Real-Time Shading with Free-Form Planar Area Lights Using Linearly Transformed Cosines

Takahiro Kuge
Waseda University

Tatsuya Yatagawa
The University of Tokyo

Shigeo Morishima
Waseda University

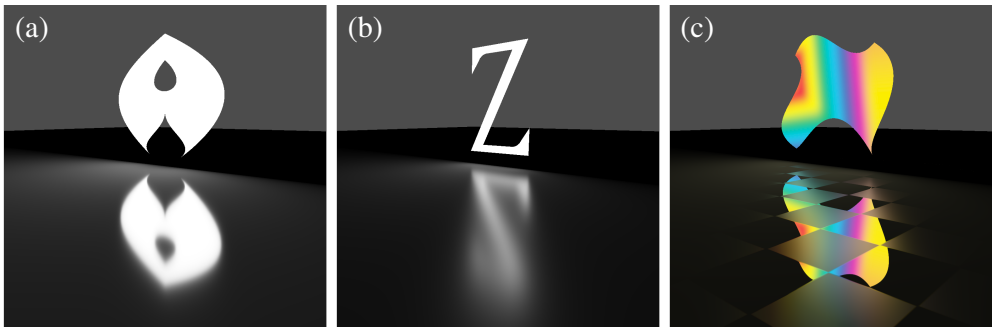


Figure 1. Rendering results for free-form planar area lights using our method. All of the results are obtained in real time using standard graphics hardware. Our method supports light sources that are (a) non-convex with internal holes and (b) in the shape of a character, provided that the contour is defined by a set of Bézier curves. We can also define (c) spatially varying light colors using a texture. The roughness, α, of the floor is 0.1 for (a) and 0.25 for (b), while it is driven by a checkerboard texture in (c).

Abstract

This article introduces a simple yet powerful approach to illuminating scenes with free-form planar area lights in real time. For this purpose, we extend a previous method for polygonal area lights in two ways. First, we adaptively approximate the closed boundary curve of the light, by extending the Ramer–Douglas–Peucker algorithm to consider the importance of a given subdivision step to the final shading result. Second, we efficiently clip the light to the upper hemisphere, by algebraically solving a polynomial equation per curve segment. Owing to these contributions, our method is efficient for various light shapes defined by cubic Bézier curves and achieves a significant performance improvement over the previous method applied to a uniformly discretized boundary curve.

 ISSN 2331-7418

1 Introduction

The area light is a standard representation of light sources in the real world and is typically used to represent artificial lighting equipment such as fluorescent lights. Therefore, efficient rendering of scenes with area lights is an important research topic in physically based rendering. Although importance sampling can be a solution for area light evaluation in offline rendering scenarios, most real-time applications cannot afford such computationally expensive sampling.

To overcome this limitation, Heitz et al. [2016] introduced a solution for real-time polygonal area lights and glossy surfaces, where the surface response is modeled by a microfacet *bidirectional reflectance distribution function* (BRDF) [Torrance and Sparrow 1967]. Their method approximates the microfacet reflection lobe with a clamped cosine distribution that has been linearly transformed. Because the integral of this distribution over the polygonal shape of the light is invariant to linear transformation, the shading calculation boils down to a simpler problem involving a Lambertian surface. Then, a closed-form solution [Baum et al. 1989] can be applied to the combination of a Lambertian surface and a polygonal area light. This method for polygonal shapes was later extended to line- and disk-shaped area lights [Heitz and Hill 2017].

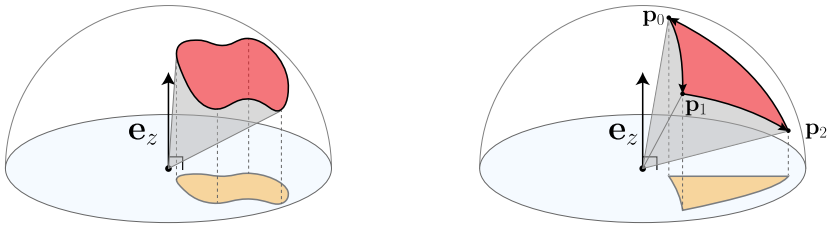
Unfortunately, despite the efforts of these previous studies, real-time illumination from free-form planar area lights has not yet been achieved. The central difficulty is the fact that the closed-form solution for polygonal area lights cannot be used for free-form shapes even when the boundary shape is defined mathematically using polynomial curves.

2 Real-Time Shading with Free-Form Planar Area Lights

To address this problem, we extend the previous method for polygonal area lights by discretizing the curved boundary of the light. In the following discussion, we assume that this boundary is defined by a set of cubic Bézier curves.

2.1 Shading with Linearly Transformed Cosines

The integration involved with area light shading originates from a study by Lambert in the 1700s [Lambert 1760], where he proved that the integration of a cosine function over a region on the unit hemisphere can be transformed into another integration over the area given by projecting the region orthogonally to the unit disk (see Figure 2(a)). Baum et al. [Baum et al. 1989] applied this solution to the computer graphics problem of calculating differential area–polygon form factors for radiosity, which is equivalent to the shading of a Lambertian surface from a polygonal area light with uniform emission. In this case, the unit disk for calculating a form factor is defined on a local shading plane, and the area on the unit disk (see Figure 2(b)) is calculated using Lambert's formula.



(a) Free shape on hemisphere (b) Polygon on hemisphere

Figure 2. The integration of the clamped cosine function over the region on the unit hemisphere (red region) is equivalent to calculating the area of its orthogonal projection to the unit disk (orange region). When a polygon is projected onto the unit hemisphere, the area on the unit disk has a closed-form solution [Lambert 1760].

Let L be the radiance of the area light and $\mathbf{p}_0, \ldots, \mathbf{p}_{N-1}$ be the vertex positions of a polygon in the local coordinate system centered on the shading point. Then, we can obtain the radiance at the shading point from the following formula:

$$I \approx \frac{L}{\pi} \cdot \frac{1}{2} \sum_{i=0}^{N-1} \operatorname{acos}(\langle \mathbf{p}_i, \mathbf{p}_j \rangle) \left\langle \frac{\mathbf{p}_i \times \mathbf{p}_j}{\|\mathbf{p}_i \times \mathbf{p}_j\|}, \mathbf{e}_z \right\rangle, \tag{1}$$

where $j = (i+1) \bmod N$, $\langle \mathbf{p}, \mathbf{q} \rangle$ is the dot product of vectors $\mathbf{p}$ and $\mathbf{q}$, and $\mathbf{e}_z = (0,0,1)^T$ is the unit vector of the z-axis. Refer to the technical report by Heitz [2017] for the geometric interpretation of this formula.

To apply Equation (1) to non-Lambertian surfaces, Heitz et al. [2016] proposed approximating the lobe shape of a microfacet BRDF by a cosine distribution. In particular, they demonstrated that the lobe shape given by the GGX normal distribution function [Walter et al. 2007] can be accurately approximated by a clamped cosine distribution that has been linearly transformed. They referred to the distribution obtained from the linear transformation as a *linearly transformed cosine* (LTC). By applying the inverse transformation to the vertices of the polygon area light, shading of glossy surfaces can then be calculated using Equation (1) owing to this approximation.

2.2 Linear Transformation for Bézier Curves

A Bézier curve $\mathbf{B}(t)$ with curve parameter $t \in [0,1]$ is defined using Bernstein's polynomial $J_{K,i}$ as follows:

$$\mathbf{B}(t) = \sum_{i=0}^{K} J_{K,i}(t)\mathbf{p}_i, \quad \text{where } J_{K,i}(t) = \binom{K}{i} t^K (1-t)^{K-i}$$

and $\mathbf{p}_i$ is the ith control point of the Kth-order Bézier curve. To apply the previous method for polygonal shapes, we need to calculate the Bézier curve $\mathbf{B}'(t)$ after linear transformation. Owing to the affine invariance of Bézier curves, we can easily obtain

the linearly transformed Bézier curve by transforming only its control points:

$$\mathbf{B}'(t) = \sum_{i=0}^{K} J_{K,i}(t)\mathbf{p}'_i, \quad \text{where } \mathbf{p}'_i = \mathbf{M}\mathbf{p}_i$$

and $\mathbf{M}$ is a linear transformation to approximate a BRDF lobe with a clamped cosine distribution. Now, analogous to the previous method, it is sufficient to consider only the case where the shading can be calculated for Lambertian surfaces. Therefore, we hereinafter denote $\mathbf{B}'$ as $\mathbf{B}$ to simplify the exposition.

2.3 Radiance-Aware Boundary Curve Discretization

We can approximate a boundary shape defined by Bézier curves as a polygon by sampling the curve parameter $t = t_0, t_1, \ldots, t_{N-1}$ and calculating vertex positions $\mathbf{B}(t_0)$, $\mathbf{B}(t_1), \ldots, \mathbf{B}(t_{N-1})$. Therefore, we can calculate the shading with a free-form planar area light by using the previous method [Heitz et al. 2016]. However, discretizing the Bézier curve with a constant number of vertex samples is inefficient because the significance of the illumination obtained by an area light varies with the shading points. In other words, when a shading point is only dimly lit by the light, we can roughly approximate the boundary shape using a small number of vertex samples. On the other hand, we need more vertex samples for a shading point strongly illuminated by the light. To consider the significance of the illumination during the discretization of the boundary curve, we propose an adaptive curve discretization based on the classic Ramer–Douglas–Peucker (RDP) algorithm [Ramer 1972; Douglas and Peucker 1973]. An overview of the algorithm is illustrated in Figure 3.

The original RDP algorithm starts with a line segment that connects two endpoints of an open curve. Then, it samples a vertex on the target curve farthest from the current approximated curve. The process is repeated until the farthest vertex is sufficiently close to the approximation curve. In an analogous manner, our method starts with a line segment but samples a new vertex such that the resulting polygonal shape makes the greatest contribution to the shading point of interest. However, determining the vertex with the largest contribution requires calculating the shading of the resulting shape many times, which is computationally expensive for real-time rendering. Instead, we sample a new vertex at the middle of the interval of curve parameter t. Suppose that the interval of the curve parameter $\mathcal{I}$ is $[t_{\min}, t_{\max}]$. Then, we always subdivide the interval into half by sampling a new parameter $t_{\text{mid}} = (t_{\min} + t_{\max})/2$. As this subdivision may fail to approximate a curve with inflection points, we initially split the default domain $[0, 1]$ of the curve parameter t into N_{div} subsections. For a cubic Bézier curve, $N_{\text{div}} = 4$ is sufficient in practice; we illustrate an example for $N_{\text{div}} = 2$ in Figure 3 for simplicity.

We now denote an interval as $\mathcal{I}_i^{(l)}$, where the superscript l represents the level of detail and the subscript i denotes an index of the interval at each detail level. Starting

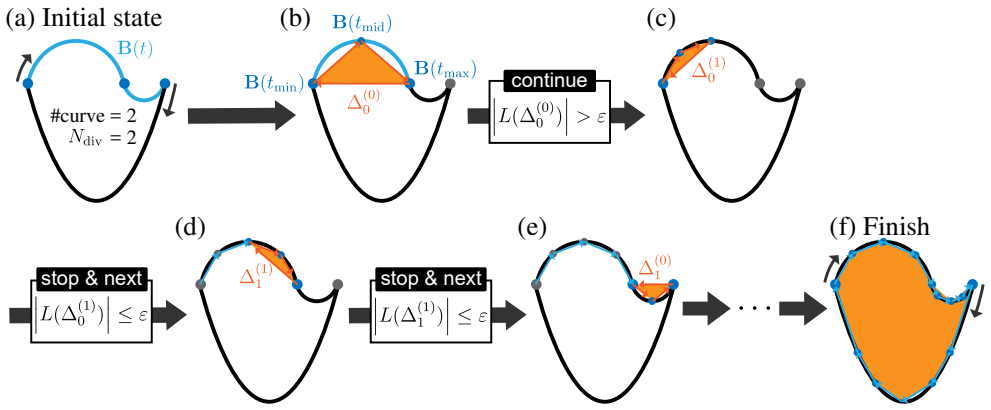


Figure 3. Boundary integration by radiance-aware discretization. A given curve is initially separated into N_{div} intervals. For each interval $\mathcal{I}_i^{(l)}$, the midpoint t_{mid} is inserted to evaluate the radiance from the triangular area $\Delta_i^{(l)}$. The interval is further subdivided when $|L_{\Delta_i^{(l)}}| > \varepsilon$. The subdivision process is performed recursively until all of the intervals have been processed as in (b)–(e). Finally, the total radiance from the area light, as well as a polygonal approximation, is calculated.

from the initial intervals $\mathcal{I}_0^{(0)}, \ldots, \mathcal{I}_{N_{\mathrm{div}}-1}^{(l)}$, we determine whether each interval is further subdivided or not. For a triangle $\Delta_i^{(l)}$ defined by three vertices $\mathbf{B}(t_{\mathrm{min}})$, $\mathbf{B}(t_{\mathrm{mid}})$, and $\mathbf{B}(t_{\mathrm{max}})$ of the interval, we calculate the contribution $L(\Delta_i^{(l)})$ to the shading point using Equation (1). We continue the subdivision only when the absolute value of the contribution is larger than a predefined threshold ε. As Figure 3(e) shows, the value of $L(\Delta_i^{(l)})$ can be negative depending on the order of the vertices. Therefore, we can calculate the contribution from only the interior region of the light without any consideration of the triangles outside the boundary. In addition, we do not need to reevaluate Equation (1) for the resulting polygon because each term summed up in Equation (1) for each edge has already been calculated during the curve discretization.

In this operation, we need to specify a threshold ε to balance computation speed and accuracy. We use different thresholds for specular and diffuse reflections, which we denote ε_{s} and ε_{d}, respectively. Here, we examine how the threshold affects the computation time and rendering quality by changing ε_{s}. Figure 4 shows the change in the rendered images for different ε_{s}, and the corresponding computation time is shown in the bottom right corner of each image. The results in the first three columns obtained using the fixed values of ε_{s} reveal two facts. First, when the threshold is insufficiently small, the reflection of the area light becomes discontinuous on an almost purely reflective surface with $\alpha = 0.01$. Hence, we need to use a small threshold when α is small. Conversely, when α is larger, e.g., $\alpha = 0.25$, the quality of the rendered images does not increase significantly when using a smaller ε_{s}, but the computation time rapidly increases. Therefore, we can use a larger threshold for a rough

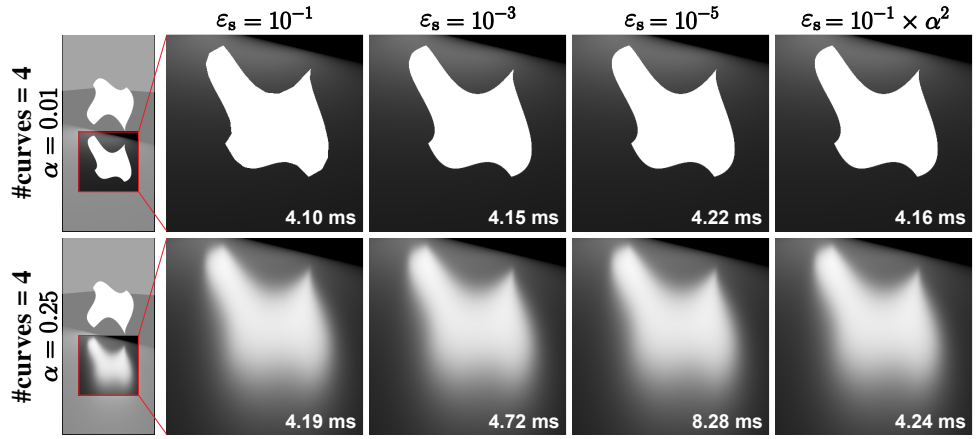


Figure 4. Comparison of lighting on a glossy floor for different radiance thresholds ε_s. The value of ε_s decreases from left to right in the first three columns, and the adaptive threshold is used in the rightmost column.

surface to prioritize computation speed over rendering quality. In summary, we need to use a smaller ε_s when α is small, whereas we can use a larger ε_s when α is large. Following this observation, we empirically set $\varepsilon_s = 0.1 \times \alpha^2$ to balance quality and computation time. For ε_d, we use a sufficiently large value, e.g., $\varepsilon_d = 1000$, because the reflection from a diffuse surface is blurrier than that of a rough specular surface. To more clearly see the effect of the adaptive threshold for ε_s, we visualize the number of polygon vertices (equal to the number of edges) at each pixel given by the RDP algorithm in Figure 5. Compared to a constant $\varepsilon_s = 10^{-5}$, which is sufficiently small

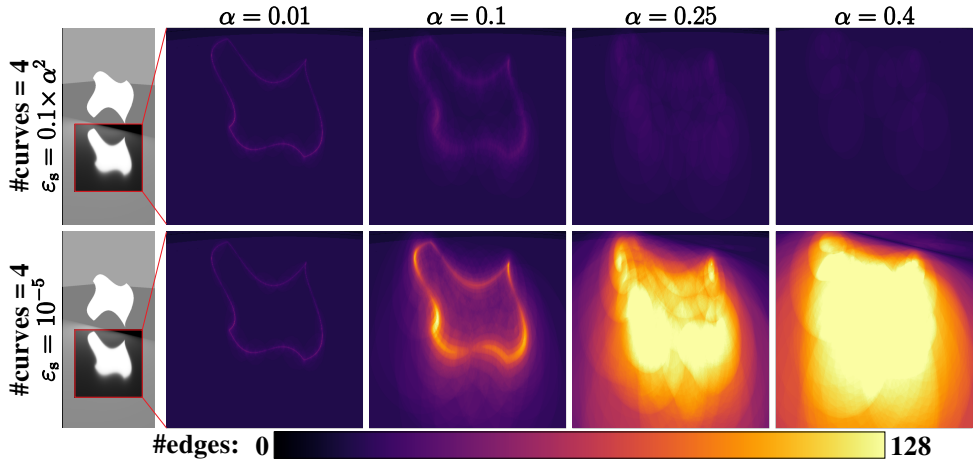


Figure 5. Visualization of the number of vertices for discrete boundary integration at each pixel. The scene is the same as the one in Figure 4. The adaptive threshold $\varepsilon_s = 0.1 \times \alpha^2$ (top row) keeps the number of vertices small even for a rough reflective surface, whereas the constant threshold $\varepsilon = 10^{-5}$ (bottom row) overly subdivides the boundary of the light.

to achieve high rendering quality for arbitrary α, the adaptive threshold conserves the number of vertices as shown in the top row, whereas the rendered results in Figure 4 ($\alpha = 0.1, 0.25$) are approximately identical.

Listing 1 provides GLSL code for the curve approximation, where the function `bezierCurve` uses De Casteljau's algorithm for the cubic Bézier curve, which can be implemented using three `mix` calls. In addition, `integrateEdge` evaluates the contribution from a single edge, for which we use an efficient rational approximation to avoid calculating `acos` in Equation (1), as introduced by Hill and Heitz [2016].

2.4 Algebraic Clipping for Cubic Bézier Curves

As reported by Heitz et al. [2016], the illumination integral in Equation (1) can only be used for a polygonal area light above the local shading plane. The linear transformation to approximate a BRDF lobe as a cosine distribution may potentially move part of a light below the plane. This can also happen to a light with a free-form boundary shape. For the LTC-based shading calculation, the shape of the light is transformed from the local coordinate system centered on a given shading point to another space called the *clamped cosine space* (see Figure 6). Therefore, the intersection points on the boundary Bézier curve are also calculated in the clamped cosine space.

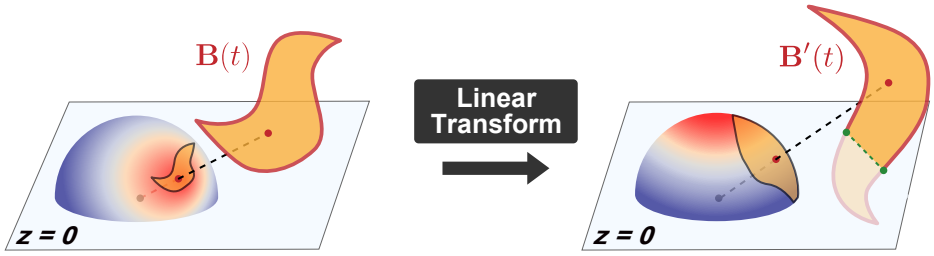
The intersection of the boundary Bézier curve is always tested with the local shading plane in the clamped cosine space, even if the shape of the reflective surface itself is not flat. Without loss of generality, we can assume that the plane is defined by $z = 0$. Because the z-coordinate of the Bézier curve can be represented by a polynomial, we can detect the intersection between a Bézier curve and the plane by solving a simple polynomial equation with regard to the curve parameter t. This polynomial equation is a cubic equation $a\,t^3 + b\,t^2 + c\,t + d = 0$, the coefficients of which are obtained from the z-coordinates of the four control points of the cubic Bézier curve:

$$
\begin{aligned}
a &= -p_{0,z} + 3(p_{1,z} - p_{2,z}) + p_{3,z}, \\
b &= 3(p_{0,z} - 2p_{1,z} + p_{2,z}), \\
c &= 3(-p_{0,z} + p_{1,z}), \\
d &= p_{0,z},
\end{aligned}
$$

where $p_{i,z}$ denotes the z-coordinate of the ith control point. Although solutions for a cubic equation can be obtained via the traditional Cardano formula, we are only interested in real-number solutions within the domain of the curve parameter, i.e., $t \in [0, 1]$. Hence, we can use a more efficient and robust method [Blinn 2007] that initially checks the number of real-number solutions and then calculates each one. Alternatively, we could have used classic Bézier clipping [Sederberg and Nishita 1990] for the clipping operation. However, we found it difficult to achieve sufficient accuracy and speed with Bézier clipping implemented as a shader program.

```
vec3 integrateCurve(Bez bez, float tStart, float tEnd,
                    int Ndiv, float eps) {
    // Initialization
    const float tRange = tEnd - tStart;
    const float interval = tRange / Ndiv;
    for (int i = Ndiv; i > 0; i--) {
        const int j = i - 1;
        const float tMin = interval * j + tStart;
        const float tMax = interval * i + tStart;
        stk[stkIndex++] = vec2(tMin, tMax);
    }

    //*****Integration with triangle dividing*****//
    vec3 res = vec3(0.0);
    while (stkIndex != 0) {
        const vec2 tmp = stk[--stkIndex];
        const float tMin = tmp.x;
        const float tMax = tmp.y;
        const float tMid = 0.5 * (tMin + tMax);

        // Sample Bézier curve
        const vec3 Btmin = bezierCurve(bez, tMin);
        const vec3 Btmid = bezierCurve(bez, tMid);
        const vec3 Btmax = bezierCurve(bez, tMax);

        // Compute radiance of triangular light
        const vec3 I01 = integrateEdge(Btmin, Btmid);
        const vec3 I12 = integrateEdge(Btmid, Btmax);
        const vec3 I20 = integrateEdge(Btmax, Btmin);
        const vec3 I = I01 + I12 + I20;

        // Comparison between radiance and threshold
        if (abs(I.z) >= thres) { // Split into two sections.
            stk[stkIndex++] = vec2(tMid, tMax);
            stk[stkIndex++] = vec2(tMin, tMid);
        }
        else { // Sum up edges.
            res += I01 + I12;
        }
    }
    return res;
}
```

Listing 1. GLSL code for integration with radiance-aware boundary discretization.



(a) Local coordinates (b) Clamped-cosine space

Figure 6. The clipping operation to eliminate regions of the light that are below the local shading plane ($z = 0$). Even when the area light is located above the plane in the local coordinate system, as in (a), it may intersect after the LTC transformation, as in (b).

Listing 2 provides GLSL code for the algebraic clipping. Here, the roots of the cubic equation are obtained via `solveEquation`, which returns the number of roots in `count` and their values in `ts`. Because the equation may also be quadratic or linear depending on the positions of the control points, it can be solved in a simpler way. For more details, we refer the reader to the source code for `solveEquation`, which we provide as supplementary material.

3 Results and Discussion

We implemented our system using C++ with OpenGL. Both of the proposed algorithms are realized in a GLSL fragment shader. For the following experimental results, we calculated both the specular and diffuse reflections using the proposed method. Algebraic clipping is performed twice (once for each reflection), though it can be omitted for the diffuse reflection when the area light does not intersect with the local shading plane. Performance analysis was performed using a PC equipped with an Intel Xeon Gold 5118 2.3 GHz CPU, NVIDIA GeForce GTX 1080 GPU, and 96 GB of system RAM. All results in this article were rendered at a resolution of 1024×1024 pixels, and performance timings are for this resolution even when only a part of the image is shown in some figures.

3.1 General Results

Figure 1 shows the rendering results for area lights with various shapes. The supplementary material includes a movie of the animated results for these scenes, where the light moving up and down is captured by a moving camera. In these scenes, the reflective properties of the glossy floors are described by GGX BRDFs [Walter et al. 2007] with different roughness parameters α. In contrast to the previous method for polygonal area lights [Heitz et al. 2016], our method can handle a significant range of light shapes as long as they are represented by cubic Bézier curves. As shown

```
void algebraicClipping(Bez bez, inout int count,
                       inout float ts[NUM_INTERSECT_MAX]) {
    // Check cases that do not need clipping.
    // NUM_CPS_IN_CURVE = 4, NUM_INTERSECT_MAX = 3
    int numCpsUnder = 0;
    for (int i = 0; i < NUM_CPS_IN_CURVE; i++) {
        numCpsUnder += (bez.cps[i].z < 0.0) ? 1 : 0;
    }
    if (numCpsUnder == 0 || numCpsUnder == NUM_CPS_IN_CURVE) {
        return;
    }

    //*****Algebraic clipping*****//
    const float P0z = bez.cps[0].z;
    const float P1z = bez.cps[1].z;
    const float P2z = bez.cps[2].z;
    const float P3z = bez.cps[3].z;

    // Compute coefficients.
    const float a = -P0z + 3.0 * P1z - 3.0 * P2z + P3z;
    const float b = 3.0 * (P0z - 2.0 * P1z + P2z);
    const float c = 3.0 * (-P0z + P1z);
    const float d = P0z;

    // Solve cubic, quadratic, or linear equation,
    // and store the parameter t in ts.
    solveEquation(a, b, c, d, count, ts);
}
```

Listing 2. GLSL code for the algebraic clipping.

in the figure, our method supports a complicated curvy shape with an internal hole, as in Figure 1(a), and a letter shape whose boundary curve is defined by a combination of Bézier curves and line segments, as in Figure 1(b). Furthermore, the previous method for a textured area light can be naturally extended to our method for free-form shapes, as shown by the result in Figure 1(c). In our implementation, the texture is prefiltered with a Gaussian filter in the same manner as the original LTC-based method [Heitz et al. 2016]. The kernel size is uniform inside the boundary of the light, whereas for exterior regions the size depends on the distance from the boundary. An alternative approach using the *form-factor vector* was also introduced by Hill and Heitz [2016], which avoids the need to prefilter exterior regions by leveraging the fact that the form-factor vector always intersects the polygon provided it is convex. However, the form-factor vector may not intersect the free-form area when its shape is non-convex or has internal holes. Hence, we used the original approach of filtering exterior regions, which does not slow down the main rendering process because all of the filtering operations can be carried out as a preprocess.

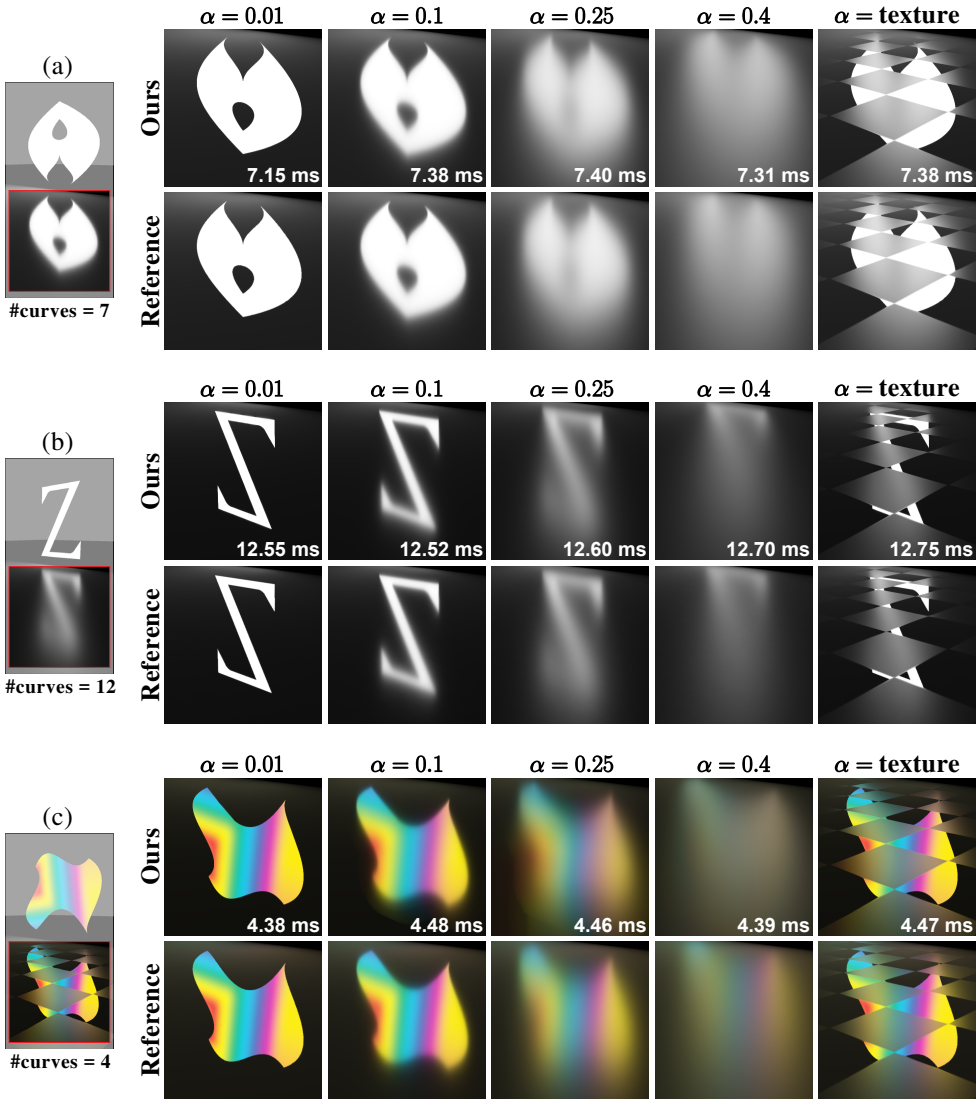


Figure 7. Comparison of our rendering results and path-traced reference images for the scenes shown in Figure 1. The results are compared using different roughness parameters α to demonstrate the robustness of our method with arbitrary roughness.

3.2 Comparison against Reference Images

To demonstrate the accuracy of our method, we compared the results against path-traced reference images rendered using Cycles, a physically based renderer bundled with Blender [2021]. Each image was rendered using 512 path samples per pixel (spp) by enabling GPU hardware acceleration. Figure 7 shows a side-by-side comparison for the scenes in Figure 1. In this comparison, we varied the value of α to demonstrate that our method works for arbitrary α. The computation time for each result is also

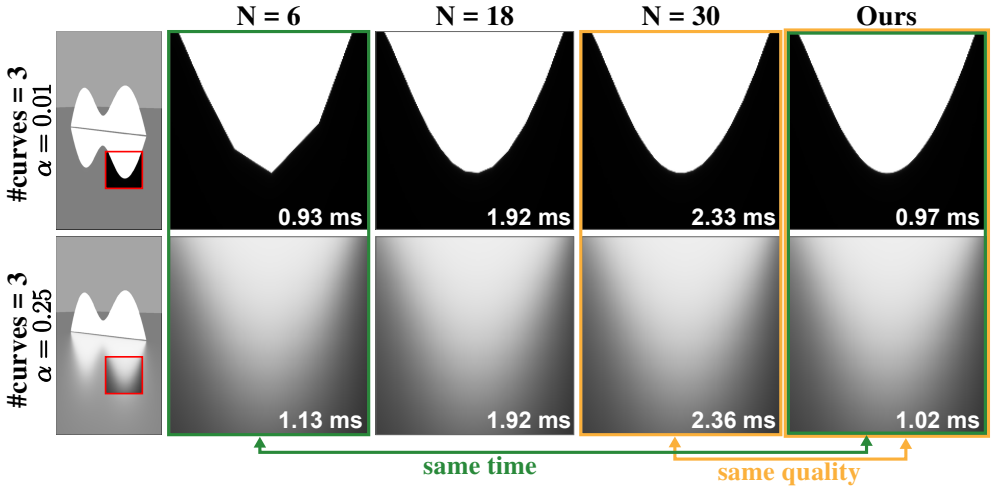


Figure 8. Comparison of the rendered images and computation times of our method using adaptive vertex sampling with the previous method [Heitz et al. 2016] using uniform vertex sampling of the polygon vertices. The presented times are for computing only specular reflection. The green and yellow rectangles highlight pairs of columns with the same computation time and rendering quality, respectively.

shown in the bottom right corner. This comparison demonstrates that our method has succeeded in rendering visually identical results to those obtained from path tracing (with the exception of the textured light, which uses a prefiltered approximation, as stated earlier). Meanwhile, the computation time of our method is significantly lower than the reference path tracer, which took approximately 20 seconds to obtain these results despite hardware acceleration.

3.3 Comparison to Uniform Curve Discretization

We compared our method to the previous method [Heitz et al. 2016] in two different ways. First, we applied the previous method to a polygonal light shape that is produced by simply sampling the vertices uniformly on the boundary curve. In this case, the intersection of the polygon and the local shading plane can be detected by checking the intersection of each edge and the plane one by one. We varied the number of polygon vertices for the previous method as $N = 6, 18, 30$ and compared the rendered images and computation times. In this experiment, the light shape was defined by two curves and one line segment, and we approximated each curve individually with N vertices, while the line segment was used without subdivision. As shown in Figure 8, the boundary shape of the light image on the floor is angular when the number of vertices is insufficient for the previous method. As the number of vertices N increases, the boundary shape approaches the original one, and the computation time increases. When $N = 6$ vertices are sampled, the computation times of the previous

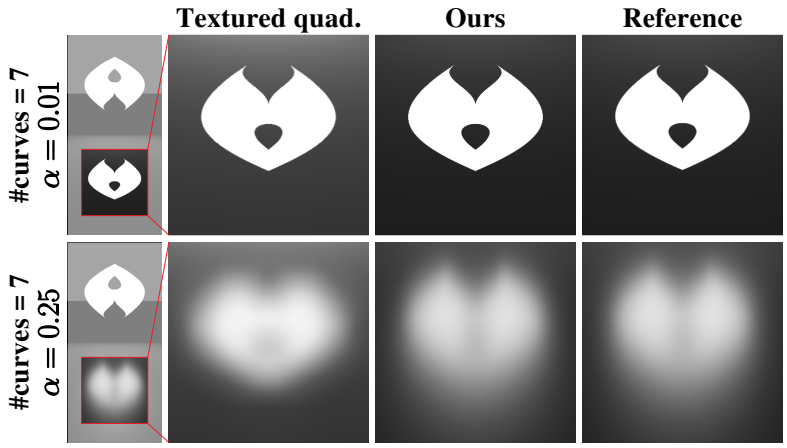


Figure 9. Comparison of the results obtained for a quadrilateral area light masked by a light shape image between our method and the reference. Though this textured light can be rendered using the previous method [Heitz et al. 2016] for polygonal area lights, the blurry appearance over the glossy surface clearly diverges from the reference.

method and ours are almost equal, whereas the approximated boundary shape of the previous method is still noticeably angular. In contrast, using $N = 30$ vertex samples results in more computation time than required by our method, whereas the quality of the results is approximately the same. Thus, the adaptive curve discretization of our method is computationally more efficient than the previous method, even though it requires an additional shading calculation for small triangles during the boundary subdivision and uses more complicated algebraic intersection testing.

3.4 Comparison to Quadrilateral Light Textured by a Light Shape Image

Second, we applied the previous method [Heitz et al. 2016] to a quadrilateral light textured by an image mask representing the light shape, with the results shown in Figure 9. As reported by the previous study, a texture with boundary black regions causes an artifact because the Gaussian filter cannot adequately approximate the real blur on glossy surfaces. This is confirmed by our experiment, where the results for a textured quadrilateral light are different than those of the reference images obtained by physically based rendering, particularly for diffuse reflection and highly glossy specular reflection with $\alpha = 0.25$. In contrast, our method does not need such an image mask to define the light shape, and the results are almost identical to the reference images.

4 Conclusion

In this article, we proposed a real-time shading calculation for area lights with a boundary shape defined by cubic Bézier curves. The proposed method can be easily implemented on top of the previous method for polygonal area lights [Heitz et al. 2016], and its performance is higher than the previous method with uniform discretization. Moreover, our technique for cubic Bézier curves can be naturally applied to other types of polynomial curves such as B-spline curves and rational Bézier curves, as long as their dimension is less than five.[1] We hope that our simple yet powerful approach for free-form planar area lights enhances the quality of experience in real-time graphics applications.

Acknowledgments

This project was jointly supported by JSPS KAKENHI (JP18K18075, JP20H04203, JP17H06101, and JP19H01129), JST ACCEL (JPMJAC1602), JST Mirai Project (JPMJMI19B2), and a Grant-in-Aid from the Waseda Institute of Advanced Science and Engineering.

Index of Supplemental Materials

The supplementary material contains a movie showing animated rendering results and reference comparisons. Source code is available under the MIT license from the authors' GitHub: https://github.com/Paul180297/BezierLightLTC.git.

References

BAUM, D. R., RUSHMEIER, H. E., AND WINGET, J. M. 1989. Improving radiosity solutions through the use of analytically determined form-factors. *ACM SIGGRAPH Computer Graphics 23*, 3, 325–334. URL: https://doi.org/10.1145/74334.74367. 2

BLENDER FOUNDATION, 2021. Blender: The Free and Open 3D Creation Suite (2.93 LTS). URL: https://www.blender.org/. 11

BLINN, J. F. 2007. How to solve a cubic equation, part 5: Back to numerics. *IEEE Computer Graphics and Applications 27*, 3, 78–89. URL: https://doi.org/10.1109/MCG.2007.60. 7

DOUGLAS, D. H., AND PEUCKER, T. K. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization 10*, 2, 112–122. URL: https://doi.org/10.3138/FM57-6770-U75U-7727. 4

HEITZ, E., AND HILL, S. 2017. Real-time line- and disk-light shading. In *ACM SIGGRAPH 2017 Courses*, Physically Based Shading in Theory and Practice, 7:1–7:8. URL: https://doi.org/10.1145/3084873.3084893. 2

[1] Polynomial equations of order five or higher cannot always be solved algebraically.

HEITZ, E., DUPUY, J., HILL, S., AND NEUBELT, D. 2016. Real-time polygonal-light shading with linearly transformed cosines. *ACM Transactions on Graphics 35*, 4, 41:1–41:8. URL: https://doi.org/10.1145/2897824.2925895. 2, 3, 4, 7, 9, 10, 12, 13, 14

HEITZ, E., 2017. Geometric derivation of the irradiance of polygonal lights. Unity Technology Technical Report. URL: https://hal.archives-ouvertes.fr/hal-01458129/. 3

HILL, S., AND HEITZ, E. 2016. Real-time area lighting: A journey from research to production. In *ACM SIGGRAPH 2016 Courses*, Advances in Real-Time Rendering in Games, Part I. URL: https://doi.org/10.1145/2897826.2940291. 7, 10

LAMBERT, J. H. 1760. *Photometria, sive de mensura et gradibus luminus, colorum et umbrae (in German)*. Sumptibus vidvae E. Klett, typis CP Detleffsen. 2, 3

RAMER, U. 1972. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing 1*, 3, 244–256. URL: https://doi.org/10.1016/S0146-664X(72)80017-0. 4

SEDERBERG, T. W., AND NISHITA, T. 1990. Curve intersection using Bézier clipping. *Computer-Aided Design 22*, 9, 538–549. URL: https://doi.org/10.1016/0010-4485(90)90039-F. 7

TORRANCE, K. E., AND SPARROW, E. M. 1967. Theory for off-specular reflection from roughened surfaces. *Journal of Optical Society of America 57*, 9, 1105–1114. URL: https://doi.org/10.1364/JOSA.57.001105. 2

WALTER, B., MARSCHNER, S. R., LI, H., AND TORRANCE, K. E. 2007. Microfacet Models for Refraction through Rough Surfaces. In *Rendering Techniques*, 195–296. URL: https://doi.org/10.2312/EGWR/EGSR07/195-206. 3, 9

Author Contact Information

Takahiro Kuge
Waseda University
3-4-1 Ohkubo, Shinjuku-ku
Tokyo, 169-8555, JAPAN
takahirolabo@gmail.com

Tatsuya Yatagawa
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku
Tokyo, 113-8656, JAPAN
tatsy@acm.org
https://tatsy.github.io/

Shigeo Morishima
Waseda University
3-4-1 Ohkubo, Shinjuku-ku
Tokyo, 169-8555, JAPAN
shigeo@waseda.jp

T. Kuge, T. Yatagawa, and S. Morishima, Real-Time Shading with Free-Form Planar Area Lights Using Linearly Transformed Cosines, *Journal of Computer Graphics Techniques (JCGT)*, vol. 11, no. 1, 1–16, 2022
http://jcgt.org/published/0011/01/01/

Received:	2021-06-23		
Recommended:	2021-07-30	Corresponding Editor:	Stephen Hill
Published:	2022-02-18	Editor-in-Chief:	Marc Olano