CHAPTER 4

# HACKING THE SHADOW TERMINATOR

*Johannes Hanika*
*KIT/Weta Digital*

## ABSTRACT

Using ray tracing for shadows is a great method to get accurate direct illumination: precise hard shadows throughout the scene at all scales, and beautiful penumbras from soft box light sources. However, there is a long-standing and well-known issue: the *terminator problem*. Low tessellation rates are often necessary to reduce the overall load during rendering, especially when dynamic geometry forces us to rebuild a bounding volume hierarchy for ray tracing every frame. Such coarse tessellation is often compensated by using smooth shading, employing interpolated *vertex normals*. This mismatch between geometric normal and shading normal causes various issues for light transport simulation. One is that the geometric shadow, as very accurately reproduced by ray tracing (see Figure 4-1, left), does in fact not resemble the smooth rendition we are looking for (see Figure 4-1, right). This chapter reviews and analyzes a simple hack-style solution to the terminator problem. Analogously to using shading

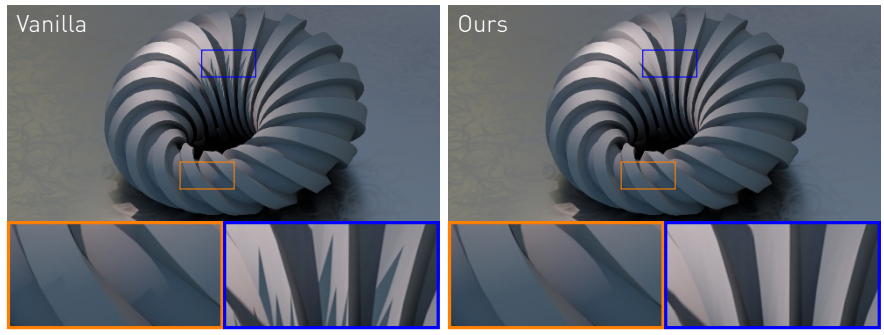


**Figure 4-1.** *Low-polygon ray tracing renders are economical with respect to acceleration structure build times, but introduce objectionable artifacts with shadow rays. This is especially apparent with intricate geometry such as this twisted shape (left). In this chapter, we examine a simple and efficient hack to alleviate these issues (right).*


A. Marrs, P. Shirley, I. Wald (eds.), *Ray Tracing Gems II*, https://doi.org/10.1007/978-1-4842-7185-8_4

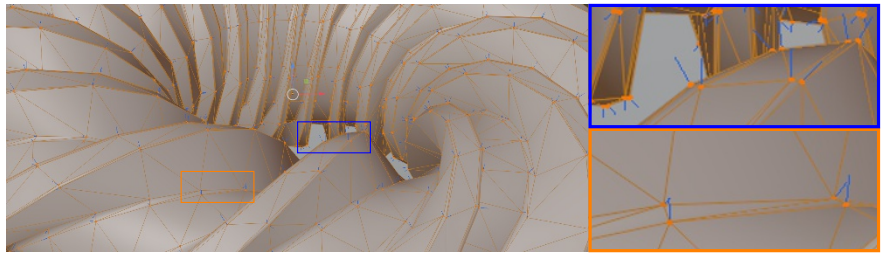

**Figure 4-2.** *A screen capture of the Blender viewport, to visualize the mesh and vertex normals used for Figure 4-1. The triangles come from twisted quads and have extremely varying normals. There are tiny and elongated triangles at the bevel borders especially at the back, which is where some artifacts remain in the render.*

normals that are smooth but inconsistent with the geometry, we will use shading points which are inconsistent with the geometry but result in smooth shadows. We show how this is closely related to quadratic Bézier surfaces but cheaper and more robust to evaluate.

## 4.1 INTRODUCTION

Ray tracing has been an elegant and versatile method to render 3D imagery for the better part of the last 50 years [1]. There has been a constant push to improve the performance of the technique throughout the years. Recently, we have seen dedicated ray tracing hardware units that even make this approach viable for real-time applications. However, since this operates hard at the boundary of the possible, some classic problems resurface. In particular, issues with low geometric complexity and simple and fast approximations are strikingly similar to issues that the community worked on in the 1980s and 1990s. In this chapter we discuss one of these: *the terminator problem*.

In 3D rendering, geometry is often represented as a polygon mesh. In fact, today triangle meshes are the ubiquitous choice. While quad meshes are often used to reduce memory footprint, individual quads are often treated as two triangles (instead of a bilinear patch) internally. We will thus limit our discussion here to triangle meshes. These can be used for intricate shapes such as Figure 4-1, which are tessellated and triangulated as illustrated in Figure 4-2. In this particular example, the shape is only very coarsely tessellated. The mesh contains long and thin triangles as well as a large variation of normals throughout one triangle (marked with blue in the image).

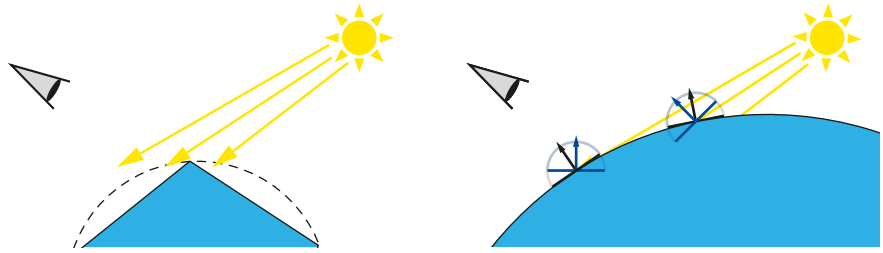

**Figure 4-3.** *Two incarnations of the terminator problem. Left: a (very) coarsely tessellated sphere is illuminated. The sphere surface (dashed) we were trying to approximate would show smooth shadow falloff, but the flat triangle surface facing the eye in this example would be rendered completely black. Using vertex normals (i.e., Phong shading) to evaluate the materials smoothly does unfortunately not render the shadow rays visible. For this to happen, we need to start the shadow rays at the dashed surface. This chapter proposes a simple and cheap way to do this. Right: the bump terminator problem is caused by mismatching hemispheres defined by the geometric normals (black) and the shading normals (blue). This kind of problem persists even when using smooth base geometry.*

To hide the discretization artifacts coming from this, shading is traditionally performed using a normal resulting from barycentric interpolation of *vertex normals* across the triangle [9]. Using normals that are inconsistent with the geometric surface normal can lead to various issues, for instance with symmetry in the light transport operator [13]. Woo et al. [16] point out a particular issue with ray traced shadows, which we illustrate again in Figure 4-3, left. Say we want to render the dashed, smooth surface, but an accurate representation is too expensive to intersect with a ray. We thus use a triangle mesh indicated by the blue solid. The triangle facing the viewpoint to the left should show a smooth falloff in shading to approximate the dashed surface well. Instead, because it is facing away from the light source, it will be rendered completely black: rays traced from the triangle surface to the light will correctly and accurately report that this surface is in shadow.

This problem is well understood and solutions using small user-driven epsilon values to push out the shading point from the triangle surface have been proposed as early as 1987 [11]. Some years later, CPU ray tracing had advanced significantly [15, 2], so objects with low tessellation became interesting for real-time display. Such meshes are prone to showing the terminator problem. Consequently, in a side note in [7, Section 5.2.9], a simple way of determining an adaptive epsilon value was proposed as an inexpensive workaround. In this chapter, we evaluate this approach in a bit more depth.

## 4.2 RELATED WORK

Over the years, many approaches have been proposed to work around the problems arising with shading normals. They can be roughly classified into three groups. The first deals with geometry terms arising in bidirectional light transport. Veach [13] observed that shading normals introduce an inconsistency between path tracing and light tracing. He proposed a correction factor based on the ratio of cosines between the geometry and shading normals. The pictures in this chapter are rendered with a unidirectional path tracer and thus do not use this correction factor. In fact, the method proposed here, as is mostly the case when working with shading normals, is not reciprocal.

The second class involves smoothing the shadow terminator by altering the shading of a microfacet material model. These approaches start from the observation that a shading normal other than the geometric normal can be pushed inside the microfacet model, as an off-center normal distribution. This idea can be turned into a consistent microsurface model [10], and from there the surface reflectance can be derived by first principles. Since the extra roughness introduced into the microsurface will lead to some overshadowing, multiple scattering between microfacets can be taken into account to brighten the look. This technique has been simplified for better adoption in practice and refined to reduce artifacts caused by the simplifications [3, 4, 5]. These approaches start from the observation that a bump map can make the reflectance extend too far into the region where the geometric normal is, in fact, shadowed (see the left two mismatching hemispheres in Figure 4-3). In this case the result will be black, but the material response is still bright, leading to a harsh shading discontinuity. The simple solution provided is to introduce a shadow falloff term that makes sure that the material evaluation smoothly fades to black. This is illustrated in Figure 4-4. As an example, we implemented Conty et al.'s method [4]. Note that our material is diffuse, and thus violates the assumptions they made about how much energy of the lobe is captured in a certain angular range. Thus, the method moves the shadow region, but not far enough by a large margin, at least for this very coarsely tessellated geometry. The best we could hope for here is to darken the gradient so much that it hides the coarse triangles completely, leading to significant look changes as compared to the base version.

In some sense Keller at al. [8, Figure 21] are also changing the material model. They bend the shading normal depending on the incoming ray, such

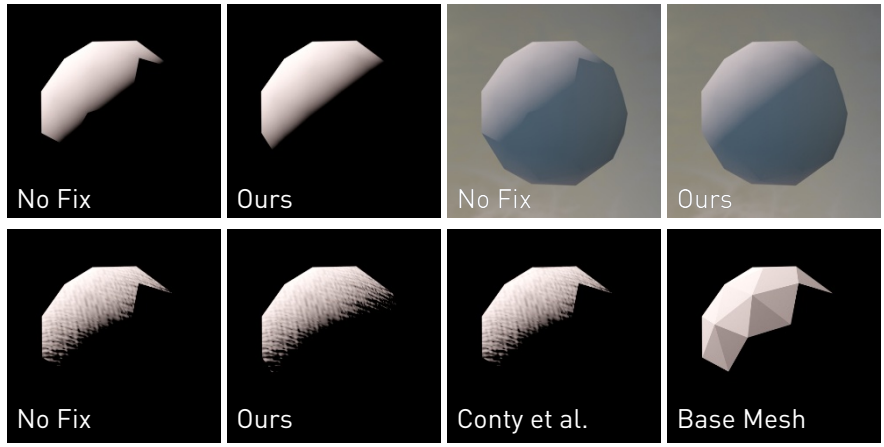


**Figure 4-4.** *This is how the problem illustrated in Figure 4-3 manifests itself in practice. Top row: without bump map; left two: direct illumination from a spotlight only; right two: with global illumination and an environment map. Note that the indirect lighting at the bottom of the sphere does not push out the shading point in any of the images. Bottom row: with a bump map applied. The hack leaves the look of the surface untouched as much as possible. Conty et al.'s shadowing term [4] does not help on the diffuse surface.*

that a perfectly reflected ray would still be just above the geometric surface. This changes the hemisphere of the shading normal, whereas in a sense we change the hemisphere of the geometric normal. At least we make sure that some shadow rays will yield a nonzero result even when cast under the geometric surface.

In general, it is hard to create a consistent microsurface model in the presence of both normal maps and vertex normals. Figure 4-5 illustrates the issue. Schüßler et al. [10] cut the surface into Fresnel lens–like microsteps, where one side (orange in the figure) corresponds to the shading normal. To complete the model and make it physically consistent, there needs to be another microfacet orientation (drawn in light blue) to close the surface. When two triangles meet at the same vertex with the same normal, the orientation of these additional microfacets lead to a discontinuity.

On the other hand, we only want to smooth out the geometric shadow terminator; i.e., we are dealing with shadow rays more than with misaligned hemispheres for geometric and shading normal. This means that we can make assumptions about slow and smooth variation of our normals across the whole triangle. It follows that the technique examined here does not work

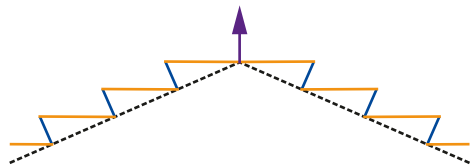

**Figure 4-5.** *The microsurface model of Schüßler et al. [10] at a vertex with a vertex normal (blue). The facets oriented toward the shading normal (orange) have a smooth transition at the vertex (facing the same direction in this closeup). However, the light blue facets orthogonal to the geometric surface (dashed) will create a discontinuous look.*

for normal maps, but could likely be combined with a microfacet model addressing the hemisphere problem.

The third approach is the obvious choice: resolve issues with coarse tessellation by tessellating more finely. A closely related technique is called *PN triangles* [14]. One constructs Bézier patches from vertices and normals, usually the cubic version [12]. Van Overveld and Wyvill [12] also mention the possibility to do quadratic, which is more closely related to the technique examined here. The main difference is that we don't want to tessellate but only fix the shadows instead.

## 4.3 MOVING THE INTERSECTION POINT IN HINDSIGHT

As a minimally invasive change to fix the harsh shadow from coarsely tessellated geometry, we want to move only the primary intersection point away from the triangle, ideally to a location on a smooth freeform surface (i.e., the dashed surface in Figure 4-3). For this, we want to look at how a simple quadratic Bézier patch is constructed from vertices and vertex normals.

Figure 4-6 shows a triangle with vertices $A, B, C$, an intersection point $P$, and an illustration of the barycentric coordinates $u, v, w$ on the left. To construct a point $P'$ on a quadratic Bézier patch defined by these vertices, these barycentric coordinates, and the vertex normals $n_A, n_B, n_C$, we use de Casteljau's algorithm. First, we construct additional control points $AB, BC, CA$. We have some freedom in how to do this; options have been discussed in the literature [14, 12]. Then, we use the barycentric coordinates $u, v, w$ to compute three additional vertices $A'$, $B'$, and $C'$ from the three triangles $(A, AB, CA)$, $(AB, B, BC)$, and $(CA, BC, C)$, respectively. These three new points form another triangle, which we interpolate once more with the barycentric coordinates $u, v, w$ to finally arrive at $P'$.

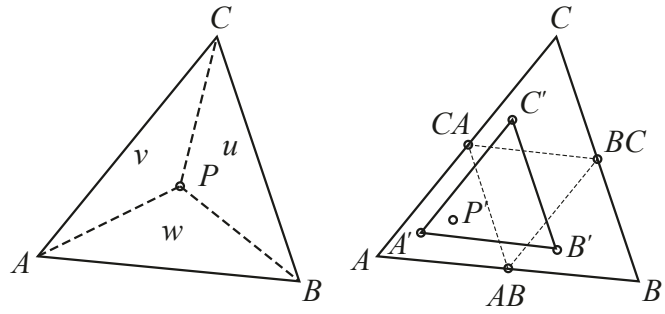


**Figure 4-6.** *Left: illustration to show our naming scheme inside a triangle. Right: schematic of the barycentric version of de Casteljau's algorithm. $P'$ is computed as the result of a quadratic barycentric Bézier patch, defined by the corner points $A, B, C$ as well as the extra nodes $AB, BC, CA$. These will in general not lie in the plane of the triangle $ABC$.*

Let's have a look how to place the extra control points such as $AB$. The patches should not have cracks between them, so the placement can only depend on the data available on the edge, for instance, $A, B, n_A, n_B$. Note that, even then, using a real Bézier patch as geometry would potentially open cracks at creases where a mesh defines different normals for the same vertex on different faces. When only moving the starting point of the shadow ray, this is not a problem. We could, for instance, place $AB$ at the intersection of three planes: the two tangent planes at $A, n_A$ and $B, n_B$ as well as the half-plane at $(A + B)/2$ with a normal in the direction of the edge $B - A$. These three conditions result in a $3 \times 3$ linear system of equations to solve. This requires a bit of computation, and there is also the possibility that there is no unique solution.

Instead, let's look at the simple technique proposed in [7, Section 5.2.9]. The procedure is similar in spirit to a quadratic Bézier patch as we just discussed it, and it is summarized in pseudocode in Listing 4-1. An intermediate triangle is constructed, and the final point $P'$ is placed in it by interpolation using the barycentric coordinates $u, v, w$. The difference to the quadratic patch is the way this triangle is constructed (named `tmpu, tmpv, tmpw` in the code listing). To avoid the need for the extra control points on each edge, the algorithm proceeds as follows: the vector from each corner of the triangle to the flat intersection point $P$ is computed and subsequently projected onto the tangent plane at this corner. This is illustrated in Figure 4-7, right. This procedure is very simple and efficient, but comes with a few properties that are different than a real Bézier surface, which we will discuss next.

**Listing 4-1.** *Pseudocode of the simple shading point offset procedure outlined in [7].*

```
// get distance vectors from triangle vertices
vec3 tmpu = P - A, tmpv = P - B, tmpw = P - C
// project these onto the tangent planes
// defined by the shading normals
float dotu = min(0.0, dot(tmpu, nA))
float dotv = min(0.0, dot(tmpv, nB))
float dotw = min(0.0, dot(tmpw, nC))
tmpu -= dotu*nA
tmpv -= dotv*nB
tmpw -= dotw*nC
// finally P' is the barycentric mean of these three
vec3 Pp = P + u*tmpu + v*tmpv + w*tmpw
```

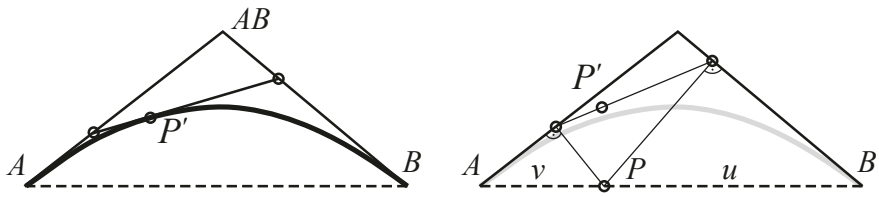


**Figure 4-7.** *Left: 2D side view visualization of a quadratic Bézier patch (for the w = 0 case), showing the line between A and B. The barycentric coordinates (u, v) are relevant for this 2D slice: de Casteljau proceeds by interpolating A and AB in the ratio v:u, as well as AB and B. The two results are then interpolated using v:u again to yield P′. Right: our simple version does not require the point AB for the computation: it projects the intersection point P on the flat triangle orthogonally to the tangent planes at A and B (note that AB lies on both of these planes, but we don't need to know where). These temporary points are then interpolated in the ratio v:u again, to yield the shading intersection point P′. The Bézier line from the left is replicated to emphasize the differences.*

## 4.4 ANALYSIS

To evaluate the behavior of our cheap approximation, we plotted a few side views in flatland, comparing a quadratic Bézier surface to the surface resulting from the pseudocode in Listing 4-1. The results can be seen in Figure 4-8.

The first row shows a few canonical cases with distinct differences between the two approaches. In the first case in Figure 4-8a, the two surfaces are identical, as both vertex normals point outward with a 45° angle to the geometric normal. In Figure 4-8b, an asymmetric case is shown; note how the point *AB* is moved toward the left. It can be seen that our surface (green) has more displacement from the flat triangle, only approximately follows the

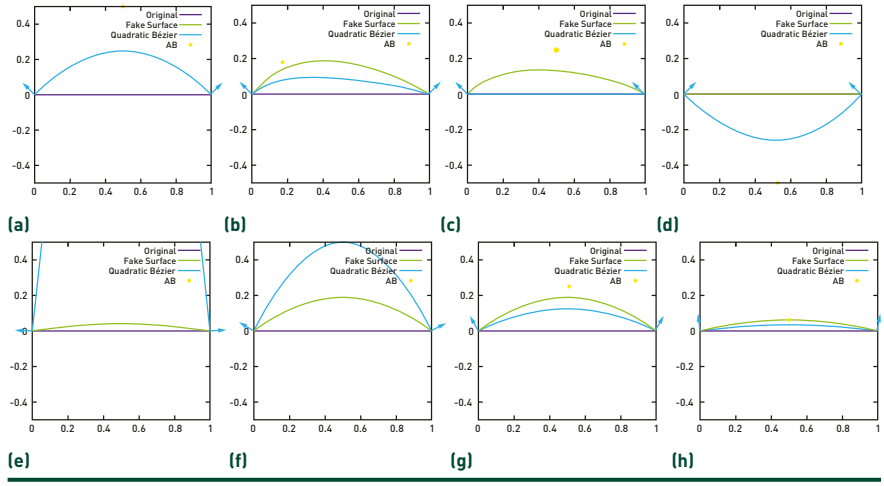


**Figure 4-8.** *2D comparison against quadratic Bézier: (a) Canonical case, equivalent. (b) Asymmetric case, no tangent at B and out of convex hull with control point AB. (c) Degenerate case not captured by quadratic Bézier. (d) Concave case clamped by `min()` in our code. (e–h) The behavior when moving AB toward the geometric surface.*

tangent planes at both vertices, and is placed outside the convex hull of the control cage of the Bézier curve. Though all this may be severe downsides for sophisticated applications, the deviations from the Bézier behavior may be acceptable for a simple offset of the shadow ray origin.

In Figure 4-8c, a degenerate case is shown: both vertex normals point in the same direction. This happens, for instance, in the shape in Figure 4-2 at the flanks of the elevated structures, where one normal is interpolated toward the top and one toward the bottom, but with both pointing the same direction. The approach to solve for *AB* by plane intersection now fails because the two tangent planes are parallel, and no quadratic Bézier can be constructed. This is a robustness concern, especially if the vertex normals aren't specifically authored for quadratic patches or are animated.

In Figure 4-8d the concave case is shown. Because we clamp away positive dot products, such a case will evaluate to the flat surface instead of bulging to the inside of the shape. This is the desired behavior: we don't push shadow ray origins inside the object.

The second row varies the distance of *AB* to the surface. In Figures 4-8e and 4-8f, *AB* is far away from the surface, where the former is almost degenerate. The quadratic Bézier patch consequently moves the curve very far away, too. This may not be the desired behavior in our case, as animation

might have caused a normal to be this extreme by accident. The cheap approximation stays much closer to the geometric surface.

Figures 4-8g and 4-8h show vertex normals closer to the geometric normal than 45°. Here, the situation turns around, and the cheap surface is farther away from the geometry than the Bézier patch. At these small deviations, however, this is much less of a concern.

In summary, the cheap surface is only approximately tangent at the vertices and violates the convex hull property with respect to the control cage.

## 4.5 DISCUSSION AND LIMITATIONS

The surface examined in this chapter does not strictly follow the tangent condition at the vertices. Because we only use it to offset the origin of the shadow ray, this is not a big issue: the shading itself will use the vertex normal to determine the upper hemisphere for lighting.

We have seen that the cheap surface does not obey the control cage of the Bézier patch. Instead, it is farther from the surface for small vertex normal variations, and closer for large variations, as compared to the Bézier surface. As the interpolation is just quadratic, it does not overshoot or introduce ringing artifacts of any kind.

The larger offsets for small variations may lead to shadow boundaries that are slightly more wobbly than expected. See, for instance, Figure 4-1 just above the blue inset. The shadow edge was already choppy because of the low tessellation both in the shadow caster and in the receiver. Offsetting the shadow ray origin seemed to aggravate this issue. However, in the foreground (to the left of the orange inset), the opposite can be observed: the shadow edges look smoother than before.

We have seen that the surface offset can work with concave objects in a sense that it will not push the shadow ray origins inside the object. However, special care has to be taken when working with transparent objects. Transmitted rays may need to apply the offset the other way around, i.e., flip all the vertex normals before evaluation.

The mesh in Figure 4-2 was modeled with bevel borders, i.e., the sharp corners consist of an extra row of small polygons to make sure that the edge appears sharp. If such creases are instead modeled using face-varying vertex normals, the shadow ray origins will have a discontinuous break at the edge.

Note that the surface will still be closed and the shading will depend on the normals, so this only affects the shadows.

Further, the method is specifically tailored for vertex normals. This means that for multi-lobe and off-center microfacet models, such as multi-modal LEADR maps [6], there is still a necessity to adapt microfacet surface models or adjust the shadowing and masking terms.

We only discussed shadow rays for evaluation of direct lighting. The same can be said about starting indirect lights when material scattering is used. In this case, the effect of the shadow terminator is a bit different and more subtle; see, for instance, the bottom half of the spheres with global illumination enabled in Figure 4-4.

## 4.6 CONCLUSION

We discussed a simple and inexpensive side note on the terminator problem from the time when CPU ray tracing became interesting for real-time applications. This method effectively resembles quadratic Bézier patches, but is cheaper to evaluate and also works in degenerate cases where creating the additional control points for such a patch would be ill-posed. It only offsets the shading point from which the shadow ray is cast; the surface itself remains unchanged. This means that low-polygon meshes will receive smooth shadows without resorting to more heavyweight tessellation approaches that may require a bounding volume hierarchy rebuild, too. In the long run, the problems of low-polygon meshes may go away as finer tessellations will become viable for real-time ray tracing. Until then, this technique has a valid use case again.

## REFERENCES

**[1]** Appel, A. Some techniques for shading machine renderings of solids. In *American Federation of Information Processing Societies*, volume 32 of *AFIPS Conference Proceedings*, pages 37–45, 1968. DOI: 10.1145/1468075.1468082.

**[2]** Benthin, C. *Realtime Ray Tracing on Current CPU Architectures*. PhD thesis, Saarland University, Jan. 2006.

**[3]** Chiang, M. J.-Y., Li, Y. K., and Burley, B. Taming the shadow terminator. In *ACM SIGGRAPH 2019 Talks*, 71:1–71:2, 2019. DOI: 10.1145/3306307.3328172.

**[4]** Conty Estevez, A., Lecocq, P., and Stein, C. A microfacet-based shadowing function to solve the bump terminator problem. In E. Haines and T. Akenine-Möller, editors, *Ray Tracing Gems*, chapter 12. Apress, 2019.

**[5]** Deshmukh, P. and Green, B. Predictable and targeted softening of the shadow terminator. In *ACM SIGGRAPH 2020 Talks*, 6:1–6:2, 2020. DOI: 10.1145/3388767.3407371.

**[6]** Dupuy, J., Heitz, E., Iehl, J.-C., Poulin, P., Neyret, F., and Ostromoukhov, V. Linear efficient antialiased displacement and reflectance mapping. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 32(6):Article No. 211, 2013. DOI: 10.1145/2508363.2508422.

**[7]** Hanika, J. *Spectral Light Transport Simulation Using a Precision-Based Ray Tracing Architecture*. PhD thesis, Ulm University, 2011.

**[8]** Keller, A., Wächter, C., Raab, M., Seibert, D., van Antwerpen, D., Korndörfer, J., and Kettner, L. The iray light transport simulation and rendering system. In *ACM SIGGRAPH 2017 Talks*, 34:1–34:2, 2017. DOI: 10.1145/3084363.3085050.

**[9]** Phong, B. T. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, June 1975. DOI: 10.1145/360825.360839.

**[10]** Schüßler, V., Heitz, E., Hanika, J., and Dachsbacher, C. Microfacet-based normal mapping for robust Monte Carlo path tracing. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 36(6):205:1–205:12, Nov. 2017. DOI: 10.1145/3130800.3130806.

**[11]** Snyder, J. and Barr, A. Ray tracing complex models containing surface tessellations. *Computer Graphics*, 21(4):119–128, 1987.

**[12]** Van Overveld, C. and Wyvill, B. An algorithm for polygon subdivision based on vertex normals. In *Proceedings of Computer Graphics International Conference*, pages 3–12, Jan. 1997. DOI: 10.1109/CGI.1997.601259.

**[13]** Veach, E. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, 1998.

**[14]** Vlachos, A., Peters, J., Boyd, C., and Mitchell, J. L. Curved PN triangles. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 159–166, 2001. DOI: 10.1145/364338.364387.

**[15]** Wald, I. and Slusallek, P. State of the art in interactive ray tracing. In *Eurographics 2001—STARs*, 2001. DOI: 10.2312/egst.20011050.

**[16]** Woo, A., Pearce, A., and Ouellette, M. It's really not a rendering bug, you see ... *IEEE Computer Graphics and Applications*, 16(5):21–26, Sept. 1996. DOI: 10.1109/38.536271.