





Precomputed Discrete Visibility Fields for Real-Time Ray-Traced Environment Lighting

Yang Xu¹ , Yuanfa Jiang¹ , Chenhao Wang¹, Kang Li^{†1} , Pengbo Zhou², and Guohua Geng¹ 

¹School of Information Science and Technology, Northwest University, Xi'an, China

²School of Arts and Communication, Beijing Normal University, Beijing, China



Figure 1: Rendering results of ray-traced environment lighting with our proposed precomputed discrete visibility fields. Left: Crytek Sponza (262 K triangles), 128 spp, 15.74 ms. Right: Amazon Lumberyard Bistro (2.8 M triangles), 64 spp, 33.67 ms.

Abstract

Rendering environment lighting using ray tracing is challenging because many rays within the hemisphere are required to be traced. In this work, we propose discrete visibility fields (DVF), which store visibility information in a uniform grid to speed up ray-traced low-frequency environment lighting for static scenes. In the precomputation stage, we compute and store the visibility and occlusion masks at the positions of the point samples of the scene using octahedral mapping. The visibility and occlusion masks of the point samples inside a grid cell are then merged by the logical OR operation. We also store the occlusion label indicating whether more than half of the pixels are occluded in the occlusion mask of each grid cell. At runtime, we exclude the rays occluded by the geometry or visible to the environment according to the information stored in the DVF. Results show that the proposed method can significantly speed up the rendering of ray-traced environment lighting and achieve real-time frame rates without sacrificing image quality. Compared to other environment lighting rendering methods based on precomputation, our method is free of tessellation or parameterization of the meshes, and the precomputation can be finished in a short time.

CCS Concepts

• *Computing methodologies* → *Rendering; Ray tracing;*

1. Introduction

Environment lighting in the scene can significantly enhance the realism of rendered images. The key to rendering environment lighting is evaluating the visibility function, which can be acquired by shadow mapping [Wil78] or ray tracing [Whi80]. However, many

shadow maps or rays are required to be created or traced because the environment light sources are distributed over the entire sphere or hemisphere, which makes rendering environment lighting at real-time frame rates challenging.

In recent years, hardware-accelerated ray tracing [NVI18] has been introduced to significantly improve the efficiency of ray tracing. However, the performance is still not able to meet the require-

[†] Corresponding author: likang@nwu.edu.cn

ment for rendering high-quality environment lighting in real-time. In most cases, noisy rendering results with a small number of ray samples are generated and then filtered by a spatiotemporal denoiser [SKW*17, FWHB21, IMF*21] to obtain the final results. However, since most denoisers exploit temporal reuse of the rendering results from previous frames, the newly visible or disoccluded pixels may become noisy when the camera is moving because temporal reprojection fails and the accumulated samples are insufficient. Besides, fast moving lights can also lead to lag or noise. Ray tracing can also be accelerated by caching visibility on the surface [CAM08a, PGSD13] or in a 3D grid [GEE20].

If the scene and environment are both static, *lightmaps* or *irradiance volumes* [GSHG98] can be precomputed to render environment lighting in real-time. A lot of previous works precomputed data for real-time environment lighting under dynamic environment lighting. The precomputed data on the surface can be either stored as vertex attributes [SKS02], texture atlases [JKG16], or scattered point samples [LZT*08]. The meshes should be tessellated, and the resolution of the textures or the point samples should be high enough to capture the lighting details, which requires both long pre-computation time and large memory storage. Otherwise, the quality of the rendered image may degrade. Additionally, parameterization of complex models may cause artifacts in detailed geometries due to discontinuities.

In this work, we present a method to render low-frequency dynamic environment lighting at real-time frame rates for static scenes. We propose a data structure called *discrete visibility field* (DVF), which stores visibility and occlusion masks in a uniform grid along with occlusion labels to determine whether the visible or occluded regions should be traced at runtime. The field is *discrete* because we do not interpolate the data from different grid cells. In the preprocessing stage, we generate point samples on the scene geometry, shoot rays from the positions of the point samples, and store the occlusion information using octahedron mapping. A uniform grid is constructed for the whole scene, and we merge the visibility and occlusion masks inside a grid cell by the logical OR operation. Besides, the occlusion label indicating whether more than half of the pixels are occluded in the occlusion mask is also stored in each grid cell. At runtime, we first locate the grid cell by the fragment position and fetch the occlusion label. If the occlusion label is 1, we trace rays in the visible region of the visibility mask and discard the unnecessary rays occluded by the geometry. Otherwise, we trace in the occluded region of the occlusion mask and exclude the rays visible to the environment. Rendering times can be significantly reduced without sacrificing image quality.

Compared to traditional ray tracing without DVFs, our method can produce comparable results because we just discard the unnecessary rays by the DVF. Therefore, the lighting details can be fully captured. Besides, our proposed method is volume-based instead of surface-based, which means it is free of tessellating the meshes or generating texture atlases for complex models. Additionally, the pre-computation time of our method is short compared to other pre-computed environment lighting methods, and the memory footprint of the precomputed data is small.

Our contributions of this work are as follows:

- A volume-based data structure called *discrete visibility field* stor-

ing visibility and occlusion masks in a uniform grid along with occlusion labels to determine whether visible or occluded regions should be traced.

- A DVF precomputation method by creating visibility and occlusion masks at the positions of the point samples of the scene and merging the masks inside a grid cell by the logical OR operation.
- A runtime rendering algorithm with the precomputed DVFs to speed up ray-traced environment lighting by discarding the unnecessary rays according to the visibility information from the DVFs during ray tracing.

2. Related Work

Precomputed Environment Lighting. Lightmaps storing the incoming irradiance in texture atlases are the most commonly used techniques to render environment lighting in practice, but dynamic environment lighting is not supported. Sloan et al. [SKS02] presented *precomputed radiance transfer* (PRT), which supports dynamic environment light sources by precomputing spherical harmonics (SH) transfer vectors that map the SH vector of the environment map to the outgoing radiance. Sloan et al. [SHHS03] compressed the PRT data by *clustered principal component analysis* (CPCA) to both reduce the memory footprint and the rendering cost. Finite element methods such as *radiosity* can be utilized to render environment lighting. Light transport paths [JKG16] representing the form factors in radiosity are precomputed to achieve real-time environment lighting. However, both PRT and radiosity need to tessellate or parameterize the meshes to capture lighting details. The surface elements can also be represented by scattered points [LZT*08], and the irradiance can be obtained by interpolating the points. Besides, volume-based representation can be adopted for precomputed lighting. Greger et al. [GSHG98] proposed *irradiance volumes* storing incoming irradiance in the probes placed at the positions of grid cells and interpolating between the probes at runtime. McGuire et al. [MMNL17] precomputed *light field probes* storing the integrated incoming radiance at the locations of the probes and suppressed light leaking by the additional distance information in the probes. Seyb et al. [SSS*20] proposed *local light grids* which store additional self-visibility terms at vertices to capture high-frequency lighting details. Zhou et al. [ZHL*05] precomputed *shadow fields* storing SH visibility vectors in multiple cube maps at the sampled points around the dynamic rigids.

Real-Time Environment Lighting without Precomputation. Ritschel et al. [RGK*08] presented *imperfect shadow maps* (ISMs), which efficiently create low-resolution shadow maps for many virtual point lights (VPLs) by point samples of the scene. Environment lighting was supported by approximating the environment map with VPLs. Barák et al. [BBH13] improved the temporal coherence of ISM using Metropolis-Hastings sampling and proposed a fast ISM creation method using GPU tessellation. Blocker accumulation can be also adopted to approximate the visibility function. Ren et al. [RWS*06] used sphere blockers and accumulated visibility in log-SH space by SH addition, and *spherical harmonic exponentiation* (SHEXP) was performed on the accumulated log-SH vector to obtain the SH visibility vector. Sloan et al. [SGNS07] splatted sphere blockers onto the screen buffer to improve efficiency. Giraud

and Nowrouzezahrai [GN15] accumulated the visibility of dynamic height fields in log-SH space and computed log-SH BRDF as well to avoid costly triple-product integration.

Ray-Traced Environment Lighting. Szécsi et al. [SSSK04] combined BRDF importance sampling with correlated sampling, which analytically computes the radiance neglecting the visibility factor as the *control variate* to reduce the variance of rendering results of environment lighting. This method is also adopted in our method when we trace the occluded region to evaluate the occluded irradiance, which is then subtracted from the unshadowed irradiance to obtain the final incoming irradiance. All-frequency environment maps can be importance sampled to reduce the variance by *cumulative density function* (CDF) inversion [CRW09] or point relaxation [ARB03, KK03, ODJ04]. However, importance sampling of environment maps performs poorly for highly glossy BRDFs. *Multiple importance sampling* proposed by Veach and Guibas [VG95] can be used to combine BRDF importance sampling with importance sampling of environment maps. *Rejection sampling* [BGH05], *resampled importance sampling* (RIS) [BGH05, TCE05], and *wavelet-based importance sampling* [CJAMJ05, CAM08b] can be utilized to sample the product of the BRDF and the environment map. Bitterli et al. [BWP*20] proposed *reservoir-based spatiotemporal importance resampling* (RESTIR) to perform RIS through spatial and temporal reuse of the samples and weighted reservoir sampling, which allows real-time rendering of environment lighting. Denoisers based on temporal reprojection and accumulation [SKW*17, FWHB21, IMF*21] are adopted in real-time ray tracing to filter the rendering results with a low sample rate.

Visibility Caching for Ray Tracing. Clarberg [CAM08a] exploited the visibility correlation to compute and store visibility maps at sparse locations on the surface, and interpolated these visibility maps to obtain visibility approximations as *control variates* to lower the variance of ray-traced direct environment lighting. Popov et al. [PGSD13] shared the cached visibility inside the cluster with similar orientations to discard most of the shadow rays. The visibility caches are generated on-the-fly and stored in a hash map. Interactive frame rates can be achieved with noticeable quality degradation. Guo et al. [GEE20] precomputed and cached voxel-to-voxel visibility in a matrix-like map. Unnecessary visibility tests can be discarded in unidirectional path tracing to acquire comparable rendering results with fewer shadow-ray queries and lower rendering time.

Potentially Visible Set. Our precomputed DVF shares some similarities with *potentially visible set* (PVS) [ARB90, TS91], such as partitioning the scene into grid cells and shooting rays from the point samples inside a cell to evaluate from-region visibility. However, PVS stores the geometries potentially visible to the grid cell while our DVF stores the visible and occluded directions using octahedral mapping in the grid cell. A conservative PVS can be obtained by occluder fusion [SDDS00] or extended projection [DDTP00]. Wonka et al. [WWS00] calculated PVS for each point sample on the boundary of the view cell and fuse the PVSs in a view cell by occluder shrinking to obtain the conservative PVS for the view cell, which is similar to merging masks inside a grid cell by the logical OR operation in our method. Bittner et al. [BMW*09]

proposed a mutation-based adaptive sampling strategy by exploiting the spatial coherence of visibility to reduce the precomputation cost of the PVS.

3. Method

The overview of our proposed method, containing both precomputation and runtime stages, is shown in Figure 2. In the precomputation stage, we construct a uniform grid for the scene, sample points on the surface geometry, create visibility and occlusion masks for each point sample, merge the masks inside a grid cell of the DVF by the logical OR operation, and compute the occlusion label for each grid cell. At runtime, we index the grid cell by the position of the surface point, determine whether to trace the visible region or the occluded region by the occlusion labels in the DVF, and trace rays inside the visible or occluded region to obtain the incoming irradiance or the irradiance occluded by the scene geometry. In the following subsections, we will describe our method in detail.

3.1. Discrete Visibility Fields

We define a *discrete visibility field* by a uniform grid. Each grid cell of the uniform grid stores a visibility mask and an occlusion mask as spherical functions. We define the visibility mask V_c of a grid cell c as:

$$V_c(c, \omega) = \begin{cases} 1, & \text{if all points inside } c \text{ is visible in direction } \omega, \\ 0, & \text{otherwise,} \end{cases} \quad (1)$$

and the occlusion mask O_c of a grid cell c as:

$$O_c(c, \omega) = \begin{cases} 1, & \text{if all points inside } c \text{ is occluded in direction } \omega, \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

Each value in the masks represents whether all surface points inside the grid cell are visible to the environment or occluded by the scene geometry in the corresponding direction. Additionally, a label which denotes whether more than half of the directions are occluded in the occlusion mask is also stored in each grid cell, which can be used to determine whether the occluded rays or the visible rays should be traced. The fields are *discrete* which means that we do not interpolate the data across the grid cells of a DVF. In the next subsection, we will introduce how to precompute the DVF for a scene.

3.2. Precomputation of DVFs

Since each value in the masks of a grid cell in the DVF denotes whether all surface points inside the grid cell are visible to the environment or occluded by the scene geometry in the corresponding direction, we can precompute the DVF for a scene by the following steps: sample points on the scene geometry, compute and store the visibility and occlusion masks for the point samples, merge the masks inside a grid cell by the logical OR operation, and compute the occlusion label for each grid cell of the DVF. Next, we will go through the precomputation process step by step.

Generate Point Samples of the Scene. The first step of precomputation is to generate point samples on the scene geometry. The point

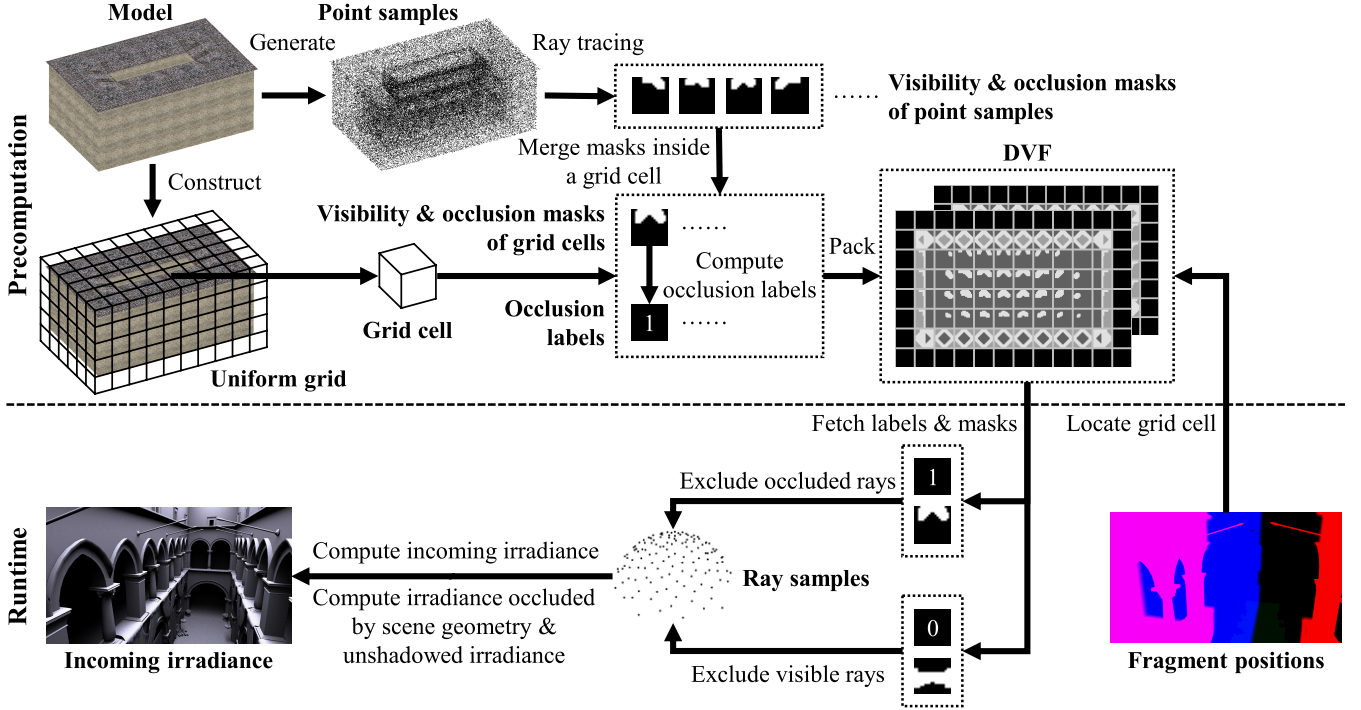


Figure 2: Overview of our proposed method. In the precomputation stage, sample points are generated on the surface geometry and a uniform grid is constructed for the scene. Visibility and occlusion masks are created for each point sample, masks inside a grid cell of the DVF are merged by the logical OR operation, and occlusion labels are computed for each grid cell. At runtime, the grid cell is indexed by the position of the surface point, the occlusion labels in the DVF are used to determine whether to compute the incoming radiance or the irradiance occluded by the scene geometry, and the unnecessary rays are excluded by the masks fetched from the DVF.

samples should lie on the surface of the meshes because only the surface points are required to be shaded at runtime. The samples should be dense enough to ensure that as many as possible surface points are considered during precomputation. Position \mathbf{p} and normal \mathbf{n} of a point sample are both required because we trace rays over the hemisphere about the surface normal in the next step of precomputation.

Create Visibility and Occlusion Masks. Once the point samples of a scene are obtained, we can create visibility and occlusion masks of the point samples using ray tracing. The visibility mask V_s of a point sample $s = (\mathbf{p}, \mathbf{n})$ can be expressed as:

$$V_s(s, \omega) = \begin{cases} 1, & \text{if sample } s \text{ is visible in direction } \omega, \\ 0, & \text{otherwise,} \end{cases} \quad (3)$$

and the occlusion mask O_p of a point sample s is defined as:

$$O_s(s, \omega) = \begin{cases} 1, & \text{if sample } s \text{ is occluded in direction } \omega, \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

To create these two masks, we first set all the values of the masks to 0. Then we shoot a set of rays from the position of the point sample over the hemisphere about the normal of the point sample. If a ray does not intersect the scene geometry, we set the value corresponding to the direction of the ray in the visibility mask as 1. Otherwise, we set the value as 1 in the occlusion mask.

Merge Visibility and Occlusion Masks. We construct a uniform grid for the whole scene to represent the DVF. The masks containing the visibility information of the scene inside a grid cell are merged into the grid cells of the DVF. For each point sample, we find the grid cell that contains the point by the position of the point. We traverse all the directions of the masks to merge the visibility information of all point samples inside a grid cell. For each direction, the logical OR operation is performed on the visibility masks of all point samples $V_s(s_1, \omega), V_s(s_2, \omega), \dots, V_s(s_N, \omega)$ inside a grid cell c to obtain the visibility mask of the grid cell $V_c(c, \omega)$, which can be expressed as:

$$V_c(c, \omega) = V_s(s_1, \omega) \vee V_s(s_2, \omega) \vee \dots \vee V_s(s_N, \omega), \quad (5)$$

where \vee denotes the logical OR operation, and N is the number of point samples inside the grid cell c . The occlusion masks of all point samples $O_s(s_1, \omega), O_s(s_2, \omega), \dots, O_s(s_N, \omega)$ inside the grid cell c can be merged to acquire the visibility mask of the grid cell $O_c(c, \omega)$ in a similar way as follows:

$$O_c(c, \omega) = O_s(s_1, \omega) \vee O_s(s_2, \omega) \vee \dots \vee O_s(s_N, \omega). \quad (6)$$

Therefore, all possible visible directions and all possible occluded directions of each grid cell can be obtained. Figure 3 gives the schematic of merging masks inside a grid cell. To simplify the description, we only show the case of merging two masks. Three or more masks can be merged in the same manner.

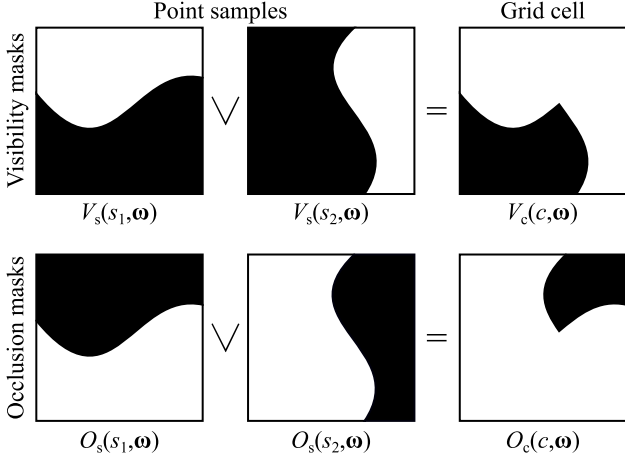


Figure 3: Schematic of merging visibility and occlusion masks inside a grid cell. \vee denotes the logical OR operation.

Compute Occlusion Labels. Once the visibility and occlusion mask of each grid cell are obtained by merging the masks of the point samples, we can compute the ratio of the number of occluded directions to the total number of directions in the occlusion mask of a grid cell as:

$$r_{\text{occ}}(c) = \frac{N_{\text{occ}}(c)}{N_{\text{occ}}(c) + N_{\text{vis}}(c)}, \quad (7)$$

where $N_{\text{occ}}(c)$ denotes the number of occluded directions, and $N_{\text{vis}}(c)$ denotes the number of visible directions in the occlusion mask of a grid cell c , respectively. Then we can compute the occlusion label l_{occ} of a grid cell c as:

$$l_{\text{occ}}(c) = \begin{cases} 1, & r_{\text{occ}}(c) > 0.5, \\ 0, & \text{otherwise.} \end{cases} \quad (8)$$

If the occlusion ratio $r_{\text{occ}}(c)$ is larger than 0.5, the occlusion label $l_{\text{occ}}(c)$ is set as 1, otherwise 0. The occlusion label is used to determine whether the rays inside the visible region or the occluded region need to be traced at runtime.

3.3. Rendering with Precomputed DVFs

To speed up ray-traced environment lighting by the precomputed DVFs, we first locate the grid cell from the DVF by the position of the surface point. The occlusion labels can be used to determine whether the rays should be traced inside the visible region or the occluded region. If the occlusion label is 1, we should trace the rays in the visible region of the visibility mask of the grid cell. Otherwise, only the rays in the occluded region of the occlusion mask need to be traced. We only consider the direct illumination of distant environment lighting and assume the scene materials are diffuse in this work. The incoming irradiance E at the surface point \mathbf{x} can be expressed as:

$$E(\mathbf{x}) = \int_{\Omega^+} L_i(\omega_i) V(\mathbf{x}, \omega_i) (\mathbf{n} \cdot \omega_i) d\omega_i, \quad (9)$$

where ρ_d is the diffuse albedo, $L_i(\omega_i)$ is the incoming radiance from the environment light source in direction ω_i , $V(\mathbf{x}, \omega_i)$ is the visibility function between \mathbf{x} and the environment, and \mathbf{n} is the surface normal. Monte Carlo ray tracing can be adopted to solve this integration. When we trace the rays inside the visible region, the incoming radiance from the environment light source is accumulated to obtain $E(\mathbf{x})$. However, if we trace the rays inside the occluded region, we can only obtain the irradiance occluded by the scene geometry, which is defined as:

$$E_{\text{occ}}(\mathbf{x}) = \int_{\Omega^+} L_i(\omega_i) O(\mathbf{x}, \omega_i) (\mathbf{n} \cdot \omega_i) d\omega_i, \quad (10)$$

where $O(\mathbf{x}, \omega_i) = 1 - V(\mathbf{x}, \omega_i)$ is the occlusion function between \mathbf{x} and the environment. Fortunately, we can subtract the occluded irradiance $E_{\text{occ}}(\mathbf{x})$ from the unshadowed irradiance E_{unshad} to acquire $E(\mathbf{x})$ as follows:

$$E(\mathbf{x}) = E_{\text{unshad}}(\mathbf{n}) - E_{\text{occ}}(\mathbf{x}), \quad (11)$$

where $E_{\text{unshad}}(\mathbf{n})$ can be efficiently evaluated by querying the irradiance environment map in the direction of the surface normal \mathbf{n} .

4. Implementation Details

4.1. Precomputation

We generate the point samples by randomly sampling the surface mesh. Positions and normals are stored for the following precomputation steps. We use a CPU ray tracer to evaluate the visibility function for creating the visibility and occlusion masks. We use the octahedral parameterization [CDE*14] which can map a spherical function to a unit square with low distortion to represent the visibility and occlusion masks. For each point sample, we create two octahedral maps to store these two masks. According to Eq. (3), 1 denotes visible and 0 denotes occluded in the first map that stores the visibility mask. Besides, 1 means occluded and 0 means visible in the second map according to Eq. (4). To create the visibility and occlusion masks, we first set all pixels of the two octahedral maps to 0, and then shoot rays from the position of the point sample over the hemisphere about the normal of the point sample. If a ray does not intersect the scene geometry, we map the direction of the ray to the unit square using octahedral mapping and set the corresponding pixel in the first octahedral map as 1. If a ray intersects the scene geometry, we set the pixel as 1 for the second octahedral map. The number of ray samples for precomputation could be small because the masks of multiple point samples are merged in the next step. We dilate the masks by one pixel to ensure that no rays are wrongly excluded.

Next, we create two octahedral maps with the same size to those of the point samples for each grid cell of the DVF and set all the pixels of the octahedral maps to 0. The merged visibility and occlusion masks inside a grid cell are acquired by the logical OR operations among the visibility and occlusion masks of the point samples as expressed in Eqs. (5) and (6), and the occlusion label is computed by Eqs. (7) and (8) and stored in each grid cell of the DVF. Rays may be wrongly discarded because the query of an octahedral map may jump to the adjacent octahedral maps near the border. To avoid this problem, each octahedral map is padded by one pixel. Our precomputation algorithm is summarized in Algorithm 1.

Algorithm 1: Precompute the DVF

Input: surface mesh \mathcal{M} , N ray samples $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_N$
Output: precomputed DVF \mathcal{V}

- 1 create N point samples s_1, s_2, \dots, s_N on \mathcal{M} ;
- 2 create octahedral map $\mathcal{O}_s^V(s)$ storing $V_s(s, \omega)$ for each s ;
- 3 create octahedral map $\mathcal{O}_s^O(s)$ storing $O_s(s, \omega)$ for each s ;
- 4 set all pixels of $\mathcal{O}_s^V(s)$ and $\mathcal{O}_s^O(s)$ as 0;
 // Create masks of point samples
- 5 **foreach** point sample $s = (\mathbf{p}, \mathbf{n})$ **do**
- 6 **foreach** ray $\mathcal{R} = (\mathbf{p}, \omega)$ **do**
- 7 randomly rotate \mathcal{R} along \mathbf{n} ;
- 8 project ω to unit square using octahedral mapping;
- 9 **if** \mathcal{R} intersects scene geometry **then**
- 10 set the corresp. pixel in $\mathcal{O}_s^V(s)$ as 1 // Eq. (4)
- 11 **else**
- 12 set the corresp. pixel in $\mathcal{O}_s^O(s)$ as 1 // Eq. (3)
- 13 **end**
- 14 **end**
- 15 dilate $\mathcal{O}_s^V(s)$ by one pixel;
- 16 dilate $\mathcal{O}_s^O(s)$ by one pixel;
- 17 **end**
- 18 construct a uniform grid \mathcal{G} for \mathcal{M} ;
- 19 create octahedral map $\mathcal{O}_c^V(c)$ for each grid cell c ;
- 20 create octahedral map $\mathcal{O}_c^O(c)$ for each grid cell c ;
- 21 set all pixels of $\mathcal{O}_c^V(c)$ and $\mathcal{O}_c^O(c)$ as 0;
 // Merge masks inside a grid cell
- 22 **foreach** point sample $s = (\mathbf{p}, \mathbf{n})$ **do**
- 23 locate the grid cell c by \mathbf{p} ;
- 24 **foreach** pixel in the octahedral maps of c **do**
- 25 $V_c(c, \omega) \leftarrow V_c(c, \omega) \vee V_s(s, \omega)$; // Eq. (5)
- 26 $O_c(c, \omega) \leftarrow O_c(c, \omega) \vee O_s(s, \omega)$; // Eq. (6)
- 27 **end**
- 28 **end**
- 29 // Compute occlusion labels
- 30 **foreach** grid cell c in \mathcal{G} **do**
- 31 $r_{\text{occ}}(c) = \frac{N_{\text{occ}}(c)}{N_{\text{occ}}(c) + N_{\text{vis}}(c)}$; // Eq. (7)
- 32 **if** $r_{\text{occ}}(c) > 0.5$ **then**
- 33 $l_{\text{occ}}(c) = 1$; // Eq. (8)
- 34 **else**
- 35 $l_{\text{occ}}(c) = 0$; // Eq. (8)
- 36 **end**

4.2. Runtime

We implemented the runtime algorithm of our method using *Vulkan Ray Tracing* [KHBW20] with the extension `VK_KHR_ray_query` allowing ray tracing in all shader stages to integrate ray tracing with rasterization. We represent the DVF as a 2D texture array instead of a 3D texture because interpolation between layers is not required. The DVF texture contains the octahedral maps representing the visibility and occlusion masks, and the occlusion labels. The octahedral maps are tiled on each layer of the texture array, which means that the x - and y -axes of one layer of the texture array represent

Algorithm 2: Rendering with the precomputed DVF

Input: precomputed DVF \mathcal{V} , incoming radiance of environment map $L_i(\omega)$, fragment position \mathbf{x} , fragment normal \mathbf{n} , N randomly rotated ray samples $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_N$
Output: incoming irradiance $E(\mathbf{x})$

- 1 $E(\mathbf{x}) \leftarrow 0$;
- 2 $E_{\text{occ}}(\mathbf{x}) \leftarrow 0$;
- 3 locate the grid cell c by \mathbf{x} ;
- 4 fetch occlusion label $l_{\text{occ}}(c)$ from \mathcal{V} ;
- 5 **foreach** ray $\mathcal{R} = (\mathbf{x}, \omega)$ **do**
- 6 project ω to unit square using octahedral mapping;
- 7 fetch visibility and occlusion masks $V(c, \omega), O(c, \omega)$;
- 8 // More than half pixels are occluded
- 9 **if** $l_{\text{occ}}(c) = 1$ **then**
- 10 // Trace inside visible region
- 11 **if** $V(c, \omega) = 1$ **then**
- 12 trace \mathcal{R} ;
- 13 **if** \mathcal{R} does not intersect scene geometry **then**
- 14 $E(\mathbf{x}) \leftarrow E(\mathbf{x}) + \frac{1}{N}L_i(\omega)$; // Eq. (9)
- 15 **end**
- 16 **end**
- 17 // Trace inside occluded region
- 18 **if** $O(c, \omega) = 1$ **then**
- 19 trace \mathcal{R} ;
- 20 **if** \mathcal{R} intersects scene geometry **then**
- 21 $E_{\text{occ}}(\mathbf{x}) \leftarrow E_{\text{occ}}(\mathbf{x}) + \frac{1}{N}L_i(\omega)$; // Eq. (10)
- 22 **end**
- 23 **end**
- 24 **end**
- 25 **if** $l_{\text{occ}}(c) = 0$ **then**
- 26 evaluate $E_{\text{unshad}}(\mathbf{n})$ by \mathbf{n} and \mathcal{E} ;
- 27 $E(\mathbf{x}) \leftarrow E_{\text{unshad}}(\mathbf{n}) - E_{\text{occ}}(\mathbf{x})$; // Eq. (11)

the two dimensions of the octahedral maps. As a result, each layer stores multiple octahedral maps with the same height. Since the visibility function is binary, we can store the visibility and occlusion information in just two bits. We use three bits of each pixel of the octahedral map to store the visibility mask, the occlusion mask, and the occlusion label, respectively.

In the fragment shader, we first locate the grid cell using the world-space fragment position, and the occlusion label is retrieved from the center pixel value of the octahedral map. We shoot and trace rays for each fragment. Based on the assumption of diffuse BRDF, the ray samples should follow a cosine-weighted distribution. The Hammersley sequence [Nie92] is used to create the low-discrepancy ray samples. To avoid the patterns caused by correlation, we rotate the ray samples along the surface normal by the values from a tiled blue noise texture [Pet16].

For each ray sample, we project the direction of the ray onto the unit square using octahedral mapping to fetch the corresponding

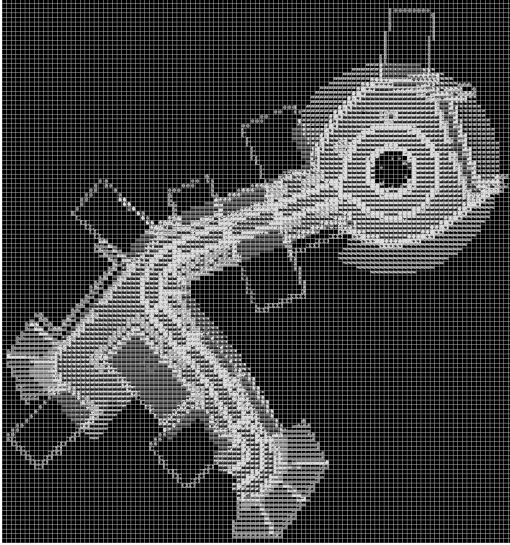


Figure 4: One layer of the texture array presenting the precomputed DVF of the Bistro scene. Each layer stores multiple tiled octahedral maps with the same height. Each pixel of the texture has 8 bits. The highest bit stores the visibility mask, the second highest bit stores the occlusion mask, and the third highest bit stores the occlusion label.

pixel from the octahedral map. Bitwise AND operations with 128 and 64 are performed on the fetched pixels to obtain the visible and occluded states from the visibility and occlusion mask of the grid cell. If the occlusion label is 1, we discard the ray if the fetched pixel is not visible in the visibility mask, and accumulate the incoming radiance from the environment map to obtain the incoming irradiance. If the occlusion label is 0, we discard the ray if the fetched pixel is not occluded in the occlusion mask and accumulate the incoming radiance occluded by the scene geometry to acquire the occluded irradiance. The occluded irradiance is then subtracted from the unshadowed irradiance to obtain the incoming irradiance, and we multiply the incoming irradiance by the diffuse albedo to obtain the outgoing radiance of the current fragment. In our implementation, we use the irradiance environment map represented by 3rd-order SH coefficients [RH01] to evaluate the incoming radiance from the environment map. Our runtime algorithm executed in the fragment shader is described in Algorithm 2.

5. Results and Discussion

To evaluate our proposed method, the rendering results of two scenes are exhibited in this section. The first scene is the Crytek Sponza (262 K triangles) [McG17] and the second scene is the exterior of the Amazon Lumbyard Bistro (2.8 M triangles) [Lum17]. The two scenes are both illuminated by the Uffizi Gallery environment map [Deb99]. We rendered all the images at a resolution of 1920×1080 on an NVIDIA GeForce RTX 3080 GPU. The size of the octahedral map is 18×18 and two additional pixels are padded around the border. The number of ray samples to precompute the

DVFs is 128. One layer of the texture array, which presents the precomputed DVF of the Bistro scene, is exhibited in Figure 4.

The comparison of the images rendered with DVFs to those rendered without DVFs, along with the rendering times and root mean squared error (RMSE), are presented in Figure 5. The columns from left to right are the textured rendering results with DVFs, the untextured rendering results with DVFs, the untextured rendering results without DVFs, and the error between rendered images with and without DVFs, respectively. The number of point samples in the precomputation stage is 10M. For the Crytek Sponza scene, the cell size of the DVF is 50 cm, the grid resolution of the DVF is $77 \times 48 \times 34$, the memory footprint of the DVF is 49,008 KB, and the number of ray samples per pixel at runtime is 128. For the Bistro scene, the cell size is 100 cm, the grid resolution is $111 \times 118 \times 34$, the memory footprint of the DVF is 173,958 KB, and the ray sample number is 64. We can see that the rendering results with DVFs are almost the same as those without DVFs. The error only appears in several pixels corresponding to geometry details. But the rendering time is significantly reduced.

We investigated the influence of the number of point samples in the precomputation stage. The point samples for precomputation, one layer of the DVFs, the images rendered by our method, the absolute error and RMSE between rendered images with and without DVFs, and the computation times of the precomputation stage in the Crytek Sponza scene using different numbers of point samples are given in Figure 6. The cell size of the DVF and the number of ray samples per pixel are the same as those of previous results. The precomputation is performed on an Intel Core i9-10980XE CPU. The results indicate that the RMSE reduces as the number of point samples gets larger. However, the time spent on precomputation gets longer when increasing the number of point samples. However, the precomputation is still fast even when 10M point samples are utilized. Besides, the rendering time is longer when the number of point samples is larger because several rays that need to be traced are wrongly discarded when the number of point samples is small.

The cell size of DVFs can also affect the rendering results. We compare the rendered images without textures by our method, the absolute error and RMSE between rendered images with and without DVFs, the rendering times, and the memory consumptions of the DVFs with different cell sizes in the Crytek Sponza scene as shown in Figure 7. The number of point samples is 10M and the number of ray samples is 128. We can see that when the cell size gets larger, both the memory consumption of the DVFs and the RMSE between rendered images with and without DVFs reduce at the price of longer time spent at runtime. The error is negligible when the grid size is larger than 100 cm.

We also compare the images rendered by our method along with the rendering times using different numbers of ray samples per pixel (spp) in the Crytek Sponza scene as exhibited in Figure 8. The rendering times without the DVF are also listed along with the speedup ratios in 1. The number of point samples is 10M and the cell size of the DVF is 50 cm. We can see from the rendering times that the speedup ratio is low when the number of ray samples is small, perhaps because other computations besides ray tracing take a larger percentage of time.

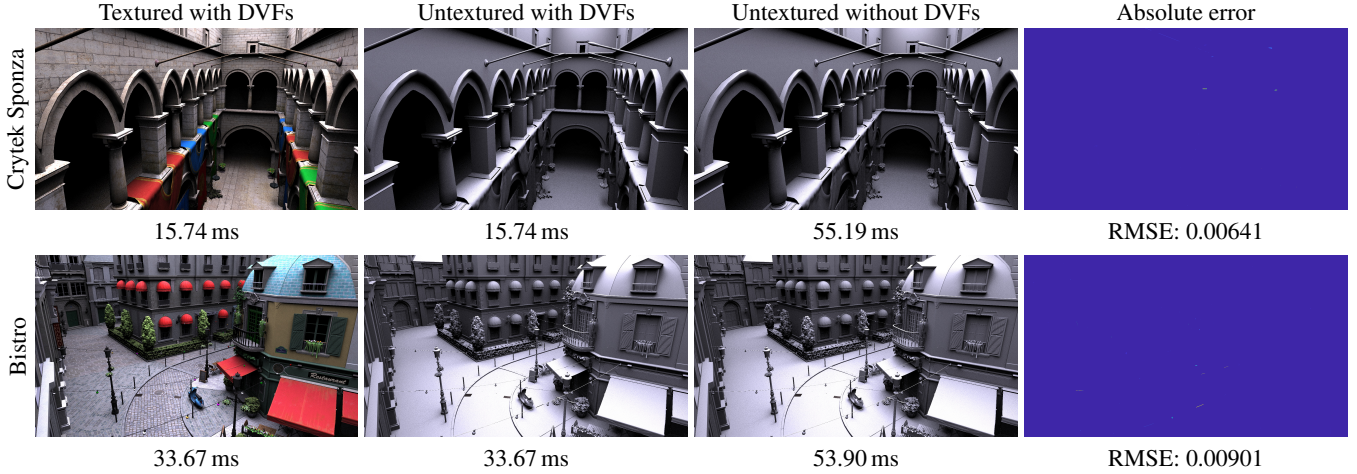


Figure 5: Comparisons of the rendering results with DVFs to those without DVFs in two scenes. From left to right: textured results with DVFs, untextured results with DVFs, untextured results without DVFs, and the error between rendered images with and without DVFs. Upper row: Crytek Sponza. Lower row: Bistro.

Table 1: Rendering time (in milliseconds) comparison of rendering results with and without the DVF using different ray samples per pixel.

spp	16	32	64	128
w/o the DVF	6.69	13.79	27.28	55.19
w/ the DVF	2.23	4.02	7.81	15.74
Speedup ratio	3.12	3.43	3.49	3.51

To demonstrate that determining whether the visible or the occluded regions should be traced using the occlusion labels stored in the DVFs can achieve optimal performance, we compare the rendered images and rendering times with occlusion labels to those only tracing the visible regions in the Bistro scene from a viewpoint on the roof. The rendered images and the visualized occlusion labels (white denotes more pixels are occluded in the occlusion masks) are displayed in Figure 9. The number of point samples in the precomputation stage is 10 M, the cell size of the DVF is 100 cm, and the number of ray samples per pixel at runtime is 64. Since most of the hemispherical domain is visible at a surface point of the roof, no rays may be excluded if we trace rays in the visible region. In this case, the occlusion labels can be introduced to determine whether the visible or occluded regions should be traced. Since the occlusion labels are 0 on the roof as shown in Figure 9 right, we trace rays in the occluded region instead, which reduces the rendering time.

6. Limitations and Future Work

Dynamic Scenes. Our method cannot support dynamic scenes because we precompute the DVF on the CPU. Scenes can be animated at runtime if the precomputation process is implemented on the GPU. With hardware-accelerated ray tracing, fast precomputation at runtime is possible. To perform the precomputation at runtime,

the algorithm to generate point samples should also be adapted to the GPU.

Indirect Illumination and Local Area Lights. We ignore indirect illumination and local area lights. They are not supported in this work because the proposed DVFs store visibility information about the environment, which cannot be applied to speed up the computation of indirect illumination and the lighting of local area lights. Further research aiming to speed up rendering indirect illumination and the lighting of local area lights will be conducted in the future.

All-Frequency Environment Maps. In this work, we only use low-frequency environment maps represented by low-order SHs as the environment light sources. We will adapt our method to efficiently render environment lighting under all-frequency environment maps in the future.

Glossy BRDFs. We assume diffuse BRDF in this work, but the adoption of glossy BRDFs is straightforward. We only need to generate the ray samples according to the glossy BRDFs of the surface points, and ignore the occlusion labels in the DVFs to trace in the visible regions for all surface points. We give the rendering results with a glossy BRDF and the comparison of the rendering times without and with DVFs in Figure 10. For the Crytek Sponza scene, the rendering time can be reduced from 54.64 ms to 10.14 ms by the DVF, and 48.75 ms to 27.16 ms for the Bistro scene. Both speedup ratios are even larger than those with diffuse BRDFs.

Point Samples Generation. In this work, we generate point samples of the scene by random sampling. However, random sampling may lead to errors compared to the rendering results without DVFs because the geometry details may be missed by sampling, especially when the number of point samples is not sufficiently high. The point samples should be generated by a smarter sampling strategy to avoid artifacts due to limited sample numbers.

Compression. We use only three bits of each pixel in the octahedral map, and the one bit storing the occlusion label in an octahedral map is wasteful because each grid cell only needs one occlusion

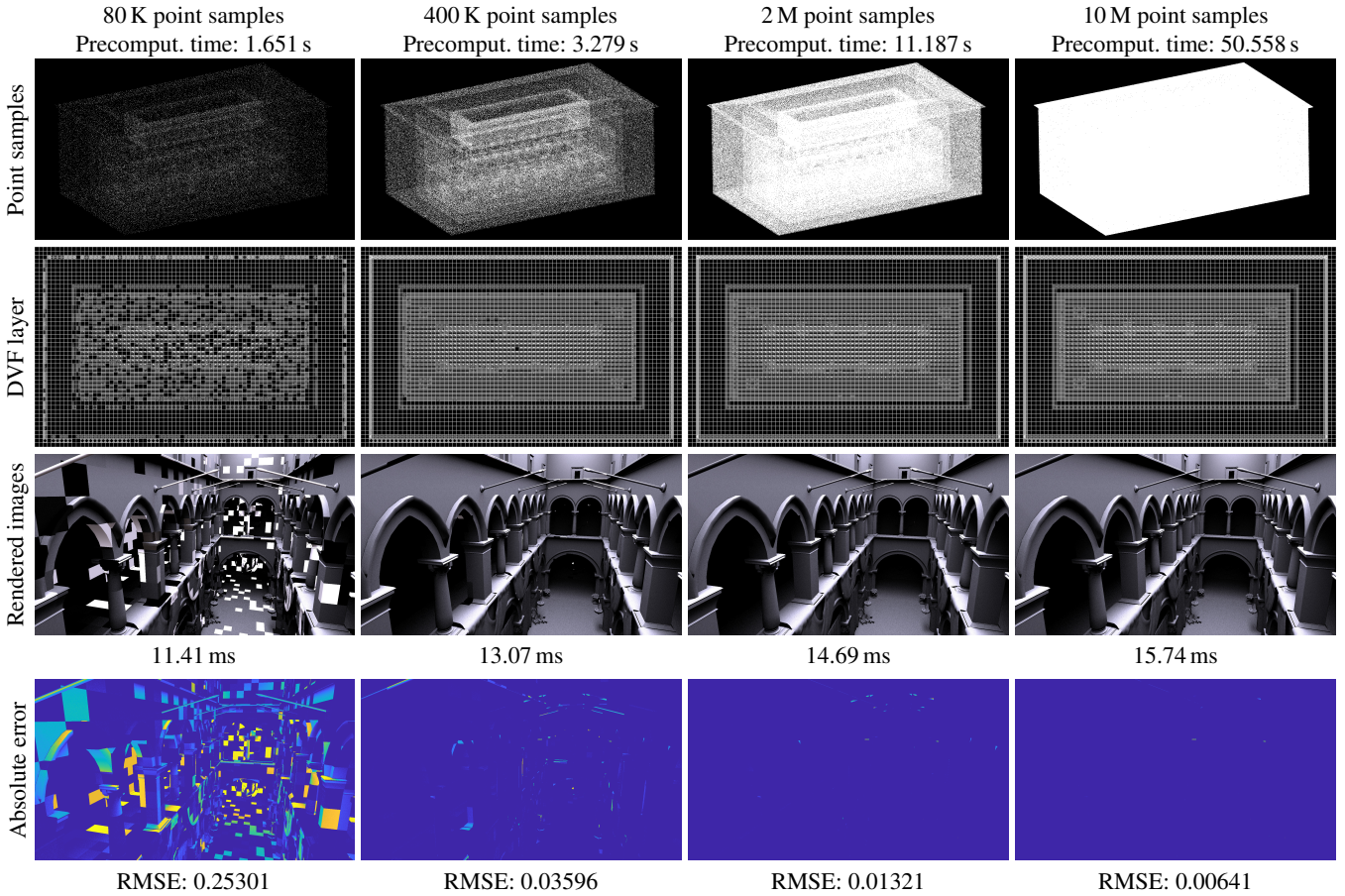


Figure 6: Comparison of the rendering results without textures in the Crytek Sponza scene using different numbers of point samples used in the precomputation stage. From left to right: 80 K, 400 K, 2 M, and 10 M point samples. From top to bottom: point samples, one layer of the DVFs, images rendered by our method, and absolute error between rendered images with and without DVFs.

label. We can pack multiple pixels of an octahedral map into one pixel of a texture layer to reduce the memory footprint of the DVF. We can also exploit the sparseness of the DVFs and store the DVFs in *sparse voxel octrees* [LK10] because a large number of pixels in the octahedral maps of the DVFs are zero. Besides, many neighborhood pixels in the octahedral maps have the same values, which can also benefit compression.

Denosing. Since our proposed method can significantly speed up ray-traced environment lighting to allow the use of a large number of ray samples, noises still exist and can be suppressed to obtain better results. SVGF [SKW*17] or learning-based denoiser [MZV*20,FWHB21,IMF*21] can be applied to our method.

Irradiance Caching. Irradiance caching computes irradiance at sparse locations in the scene and interpolates these irradiance samples to obtain the final results. Therefore, the number of scene points that need to be shaded by ray tracing can be significantly reduced. Screen-space irradiance caching [WWZ*09] can be introduced in our work to further improve rendering efficiency.

7. Conclusion

We present *discrete visibility fields*, which store the visibility and occlusion masks in a uniform grid to speed up ray-traced direct illumination of low-frequency environment lighting for static scenes by excluding the unnecessary rays. At precomputation time, we first construct a uniform grid for the scene and generate point samples of the scene. We then create visibility and occlusion masks at the positions of the point samples and merge the visibility and occlusion masks of the point samples inside the grid by the logical OR operation. We also store a label indicating whether more than half of the directions are occluded in the occlusion mask of each grid cell. The octahedral parameterization is adopted to represent the visibility and occlusion masks. At runtime, we exclude the rays that are not visible to the environment or the rays that are not occluded by the scene geometry according to the visibility and occlusion masks in the DVF, which can significantly reduce the rendering times.

Our method is simple to implement, and the precomputation time is short. Compared to the existing environment lighting rendering methods based on precomputation, the main advantage of our

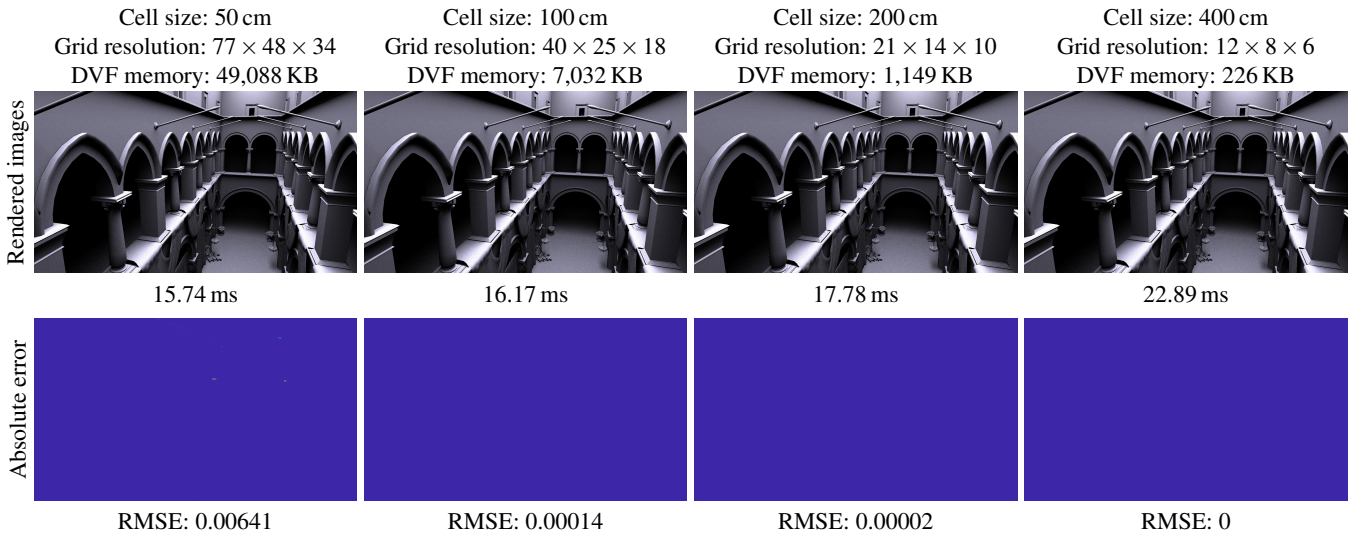


Figure 7: Comparison of the rendering results without textures in the Crytek Sponza scene using DVFs with different cell sizes. From left to right: 50, 100, 200, and 400 cm. Upper row: rendering results of our method. Lower row: absolute error between rendered images with and without DVFs.



Figure 8: Comparison of the rendering results without textures in the Crytek Sponza scene using different numbers of ray samples per pixel. From left to right: 16, 32, 64, and 128 spp.

method is that we can produce comparable results to traditional ray tracing because we only discard the unnecessary rays during ray tracing. Besides, our method is free of tessellation or parameterization of the meshes.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. This work was supported by the National Key R&D Program of China (2020YFC1523303, 2019YFC1521102), the Key Research and Development Program of Shaanxi (2019ZDLSF07-02, 2019GY-215), the Major Research and Development Project of Qinghai (2020-SF-143), and the Scientific Research Program Funded by Shaanxi Provincial Education Department (21JK0931).

References

[ARB90] AIREY J. M., ROHLF J. H., BROOKS F. P.: Towards image realism with interactive update rates in complex virtual building environments. *SIGGRAPH Comput. Graph. (Proc. I3D '90)* 24, 2 (Mar. 1990), 41–50. doi:10.1145/91385.91416.3

[ARBJ03] AGARWAL S., RAMAMOORTHY R., BELONGIE S., JENSEN H. W.: Structured importance sampling of environment maps. *ACM Trans. Graph. (Proc. SIGGRAPH '03)* 22, 3 (July 2003), 605–612. doi:10.1145/882262.882314.3

[BBH13] BARÁK T., BITTNER J., HAVRAN V.: Temporally coherent adaptive sampling for imperfect shadow maps. *Comput. Graph. Forum (Proc. EGSR '13)* 32, 4 (July 2013), 87–96. doi:10.1111/cgf.12154.

[BGH05] BURKE D., GHOSH A., HEIDRICH W.: Bidirectional importance sampling for direct illumination. In *Proc. EGSR '05* (2005), pp. 147–156. doi:10.2312/EGWR/EGSR05/147-156.3

[BMW*09] BITTNER J., MATTAUSCH O., WONKA P., HAVRAN V., WIMMER M.: Adaptive global visibility sampling. *ACM Trans. Graph. (Proc. SIGGRAPH '09)* 28, 3 (July 2009), 94. doi:10.1145/1531326.1531400.3

[BWP*20] BITTERLI B., WYMAN C., PHARR M., SHIRLEY P., LEFOHN A., JAROSZ W.: Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Trans. Graph. (Proc. SIGGRAPH '20)* 39, 4 (July 2020), 148. doi:10.1145/3386569.3392481.3

[CAM08a] CLARBERG P., AKENINE-MÖLLER T.: Exploiting visibility correlation in direct illumination. *Comput. Graph. Forum (Proc. EGSR '08)* 27, 4 (Sept. 2008), 1125–1136. doi:https://doi.org/10.1111/j.1467-8659.2008.01250.x.2,3

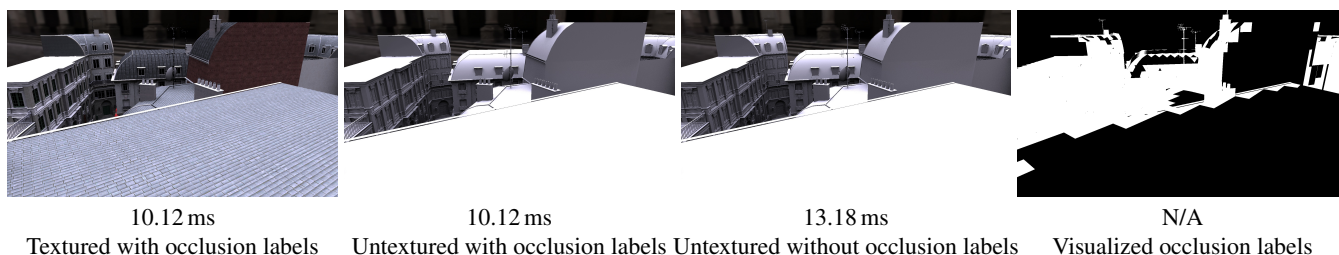


Figure 9: Comparison of the rendering results in the Bistro scene with and without occlusion labels stored in the DVFs. Left to right: Visualized occlusion labels, textured rendered image with occlusion labels to decide whether the visible or the occluded regions should be traced, untextured rendered image with occlusion labels, and untextured rendered image with rendered image without occlusion labels, which traces rays in the visible regions.

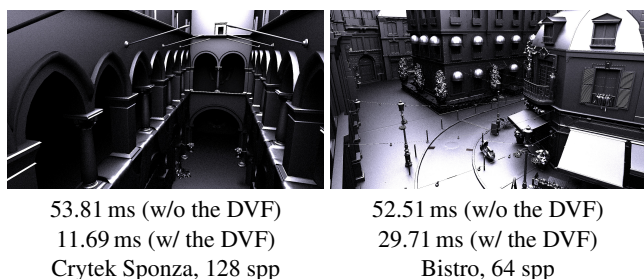


Figure 10: Rendering results with a glossy BRDF in two scenes. Left: Crytek Sponza. Right: Bistro.

[CAM08b] CLARBERG P., AKENINE-MÖLLER T.: Practical product importance sampling for direct illumination. *Comput. Graph. Forum (Proc. Eurographics '08)* 27, 2 (Apr. 2008), 681–690. doi:10.1111/j.1467-8659.2008.01166.x. 3

[CDE*14] CIGOLLE Z. H., DONOW S., EVANGELAKOS D., MARA M., MCGUIRE M., MEYER Q.: A survey of efficient representations for independent unit vectors. *J. Comput. Graph. Tech.* 3, 2 (April 2014), 1–30. URL: <http://jcgt.org/published/0003/02/01/5>

[CJAMJ05] CLARBERG P., JAROSZ W., AKENINE-MÖLLER T., JENSEN H. W.: Wavelet importance sampling: efficiently evaluating products of complex functions. *ACM Trans. Graph. (Proc. SIGGRAPH '05)* 24, 3 (July 2005), 1166–1175. doi:10.1145/1073204.1073328. 3

[CRW09] CLINE D., RAZDAN A., WONKA P.: A comparison of tabular PDF inversion methods. *Comput. Graph. Forum* 28, 1 (Feb. 2009), 154–160. doi:10.1111/j.1467-8659.2008.01197.x. 3

[DDTP00] DURAND F., DRETTAKIS G., THOLLOT J., PUECH C.: Conservative visibility preprocessing using extended projections. In *Proc. SIGGRAPH '00* (2000), p. 239–248. doi:10.1145/344779.344891. 3

[Deb99] DEBEVEC P.: Light probe image gallery, 1999. URL: <https://www.pauldebevec.com/Probes/>

[FWHB21] FAN H., WANG R., HUO Y., BAO H.: Real-time Monte Carlo denoising with weight sharing kernel prediction network. *Comput. Graph. Forum (Proc. EGSR '21)* 40, 4 (July 2021), 15–27. doi:10.1111/cgf.14338. 2, 3, 9

[GEE20] GUO J. J., EISEMANN M., EISEMANN E.: Next event estimation+: Visibility mapping for efficient light transport simulation. *Comput. Graph. Forum (Proc. Pacific Graphics '20)* 39, 7 (Nov. 2020), 205–217. doi:10.1111/cgf.14138. 2, 3

[GN15] GIRAUD A., NOWROUZEZHAI D.: Practical Shading of Height Fields and Meshes using Spherical Harmonic Exponentiation. In *Proc. EGSR '15* (2015), p. 1–8. doi:10.2312/sre.20151161. 3

[GSHG98] GREGER G., SHIRLEY P., HUBBARD P. M., GREENBERG D. P.: The irradiance volume. *IEEE Comput. Graph. Appl.* 18, 2 (Mar. 1998), 32–43. doi:10.1109/38.656788. 2

[IMF*21] IŞIK M., MULLIA K., FISHER M., EISENMANN J., GHARBI M.: Interactive Monte Carlo denoising using affinity of neural features. *ACM Trans. Graph. (Proc. SIGGRAPH '21)* 40, 4 (July 2021), 37. doi:10.1145/3450626.3459793. 2, 3, 9

[JKG16] JENDERSIE J., KURI D., GROSCH T.: Precomputed illumination composition for real-time global illumination. In *Proc. I3D '16* (2016), p. 129–137. doi:10.1145/2856400.2856407. 2

[KHBW20] KOCH D., HECTOR T., BARCZAL J., WERNESSE E.: Ray tracing in Vulkan, Dec. 2020. URL: <https://www.khronos.org/blog/ray-tracing-in-vulkan>. 6

[KK03] KOLLIG T., KELLER A.: Efficient illumination by high dynamic range images. In *Proc. EGSR '03* (2003), pp. 45–51. doi:10.2312/EGWR/EGWR03/045-051. 3

[LK10] LAINE S., KARRAS T.: Two methods for fast ray-cast ambient occlusion. *Comput. Graph. Forum (Proc. EGSR '10)* 29, 4 (Aug. 2010), 1325–1333. doi:10.1111/j.1467-8659.2010.01728.x. 9

[Lum17] LUMBERYARD A.: Amazon Lumberyard Bistro, open research content archive (ORCA), July 2017. URL: <http://developer.nvidia.com/orca/amazon-lumberyard-bistro>. 7

[LZT*08] LEHTINEN J., ZWICKER M., TURQUIN E., KONTKANEN J., DURAND F., SILLION F., AILA T.: A meshless hierarchical representation for light transport. *ACM Trans. Graph. (Proc. SIGGRAPH '08)* 27, 3 (Aug. 2008), 37. doi:10.1145/1360612.1360636. 2

[McG17] MCGUIRE M.: Computer graphics archive, July 2017. URL: <https://casual-effects.com/data>. 7

[MMNL17] MCGUIRE M., MARA M., NOWROUZEZHAI D., LUEBKE D.: Real-time global illumination using precomputed light field probes. In *Proc. I3D '17* (2017), p. 2. doi:10.1145/3023368.3023378. 2

[MZV*20] MENG X., ZHENG Q., VARSHNEY A., SINGH G., ZWICKER M.: Real-time Monte Carlo denoising with the neural bilateral grid. In *Proc. EGSR '20* (2020). doi:10.2312/sr.20201133. 9

[Nie92] NIEDERREITER H.: *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, 1992. doi:10.1137/1.9781611970081. 6

[NVI18] NVIDIA: *NVIDIA Turing GPU Architecture*. WP-09183-001_v01, 2018. 1

[ODJ04] OSTROMOUKHOV V., DONOHUE C., JODOIN P.-M.: Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph. (Proc. SIGGRAPH '04)* 23, 3 (Aug. 2004), 488–495. doi:10.1145/1015706.1015750. 3

- [Pet16] PETERS C.: Free blue noise textures, Dec. 2016. URL: <https://momentsingraphics.de/BlueNoise.html>. 6
- [PGSD13] POPOV S., GEORGIEV I., SLUSALLEK P., DACHSBACHER C.: Adaptive quantization visibility caching. *Comput. Graph. Forum (Proc. Eurographics '13)* 32, 2pt4 (May 2013), 399–408. doi:10.1111/cgf.12060. 2, 3
- [RGK*08] RITSCHER T., GROSCH T., KIM M. H., SEIDEL H. P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph. (Proc. SIGGRAPH Asia '08)* 27, 5 (Dec. 2008), 129. doi:10.1145/1409060.1409082. 2
- [RH01] RAMAMOORTHY R., HANRAHAN P.: An efficient representation for irradiance environment maps. In *Proc. SIGGRAPH '01* (2001), p. 497–500. doi:10.1145/383259.383317. 7
- [RWS*06] REN Z., WANG R., SNYDER J., ZHOU K., LIU X., SUN B., SLOAN P.-P., BAO H., PENG Q., GUO B.: Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM Trans. Graph. (Proc. SIGGRAPH '06)* 25, 3 (July 2006), 977–986. doi:10.1145/1141911.1141982. 2
- [SDDS00] SCHAUFLEER G., DORSEY J., DÉCORET X., SILLION F. X.: Conservative volumetric visibility with occluder fusion. In *Proc. SIGGRAPH '00* (2000), pp. 229–238. doi:10.1145/344779.344886. 3
- [SGNS07] SLOAN P.-P., GOVINDARAJU N. K., NOWROUZEZAHRAI D., SNYDER J.: Image-based proxy accumulation for real-time soft global illumination. In *Proc. Pacific Graphics '07* (Oct. 2007), pp. 97–105. doi:10.1109/PG.2007.28. 2
- [SHHS03] SLOAN P.-P., HALL J., HART J., SNYDER J.: Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph. (Proc. SIGGRAPH '03)* 22, 3 (July 2003), 382–391. doi:10.1145/882262.882281. 2
- [SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph. (Proc. SIGGRAPH '02)* 21, 3 (July 2002), 527–536. doi:10.1145/566654.566612. 2
- [SKW*17] SCHIED C., KAPLANYAN A., WYMAN C., PATNEY A., CHAITANYA C. R. A., BURGESS J., LIU S., DACHSBACHER C., LEFOHN A., SALVI M.: Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination. In *Proc. HPG '12* (2017), p. 2. doi:10.1145/3105762.3105770. 2, 3, 9
- [SSS*20] SEYB D., SLOAN P.-P., SILVENNOINEN A., IWANICKI M., JAROSZ W.: The design and evolution of the UberBake light baking system. *ACM Trans. Graph. (Proc. SIGGRAPH '20)* 39, 4 (July 2020). doi:10.1145/3386569.3392394. 2
- [SSSK04] SZÉCSI L., SBERT M., SZIRMAY-KALOS L.: Combined correlated and importance sampling in direct light source computation and environment mapping. *Comput. Graph. Forum (Proc. Eurographics '04)* 23, 3 (Aug. 2004), 585–593. doi:10.1111/j.1467-8659.2004.00790.x.
- [TCE05] TALBOT J. F., CLINE D., EGBERT P.: Importance resampling for global illumination. In *Proc. EGSR '05* (2005), pp. 139–146. doi:10.2312/EGWR/EGSR05/139-146. 3
- [TS91] TELLER S. J., SÉQUIN C. H.: Visibility preprocessing for interactive walkthroughs. *SIGGRAPH Comput. Graph. (Proc. SIGGRAPH '91)* 25, 4 (July 1991), 61–70. doi:10.1145/122718.122725. 3
- [VG95] VEACH E., GUIBAS L. J.: Optimally combining sampling techniques for Monte Carlo rendering. In *Proc. SIGGRAPH '95* (1995), pp. 419–428. doi:10.1145/218380.218498. 3
- [Whi80] WHITTED T.: An improved illumination model for shaded display. *Commun. ACM* 23, 6 (June 1980), 343–349. doi:10.1145/358876.358882. 1
- [Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph. (Proc. SIGGRAPH '78)* 12, 3 (Aug. 1978), 270–274. doi:10.1145/965139.807402. 1
- [WWS00] WONKA P., WIMMER M., SCHMALSTIEG D.: Visibility preprocessing with occluder fusion for urban walkthroughs. In *Proc. EGWR '00* (2000), pp. 71–82. doi:10.1007/978-3-7091-6303-0_7. 3
- [WWZ*09] WANG R., WANG R., ZHOU K., PAN M., BAO H.: An efficient gpu-based approach for interactive global illumination. *ACM Trans. Graph. (Proc. SIGGRAPH '09)* 28, 3 (July 2009), 91. doi:10.1145/1531326.1531397. 9
- [ZHL*05] ZHOU K., HU Y., LIN S., GUO B., SHUM H.-Y.: Precomputed shadow fields for dynamic scenes. *ACM Trans. Graph. (Proc. SIGGRAPH '05)* 24, 3 (July 2005), 1196–1201. doi:10.1145/1073204.1073332. 2