Received April 20, 2021, accepted April 27, 2021, date of publication May 28, 2021, date of current version June 11, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3084611

All Frequency Direct Illumination Using Visibility Correspondence Generated With Spherical Voronoi Diagrams

HO CHUN LEUNG, (Member, IEEE), ZHENNI WANG, TZE YUI HO, CHI SING LEUNG, (Senior Member, IEEE), AND ERIC WING MING WONG, (Senior Member, IEEE)
Department of Electrical Engineering, City University of Hong Kong, Hong Kong, SAR, China

Corresponding author: Chi Sing Leung (eeleungc@cityu.edu.hk)

This work was supported by the Research Grant from the Government of the Hong Kong Special Administrative Region under Grant CityU 11259516.

ABSTRACT Vectorized visibility is a powerful visibility representation for rendering direct illumination with all frequency quality. However, the existing rendering method requires the visibility functions to be synthesized three times, followed with three radiance evaluation, for each triangle of the 3D model. We propose an alternative method to enhance vectorized visibility, such that we can directly interpolate the vectorized visibility, and therefore, reduce the computation bottleneck, i.e. the radiance evaluation, to just once. To facilitate the interpolation, we need to generate the correspondence among the three sampled visibility functions for each triangular patch. This paper uses spherical Voronoi diagram as a tool to conduct a preliminary correspondence generation, instead of doing brute force search. Additional treatments are also implemented to ensure that the interpolated visibility functions have smooth transition across the 3D model. With our method, we have more flexibility to manipulate the visibility functions in the favour of rendering speed and render the all frequency direct illumination twice as fast as the previous method.

INDEX TERMS Direct illumination, rendering, visibility interpolation, spherical Voronoi diagram.

I. INTRODUCTION

For direct illumination, rendering the all frequency lighting effect is a challenging field in computer graphic. In order to render the all frequency lighting effect in real time, low computational complexity is required. Some methods have their emphases on reducing the complexity using importance sampling [1]–[5], where the radiance evaluation and the sampling process are the bottleneck to the rendering speed. To further reduce the complexity, pre-computed radiance transfer (PRT) methods are proposed by [6]–[11], which embed the lighting features of an element into a transfer vector based on spherical basis. However, these methods are mainly per vertex based, and the radiance evaluation is conducted at the vertices. For on-screen pixels, their radiance values are interpolated from the neighboring vertices. Hence, for the coarsely tessellated 3D models, aliasing artifacts may appear around the regions with high frequency changes because of the interpolation.

Vectorized visibility [12] is a visibility representation which represents visibility functions in vector form for the direct illumination rendering. Each vectorized visibility is represented by some sequences of position vectors. Given a 3D model (presumably a triangular mesh), the vectorized visibility is pre-computed for each of the vertices. To render the 3D model, for each on-screen pixel, three visibility functions need to be synthesized with the three vectorized visibility from the vertices of a triangle. Then, the radiance evaluation is performed three times, one for each synthesized visibility function.

If the vectorized visibility could be directly interpolated, then we could replace the three vectorized visibility by a single interpolated vectorized visibility and reduce the computation bottleneck, i.e. the radiance evaluation, to just once. However, interpolating vectorized visibility is not feasible for

The associate editor coordinating the review of this manuscript and approving it for publication was Liang-Bi Chen.

the method proposed by [12], since an individual vectorized visibility does not have common metrics with the others.

To facilitate the interpolation, it would be equivalent to subsidize the vectorized visibility with some meaningful correspondence information. This paper presents a method to generate the correspondence for the vectorized visibility. To generate the correspondence, we need to search for a meaningful combination among all permutations of the position vectors from the vectorized visibility, where the number of permutations increases exponentially w.r.t. the number of position vectors. A brute force search is not a practical solution.

Instead of doing a brute force search, we use spherical Voronoi diagram as a tool to conduct a preliminary correspondence generation. The three vectorized visibility of a triangle are treated as three groups of point sites and added to a spherical Voronoi diagram. Examining the Voronoi diagram, we could find that some Voronoi vertices would attach to all three groups simultaneously (see Fig. 2(b) and (c)). Taking into account the fact that each Voronoi vertex corresponds to a closest triple of sites, these Voronoi vertices provide us an ideal shortcut to find the preliminary correspondence information.

From the macroscopic point of view, the preliminary correspondence information only reflects the intra triangle relations. A triangular mesh is almost always intended to be a 3D model with smooth surfaces. Hence, triangles are connected, and the correspondence information of all the triangles is interrelated. In other words, the correspondence information should be consistent for the whole 3D model, and the correspondence generation should be a global event. To achieve these, additional treatments are implemented which propagate the preliminary correspondence information through the neighboring triangles and spread them across the 3D model.

The contributions of this paper are summarized as follows.

- A method to generate the correspondence for the vectorized visibility using spherical Voronoi diagram.
- The vectorized visibility is directly interpolated for the all frequency direct illumination rendering.
- The rendering results show that our method is twice as fast as the vectorized visibility method [12], while providing a bit more accurate rendering results.

The organization of this paper is as follows. Section II reviews the related works. Section III presents the details of our rendering implementation and our method for the correspondence generation. Section IV presents the results and their analysis, and Section V concludes our work.

II. BACKGROUND

The PRT methods are proposed by [6]–[8]. The general idea is to represent the Bidirectional Reflectance Distribution Functions (BRDFs) by transfer vectors or matrices. The authors develop the PRT methods based on spherical harmonics (SH). In [13], the authors encode the all frequency reflectance properties in [9], [14], [15] by using the cubemap-based spherical wavelets. However, visual artifacts may occur in the local illumination when discrete spherical wavelets are used [11]. Some PRT methods based on the spherical radial basis functions (SRBFs) are also proposed for the direct illumination with fixed BRDFs [11], [16], [17]. To relax the fixed BRDF restriction, [10] proposes to edit the BRDF interactively during the rendering process, but this method could only be applied to the static lighting environment and some fixed viewing angles. For the glossy 3D model rendering, [18] embeds the BRDF under various settings into the pre-computed transfer tensor (PTT). It allows the users to modify the BRDF, the view point, and the lighting environment. On the other hand, the requirement of a pre-computed BRDF could also be a problem, and [19] introduces the pre-computed visibility cut method which could do the BRDF editing without requiring a pre-computed BRDF.

In [20], the authors propose to pre-compute the light transport links based on SH for the illumination. With the pre-computed links, both the direct and indirect illumination could be handled by propagating the SH illumination through the light maps. However, bound by the nature of SH basis, the frequency of shadow is blurry. [21] introduces a hybrid global illumination method, which is a balanced solution involving some state of art methods.

A. PRT METHODS AND SUMMED AREA TABLE

The Summed Area Table (SAT) algorithm [22] is an alternative filtering algorithm for texture mapping. Its major advantage is that it can provide an accurate integral for a vast region with just a handful of lookup values. By transforming an image into an SAT, the summed value over an axis-aligned rectangle can be accurately and efficiently obtained with just four SAT lookups. In [23], [24] and [25], the SAT algorithm is used as a shortcut to accelerate the rendering of glossy reflection and global illumination.

Specific to [25], the real live scene from a fixed viewpoint is synthesized under an arbitrary natural illumination, and the rectangular constant basis functions are used for the approximation of the visibility functions and the BRDFs. Unfortunately, the rectangular constant basis does not match the spherical nature of the visibility functions and the BRDFs. Analogous to the rectangular constant basis functions, [26] proposes the spherical piecewise constant basis functions for the all frequency PRT. It is further enhanced by [27], which approximates spherical functions with piecewise constants and the triple product concept.

A two-stage method for determining the set of silhouette edges is proposed by [28]. Given a single shadow and a local visibility function, the physically based soft shadows are generated. Afterwards, a more advanced data structure is proposed by [29] to enhance the mathematical model. These two methods aim for generating the shadows, rather than rendering the illumination from lighting environments. To utilize the idea of silhouette edges for the direct illumination, [30] evaluates the lighting with the 1D SAT model along the zenith lines.

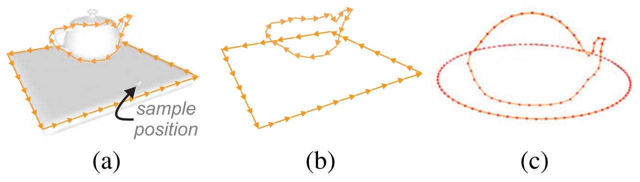


FIGURE 1. The vectorized visibility. (a) is a vectorized visibility drawn on a 3D model sampled at a specific sample position. (b) is the vectorized visibility Q, which is also some sequences of position vectors. Visibility functions can be synthesized from Q by using (3) at arbitrary surface points, regardless where the sample position is. (c) is a visibility function Q_s synthesized at the surface point s. Q_s is a collection of unit vectors sequences, which is also an essential property for our correspondence generation.

To a good portion of the mentioned PRT methods, the qualities of shadows and highlight are per vertex based, e.g. [6]–[9], [13], [14]. In other words, the 3D model ought to be densely tessellated in order to preserve the fidelity of the high frequency contents. The difficulty comes from the binary nature of the visibility functions. Owing to the binary nature, interpolating two visibility functions in the ordinary sense would not resemble the visibility functions in between. To enable the visibility interpolation, the visibility function requires some special treatment. Two of such methods are the spherical signed distance function (SSDF) method [31], [32] and the vectorized visibility method [12].

The SSDFs are proposed by [31]. Each SSDF is a sampled hemi spherical function. It encodes a visibility function by recording the signed angular distance to the closest visibility boundary for each of its hemi spherical samples. By providing each 3D model vertex an SSDF and exploiting the inner product of spherical Gaussian functions, the SSDF method [31] can interpolate the visibility function and efficiently evaluate the radiance value, per pixel. Afterwards, it is extended by [32] to handle multiple specular shadows.

B. VECTORIZED VISIBILITY

Visibility functions are usually represented by using sampled binary functions. However, besides sampled binary functions, a visibility function can also be expressed as some sequences of unit vectors, which form some spherical polygons, as shown in Fig. 1(c). To represent all the visibility functions within a small neighborhood of a 3D model vertex, [12] uses the vectorized visibility, i.e. some sequences of position vectors as shown in Fig. 1(b).

The vectorized visibility is precomputed for every vertex and is expressed as:

$$Q = \left\{ \left\{ \boldsymbol{q}_{1,1}, \cdots, \boldsymbol{q}_{1,n_1} \right\}, \cdots, \left\{ \boldsymbol{q}_{m,1}, \cdots, \boldsymbol{q}_{m,n_m} \right\} \right\}, \quad (1)$$

where Q contains m position vector sequences, $\boldsymbol{q}_{i,j}$ is the j-th position vector of the i-th sequence, and the i-th sequence contains n_i position vectors. For readability, we refer to (1) as:

$$Q = \left\{ \boldsymbol{q}_{ij} \right\}. \quad (2)$$

Given a vectorized visibility Q, we can synthesize the visibility function Q_s for a surface point $\boldsymbol{s}$, which is given by

$$Q_s = \left\{ \text{normalize}(\boldsymbol{q}_{ij} - \boldsymbol{s}) \right\}. \quad (3)$$

The radiance equation for rendering direct illumination in general [33] is given by

$$L = \int_{\Omega} f_r(\boldsymbol{l})\, G(\boldsymbol{l})\, L_e(\boldsymbol{l})\, V(\boldsymbol{l})\, d\boldsymbol{l}, \quad (4)$$

where f_r is the BRDF, L_e is the lighting environment, G is the geometry term, V is the visibility function, $\boldsymbol{l}$ is the lighting direction and Ω is the spherical domain. Note that, depending on the discretization, the solid angle is not necessarily a constant. For instance, if latitude-longitude map is assumed, (4) will have an implicit cosine term for the solid angle. The diffuse component and the specular component are added together to form the final radiance value, i.e.

$$L = L_d + L_s, \quad (5)$$

where L_d is the radiance from the diffuse component, and L_s is the radiance from the specular component.

For the diffuse component, the BRDF f_r is a constant function, and the geometry term G is usually chosen to be $\max(\boldsymbol{n} \cdot \boldsymbol{l}, 0)$. Substituting to (4) with f_r being a constant function and $G = \max(\boldsymbol{n} \cdot \boldsymbol{l}, 0)$, we have

$$L_d = \int_{\Omega} k_d \max(\boldsymbol{n} \cdot \boldsymbol{l}, 0)\, L_e(\boldsymbol{l})\, V(\boldsymbol{l})\, d\boldsymbol{l}, \quad (6)$$

where the diffuse reflectance k_d is a constant, and $\boldsymbol{n}$ is the surface normal.

The visibility function V can be synthesized from a vectorized visibility, and the synthesized visibility function (3) can serve as the domain of the integral. Taken into account these facts, we can replace the integration domain Ω with a synthesized visibility function. Then, (6) can be rewritten as

$$L_d = k_d \int_A \max(\boldsymbol{n} \cdot \boldsymbol{l}, 0) L_e(\boldsymbol{l})\, d\boldsymbol{l}, \quad (7)$$

where A is a synthesized version of the visibility function V synthesized from a vectorized visibility.

Since the illumination from the lower hemisphere has been blocked by the visibility, the max function becomes redundant. Removing the max function, we have

$$L_d = k_d \int_A \boldsymbol{n} \cdot \boldsymbol{l}\, L_e(\boldsymbol{l})\, d\boldsymbol{l}. \quad (8)$$

As $\boldsymbol{n}$ is independent to $\boldsymbol{l}$, we can factorize $\boldsymbol{n}$ out of the integral [30], thus (8) can be rewritten as

$$L_d = k_d\, \boldsymbol{n} \cdot \int_A \boldsymbol{l} L_e(\boldsymbol{l}) d\boldsymbol{l}. \quad (9)$$

$\boldsymbol{l} L_e(\boldsymbol{l})$ is transformed to three SATs, and we refer to these SATs of $\boldsymbol{l} L_e(\boldsymbol{l})$ as ρ.

To compute (9), [12] employs a *non axis-aligned* line segment SAT evaluation, i.e. *vsat*. The integration domain A in (9) is a sequence of unit vectors $\{\boldsymbol{a}_i\}$, each consecutive pair of unit vectors $\{\boldsymbol{a}_i, \boldsymbol{a}_{i+1}\}$ represents a line segment, and we can draw a bounding box for the line segment. Using the SAT lookup directly, we can obtain the integration of the upper and the lower boundaries (say $\boldsymbol{E}$ and $\boldsymbol{F}$). Since the integration of the line segment $\{\boldsymbol{a}_i, \boldsymbol{a}_{i+1}\}$ has to be somewhere in between $\boldsymbol{E}$ and $\boldsymbol{F}$, the *vsat* is defined as

$$\begin{aligned} \boldsymbol{E} &= \mathbf{SAT}(\rho, \phi_{a_{i+1}}, \theta_{a_i}) - \mathbf{SAT}(\rho, \phi_{a_i}, \theta_{a_i}), \\ \boldsymbol{F} &= \mathbf{SAT}(\rho, \phi_{a_{i+1}}, \theta_{a_{i+1}}) - \mathbf{SAT}(\rho, \phi_{a_i}, \theta_{a_{i+1}}), \\ vsat(\rho, \boldsymbol{a}_i, \boldsymbol{a}_{i+1}) &= (1-\beta)\boldsymbol{E} + \beta\boldsymbol{F}, \end{aligned} \tag{10}$$

where $\boldsymbol{a}_i = (\phi_{a_i}, \theta_{a_i})$ and $\boldsymbol{a}_{i+1} = (\phi_{a_{i+1}}, \theta_{a_{i+1}})$ are unit vectors in spherical coordinates, **SAT** is the SAT lookup operation, and β is a scalar interpolation factor.

vsat is indeed a simple operation. It is just utilizing the SAT evaluation from a different perspective. To be specific, although ρ is initially designed for evaluating the integration, we can interpret the evaluation result of ρ, i.e. $\boldsymbol{F} - \boldsymbol{E}$, as a sum of luminance weighted directions. Normalizing $\boldsymbol{F} - \boldsymbol{E}$, we have a weighted average direction which is somewhere inside the bounding box, and the weighted average direction reviews the direction of the dominating light source in the vicinity. Based on the perpendicular distance from the weighted average direction to the line segment and a 1D lighting assumption, we have an interpolation factor that can review the influence of the dominating light source, which is given by

$$\beta = \frac{\int_0^\infty \operatorname{sech}^2(\alpha'(x-x_c))dx}{\int_{-\infty}^\infty \operatorname{sech}^2(\alpha'(x-x_c))dx} = \frac{1}{2} + \frac{1}{2}\tanh(\alpha' x_c), \tag{11}$$

where α' is a constant dependent on environment map resolution, and x_c is the perpendicular distance. α' should roughly correspond to the pixel size of a given environment map, in particular $\alpha' = 5.8$ for $6 \times 256 \times 256$ cubemaps.

By using (10), the integration of each line segment can be evaluated efficiently by fetching just four values from the SAT ρ. Then, $\int_A \boldsymbol{l} L_e(\boldsymbol{l}) d\boldsymbol{l}$ can be approximated by using a series of *vsat*, which is given by

$$\int_A \boldsymbol{l} L_e(\boldsymbol{l}) d\boldsymbol{l} \approx VSAT(\rho, A) = \sum_i vsat(\rho, \boldsymbol{a}_i, \boldsymbol{a}_{i+1}). \tag{12}$$

As a series of *vsat*, (12) inherits the outstanding computational efficiency of the SAT algorithm.

For the specular component, the BRDF f_r and the geometry term G in [12] are chosen to be the Phong lighting function [34], [35] and a constant function 1 respectively. Substituting to (4), we have

$$L_s = \int_\Omega k_s \max(\boldsymbol{r} \cdot \boldsymbol{l}, 0)^\alpha \, L_e(\boldsymbol{l}) \, V(\boldsymbol{l}) \, d\boldsymbol{l}, \tag{13}$$

where k_s is the glossy reflectance, $\boldsymbol{r}$ is the reflected camera direction, and α is the shininess of the surface. Using a computation-friendly version of (13) and following the steps (7)-(9) in an analogous manner, we have

$$L_s = k_s \, \gamma(\boldsymbol{r}, \xi) \int_\Omega \max(\boldsymbol{r} \cdot \boldsymbol{l}, 0)^\alpha \, L_e(\boldsymbol{l}) \, d\boldsymbol{l}, \tag{14}$$

where γ is the intensity ratio of the radiance values with and without the visibility influence. The approximation for γ is given by

$$\gamma \approx \frac{\int_A Cap(\boldsymbol{l}, \boldsymbol{r}, \xi) L_e(\boldsymbol{l}) d\boldsymbol{l}}{\int_W Cap(\boldsymbol{l}, \boldsymbol{r}, \xi) L_e(\boldsymbol{l}) d\boldsymbol{l}}, \tag{15}$$

where W is a spherical circle centered at $\boldsymbol{r}$ with the radius ξ, and

$$Cap(\boldsymbol{l}, \boldsymbol{r}, \xi) = \begin{cases} \boldsymbol{l} \cdot \boldsymbol{r} - cos(\xi), & \text{if } \boldsymbol{l} \cdot \boldsymbol{r} < cos(\xi), \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

(15) can be evaluated using (12) twice, and therefore (14) is inherently computationally efficient.

C. DYNAMIC TESSELLATION SHADING

Rendering direct illumination with vectorized visibility can be done with three kinds of shading, the per vertex shading, the per fragment shading, and the dynamic tessellation shading.

The per vertex shading can provide a fast rendering speed, but its rendering results are potentially plagued with aliasing problems (see Fig. 10(h)). While the rendering results of some triangles are indeed being severely affected though, the majority of the triangles do not share the same problem. For instance, the triangles on the rightmost side of the screen in Fig. 10(h) are practically having the same rendering results as the ground truth (see Fig. 10(b)). To handle the aliasing problem, a simple method is to subdivide all triangles unconditionally, i.e. subdividing the whole 3D model. However, subdividing the whole 3D model is unnecessary, since most of the triangles would not even be affected by the aliasing problem in the first place. The dynamic tessellation shading is a method designed for utilizing this insight.

The dynamic tessellation shading is for boosting the rendering speed while maintaining the rendering quality. The main idea is to subdivide the triangles of the 3D model on demand while performing the computationally more expensive radiance evaluation only at the vertices of the subdivided triangles. The process begins with an estimation to the content frequency of the triangles. It estimates the content frequency indirectly using the vectorized visibility and the lighting environment. For the triangles with higher frequency contents, e.g. shadow boundaries, the triangles get a finer subdivision.

Suppose we have a triangle with vertices $\{t_1, t_2, t_3\}$, and t_1, t_2 and t_3 have the vectorized visibility Q^{t1}, Q^{t2} and Q^{t3} respectively. Then, the radiance values of these vertices under the influence of a lighting environment L can be evaluated using (9) and (14) as mentioned above. This gives us a set of three radiance values $\{I_{Q^{t1}}, I_{Q^{t2}}, I_{Q^{t3}}\}$ for the three vertices. Blurring L using a Phong kernel with $\alpha = 256$, we have a blurry lighting environment L'. Given L', there will be another set of three radiance values $\{I'_{Q^{t1}}, I'_{Q^{t2}}, I'_{Q^{t3}}\}$

The difference Δ between the two sets of radiance can serve as a metric to measure the content frequency of a triangle, which is given by

$$\begin{aligned}\Delta_{t_1} &= \left|I_{Q^{t1}} - I'_{Q^{t1}}\right|,\\ \Delta_{t_2} &= \left|I_{Q^{t2}} - I'_{Q^{t2}}\right|,\\ \Delta_{t_3} &= \left|I_{Q^{t3}} - I'_{Q^{t3}}\right|,\\ \Delta &= max(\Delta_{t_1}, \Delta_{t_2}, \Delta_{t_3}).\end{aligned} \tag{17}$$

Then, the number of subdivisions for the triangle is

$$n = 2^{round(\zeta\Delta)}, \tag{18}$$

where ζ is a parameter to control the tessellation quality.

Note that the radiance differences between the radiance under the influence of a blurred lighting environment w.r.t. that of the original lighting environment will tend to be bigger when there are high frequency contents. This fact enables the above process to provide an estimation of triangle tessellation levels that can adjust itself on demand to reflect the presence of high frequency contents.

D. PER FRAGMENT SHADING

For the per fragment shading, the vectorized visibility is used to calculate the radiance values by using (9) and (14) for each pixel on the rendered image. As mention in Section II, for each triangle, we have three vertices $\{t_1, t_2, t_3\}$ and three vectorized visibility $\{Q^{t1}, Q^{t2}, Q^{t3}\}$. Each pixel gives us a surface point $\boldsymbol{s}$, and three visibility functions $\{Q_s^{t1}, Q_s^{t2}, Q_s^{t3}\}$ are synthesized by using (3). Then, three radiance values $\{I_{Q_s^{t1}}, I_{Q_s^{t2}}, I_{Q_s^{t3}}\}$ are calculated by using (9) and (14). Finally, the radiance value for the surface point $\boldsymbol{s}$ is calculated by

$$I_s = \lambda_1 I_{Q_s^{t1}} + \lambda_2 I_{Q_s^{t2}} + \lambda_3 I_{Q_s^{t3}}, \tag{19}$$

where λ_1, λ_2 and λ_3 are the barycentric coordinates. Q^{t1}, Q^{t2} and Q^{t3} potentially have different resulting radiance values. This makes it necessary to interpolate the three resulting radiance values with (19) to ensure a smooth transition for the rendered image.

E. VORONOI DIAGRAMS

As mentioned, we will be using Voronoi diagram as a tool to conduct our preliminary correspondence generation. The Voronoi diagram [36] itself is a tool for solving computational geometry problems. For instance, it is applied to the page segmentation and path planning in [37], [38]. Given a set of sites in a 2D space, it partitions the space into some convex regions, namely Voronoi cells, and each site is associated with a Voronoi cell. The vertices of Voronoi cells are called Voronoi vectices. The spherical Voronoi diagram [39] is an extension of the Voronoi diagram. Analogously, given a set of sites on a sphere, the spherical Voronoi diagram will partition the spherical surface into some Voronoi cells, where each Voronoi cell is a convex spherical polygon.

F. MONTE CARLO METHODS

Besides the PRT methods, we can also use ray tracing [40], [41] to render direct illumination. Since the introduction of the Turing architecture [42], NVIDIA RTX [43] provides dedicated hardware to accelerate ray tracing, which boosts the performance of any compatible ray tracing implementations dramatically. As an auxiliary to the ray tracing solution, NVIDIA provides also an OptiX AI denoiser [44], [45] to denoise images such that smooth images can be produced with just a few samples per pixel (spp).

Monte Carlo methods, or importance sampling, are essential components to any ray tracing applications. The Monte Carlo methods estimate the integral (4) by using

$$L = \frac{1}{N}\sum_{i=1}^{N}\frac{f(\boldsymbol{l}_i)}{p(\boldsymbol{l}_i)}, \tag{20}$$

where $f = f_r \cdot G \cdot L_e \cdot V$, p is the probability distribution function (PDF), and $\boldsymbol{l}_i$ is the i-th random sample sampled from p. The PDF p can be chosen arbitrarily as long as it is non-zero for the whole spherical surface. A straightforward choice of p is $p \propto 1$, and it will give us a uniform sampling. To generate uniform random samples for spherical surface, the sampling equations will consist of a cosine inverse. However, in general, the rendering results will have lower variance if p better resembles f; ergo, an inevitable conclusion is p would be ideal if $p \propto f$.

We would naturally encounter multiple sources of PDFs during rendering, e.g. the BRDF f_r and the lighting environment L_e. If we choose to draw samples proportional to the BRDF (i.e. $p \propto f_r \cdot G$), and the cumulative distribution function inverse of p is available in closed form (e.g. [33]), the process of drawing samples can be as simple as just evaluating some closed form equations. If we choose to draw samples proportional to the lighting environment (i.e. $p \propto L_e$), we can accelerate the process of drawing samples using the alias method [46], [47]. It offers an efficient sample drawing mechanism, which allows us to draw a sample in $O(1)$ time.

Inheriting the good qualities of individual sources, multiple importance sampling (MIS) [48] draws samples from multiple sources simultaneously to reduce the variance. It combines samples from multiple distributions to construct an estimator by using some heuristics, e.g. the balance heuristic. To further reduce the variance, there are methods to approximate the ideal PDF, e.g. [49], [50] and [51].

The *local environment map sampling* [50] approximates the ideal PDF by utilizing the coherence between the BRDF and the environment map. It treats all required ideal PDFs as a 4D function of incoming directions and normal directions, discrete samples the 4D function, and applies a customized compression to reduce the memory consumption to a practical level. Importance Driven Environment Map Sampling (IDES) [51] also approximates the ideal PDF by utilizing this coherence. However, instead of trying to precompute all possible required PDFs, it uses a path tracing pre-pass to collects the coherence, and its approximated PDF can better reflect the importance from the camera to the environment.

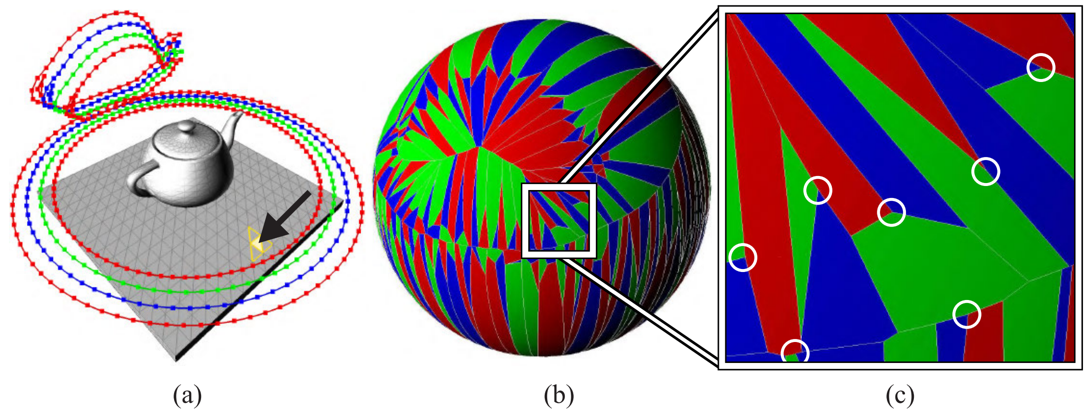


FIGURE 2. The spherical Voronoi diagram and the RGB tripoints of a triangle. (a) shows the three synthesized visibility functions of the indicated triangle where the correspondence information is awaiting to be filled. It is prepared by laying down the unit vectors on behalf of the position vector entities, and it serves as the 3D interpretation for the correspondence information. Using this interpretation, we can visualize each correspondence triple as a group of three line segments connecting red, green and blue dots, and we refer to (a) as a 3D entity layout. (b) is the spherical Voronoi diagram after adding to it all three synthesized visibility functions. As we are interested in the relationship among the three vectorized visibility, we color the Voronoi cells with three different colors, red, green, and blue. Each color represents one of the vectorized visibility. (c) is the blowup of (b) which indicates the RGB tripoints. We can observe that some Voronoi vertices have different colors for all their attached Voronoi cells. We refer to these Voronoi vertices as RGB tripoints. Each of these RGB tripoints offers a correspondence triple, i.e. a mapping of three entities.

III. THE PROPOSED ALGORIHM

Hypothetically, if we could directly interpolate the vectorized visibility, we could use the interpolated vectorized visibility to perform only one radiance evaluation (instead of three) and reduce the computational cost significantly.

To facilitate this insight, we develop a method to generate the correspondence for the vectorized visibility, which enables us to add the vectorized visibility in a meaningful sense. As a vectorized visibility by itself is a sequence of position vectors, it supports scalar multiplicity naturally. With these two properties, the vectorized visibility can be directly interpolated.

Having the option to interpolate the vectorized visibility, we can evaluate the radiance I_s by doing the interpolation before the integration. Therefore, (19) can be rewritten as

$$I_s = I_{\lambda_1 Q_s^{t1} + \lambda_2 Q_s^{t2} + \lambda_3 Q_s^{t3}}. \tag{21}$$

A. THE CORRESPONDENCE GENERATION

To generate the correspondence for the vectorized visibility, we begin with generating a bunch of spherical Voronoi diagrams, one for each 3D model triangle. As mentioned above, each triangle $\{t_1, t_2, t_3\}$ has three vectorized visibility Q^{t_1}, Q^{t_2} and Q^{t_3}, i.e. three sets of position vector sequences. Synthesizing three visibility functions at the triangle center t_c using (3), we have three visibility functions $Q^{t_1}_{t_c}$, $Q^{t_2}_{t_c}$ and $Q^{t_3}_{t_c}$. These three visibility functions are three sets of unit vector sequences (see Fig. 2(a)). Adding the three sets of unit vector sequences to the same spherical Voronoi diagram, we have Fig. 2(b). Since we are interested in the relationship among the three sets of unit vector sequences, the spherical Voronoi diagram is presented in Red-Green-Blue color coding, where each color represents a set.

B. THE CANDIDATES FOR CORRESPONDENCE

In general, each Voronoi vertex has precisely three Voronoi cells attached to it, and therefore, Voronoi vertices are tripoints in general. Looking at the Voronoi vertices in Fig. 2(b), we can observe an interesting phenomenon, i.e. some Voronoi vertices happen to have different colors for all their attached Voronoi cells (see Fig. 2(c)). We refer to these Voronoi vertices as RGB tripoints.

Each of these RGB tripoints offers a mapping of three entities which identifies three specific position vectors in the three involving vectorized visibility. Geometrically, each RGB tripoint gives us a closest triple of sites. These properties make the RGB tripoints the good candidates for the preliminary correspondence generation.

The three visibility functions, though sampled differently, are indeed representing the same visibility function, thus the RGB tripoints which have their sites geometrically closer to each other can better reflect this fact. Therefore, we define also a metric for measuring the quality of the tripoints, namely the tripoint distance, which is given by

$$\begin{aligned} D_{RGB}(\boldsymbol{v}_r, \boldsymbol{v}_g, \boldsymbol{v}_b) &= cos^{-1}(\boldsymbol{v}_r \cdot \boldsymbol{v}_g) + cos^{-1}(\boldsymbol{v}_g \cdot \boldsymbol{v}_b) \\ &\quad + cos^{-1}(\boldsymbol{v}_b \cdot \boldsymbol{v}_r), \end{aligned} \tag{22}$$

where $\{\boldsymbol{v}_r, \boldsymbol{v}_g, \boldsymbol{v}_b\}$ are the unit vectors from the three adjacent sites.

C. SELECTING TRIPLES FOR CORRESPONDENCE

As mentioned, each RGB tripoint offers a mapping of three entities which identifies three specific position vectors in the three involving vectorized visibility. We call this mapping *a correspondence triple*, and we will refer to the *correspondence triples* as *triples* for the rest of the discussion. A triple can be interpreted from the perspective of the sequence

Algorithm 1 Identify the Occurrence of the Cross-Link Situation

```
Require: The selected triple $t_s$; the targeted triple set $T = \{t_1, t_2, \dots, t_n\}$, where n is the size of the set;
Ensure: true or false
 1: ind($t$,$v$) = get the position vector index of $t$ in visibility $v$
 2: if $n >= 2$ then
 3:     $ai$ = ind($t_s$,a);
 4:     $bi$ = ind($t_s$,b);
 5:     j=0;
 6:     for $j < n$ do
 7:         $aj_{upper}$ = ind($t_j$,a);
 8:         $aj_{lower}$ = ind($t_{((j+1)\ mod\ n)}$,a);
 9:         if $(aj_{upper} < aj_{lower} \cap aj_{upper} < ai \cap ai < aj_{lower}) \wedge (aj_{upper} > aj_{lower} \cap (ai < aj_{lower} \wedge aj_{upper} < ai))$ then
10:             $bj_{upper}$ = ind($t_j$,b);
11:             $bj_{lower}$ = ind($t_{(j+1)\ mod\ n)}$,b);
12:             if $(bj_{upper} < bj_{lower} \cap bj_{upper} < bi \cap bi < bj_{lower}) \wedge (bj_{upper} > bj_{lower} \cap (bi < bj_{lower} \wedge bj_{upper} < bi))$ then
13:                 return true;
14:             else
15:                 return false;
16:             end if
17:         end if
18:     end for
19: end if
20: return true;
```

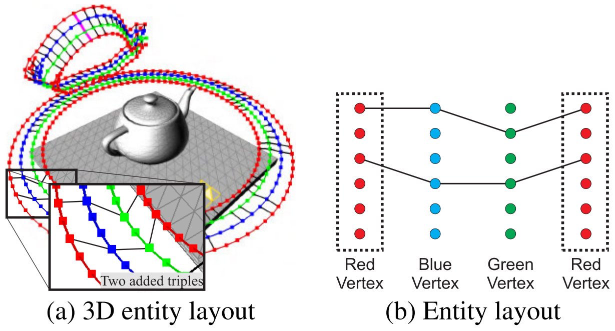


FIGURE 3. The entity layout. (a) is the 3D entity layout of a selected triangle after some iterations during the selection process, and some triples have been added to the layout in the process. The blowup in (a) highlights two groups of three line segments connecting red, green and blue dots, which indicates two added triples. (b) is the entity layout of the blowup in (a) presented in 2D, which is prepared by laying down the triples and the position vector entities according to the sequence ordering. Each line is a triple, each dot is a position vector entity, and each column of dots represents a position vector sequence. By stripping off the 3D info, the entity layout in 2D better, and more directly, indicates the behaviour of the triples.

ordering of the involving vectorized visibility. Laying down the triples and the position vector entities according to the sequence ordering, we have an entity layout of a triangle analogous to Fig. 3.

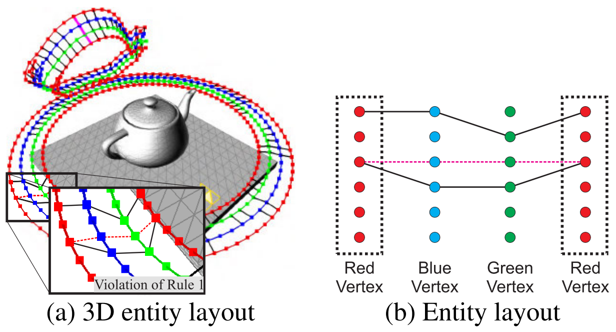


FIGURE 4. The violation of Rule 1. The red triple, indicating a nominated triple, goes through a position vector entity of the black triple below. In such case, we reject the nominated triple to reserve the flexibility of using partially overlapped triples for the lateral stages of the correspondence generation.

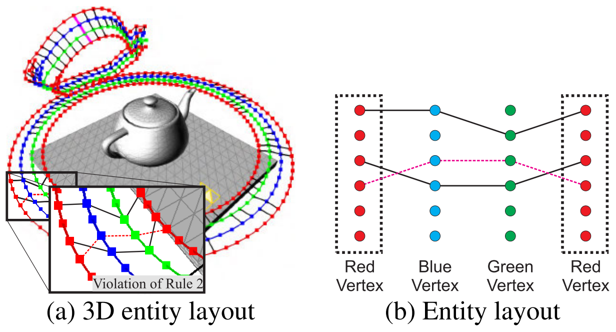


FIGURE 5. The violation of Rule 2. The red triple goes across a triple that is already in the entity layout. In such case, we say the red triple has a cross-link and reject the triple. This can ensure the ordering of the output triple lists being consistent.

To generate the correspondence, we will collect all RGB tripoints, record their triples, and sort these triples in an ascending order according to their tripoint distance values. Each triangle has a sorted list. A 3D model has many triangles, so a 3D model has many of such sorted lists. For each triangle, we will select triples from its sorted list, and record them to a new triple list, namely the output triple list. After selecting and adding triples to triangles until no more triples can be added, then completing partial triples until no more partial triples can be completed (see Section III-D), and then completing all the remaining partial triples with the relaxed constraints (also in Section III-D), the output triple lists of all the triangles will be the final output of the correspondence generation.

The selection process is an iterative process. First, we randomly select a triangle. The selected triangle will nominate and remove its best triple from its sorted list. The nominated triple may be rejected according to the two following rules. The two rules are

1) If the nominated triple goes through any position vector entity that is already in the output triple list (see Fig. 4), the nominated triple is rejected.
2) Laying down both the position vector entities and the triples already in the output triple list, we have the entity layout of the selected triangle (see Fig. 3). When the nominated triple goes across any triple on the entity

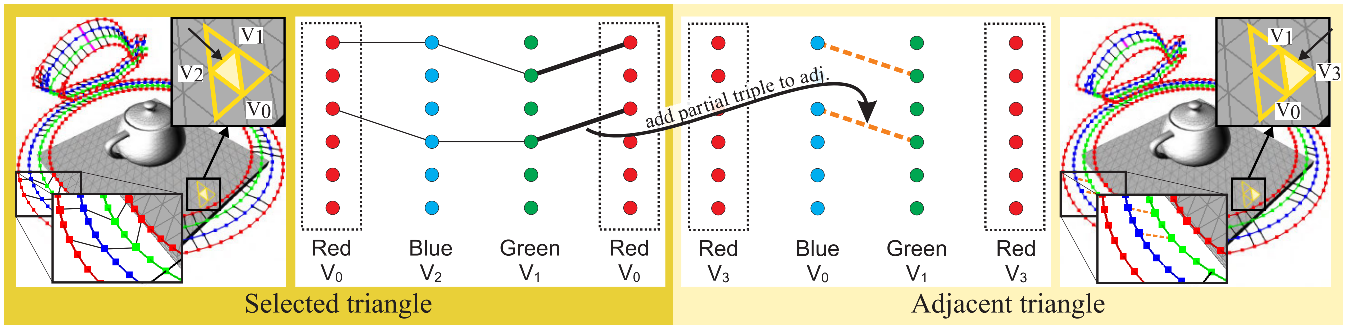


FIGURE 6. Adding partial triples to adjacent triangles. When we add a triple to the selected triangle, we add also a partial triple to each of its adjacent triangles. The shared triples are to address the fact that the triangles are intended to represent some smooth surfaces.

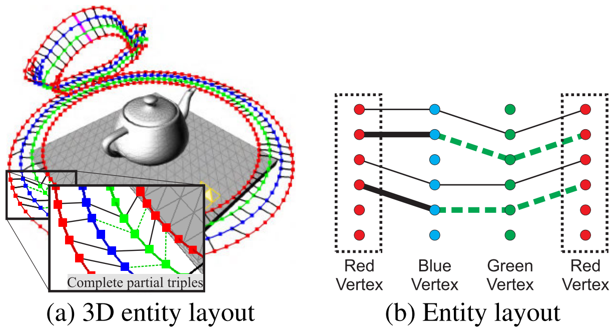


FIGURE 7. Completing the partial triples. Each partial triple is a triple with just two position vector entities. We assign an extra entity to make it a completed triple. In the figure, the two black bold line segments are the partial triples, where the green dashed line segments connect the partial triples through the extra entities.

layout, we say the nominated triple has a cross-link (see Fig. 5), and the nominated triple is rejected.

The reasons behind Rule 1 and Rule 2 are not quite the same. Rule 1 is to reserve the flexibility of using partially overlapped triples for the lateral stages of the correspondence generation. Rule 2, on the other hand, is to avoid an obviously avoidable rendering artifact. Suppose the output triple list does have a cross-link as shown in Fig. 5. If we look at the ordering of the triples from the perspective of the red dots topside downwards, the red link is in the third place. On the other hand, from the perspective of the green dots, the red link is in the second place.

This inconsistent ordering will almost certainly cause the interpolated visibility functions to become geometrically impossible and results in some rendering artifacts. Rule 2 can ensure the ordering being consistent, thus avoiding the rendering artifacts. Algorithm 1 summarizes the steps to identify the occurrence of a cross-link.

D. PROPAGATING THE PARTIAL TRIPLES

A triangular mesh is almost always intended to be a 3D model with smooth surfaces. As a result, when we add a triple to the selected triangle, part of the triple should be shared by the adjacent triangles. To address this, we need to add a partial triple to each of the adjacent triangles. In other words, each triple is bundled with three partial triples. Fig. 6 illustrates this process, where the added triple on the selected triangle sends a partial triple to the adjacent. Rule 1 and Rule 2 are also applied to the partial triples.

If any partial triples violates the rules, the nominated triple and all its partial triples are rejected. Otherwise, they are added to the output triple list of the selected triangle and its adjacent triangles. This selection process is repeated until no more triples can be added to the output triple lists.

Each partial triple is a triple with just two position vector entities. We assign an extra entity to make it a completed triple. When we assign the extra entity, we still need to maintain the consistency of the output triple list ordering. This requirement limits the choices of the extra entity to the entities in between the immediate neighbours of the partial triple. Fig. 7 illustrates the assignment, where an extra entity is chosen in between the two immediate neighbours of the black bold partial triple.

Analogous to adding triple to a triangle, for each completed partial triple, we need to add two new partial triples to the adjacent triangles. In other words, the partial triples propagate through the neighbouring triangles and spread across the 3D model. During the propagation, Rule 1 and Rule 2 are also applied. Fig. 8 illustrates a partial triple added to an adjacent triangle because of a completed partial triple. We repeatedly examine and try to complete individual partial triple, including the newly generated partial triples during the process, until all partial triples are examined.

After the above processes, most partial triples should have been completed, and most position vector entities should have been visited by a triple. For the remaining, we ignore Rule 1 and stop adding partial triples to the adjacent triangles. By relaxing these constraints, we can add new triples such that all the remaining position vector entities are visited by at least one triple, and we can complete all the remaining partial triples. Fig. 9 illustrates an output triple list filled with the relaxed constraints. Filling output triple lists with the relaxed constraints can create seams in the rendering results along the common edges (the edges shared by two triangles). However, these seams are unlikely to be picked up visually due to the shared partial triples scattered all over the 3D model.

IV. RESULTS AND ANALYSIS

To evaluate our method, there are quite a few aspects that can only be examined in detail by using the actual rendering

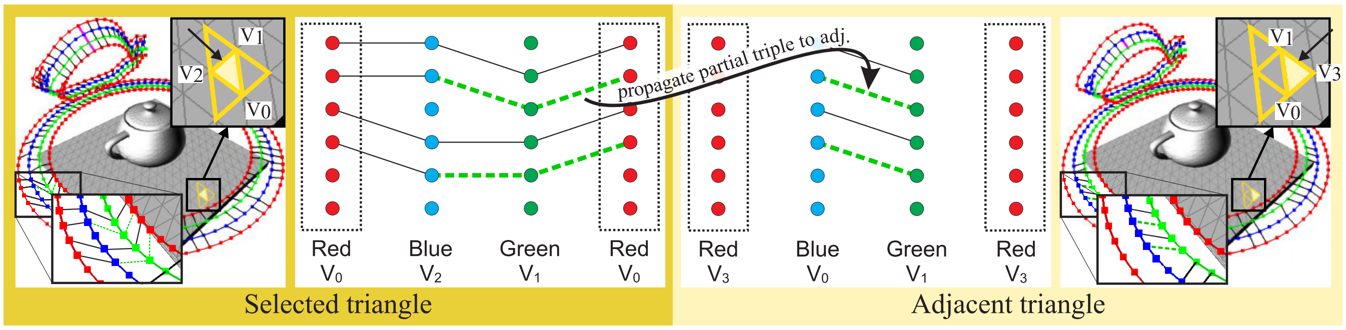


FIGURE 8. The propagation of partial triples. Analogous to adding triple to a triangle, for each completed partial triple, we need to add two new partial triples to the adjacent triangles. This process will propagate the partial triples through the neighbouring triangles and spread them across the 3D model.

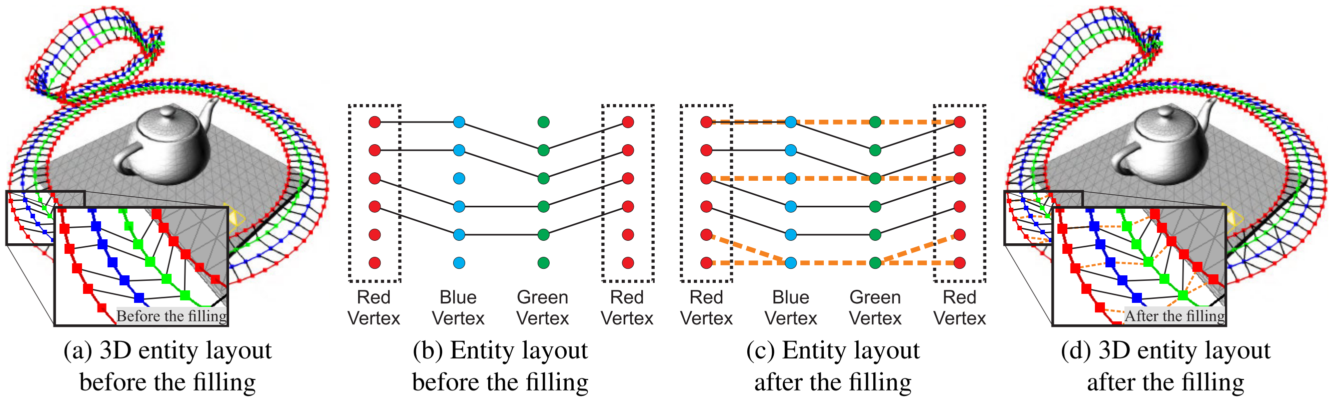

Before the filling
Red Vertex
Blue Vertex
Green Vertex
Red Vertex
Red Vertex
Blue Vertex
Green Vertex
Red Vertex
After the filling
(a) 3D entity layout before the filling
(b) Entity layout before the filling
(c) Entity layout after the filling
(d) 3D entity layout after the filling

FIGURE 9. The output triple list filled with the relaxed constraints. For the remaining partial triples and position vector entities, we ignore Rule 1 and stop adding partial triples to the adjacent triangles. Relaxing these constraints, we can add new triples to complete any output triple list.

TABLE 1. The general information of 3D models. These 3D models are all coarsely tessellated.

	No. of vertices	No. of triangles	Vectorized vis. data size	Output triple list data size	Vectorized vis. generation time	Correspondence generation time
Avocado	484	882	0.72 MB	4.03 MB	11.9 s	35.6 s
Duck	2350	4692	3.66 MB	21.15 MB	42.5 s	62.3 s
Teapot	2024	3706	3.04 MB	17.07 MB	46.3 s	41.9 s

application. The rendering application that we used is the direct illumination rendering. The lighting environment is St. Peter church from [52], which is a cubemap of resolution of $6 \times 256 \times 256$. We convert the lighting environment to latitude-longitude format, and obtain a resolution 1024×512 texture. The performance is evaluated by using a PC equipped with an Intel(R) Core(TM) i5-8400 CPU, 16GB RAM and a NVIDIA GeForce RTX 2080 Ti GPU.

Table 1 summarizes some general information of the 3D models, Avocado, Duck and Teapot. These 3D models are all coarsely tessellated. With respect to the other PRT methods which could take hours to pre-compute, the pre-computation time of our method is relatively short, which is around a minute. For instance, the pre-computation time for the 3D model Avocado, which has 484 vertices, is less than 50 seconds. The pre-computation time depends on the number of vertices, and we can observe that the pre-computation time is not growing exponentially along with the number of vertices. For instance, w.r.t the 3D model Avocado, the 3D model Duck has 4x vertices but just a double pre-computation time.

As mentioned in Section II, direct illumination rendering can be done with two kinds of shading, one is the per fragment shading, and the other is the dynamic tessellation shading. Both kinds of shading are used for the performance evaluation. We render our ground truth images using a brute force integration, where the shadow volume algorithm [28] is set in place to generate the visibility for the ground truth images.

Our rendering implementation inherits most of the implementation from the previous method, i.e. the vectorized visibility method [12]. The major difference is that our method uses (21) to calculate the radiance values while the previous method uses (19). This is equivalent to interpolating the vectorized visibility directly and performing the radiance evaluation once, instead of synthesizing three visibility functions and performing the radiance evaluation three times.

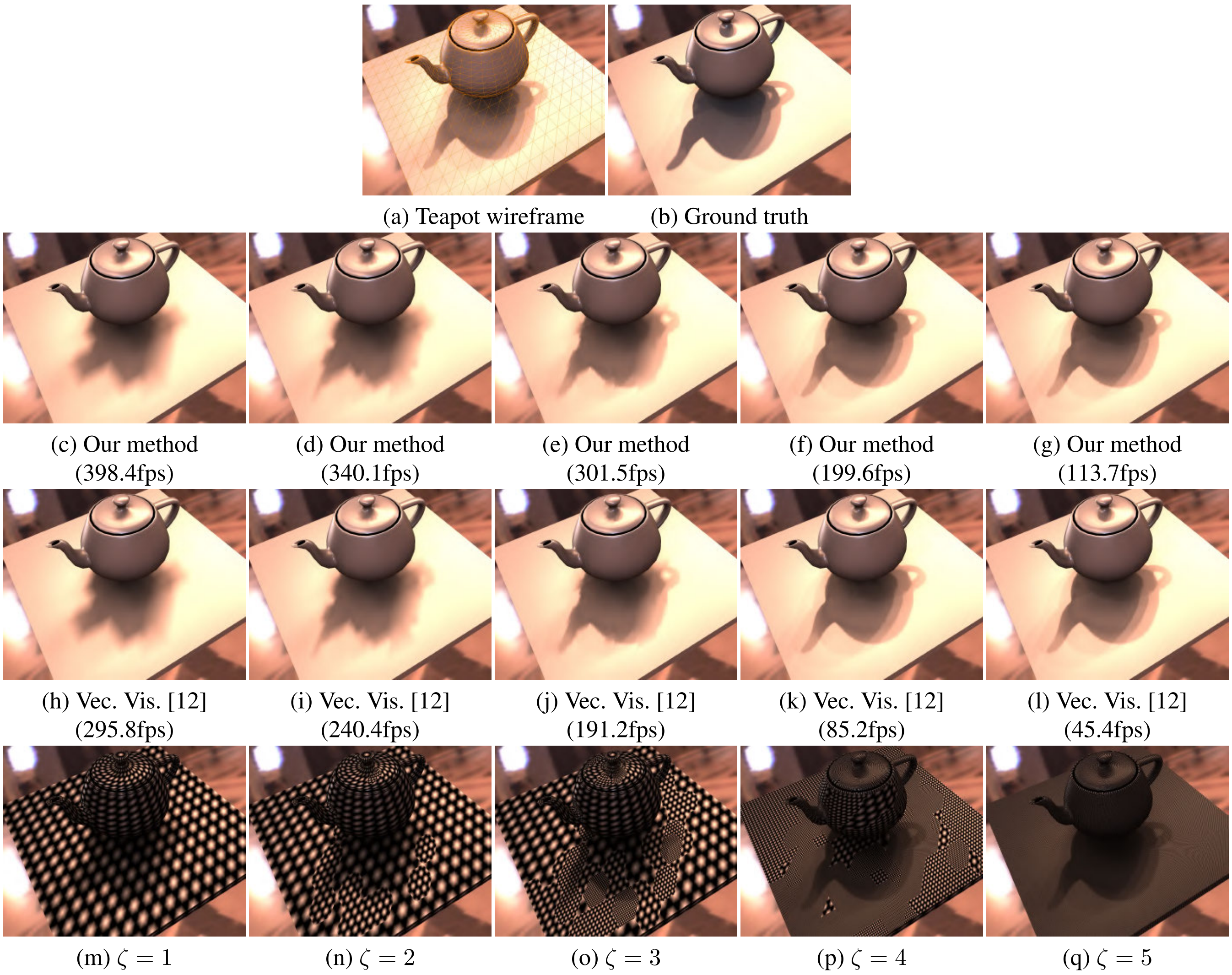

(a) Teapot wireframe
(b) Ground truth
(c) Our method (398.4fps)
(d) Our method (340.1fps)
(e) Our method (301.5fps)
(f) Our method (199.6fps)
(g) Our method (113.7fps)
(h) Vec. Vis. [12] (295.8fps)
(i) Vec. Vis. [12] (240.4fps)
(j) Vec. Vis. [12] (191.2fps)
(k) Vec. Vis. [12] (85.2fps)
(l) Vec. Vis. [12] (45.4fps)
(m) $\zeta = 1$
(n) $\zeta = 2$
(o) $\zeta = 3$
(p) $\zeta = 4$
(q) $\zeta = 5$

FIGURE 10. **The rendering results with the dynamic tessellation shading. The tessellation parameter ζ varies. Our method has a similar rendering quality w.r.t. Vec. Vis. [12], while providing a significant improvement to the rendering speed.**

The rendering results with the dynamic tessellation shading using our method and the previous method, along with the ground truth, are shown in Fig. 10. The tessellation level varies dynamically depending on the content frequency and the tessellation parameter ζ. The rendering quality of both methods improves gradually upon ζ increases. Fig. 10(c) to Fig. 10(g) are the rendering results of our method, and Fig. 10(h) to Fig. 10(l) are the rendering results of the previous method. As shown in the figures, given the same ζ, the rendering quality is visually similar.

We can also observe the similar rendering quality in terms of PSNR. Fig. 11 shows the PSNR values of our method and the previous method w.r.t. the ground truth, where the curves share the same trend. The PSNR of both methods increases gradually upon ζ increases and saturates to a similar PSNR value. Note that although the rendering quality is similar, ours is subtly better when the tessellation quality is high enough. For instance, when $\zeta = 4$, the PSNR of our method is 29.58 dB. It is 0.11 dB higher than the PSNR 29.47 dB of the previous method.

The subtly better rendering quality is easier to observe by using the per fragment shading. Fig. 12 shows a pair of per fragment rendering results that we zoom in to emphasize the shadow boundaries. The differences come from the fact that we are directly interpolating the vectorized visibility before the radiance evaluation using (21), instead of interpolating the radiance values using (19), so we have the more accurate visibility functions and, therefore, the subtly better rendering quality.

A. FRAME RATE COMPARISONS

Although just having a similar rendering quality to the previous method, our method provides a significant improvement in terms of rendering speed, which is also a consequence of directly interpolating the vectorized visibility before the radiance evaluation. As mentioned in Section I, our method can reduce the number of radiance evaluation per shading pixel (or per shading vertex) to one-third of that required by the previous method. However, in practice, a 3 times as fast rendering speed is unlikely due to the overhead.

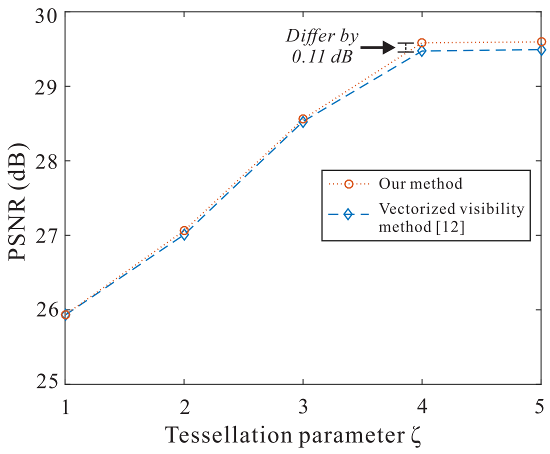


FIGURE 11. The PSNR for our method and the previous method [12] given different tessellation parameter ζ. As mentioned, our method has a similar rendering quality w.r.t. the previous method [12]. The PSNR curves confirm this visual observation. Although we describe the rendering quality being similar, ours is indeed subtly better, specifically 0.11 dB higher in terms of PSNR when the tessellation quality is high enough.

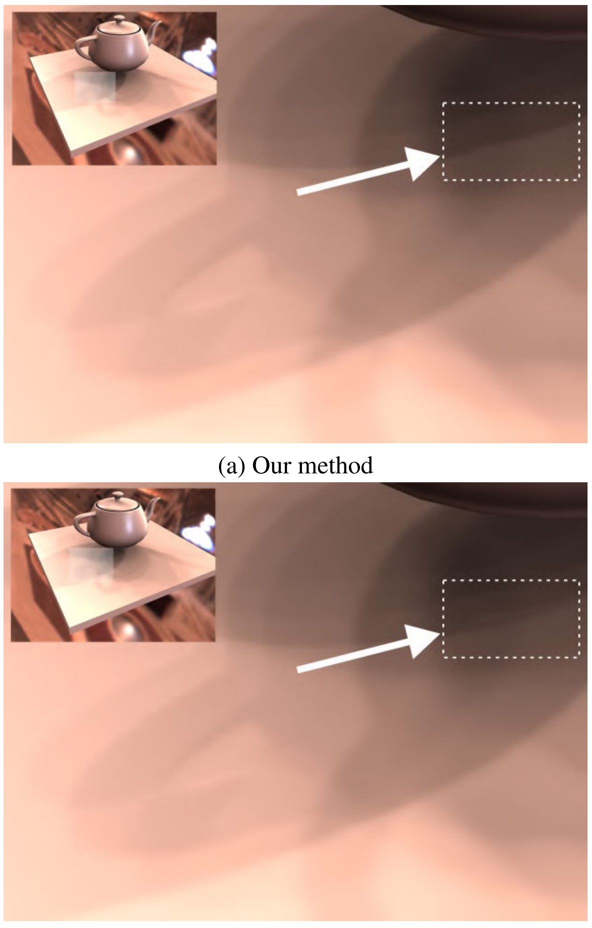


FIGURE 12. The rendering results with the per fragment shading. To visually appreciate the subtly better rendering quality, we would have to switch to the per fragment shading and zoom-in to the shadow boundaries.

Fig. 13 shows the frame rate of our method and the previous method for the dynamic tessellation shading. We can see in

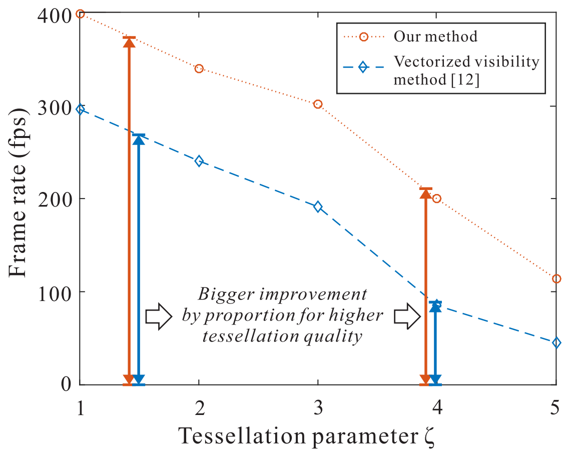


FIGURE 13. The frame rate for our method and the previous method [12] given different tessellation parameter ζ. Our method is consistently faster, and the frame rate gains of our method are proportionally bigger for the higher tessellation quality. Due to the influence of the overhead, a 2x times as fast rendering speed is probably the best that we can get from our method.

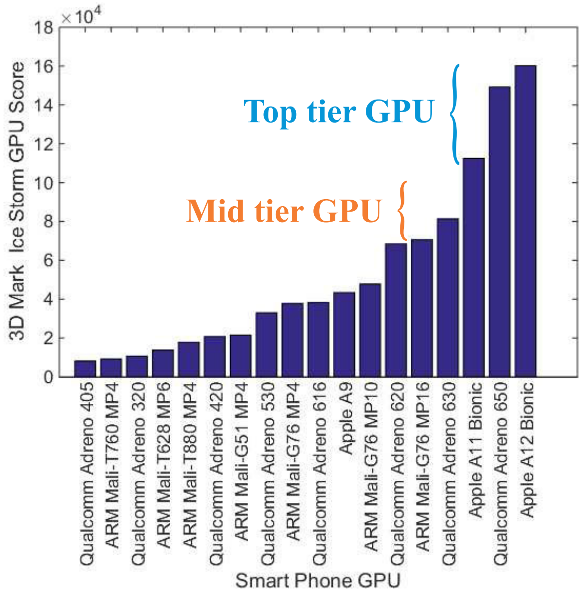


FIGURE 14. The 3D Mark Ice Storm GPU score for smart phone GPUs from [53]. We estimate that only the top tier GPUs can fulfill the 40 to 60 fps expectation using the previous method [12]. On the contrary, by using our method, the computational power requirement of the GPU can be reduced to the mid-tier GPUs. In other words, our method could potentially reduce the computational power requirement without damaging the rendering quality.

the figure that our method is consistently faster. Not only do our method faster, the frame rate gains of our method are also proportionally bigger for the higher tessellation quality. When $\zeta = 1$, the frame rate of our method is 398.4 fps, and the previous method is 295.8 fps. It gives us a frame rate ratio of 1.35. When the tessellation quality increase to $\zeta = 4$, the frame rate ratio increases to 2.3. If the tessellation quality further increases, the frame rate ratio will get closer to the upper bound 3. Due to the influence of the overhead, a 2x times as fast rendering speed is probably the best that we can get from our method.

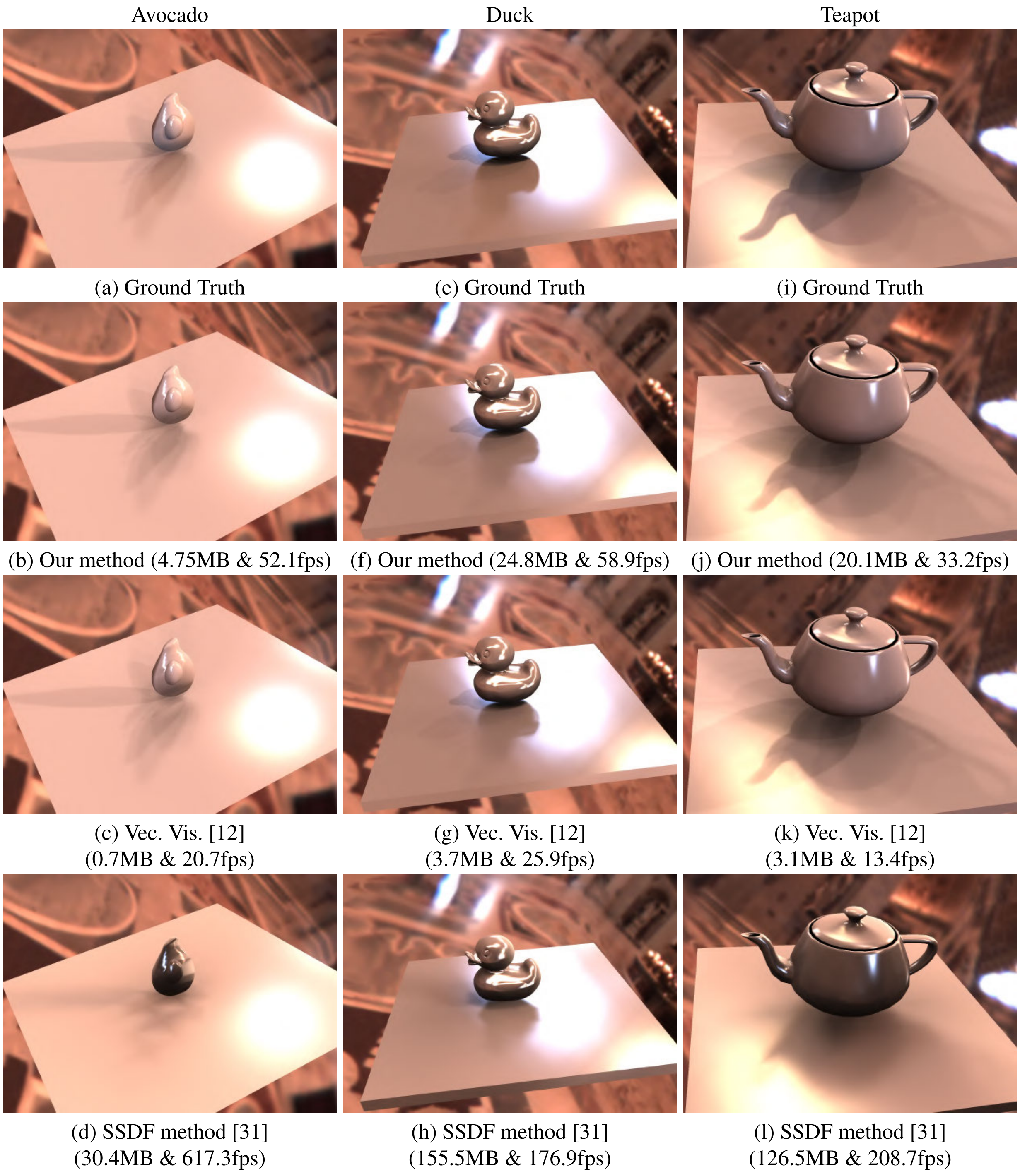

Avocado
Duck
Teapot
(a) Ground Truth
(e) Ground Truth
(i) Ground Truth
(b) Our method (4.75MB & 52.1fps)
(f) Our method (24.8MB & 58.9fps)
(j) Our method (20.1MB & 33.2fps)
(c) Vec. Vis. [12] (0.7MB & 20.7fps)
(g) Vec. Vis. [12] (3.7MB & 25.9fps)
(k) Vec. Vis. [12] (3.1MB & 13.4fps)
(d) SSDF method [31] (30.4MB & 617.3fps)
(h) SSDF method [31] (155.5MB & 176.9fps)
(l) SSDF method [31] (126.5MB & 208.7fps)

FIGURE 15. The rendering results of our method, Vec. Vis. [12] and the SSDF method [31] with the per fragment shading.

Our method provides a 2x as fast rendering speed with a similar rendering quality (indeed, subtly better). For the devices with less computational power, e.g. the mobile devices, this property can be an important factor. Limited by the devices that we have accessed to, we can only provide some weaker evidence to support this claim. We obtain the 3DMark Ice Storm GPU scores from the website [53] (see Fig. 14) and use these GPU scores as our reference to the computational power of the mobile devices.

Consider the 521k GPU score of RTX 2080 Ti, the frame rate reported in Fig. 13 and a target frame rate of 40 to 60 fps. We can estimate that, using the previous method and tessellation quality $\zeta = 3$, only the top tier mobile devices, which have GPU scores ranging from 109k to 164k, can fulfill the 40 to 60 fps expectation. On the contrary, by using our method, the computational power requirement of the mobile devices can be reduced to the mid-tier mobile devices, which have GPU scores ranging from 69k to 104k. In other words,

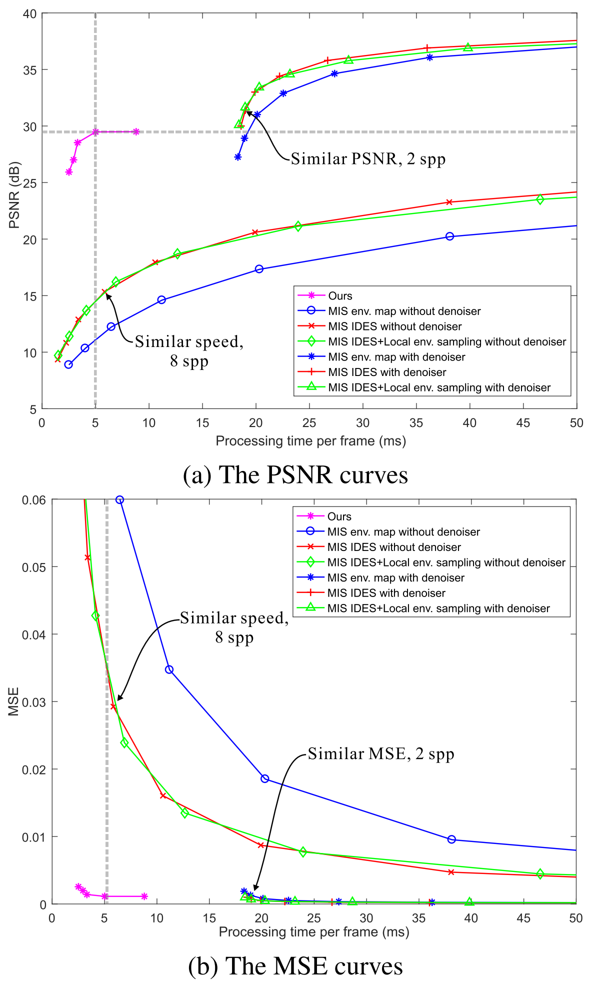


FIGURE 16. The PSNR and MSE of our method and the six Monte Carlo comparison targets. As expected, the PSNR of MIS IDES is consistently higher than that of directly sampling the environment map by a significant margin, 3.4 dB to be specific. Applying the denoiser to MIS IDES further increases the PSNR roughly by 15 dB given the same amount of processing time. On the other hand, when we have a tight window of processing time, e.g. 5 ms per frame, our method provides a margin of 14.24 dB PSNR improvement w.r.t. MIS IDES. If similar PSNR is assumed, it provides a 4 times as fast rendering speed w.r.t. MIS IDES with denoising. The corresponding MSE curves (b) are also provided for reference.

our method could potentially reduce the computational power requirement without damaging the rendering quality.

B. MEMORY CONSUMPTION

An individual position vector $\boldsymbol{q}_{ij}$ in a vectorized visibility $Q = \{\boldsymbol{q}_{ij}\}$ can be uniquely identified by using a pair of indices (i, j). Three position vector entities make a triple, so a triple has six indices. Each triangle has an output triple list, so the memory size in byte for a triangle is $6 \times \text{sizeof}(int) \times$ *number of triples* bytes. The sum of these memory sizes gives us the total memory size for the output triple lists of a 3D model.

Table 1 summarizes also the memory sizes of the output triple lists. The memory sizes of the output triple lists are ranging from 4.03 MB to 21.15 MB, while the memory sizes of the initial data, i.e. the vectorized visibility, are ranging from 0.72 MB to 3.66 MB. Comparing to the initial data, we can see that the memory sizes of the output triple lists are larger. This is because we provide each triangle an individual output triple list. As each output triple list is a many to many mapping, the number of triples per triangle will be larger than the number of involving position vectors. However, the memory requirement is still within a comfortable range to the nowadays mobile devices.

C. COMPARISONS TO PRT METHODS

We also compare the direct illumination rendering results using the per fragment shading, and the rendering results are shown in Fig. 15. There are just a handful of rendering methods that can synthesize visibility functions with per vertex visibility samples for the per fragment shading. Two of such methods are the vectorized visibility method and the SSDF methods.

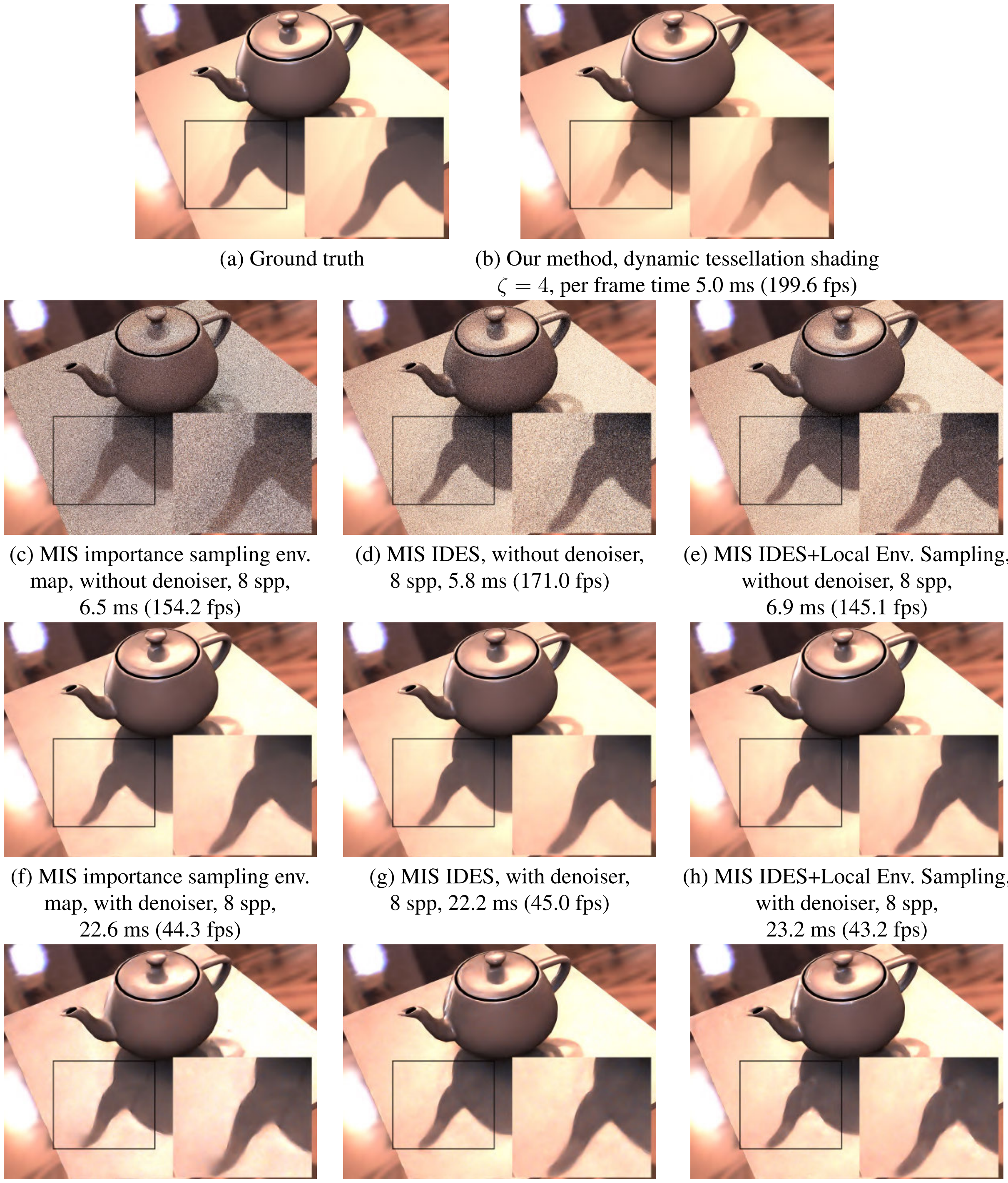
Our method inherits the vectorized visibility method, so it also allows such an application. Inheriting the advantages of the vectorized visibility method (see Fig. 15(k)), our method provides high quality rendering results (see Fig. 15(j)) even with a coarsely tessellated 3D model (see Fig. 10(a)) and preserves similar visual details for the shadows compared to the ground truth (see Fig. 15(i)). Besides the inherited advantages, analogous to the dynamic tessellation shading, our method also provides a 2x rendering speed for the per fragment shading. In particular, specific to Fig. 15(j) and Fig. 15(k), our method is 2.5 times as fast as the previous method. The same observation can also be concluded by comparing Fig. 15(b)(c) against (a), so as Fig. 15(f)(g) against (e).

The SSDF methods synthesize the visibility functions by interpolating the SSDFs. Fig. 15(d), Fig. 15(h) and Fig. 15(l) are the SSDF rendering results rendered using the SSDF method [31]. As reported in [12], the vectorized visibility method has a better visibility interpolation ability than the SSDF methods. Our method inherits the advantages of the vectorized visibility method, so it also has a better visibility interpolation ability than the SSDF methods. However, it is important to point out that the visibility interpolation ability of the SSDF methods can improve dramatically by over sampling the 3D model surfaces for the visibility functions and applying to them a cluster PCA compression afterwards. Taking into account this possibility, the SSDF methods might be able to provide a faster rendering speed at the same rendering quality even w.r.t. our 2x rendering speed remedy to the vectorized visibility method. It would be interesting to analyze these methods from the rendering speed perspective.

D. COMPARISONS TO MONTE CARLO METHODS

1) THE EXPERIMENTAL SETUP

To include Monte Carlo methods for the performance evaluation, we implement IDES for the direct illumination rendering. We adjust IDES to an RTX compatible implementation using the alias method [47], MIS [48], Vulkan [54], NIVIDIA RTX [43] and the OptiX AI denoiser [45]. By fully utilizing the hardware acceleration, the implementation achieves a real time rendering speed. For



(a) Ground truth

(b) Our method, dynamic tessellation shading $\zeta = 4$, per frame time 5.0 ms (199.6 fps)

(c) MIS importance sampling env. map, without denoiser, 8 spp, 6.5 ms (154.2 fps)

(d) MIS IDES, without denoiser, 8 spp, 5.8 ms (171.0 fps)

(e) MIS IDES+Local Env. Sampling, without denoiser, 8 spp, 6.9 ms (145.1 fps)

(f) MIS importance sampling env. map, with denoiser, 8 spp, 22.6 ms (44.3 fps)

(g) MIS IDES, with denoiser, 8 spp, 22.2 ms (45.0 fps)

(h) MIS IDES+Local Env. Sampling, with denoiser, 8 spp, 23.2 ms (43.2 fps)

(i) MIS importance sampling env. map, with denoiser, 2 spp, 18.3 ms (52.8 fps)

(j) MIS IDES, with denoiser, 2 spp, 19.0 ms (52.7 fps)

(k) MIS IDES+Local Env. Sampling, with denoiser, 2 spp, 19.0 ms (52.7 fps)

FIGURE 17. The visual evaluation for our method and the six Monte Carlo comparison targets. Given 5 ms per frame, the rendering result of our method (b) is smooth and preserves the fidelity of shadows. On the other hand, the rendering result of MIS IDES at 8 spp (d) still has a significant amount of pepper noise. Therefore, our method provides a large margin of performance improvement, i.e. 14.24 dB in terms of PSNR. With the denoiser, the rendering result of MIS IDES at 2 spp (j) has a similar PSNR as (b). However, similar PSNR only reflects similar visual quality from macroscopic point of view, which does not necessarily imply an equivalent visual quality. For instance, (b) captures local features like the subtle shadow transitions, which are lost in (j) because of not enough spp.

instance, with 8 spp and the OptiX AI denoiser, it can produce visually appealing smooth images at 45 fps (see Fig. 17(g)).

Note that IDES consists of two components, the first component is an improved selection of outgoing directions from a

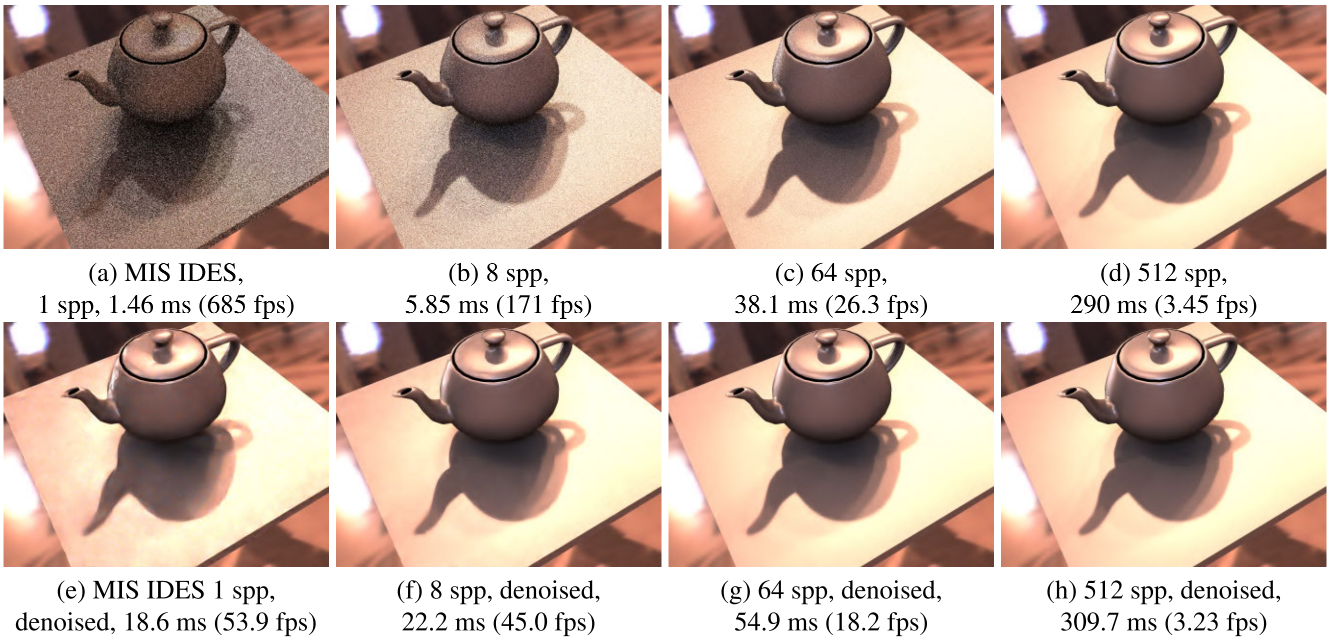


FIGURE 18. A sequence of rendering results of MIS IDES. (a)-(d) are the rendering results of MIS IDES with the increasing number of spp. At 8 spp, MIS IDES is far from converged, and the rendering result in (b) has a significant amount of pepper noise. By increasing the number of spp, the rendering quality gradually improves at the cost of longer processing time. Applying the denoiser (e)-(h) produces visually appealing smooth rendering results with just a few spp. Visually appealing though, the subtle shadow transitions are lost, and the processing time is longer due to the overhead.

given environment map, and the second is an efficient starting position sampling for light paths. It is less obvious of how to implement an RTX compatible adjustment for the lateral one, and it is unfair to compare algorithms with hardware support vs. those without. Therefore, we compare to the first component of IDES only, i.e. the *approximated directional distribution*.

As documented in [51], we prepare the initial luminance environment map, prepare the path map, and element wise multiply them to obtain the *approximated directional distribution*. Then, we feed the distribution to the alias method for accelerating the sampling process. It should be noticed that before feeding to the alias method, the distribution should be multiplied with the solid angles, in order to correct the non-uniform sampling problem of the latitude-longitude map. The resulting probability table and the alias table from the alias method, together with the *approximated directional distribution* before the solid angle multiplication, are uploaded to the GPU as three Vulkan storage buffers.

In the raygen shader, we use the balance heuristic, MIS [48], to combine the samples from the BRDF and the *approximated directional distribution*. To simplify the discussion, we refer to this usage of MIS and IDES as MIS IDES. Then, we substitute the distribution with the distributions from two other methods. The first one is the environment map itself, and it serves as the baseline for the included Monte Carlo methods in our performance evaluation. The second one is the distribution from applying both the *approximated directional distribution* and [50]. Considering also with or without denoising, we have the six comparison targets in the direct illumination rendering results shown in Fig. 16.

2) COMPARISON

Fig. 16(a) shows how the PSNR evolves with varying spp. The curves are obtained by plotting the PSNR against per frame processing time. As expected, the PSNR of MIS IDES is consistently higher than that of the baseline by a significant margin, 3.4 dB to be specific. Interestingly, MIS IDES is increasingly faster than the baseline with the same number of spp, which is probably because IDES exhibits a better ray coherence.

Also shown in the figure is the PSNR of our method with varying tessellation quality. We can see in the figure that given the same amount of processing time, our method has a higher PSNR than MIS IDES. Given 5 ms per frame, the PSNR of our method is 29.58 dB, while the PSNR of MIS IDES at 8 spp is 15.34 dB. It gives us a performance improvement of 14.24 dB in terms of PSNR.

This performance improvement can also be observed visually. Fig. 17 shows the direct illumination rendering results of our method and our six comparison targets. Comparing Fig. 17(b) (rendered in 5.0 ms) to Fig. 17(d) (rendered in 5.8 ms), we can see that our method provides a smooth rendering result that can preserve the fidelity of shadows, while MIS IDES at 8 spp still exhibits a significant amount of pepper noise. At 8 spp, MIS IDES is far from converged, and therefore we have the large margin of performance improvement. See Fig. 18 for a sequence of rendering results that is approaching to its converged state.

By applying the denoiser, smooth rendering results can be obtained with just a few spp (see Fig. 17(c)-(e) vs. Fig. 17(f)-(h) and Fig. 18(a)-(d) vs. Fig. 18(e)-(h)). In terms of PSNR, denoising a rendering result significantly

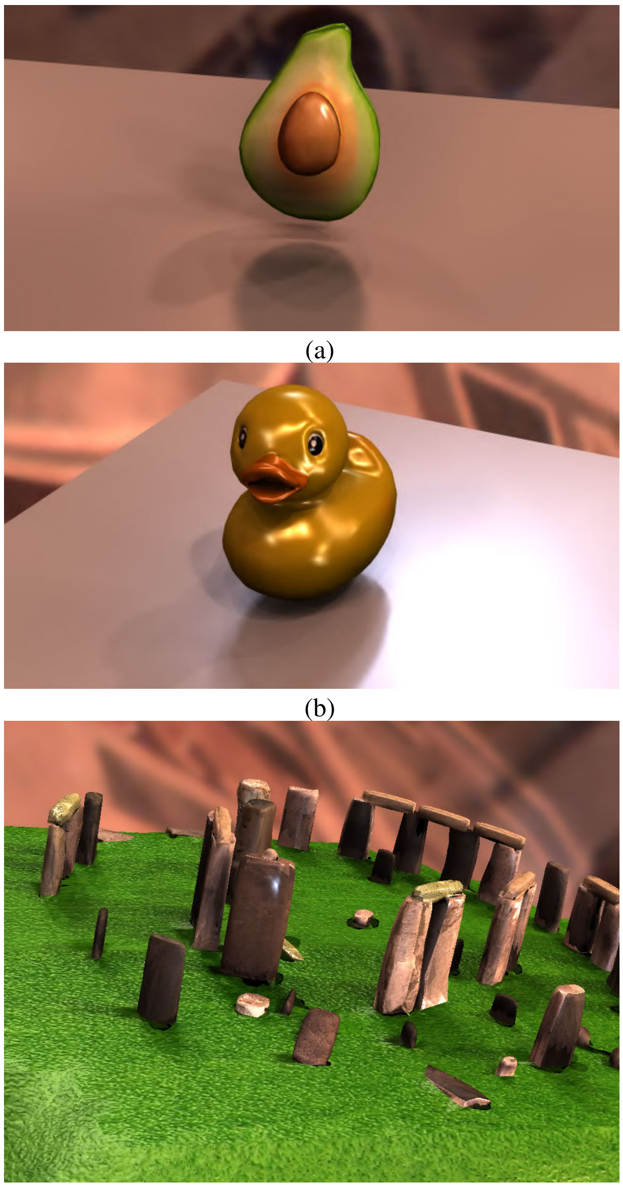

(a)
(b)
(c)

FIGURE 19. More rendering results.

increases the PSNR, ranging from as high as 20 dB at 2 spp, gradually reducing to 19 dB at 8 spp, and eventually to 8 dB at 512 spp (see Fig. 16(a)).

However, because of the overhead of applying the denoiser, it is infeasible to have a same speed comparison. The fastest rendering speed with the denoiser is 18.56 ms at 1 spp whereas the slowest of our method is 8.79 ms. Instead, we perform a similar PSNR comparison. With the similar PSNR, the processing time of our method is 5 ms, while that of MIS IDES with the denoiser at 2 spp is 19 ms. Our method has a 4 times as fast rendering speed w.r.t. MIS IDES with the denoiser. Note that PSNR only reflects visual quality from macroscopic point of view. The rendering result of our method Fig. 17(b) captures local features like the subtle shadow transitions, which are lost in Fig. 17(j) because of not enough spp.

It is possible to apply [50] on top of IDES. However, we did not observe a further performance improvement (see Fig. 16(a) and Fig. 17). This is probably because their treatment to approximate the ideal PDF is overlapping, where both are utilizing the coherence between the BRDF and the environment map.

Lastly, the material attributes of the 3D models in the rendering results above are all, deliberately, plain white to enhance the visual comparison. In general, 3D models would look better with material attributes like texture mapping, material color, vertex color, etc. Readers may refer to Fig. 19 for the rendering results with more decor.

V. CONCLUSION

In this paper, we propose a method to generate the correspondence for the vectorized visibility, and we evaluate our method using the all frequency direct illumination rendering. The basic idea is to use spherical Voronoi diagram to conduct a preliminary correspondence generation. Propagating the preliminary results through the neighbouring triangles and spreading them across the 3D model give us the output triple lists, i.e. the correspondence for the vectorized visibility.

With the output triple lists, we can directly interpolate the vectorized visibility for either the dynamic tessellation shading or the per fragment shading. Hence, our method can provide a 2x as fast rendering speed with a similar rendering quality w.r.t. the vectorized visibility method [12]. This property can potentially make the usages of the vectorized visibility more suitable for devices with less computational power, e.g. the mobile devices.

ACKNOWLEDGMENT

A high quality video demonstration is available at https://drive.google.com/file/d/1Vn8fhSDdbAIi3Dvxm1SlqV2DpXMP1dJ8/view?usp=sharing.

REFERENCES

[1] P. Bekaert, M. Sbert, and Y. D. Willems, "Weighted importance sampling techniques for Monte Carlo radiosity," in *Rendering Techniques*. Vienna, Austria: Springer, 2000, pp. 35–46.

[2] S. Agarwal, R. Ramamoorthi, S. Belongie, and H. W. Jensen, "Structured importance sampling of environment maps," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 605–612, Jul. 2003.

[3] V. Ostromoukhov, C. Donohue, and P.-M. Jodoin, "Fast hierarchical importance sampling with blue noise properties," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 488–495, Aug. 2004.

[4] L. Wan, T.-T. Wong, and C.-S. Leung, "Spherical Q^2-tree for sampling dynamic environment sequences," in *Proc. 16th Eurographics Conf. Rendering Techn.* Aire-la-Ville, Switzerland: Eurographics Association, 2005, pp. 21–30.

[5] L. Wan, S.-K. Mak, T.-T. Wong, and C.-S. Leung, "Spatiotemporal sampling of dynamic environment sequences," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 10, pp. 1499–1509, Oct. 2011.

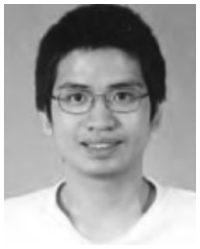
[6] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 527–536, Jul. 2002.

[7] P.-P. Sloan, J. Hall, J. Hart, and J. Snyder, "Clustered principal components for precomputed radiance transfer," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 382–391, Jul. 2003.

[8] J. Kautz, J. Snyder, and P.-P. J. Sloan, "Fast arbitrary BRDF shading for low-frequency lighting using spherical harmonics," in *Proc. 13th Eurograph. Symp. Rendering Techn.*, Pisa, Italy, Jun. 2002.
[9] X. Liu, P.-P. J. Sloan, H.-Y. Shum, and J. Snyder, "All-frequency precomputed radiance transfer for glossy objects," in *Proc. 15th Eurograph. Symp. Rendering Techn.*, Norrköping, Sweden, Jun. 2004.
[10] A. Ben-Artzi, R. Overbeck, and R. Ramamoorthi, "Real-time BRDF editing in complex lighting," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 945–954, Jul. 2006.
[11] P.-M. Lam, T.-Y. Ho, C.-S. Leung, and T.-T. Wong, "All-frequency lighting with multiscale spherical radial basis functions," *IEEE Trans. Vis. Comput. Graphics*, vol. 16, no. 1, pp. 43–56, Jan. 2010.
[12] T.-Y. Ho, Y. Xiao, R.-B. Feng, C.-S. Leung, and T.-T. Wong, "All-frequency direct illumination with vectorized visibility," *IEEE Trans. Vis. Comput. Graphics*, vol. 21, no. 8, pp. 945–958, Aug. 2015.
[13] R. Ng, R. Ramamoorthi, and P. Hanrahan, "All-frequency shadows using non-linear wavelet lighting approximation," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 376–381, Jul. 2003.
[14] R. Wang, J. Tran, and D. Luebke, "All-frequency relighting of glossy objects," *ACM Trans. Graph.*, vol. 25, no. 2, pp. 293–318, Apr. 2006.
[15] W. Sun and A. Mukherjee, "Generalized wavelet product integral for rendering dynamic glossy objects," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 955–966, Jul. 2006.
[16] Y.-T. Tsai and Z.-C. Shih, "All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 967–976, Jul. 2006.
[17] C.-S. Leung, T.-T. Wong, P.-M. Lam, and K.-H. Choy, "An RBF-based compression method for image-based relighting," *IEEE Trans. Image Process.*, vol. 15, no. 4, pp. 1031–1041, Apr. 2006.
[18] X. Sun, K. Zhou, Y. Chen, S. Lin, J. Shi, and B. Guo, "Interactive relighting with dynamic BRDFs," *ACM Trans. Graph.*, vol. 26, no. 3, p. 27, Jul. 2007.
[19] O. Akerlund, M. Unger, and R. Wang, "Precomputed visibility cuts for interactive relighting with dynamic BRDFs," in *Proc. 15th Pacific Conf. Comput. Graph. Appl. (PG)*, Oct. 2007, pp. 161–170.
[20] J. Jendersie, D. Kuri, and T. Grosch, "Precomputed illuminance composition for real-time global illumination," in *Proc. 20th ACM SIGGRAPH Symp. Interact. 3D Graph. Games*, Feb. 2016, pp. 129–137.
[21] S. Elcott, K. Chang, M. Miyamoto, and N. Metaaphanon, "Rendering techniques of final fantasy XV," in *Proc. ACM SIGGRAPH Talks*, Jul. 2016, p. 48.
[22] F. C. Crow, "Summed-area tables for texture mapping," *ACM SIGGRAPH Comput. Graph.*, vol. 18, no. 3, pp. 207–212, Jul. 1984.
[23] J. Hensley, T. Scheuermann, M. Singh, and A. Lastra, "Interactive summed-area table generation for glossy environmental reflections," in *Proc. ACM SIGGRAPH Sketches SIGGRAPH*, 2005, p. 34.
[24] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra, "Fast summed-area table generation and its applications," *Comput. Graph. Forum*, vol. 24, no. 3, pp. 547–555, Sep. 2005.
[25] W. Matusik, M. Loper, and H. Pfister, "Progressively-refined reflectance functions from natural illumination," in *Proc. 15th Eurograph. Symp. Rendering Techn.*, Norrköping, Sweden, Jun. 2004.
[26] K. Xu, Y.-T. Jia, H. Fu, S.-M. Hu, and C.-L. Tai, "Spherical piecewise constant basis functions for all-frequency precomputed radiance transfer," *IEEE Trans. Vis. Comput. Graphics*, vol. 14, no. 2, pp. 454–467, Mar. 2008.
[27] E. Cheslack-Postava, R. Wang, O. Akerlund, and F. Pellacini, "Fast, realistic lighting and material design using nonlinear cut approximation," *ACM Trans. Graph.*, vol. 27, no. 5, p. 128, 2008.
[28] S. Laine, T. Aila, U. Assarsson, J. Lehtinen, and T. Akenine-Möller, "Soft shadow volumes for ray tracing," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1156–1165, 2005.
[29] J. Lehtinen, S. Laine, and T. Aila, "An improved physically-based soft shadow volume algorithm," *Comput. Graph. Forum*, vol. 25, no. 3, pp. 303–312, Sep. 2006.
[30] D. Nowrouzezahrai, I. Baran, K. Mitchell, and W. Jarosz, "Visibility silhouettes for semi-analytic spherical integration," in *Computer Graphics Forum*, vol. 33, no. 1. Hoboken, NJ, USA: Wiley Online Library, 2014, pp. 105–117.
[31] J. Wang, P. Ren, M. Gong, J. Snyder, and B. Guo, "All-frequency rendering of dynamic, spatially-varying reflectance," *ACM Trans. Graph.*, vol. 28, no. 5, p. 133, 2009.
[32] R. Wang, M. Pan, W. Chen, Z. Ren, K. Zhou, W. Hua, and H. Bao, "Analytic double product integrals for all-frequency relighting," *IEEE Trans. Vis. Comput. Graphics*, vol. 19, no. 7, pp. 1133–1142, Jul. 2013.
[33] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, "Microfacet models for refraction through rough surfaces," in *Proc. 18th Eurograph. Symp. Rendering Techn.*, Grenoble, France, Jun. 2007.
[34] B. T. Phong, "Illumination for computer generated pictures," *Commun. ACM*, vol. 18, no. 6, pp. 311–317, Jun. 1975.
[35] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering*. Boca Raton, FL, USA: CRC Press, 2008.
[36] F. Aurenhammer, "Voronoi diagrams—A survey of a fundamental geometric data structure," *ACM Comput. Surv.*, vol. 23, no. 3, pp. 345–405, Sep. 1991.
[37] K. Kise, A. Sato, and M. Iwata, "Segmentation of page images using the area Voronoi diagram," *Comput. Vis. Image Understand.*, vol. 70, no. 3, pp. 370–382, Jun. 1998.
[38] P. Bhattacharya and M. L. Gavrilova, "Voronoi diagram in optimal path planning," in *Proc. 4th Int. Symp. Voronoi Diagrams Sci. Eng. (ISVD)*, Jul. 2007, pp. 38–47.
[39] J. M. Augenbaum and C. S. Peskin, "On the construction of the Voronoi mesh on a sphere," *J. Comput. Phys.*, vol. 59, no. 2, pp. 177–192, Jun. 1985.
[40] T. Whitted, "An improved illumination model for shaded display," in *Proc. ACM SIGGRAPH Courses SIGGRAPH*, 2005, p. 4.
[41] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," in *Proc. 11th Annu. Conf. Comput. Graph. Interact. Techn.*, 1984, pp. 137–145.
[42] *Graphics Cards With Turing GPU Architecture*. Nvidia Corporation, Santa Clara, CA, USA, 2018. [Online]. Available: https://www.nvidia.com/en-us/geforce/turing
[43] *NVIDIA RTX Platform*, Nvidia Corporation, Santa Clara, CA, USA, 2018. [Online]. Available: https://developer.nvidia.com/rtx
[44] C. R. A. Chaitanya, A. S. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai, and T. Aila, "Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–12, Jul. 2017.
[45] *NVIDIA OptiX AI-Accelerated Denoiser*, Nvidia Corporation, Santa Clara, CA, USA, 2017. [Online]. Available: https://developer.nvidia.com/optix-denoiser
[46] A. J. Walker, "New fast method for generating discrete random numbers with arbitrary frequency distributions," *Electron. Lett.*, vol. 10, no. 8, pp. 127–128, Apr. 1974.
[47] M. D. Vose, "A linear algorithm for generating random numbers with a given distribution," *IEEE Trans. Softw. Eng.*, vol. 17, no. 9, pp. 972–975, Sep. 1991.
[48] E. Veach and L. J. Guibas, "Optimally combining sampling techniques for monte carlo rendering," in *Proc. 22nd Annu. Conf. Comput. Graph. Interact. Techn.*, 1995, pp. 419–428.
[49] D. Burke, A. Ghosh, and W. Heidrich, "Bidirectional importance sampling for illumination from environment maps," in *Proc. ACM SIGGRAPH Sketches SIGGRAPH*, 2004, p. 112.
[50] J. Lawrence, S. Rusinkiewicz, and R. Ramamoorthi, "Adaptive numerical cumulative distribution functions for efficient importance sampling," in *Proc. 16th Eurograph. Symp. Rendering Techn.*, Konstanz, Germany, Jun./Jul. 2005.
[51] T. Bashford-Rogers, K. Debattista, and A. Chalmers, "Importance driven environment map sampling," *IEEE Trans. Vis. Comput. Graphics*, vol. 20, no. 6, pp. 907–918, Jun. 2014.
[52] P. E. Debevec and J. Malik, "Recovering high dynamic range radiance maps from photographs," in *Proc. 24th Annu. Conf. Comput. Graphics Interact. Techn.*, Reading, MA, USA: Addison-Wesley, 1997, pp. 369–378, doi: 10.1145/258734.258884.
[53] *Notebookcheck.net*. [Online]. Available: https://www.notebookcheck.net
[54] *Vulkan Overview*. The Khronos Group Inc., Beaverton, OR, USA, 2016. [Online]. Available: https://www.khronos.org/vulkan

HO CHUN LEUNG (Member, IEEE) received the B.Eng. degree in information engineering and the Ph.D. degree in electrical engineering from the City University of Hong Kong, Hong Kong, SAR, China, in 2015 and 2020, respectively. His research interests include machine learning, neural networks, computer graphics, and telecommunication.

ZHENNI WANG received the B.S. degree from Hunan University, in 2017. She is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, City University of Hong Kong. Her research interest includes computer graphics.



TZE YUI HO received the B.Sc. degree in mathematics from the Hong Kong University of Science and Technology, in 2002, and the M.Phil. and Ph.D. degrees in electronic engineering from the City University of Hong Kong, in 2007 and 2010, respectively. His research interests include pre-computed radiance transfer (PRT) and GPU programming for realistic rendering.

CHI SING LEUNG (Senior Member, IEEE) received the Ph.D. degree in computer science from the Chinese University of Hong Kong, in 1995. He is currently a Professor with the Department of Electrical Engineering, City University of Hong Kong. He has published over 120 journal articles in the areas of digital signal processing, neural networks, and computer graphics. His research interests include neural computing and computer graphics. Dr. Leung was a member of Organizing Committee of ICONIP2006. He is a Governing Board Member of the Asian Pacific Neural Network Assembly (APNNA) and the Vice President of APNNA. In 2005, he received the 2005 IEEE TRANSACTIONS ON MULTIMEDIA Prize Paper Award for his article The Plenoptic Illumination Function. He was the Program Chair of the ICONIP2009 and ICONIP2012. He is/was the Guest Editor of several journals, including *Neural Computing and Applications, Neuro Computing*, and *Neural Processing Letters.*

ERIC WING MING WONG (Senior Member, IEEE) received the B.Sc. and M.Phil. degrees in electronic engineering from the Chinese University of Hong Kong, Hong Kong, in 1988 and 1990, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Massachusetts, Amherst, MA, USA, in 1994. He is an Associate Professor with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong. His research interests include analysis and design of telecommunications and computer networks, energy-efficient data center design, green cellular networks, and optical networking.