesh. Then our method subdivides the silhouette loops until the contours are sufficiently smooth and project to smooth shadow boundaries. This approach decouples the shadow smoothness from the tessellation of the geometry and can be used to maintain equally high shadow quality for multiple LOD levels. It causes only a minimal change to the fill rate, which is the well-known bottleneck of shadow volumes, and hence has only small overhead.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—

Keywords: real-time shadows, silhouettes, shadow volumes

1. Introduction

Shading discontinuities due to a coarse mesh, which can be successfully avoided using vertex normals and Phong shading, are still revealed in the silhouettes of the mesh and the projected shadow boundaries. Even worse, for shadows the size of the shadow projection can vary almost arbitrarily in screen-space, and results in distracting shadow discontinuities during animation. Imagine for example a sunset scenario or a case where the light source is close to the caster (as can be seen in Figure 1). The benefits of a finely tuned distance-based geometric level-of-detail (LOD) system can be destroyed by an edgy shadow, which will likely pop during LOD transitions.

While solutions have been proposed to improve the smoothness of object silhouettes in screen space [BA08], to our knowledge no solution has been proposed to address the tessellation uncovering problem for shadow boundaries (see Figure 1). The naive solution of finely subdividing the whole mesh for shadow casting would cause an unnecessary overhead and defeat the purpose of using the coarse mesh in the first place. Another idea, adaptively subdividing the mesh only at the silhouettes as seen from a light position (using some subdivision scheme like [CC78, BA08, WTW*08]) would cause visible artifacts in dynamic scenes, because the finely tessellated area on the mesh caused by the shadow silhouette would wander around if the light or the object moves. Additionally, a practical solution should guarantee C_1 smoothness in the limit and should not depend on user parameters for convincing results.

Figure 1: *Discrete shadows reveal the coarse tessellation of the underlying mesh due to discontinuities in the shadow boundaries* (left). *This can be avoided with our smooth contour reconstruction scheme* (right). *Also note the improved self shadows on the ghost.*

[1] e-mail:mattausch@ifi.uzh.ch
[2] e-mail:scherzer@mpi-inf.mpg.de
[3] e-mail:wimmer@cg.tuwien.ac.at
[4] e-mail:takeo@acm.org

In this paper, we propose a solution to this problem that works directly on the shadow silhouettes without modifying the mesh, thus avoiding changing tessellations in dynamic scenes. Our approach creates a Hermite spline approximation of the shadow boundary contours of the underlying smooth mesh (see Figure 3). We then employ a fast 1D subdivision scheme that adaptively subdivides the contour until a given approximation error in world space is reached. For rendering shadows from the contours, we construct shadow volumes in a way that artifacts due to the mismatch of scene geometry and the smooth contours are avoided.

Note that from the two competing real-time shadowing algorithms, shadow volumes [Cro77] and shadow maps [Wil78], *only* shadow volumes are suited for our purpose since they are constructed from the silhouettes of the shadow caster and are therefore amenable to smooth subdivision schemes. For shadow maps, the shadow boundary is created when evaluating the shadow test on the receiver. Here the geometric information of the caster is not available and reconstruction is often not feasible, due to undersampling [McC00, SCH03]. Using different LOD levels or more finely tessellated geometry for shadow-map rendering can only cause severe self-shadowing artifacts.

2. Related Work

The *shadow-volume* algorithm [Cro77], and in particular the GPU-friendly stencil shadow-volume algorithm [Hei91, EK02], is a well-known method for real-time shadow generation, and has been used in high-profile games like Doom 3. Shadow volumes are created by extending the silhouette of an object to infinity with respect to the light source, and then determining for each receiver point whether it is inside (in shadow) or outside this volume (not in shadow). Shadow volumes have a certain look-and-feel which is well suited for certain styles of rendering (e.g., cartoon or NPR rendering) and have been an active area of research for over 3 decades [SOA11]. They produce high-quality shadows and avoid the well-known resampling artifacts of the alternative real-time shadowing technique, shadow mapping [Wil78]. We believe that a revival of shadow volume algorithms is possible due to recent hardware developments: Memory access is becoming increasingly expensive compared to geometric computations, and hence geometric and analytic solutions to rendering problems have been revisited [LAC*11, GBAM11]. As the required sampling rate for shadow maps is not easily predictable due to the arbitrarily large projective error [LGQ*08], huge shadow maps and multiple passes [Eng06] are required to ensure that *all* parts of a scene have sufficiently high shadow quality. Shadow volumes, in comparison, guarantee shadows with equal quality for all parts of a scene. However, the shadow quality is still influenced by the *tessellation* of the shadow caster, since insufficient tessellation will become apparent at projected shadow boundaries. Using our approach, we can lift this restriction with a small computational overhead.

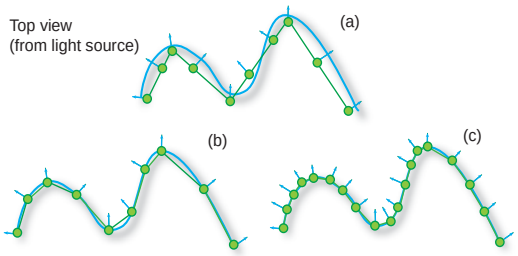

Figure 2: *(a) Hertzmann and Zorin: The silhouette vertices (green) as seen from the light source do not approximate the contours of the underlying smooth mesh (blue) very well. (b) To get a better approximation, we displace the positions of the vertices by using the vertex normals. (c) With recursive subdivision, we approximate the underlying mesh well.*

Our work is also closely related to silhouette refinement methods for meshes [CC78, BA08, WTW*08]) as discussed before.

3. Smooth Silhouette Reconstruction

In the following, we explain our algorithm for creating smooth silhouettes from a coarse mesh, which are subsequently used to render smooth shadow boundaries (note that we always talk about the silhouette of an object as seen from the current light-source position). Our algorithm is inspired by the simplicity and efficiency of the Phong Tessellation algorithm [BA08], which creates a smooth mesh using a subdivision scheme related to the Phong shading technique. However, that concept can lead to discontinuities which can only be partly alleviated with a user-defined blending factor and lead to visible (self-)shadowing artifacts. Instead, as proposed for meshes by Wang et al. [WTW*08], we use Hermite splines to reconstruct the underlying smooth mesh contours with respect to the vertex normals. However, in contrast to Wang et al.'s algorithm, we work on the 1D silhouette directly and not on the mesh itself, avoiding the remeshing step. Our algorithm creates smooth silhouette loops using the following steps (see Figure 2):

a) Find the vertex positions on the accurate silhouette path [HZ00].
b) Displace the silhouette vertices according to a Hermite Spline approximation of the underlying smooth mesh contours.
c) Adaptively subdivide each resulting silhouette loop until it is sufficiently smooth.

3.1. Silhouette Vertex Interpolation

The usual approach for real-time shadows is to create the silhouettes from the edges of the discrete triangle

mesh [EK02], which we call *discrete silhouettes* in the following. However, Hertzmann and Zorin introduced a better way to reconstruct the silhouettes of the underlying smooth mesh as defined by the vertex normals [HZ00], which we call *accurate silhouettes* in the following. A silhouette vertex of the underlying smooth mesh is lying on an edge where one of the smooth surface normals of the edge vertices ($\mathbf{v_i}$, $\mathbf{v_j}$) is front-facing with respect to the light direction, and the other normal back-facing. The silhouette vertex position is obtained by interpolating between $\mathbf{v_i}$ and $\mathbf{v_j}$ so that $\mathbf{n} \cdot \mathbf{l} = 0$. We get the interpolation factor u by solving for $(1-u)\mathbf{n_i} \cdot (\mathbf{l} - \mathbf{v_i}) + u\mathbf{n_j} \cdot (\mathbf{l} - \mathbf{v_j}) = 0$. $\mathbf{l}$ is the light position and $\mathbf{n_i}$ the vertex normal of $\mathbf{v_i}$.

The method of Hertzmann and Zorin already has some benefit for shadowing, e.g., that it partly eliminates the self-shadowing artifacts caused by the discrete silhouettes, and a similar silhouette-generation algorithm was used for shadowing subdivision surfaces [TDC06]. We use their method of computing the accurate silhouette position (as given by u) as basis for our contour reconstruction method. Note that we *only* adapted the accurate vertex interpolation from the paper of Hertzmann and Zorin, while we did not use their 4D hypercube method for silhouette detection. Instead, we loop over all edges to find an edge which contains a silhouette vertex, and then follow each contour line until we obtain a connected loop of silhouette edges. These connected loops serve then as the input to our subdivision algorithm.

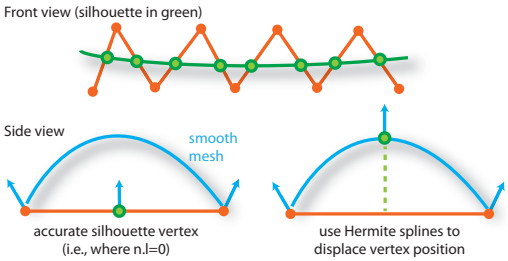

Figure 3: *The vertex positions of the accurate silhouette path on the smooth mesh can be reconstructed using the vertex normals [HZ00]. Using Hermite splines, we displace each silhouette vertex to lie on the smooth surface.*

3.2. Silhouette Vertex Displacement

However, accurate silhouettes do not avoid the tessellation artifacts that appear at the shadow boundaries when projecting the silhouette onto the receiver. The reconstructed smooth mesh contours can be significantly improved as shown in Figure 3. Once we found the accurate silhouette position (i.e., the interpolation factor u between the edge vertices), we *displace* each silhouette vertex from its position on the discrete mesh by using cubic Hermite splines. A cubic Hermite spline of u in $[0..1]$ is defined in matrix form

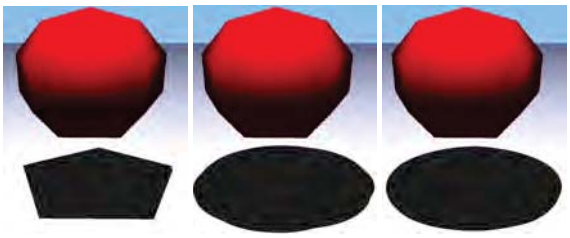

Figure 4: *Silhouette reconstruction on a coarsely tessellated sphere. (left) Hertzmann and Zorin [HZ00]. (middle) Using the interpolation formula from Phong Tessellation [BA08] instead of Hermite splines to generate the 1D shadow silhouette leads to discontinuities. (right) Our reconstruction scheme using Hermite splines.*

as:

$$\mathbf{p}(u) = \begin{pmatrix} u^3 \\ u^2 \\ u \\ 1 \end{pmatrix}^T \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p_0} \\ \mathbf{p_1} \\ \mathbf{t_0} \\ \mathbf{t_1} \end{pmatrix} \quad (1)$$

$\mathbf{p_0}$ and $\mathbf{p_1}$ are the start and the end point of the segment, respectively, and $\mathbf{n_0}$ and $\mathbf{n_1}$ are the associated interpolated normals. $\mathbf{t_0}$ and $\mathbf{t_1}$ denote the tangent vectors at the start and at the end point. To compute a tangent $\mathbf{t_0}$ from normal $\mathbf{n_0}$, we first project the silhouette edge $\mathbf{e} = \mathbf{p_1} - \mathbf{p_0}$ onto $\mathbf{n_0}$, and then subtract the result from $\mathbf{e}$ to get the tangential part $\mathbf{t} = \mathbf{e} - (\mathbf{n_0} \cdot \mathbf{e})\mathbf{n}$. To use t in equation 1, it has to be normalized and scaled with respect to the length of $\mathbf{e}$ (Wang et al. suggest to assume a curved silhouette segment for the length [WTW*08]).

3.3. Adaptive Silhouette Subdivision

In the final step (c) of the algorithm, we successively improve the contour reconstruction by recursively subdividing the silhouette loop. Starting with a silhouette loop from step (b), a new intermediate silhouette vertex $\mathbf{p}$ is obtained by evaluating Equation 1 at position $u = 0.5$. We adaptively subdivide with respect to an approximation error ε, which is measured by the distance to the smooth mesh given by the length of the *displacement* of $\mathbf{p}$. Hence the termination criterion is given by:

$$\left|\frac{\mathbf{p_1} + \mathbf{p_0}}{2} - \mathbf{p}\right| < \varepsilon. \quad (2)$$

The pseudo-code for creating a smooth silhouette segment using our adaptive subdivision scheme is shown in Listing 1. This function has to be called for each silhouette edge in a loop. With this method, all contours in the scene have an approximation error less than ε in *world space*. Note that giving an error measurement of the shadow boundary in *screen space* (which would be the ideal) is difficult since the size of the shadow projection is not known. However, choosing

```
1  ComputeVertex(loop,p0,p1,t0,t1) {
2    // new silhouette vertex
3    p=HermiteInterpolate(p0,p1,t0,t1,u=0.5);
4    if (|p,(p0,p1)/2| > eps) {
5      t=(t0+t1)/2; // interpolate tangent
6      // recursion
7      ComputeVertex(loop,p0,p,t0,t);
8      ComputeVertex(loop,p,p1,t,t1);
9    }
10   else loop.append(p0); // teminate
11 }
```

Listing 1: *Adaptive silhouette loop subdivision taking vertices p_0, p_1, and tangents t_0, t_1 as input*

a small enough value for ε will usually make all shadow boundaries visually smooth. A visual example of the smooth contours resulting from our reconstruction scheme can be seen in Figure 4.

Our subdivision algorithm can handle crease edges, where a different surface normal is used on either side, as can be seen in Figure 5, left. When such an edge is encountered during step (a) of our algorithm (the edge interpolation), we add two silhouette vertices with either surface normal to the silhouette segment (which effectively results in a zero-length silhouette edge). Then the Hermite interpolation automatically adapts for the non-smooth normal as a discontinuity is introduced if the vertices at the segment end points differ from each other.

4. Smooth Shadow Volume Construction

In this section we describe how to use the smooth silhouettes for shadow casting. We observe that naively extruding shadow volumes from the newly computed smooth boundaries instead of the original silhouette creates artifacts. For example, gaps will appear between the shadow volume faces extruded from the smooth contours and the discrete mesh. This causes wrongly classified shadows and subsequently leads to inverted shadows near the silhouette, as shown in

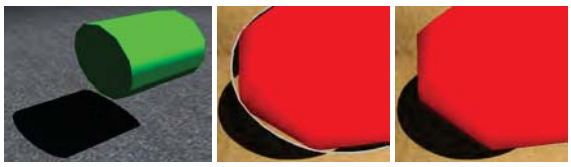

Figure 5: *(Left) Our subdivision algorithm correctly accounts for crease edges like those between the top and the side of the cylinder. (Middle) Shadows may appear inverted if the shadow volume is not properly closed after the subdivision process (smooth contour shown in light gray). (Right) Our smooth shadow volumes cleanly close the gaps.*

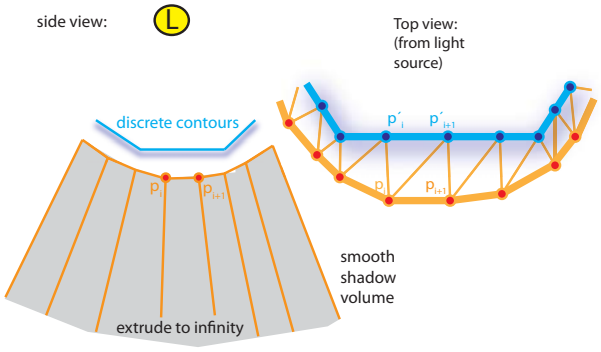

Figure 6: *Triangulation closes the gaps between the smooth contour and the original silhouettes. Note that the offsets from the accurate silhouettes in blue stem from the Hermite interpolation.*

Figure 5, left. We construct the shadow volumes so that these gaps are cleanly closed (Figure 5, right).

Our approach is depicted in Figure 6. We use the standard shadow-volume algorithm to extrude the smooth contour edges from $\mathbf{p_i}$ to $\mathbf{p_{i+1}}$ to infinity (Figure 6). Further, we close the gaps that appear between the smooth shadow-volume faces and the discrete mesh contours. For each smooth silhouette vertex $\mathbf{p}$, we keep track of the discrete mesh position $\mathbf{p'}$ on the original accurate silhouette (i.e., the silhouette from algorithm step (a)). This means that whenever a silhouette edge is subdivided to obtain a new vertex $\mathbf{p}$ from $\mathbf{p_i}$ and $\mathbf{p_{i+1}}$ using Equation 1, a vertex $\mathbf{p'}$ is calculated as $\mathbf{p'} = (1-u)\mathbf{p'_i} + u\mathbf{p'_{i+1}}$ and stored along with $\mathbf{p}$. Finally we triangulate the area between consecutive vertex pairs $\mathbf{p_i}$, $\mathbf{p_{i+1}}$, and $\mathbf{p'_i}$, $\mathbf{p'_{i+1}}$ (Figure 6) to close the gaps.

5. Implementation and Results

Currently, still both the silhouette extraction methods as well as the extrusion of the shadow volumes are implemented in software, leaving a GPU implementation to future work. To extract the discrete silhouettes, we simply loop over all edges and compare the adjacent face normals. In order to avoid self shadowing artifacts (light leaks) due to the difference of the accurate silhouettes of Hertzmann and Zorin to the actual silhouettes of the discrete mesh, we use a small offset of *0.15* along the negative normal direction of the accurate silhouettes.

In order to efficiently find the silhouettes, we require to know the mesh connectivity. Hence for each edge, we store the indices of the 2 triangles that are adjacent to this edge, and for each vertex a list of adjacent edges (used to trace the silhouette loops for the accurate silhouettes). For the Dragon model (100K triangles), these additional data structures require about 1 MB. Please note that similar structures are required by most efficient silhouette detection algorithms.

Scene	tris	discrete	accurate	smooth
Dragon	100k	11.1 (8.3)	10.0 (8.4)	13.1 (8.8)
Paolo	30k	5.0 (4.9)	5.5(4.9)	5.4 (4.9)
Paolo l.	6k	4.8 (4.5)	4.4 (4.3)	4.6 (4.4)
Ghost	1k	4.3 (4.1)	4.1 (4.1)	4.1 (4.1)

Table 1: *This table shows typical render timings (in milliseconds) using different silhouette algorithms for the shadow volumes, i.e., the commonly used discrete mesh silhouettes, accurate silhouettes, and our smooth silhouette reconstruction scheme with subdivision until the maximal error is* 0.01. *In brackets: Timings without regenerating the silhouettes.*

Table 1 compares the render times of discrete mesh silhouettes (as they are most commonly used in real-time rendering) to our smooth silhouette reconstruction algorithm using subdivision until a threshold $\varepsilon = .01$ is reached (resulting in visually completely smooth shadow boundaries). The results were generated on a laptop with a GeForce Quadro 2000M GPU and an Intel Core i7-2820QM (using a single core). The shadows are rendered with stencil shadow volumes [EK02], using 3 16bit render targets with a resolution of 1024×768 in a deferred rendering pipeline. Note that stencil shadow volumes require watertight models, otherwise there may be light leaks. We exploit the properties of the described accurate silhouette reconstruction method (i.e., that it creates long connected loops, which discrete silhouettes usually don't do) for accelerating the rendering process of the shadow volumes in terms of the type of primitive we can use for rendering. In particular, this allows us to use a single OpenGL primitive (e.g., GL_TRIANGLE_STRIP) per loop.

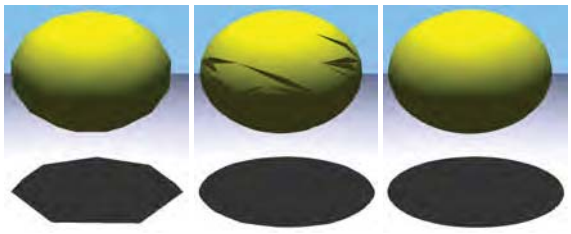

Figure 7: *Smoothing the mesh and shadow contours of a coarse sphere (a) by applying Phong tessellation (blend factor 0.5) for both light and camera view silhouettes will reveal the introduced discontinuities as self shadows (B). Restricting Phong tessellation to the camera view mesh contours only and using our smooth silhouette reconstruction scheme for the light view silhouettes avoids any discontinuities (C).*

All methods are approximately equal in performance for a lower number of triangles. The additional computational complexity due to the recursive Hermite Spline computation starts having a noticeable impact on performance for the more complex Stanford Dragon model. We believe that this overhead of our method can be minimized using an efficient GPU implementation of the recursive subdivision scheme. Most of the performance variations come from the different silhouette reconstruction schemes and not from the rendering, as the timings are quite similar if the silhouettes are not recomputed (shown in brackets).

Figure 8: *Using the discrete mesh silhouettes (left) causes self-shadowing artifacts on the Paolo model. The self-shadows can be improved using accurate silhouettes (middle), and further improved with smooth silhouettes (right).*

Figure 4 compares our method to a hypothetical algorithm where the interpolation formula used in Phong tessellation is used instead of Hermite splines to create the smooth contour in our algorithm. It can be seen that this approach would reveal discontinuities.

Figure 7 compares our method to a Phong Tessellation variant where not only the screen-space silhouette of the geometry is refined, but also the silhouette seen from the light. As can be seen, the introduced discontinuities due to the local operator are strongly visible in the shadows. Using our algorithm in combination with Phong Tessellation for the object contours gives satisfactory results while also avoiding morphing artifacts.

Figure: 8 shows another benefit of our reconstruction scheme. Typical self-shadow artifacts that appear for the standard approach of using discrete mesh silhouettes for shadow volumes (left) are reduced with the method of Hertzmann and Zorin. (middle), and practically removed with smooth silhouette reconstruction (right).

Figure 9: *Closeup of a shadow of an arm of the Paolo model (6K triangles). While the discrete shadow is quite well tessellated (left), our algorithm improves the smoothness as can be seen in the closeup (middle) and the full image (right).*

Figure 9 shows the benefit of our reconstruction scheme

Figure 10: *Closeup of the Stanford dragon model. Due to the projection, this finely tessellated model (100K triangles) can still reveal discontinuities (top), which can be removed using our method (bottom and right).*

on a model which would typically be used in a game, with several thousand vertices. In Figure 10, discontinuities are still visible in the discrete shadows of the rather complex Stanford Dragon model (100K triangles), whereas our method produces a smooth shadow boundary. This example shows that the degree of model tessellation required to guarantee a smooth shadow can be almost arbitrarily high, and hence underlines the usefulness of our algorithm in practical scenes.

The particular shadowing method is orthogonal to our algorithm (as long as it operates on the silhouettes). Hence it is straightforward to use our tessellated silhouettes as input to a soft-shadow volume algorithm like penumbra wedges [AAM03] (refer to Figure 11).

6. Conclusions

To the best of our knowledge, we have presented the first method for rendering smooth shadow boundaries. The degree of smoothness can be chosen globally independent of the tessellation of the geometry. The simplicity and efficiency of our approach makes it well suited for practical use. Our technique can be well combined with algorithms that refine camera-space silhouettes, so that both object and shadow boundaries become smooth. Compared to naive approaches based on smooth screen-space silhouettes, our approach avoids shadow artifacts and visible mesh retessellation artifacts.

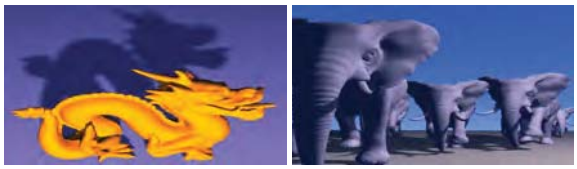

Figure 11: *Our algorithm applied to soft shadows generated with penumbra wedges [AAM03].* Left: *dragon scene;* right: *elephant scene (423K triangles) with overlapping shadows and self shadows (e.g., on the trunk).*

References

[AAM03] ASSARSSON U., AKENINE-MÖLLER T.: A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Trans. Graph. 22*, 3 (July 2003), 511–520. 6

[BA08] BOUBEKEUR T., ALEXA M.: Phong tessellation. *ACM Trans. Graph. 27* (2008), 141:1–141:5. 1, 2, 3

[CC78] CATMULL E., CLARK J.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6 (1978), 350–355. 1, 2

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. *Computer Graphics 11* (1977), 242–248. 2

[EK02] EVERITT C., KILGARD M.: *Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering*. Tech. rep., NVIDIA Corporation, mar 2002. 2, 3, 5

[Eng06] ENGEL W. (Ed.): *Cascaded Shadow Maps*, vol. 5 of *ShaderX*. Charles River Media, 2006, pp. 197–206. 2

[GBAM11] GRIBEL C. J., BARRINGER R., AKENINE-MÖLLER T.: High-quality spatio-temporal rendering using semi-analytical visibility. *ACM Trans. Graph. 30*, 4 (2011), 54:1–54:12. 2

[Hei91] HEIDMANN T.: Real shadows, real time. *Iris Universe 18* (1991), 28–31. Silicon Graphics, Inc. 2

[HZ00] HERTZMANN A., ZORIN D.: Illustrating smooth surfaces. In *Proceedings of SIGGRAPH 2000* (New York, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 517–526. 2, 3

[LAC*11] LEHTINEN J., AILA T., CHEN J., LAINE S., DURAND F.: Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. 30*, 4 (2011), 55:1–55:12. 2

[LGQ*08] LLOYD D. B., GOVINDARAJU N. K., QUAMMEN C., MOLNAR S. E., MANOCHA D.: Logarithmic perspective shadow maps. *ACM Trans. Graph. 27*, 4 (2008), 1–32. 2

[McC00] MCCOOL M.: Shadow volume reconstruction from depth maps. *ACM Transactions on Graphics 19* (2000), 1–26. 2

[SCH03] SEN P., CAMMARANO M., HANRAHAN P.: Shadow Silhouette Maps. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2003) 22*, 3 (2003), 521–526. 2

[SOA11] SINTORN E., OLSSON O., ASSARSSON U.: An efficient alias-free shadow algorithm for opaque and transparent objects using per-triangle shadow volumes. *ACM Trans. Graph. 30*, 6 (2011), 153:1–153:10. 2

[TDC06] TANG M., DONG J.-X., CHOU S.-C.: *Real-Time shadow volume algorithm for subdivision surface based models*. CGI'06. Springer, Berlin, Heidelberg, 2006, pp. 538–545. 3

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *Computer Graphics (Proc. SIGGRAPH 78) 12*, 3 (1978), 270–274. 2

[WTW*08] WANG L., TU C., WANG W., MENG X., CHAN B., YAN D.: Silhouette smoothing for real-time rendering of mesh surfaces. *Visualization and Computer Graphics, IEEE Transactions on 14*, 3 (2008), 640 –652. 1, 2, 3