# Interactive Global Illumination Effects Using Deterministically Directed Layered Depth Maps

F. P. Aalund[†], J. R. Frisvad, and J. A. Bærentzen

Technical University of Denmark

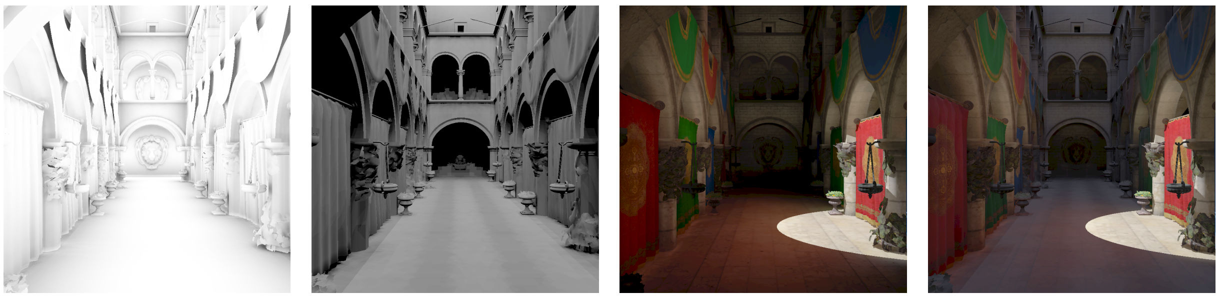


Figure 1: All images are generated using rasterization and layered depth maps. From left to right: Ambient obscurance, ambient occlusion, single-bounce indirect lighting, and environment lighting combined with indirect lighting.

**Abstract**

*A layered depth map is an extension of the well-known depth map used in rasterization. Multiple layered depth maps can be used as a coarse scene representation. We develop two global illumination methods which use said scene representation. The first is an interactive ambient occlusion method. The second is an interactive single-bounce indirect lighting method based on photon differentials. All of this is implemented in a rasterization-based pipeline.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

## 1. Introduction

An efficient solution to the problem of rendering with global illumination is perhaps the single problem that has received most attention in real-time graphics recently. While great inroads have been made, most methods are still limited and require the combination of several elements — which in turn are usually based on rasterization or ray tracing.

Despite the many similarities between rasterization and ray-tracing, the traversal order is a distinctive difference. Rasterization only requires *local* scene information (a single primitive) each step. Ray-tracing, however, requires *global* scene information (all primitives) each step. Consequently, rasterization has a smaller memory footprint for complex scenes. This is a key advantage of rasterization that is pivotal to the way modern GPUs operate [AMHH08]. Simultaneously, this is a significant limitation of rasterization since only local information can be used in shading.

Overcoming the local information limitation of rasterization in real-time rendering has been the focus of recent research [RGS09, CNS*11, NRS14, ZRD14]. Common for these methods is the use of an auxiliary data structure

[†] e-mail: frederikaalund+ldm2015@gmail.com

which contains a coarse representation of the scene geometry. Programmable GPU features such as fragment shaders are adapted to construct said data structures in real-time.

During rasterization, a shader can then query the auxiliary data structure for global scene information. The latter can be used to implement global illumination and thus overcome the local limitation of rasterization. This is an ideal combination of the performance characteristics of rasterization with the physical correctness of global illumination.

As in some previous work [KBW06, BHKW07, ZHS08, NSS10, HHZ*14], we use an auxiliary data structure based on layered depth maps (LDMs). Layered, in the sense that all depth values (not just a single one) are stored in the map. The novelty in our approach is that each LDM is pre-sorted which in turn allows for a fast tracing algorithm.

We also present two global illumination methods which use rasterization in combination with our auxiliary data structure: Ambient occlusion and single-bounce indirect lighting. These methods are meant to demonstrate the applicability of our auxiliary data structure. We use a path traced reference to evaluate the image quality of our results. Furthermore, we compare the ambient occlusion implementation with a screen-space approach.

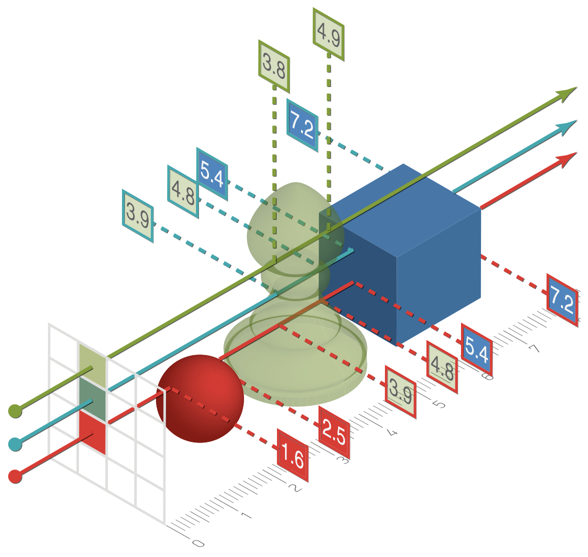


Figure 2: A layered depth map (LDM). Each pixel stores multiple depth values. Here, three view rays are shown. The depth values are given in the colored squares along each view ray. Both front and back faces are rasterized. Note that multiple fragments may map to the same pixel.

## 2. Related Work

First, previous work on LDMs in the context of transparency is presented. Second, related indirect lighting methods are described.

### 2.1. LDMs

LDMs were invented to solve the issue of rendering transparent objects in a rasterization pipeline [MCTB11]. Specifically, to implement so-called order-independent transparency (OIT). The primitives are sent through the pipeline in any order and the LDM will automatically depth-sort the corresponding fragments (Figure 2). Previous work on LDMs in the context of OIT can be readily applied to our use case.

**Early Approaches.** LDMs were introduced with the anti-aliased, area-averaged, accumulation buffer (A-buffer) [Car84]. The A-buffer is essentially a per-pixel singly linked list of fragments sorted according to depth. The A-buffer was introduced before the emergence of modern GPUs and the proposed implementation is meant for offline use in the REYES system. The $Z^3$ data structure seeks to improve on the anti-aliasing of the A-buffer by also storing image-space derivatives of the depth value [JC99]. Furthermore, the authors suggest to store a constant $k$ fragments per pixel.

The recirculating fragment buffer (R-buffer) is a pointer-less derivative of the A-buffer. [Wit01]. The R-buffer is essentially a FIFO buffer of the incoming fragments. In other words, the R-buffer provides unique storage for all fragments. The fragment-stream buffer (F-buffer) is a new render target that stores all incoming fragments in a FIFO buffer [MP01]. It is a general proposal meant to be used with a programmable pipeline (such as OpenGL 2.x and above).

A LDM with a constant, $k$, number of layers is called a $k$-buffer [CICS05, BCL*07, MB07]. Note that $Z^3$ already introduced this idea [JC99]. All $k$-buffers use multiple render targets (MRT) to capture multiple fragments at a time. The various implementations use different tricks to sort and store multiple fragments per pixel. E.g., by using read-modify-write operations [CICS05] or by exploiting multisample textures and stencil routing [MB07].

**Modern Approaches.** The previous implementations were limited by the hardware of their time. On graphics cards, LDMs were originally constructed using the so called depth peeling method which employed the shadow map depth test to "peel off" each layer of the LDM [Eve01]. Atomic operations and shader storage buffer objects (SSBOs) introduced with the OpenGL 4.3 Core Profile [SAF*13] allow for more advanced implementations. One example is the use of an SSBO to store per-pixel fixed-length arrays (PPFLA) of depth values [LHLW09, LHLW10, Cra10a]. This approach is similar to $Z^3$ and the $k$-buffer. That is, it has constant constant memory requirements (up to a fixed $k$) but with a different implementation that uses newer hardware features. Depth-sorting is done as a post-processing pass on the per-pixel arrays. The sorting pass can be skipped by depth-sorting during construction in order to construct pre-sorted per-pixel fixed-length arrays (PSPPFLA) [LHLW09, LHLW10].

Inspired by the A-buffer, per-pixel singly linked lists (PP-SLL) can be constructed using OpenGL 4.x [YHGT10, YM10, GT10, Thi11]. Paged per-pixel singly linked lists (PPPSLL) store multiple depth values per node which decreases the memory overhead [Cra10b]. Insertion sort can be during construction of PPSLL to get pre-sorted per-pixel singly linked lists (PSPPSLL) [LHL14].

The layered buffer or list buffer (l-buffer) is a pointer-less A-buffer derivative [Lip10]. Depth values are stored in per-pixel contiguous variable-length arrays thus not requiring any pointer indirection. The downside of the l-buffer is the complicated construction wich requires at least seven passes [Lip10]. The l-buffer has been optimized in several iterations. First, a simpler, 4-step construction method with the dynamic fragment buffer (DF-buffer) using a parallel prefix-sum algorithm [CTBM12]. Second, an alternative parallel prefix-sum implementation optimized for pixel sparsity (when many pixels are without fragments) called the sparsity-aware buffer (S-buffer) [VF12]. Third, the dequeue buffer (D-buffer) presents various micro-optimizations [Lip13]. The D-buffer can be pre-sorted in order to construct pre-sorted per-pixel variable-length arrays (PSPPVLA) [Kub14].

The hashed A-buffer (HA-buffer) is a hash map of depth values [LHL13]. The hash table itself is actually a simple contiguous array of entries stored in an SSBO. It is the operations on said array that define the hash map. One of the key benefits of the HA-buffer is that the depth-sorting can be combined with the hash function. As such, the HA-buffer can be constructed in a single pass over the scene geometry.

## 2.2. Real-time Global Illumination

First, previous work on interactive ambient occlusion is presented. Second, general indirect lighting methods are described.

**Ambient Occlusion.** Ambient occlusion (AO) [ZIK*98, Lan02] is a visibility term in the rendering equation used to attenuate ambient lighting. In offline rendering, AO is often computed using a Monte Carlo integration over directions in the unit hemisphere [PH04]. In real-time rendering, the simplest solution is to use a constant AO term [AMHH08]. This approach avoids computing the integral altogether which is the cheapest option performance-wise. However, the result is flat-shaded surfaces since the constant model lacks any kind of directionality.

Recently, screen-space (or image-space) AO methods have been used in real-time rendering. Said methods typically use the depth map as a coarse scene representation from which global information can be queried [SA07, Mit07]. These methods are known as SSAO methods. The early SSAO approaches estimate AO by using a point sampling strategy [SA07, Mit07]. Each point corresponds to a depth buffer look-up which determines whether the point is occluded or not. The average of all such queries is used as the AO term. A later screen-space approach approximates AO as a function of the unoccluded horizon (HBAO) [BSD08]. This is done by ray-marching the depth buffer in screen-space. The ray-marching makes HBAO slower than earlier SSAO methods but HBAO produces high quality results. Some SSAO methods use multi-view or layered depth buffers to get more scene information [SA07, VPG13].

There are many other variations of SSAO [FM08, LS10, MOBH11, Mit12]. Common for them all is that they fit within a real-time rendering budget both in terms of performance and memory use. We have similar requirements for our AO model. We therefore implement a screen-space method (HBAO) that we can compare with our method based on LDMs.

Another approach is to compute a dynamic sparse voxel octree during primitive traversal. The octree serves as a coarse scene representation and can be queried efficiently with voxel cone tracing to approximate AO [CNS*11]. However, voxel cone tracing is limited to diffuse and glossy light models. In contrast, LDMs can be used to trace specular reflections and refractions.

**Indirect Lighting.** Indirect lighting covers all lighting beyond the initial bounce. In offline rendering, path tracing is often used to compute indirect lighting [PH04]. Reflective shadow maps (RSMs) is a real-time virtual point light (VPL) approach that models single-bounce indirect lighting [DS05]. A RSM augments the shadow map by also storing the radiant flux, surface normal, and surface position. As such, each pixel in the reflective shadow map is a VPL representing an LD path (using light transport notation [Hec90]). Reflected radiance from the camera's view is then integrated in screen-space. That is, nearby pixels (nearby VPLs are sampled in a fragment shader and the result is accumulated (LDDE paths). Imperfect shadow maps (ISMs) are coarse approximations of shadow maps [RGK*08]. As such, ISMs can be generated much faster. ISMs can be combined with RSMs to compute the visibility term of each VPL. This removes the light leak problem of RSMs. Imperfect reflective shadow maps (IRSMs) [RGK*08] are the ISM analogues to RSM. By augmenting each of the ISM with light information, multiple light bounces of indirect light can be computed (LD+E paths). Of course, this adds additional complexity to the method and thus degrades performance. LDMs can be combined with RSMs in so-called layered reflective shadow maps (LRSMs) [SRS14]. This allows light data to be decoupled from a voxel-based representation of the scene geometry.

Global ray-bundles can be used in a combined Monte Carlo and finite element method to compute indirect lighting

[SS96]. This method introduced the idea of tracing bundles of parallel rays. Later methods (including ours) use modern hardware features to further improve the performance of the global ray-bundle technique [SKP98, HHGM10].

A hybrid VPL and path tracing approach can be implemented using LDMs [TO12b]. The idea is to combine the two approaches into a single bidirectional algorithm. To do so, a RSM is first used to generate the VPLs along with a regular shadow map for each VPL (forming LD paths). Then, global ray-bundles are generated using LDMs (forming DD paths). Lastly, DE paths are rendered from the camera into a G-buffer. Any combination of the above-mentioned paths can be connected. E.g., an LDDDE path by tracing from the eye to the first surface (using the G-buffer), then reflecting towards another surface (using a LDM), and lastly towards a VPL (using the corresponding shadow map for visibility). This results in a two-bounce global illumination method. Additional bounces can be added by tracing the LDMs again for more DD paths. That is, LD*E paths are possible though at the cost of performance for each additional bounce. The authors suggest to use PPSLL to implement the LDMs [TO12b].

The principle behind ISMs can also be applied to ray-bundle tracing [TO12a]. In this approach, the LDMs are generated from coarse scene representation (using the ISM reduction technique). This results in faster LDM construction. However, light leaks can now occur since the scene representation is much coarser.

In the aforementioned methods, the LDMs have been used to trace rays in the direction which the LDM is oriented. This requires the construction of a LDM for each direction. Another approach is to trace in arbitrary directions by ray-marching through each LDM [LR98, BHKW07]. This approach is analogous to ray-marching a depth map (as done in HBAO). This adds additional complexity to the tracing algorithm. On the other hand, fewer LDMs need to be generated. In fact, only three LDMs in orthogonal directions are necessary; a so-called layered depth cube [LR98]. When tracing in direction, $\omega$, the LDM which is oriented closest to $\omega$ is chosen. Then the chosen LDM is ray-marched. In practice, rays that are almost orthogonal to the chosen LDM will be costly to ray-march. The more LDMs, the less so-called pixel crossings (and the faster performance) [NSS10]. As such, it is sometimes more performant to use more than the theoretical minimum number of three orthogonal LDMs in practice. The optimal number depends on the underlying implementation. The ray-marching scheme is not perfect. Rays may miss intersections for steep depth values. A tolerance threshold for the intersection test can mitigate this issue [NSS10]. Applications include: Whitted ray-tracing of reflections (offline) [BHKW07] and refractions (real-time) [UPSK07], glossy reflections and soft shadows (offline) [NSS10], caustic photon tracing (interactive) [KBW06], and path tracing (real-time) [HHZ*14]. We do not use ray-marching but trace in directions parallel to each LDM. This way, no intersections are missed and no expensive pixel crossings occur. However, a LDM must be generated for each sample direction which may be expensive.

Layered screen-space information can also be used for global illumination with the so-called deep G-buffer [MMNL14]. A two-layered G-buffer is used to approximate both indirect lighting and ambient occlusion. Deep screen space [NRS14] is another recent approach which extends screen-space information. Here, each primitive is tessellated into surface elements (surfels) and shading effects are applied by splatting said surfels.

## 3. Method

The idea is to compute all indirect lighting with ray tracing based on LDMs. Direct lighting is computed using conventional rasterization. As such, our methos is a rasterization and ray tracing hybrid. We chose to implement our LDMs as PSPPSLL. The latter can be constructed efficiently in a single pass over the scene geometry (for each LDM). This enables us to construct the LDMs each frame in order to support dynamic scenes (no pre-computation requirements). The LDM construction itself is identical to previous work on PSPPSLL [LHL14]. We use multiple LDMs each oriented in a different direction (Figure 3). The LDMs are then used as a coarse scene approximation in which rays can be traced. The tracing itself is a key component of all our indirect lighting methods and is explained in the next section.

### 3.1. Tracing in a LDM

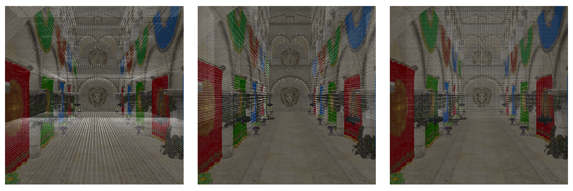
Given a ray, $r = (x, d)$ with initial point $x$ and direction $d$, a trace should return the position of the first geometric intersection, $x_f$, between $r$ and the scene geometry,

$$x_f = \text{trace}(r) = \text{trace}(x, d)$$

The trace$(r)$ query can be broken down into four steps:

1. **Find Map.** Find the LDM corresponding to $d$.
2. **Find Pixel.** Find the pixel, $p$, which corresponds to $x$ in the LDM.
3. **Find Depth Value.** Find the depth value, $z_x$, corresponding to $x$ in the depth value sequence of $p$.
4. **Compute Intersection.** Use $z_{x-1}$ to construct $x_f$.

With orthographic multi-view LDMs, there is direct mapping between the sample directions and the LDMs. As such, the **Find Map** step is trivial. The **Find Pixel** step must find the right $p$ to sample from. The solution is to use re-projection (as done in shadow mapping). That is, $x$ is projected from world coordinates into normalized device coordinates (NDC) by the LDM's view-projection matrix. The NDC directly give the position, $p$. In the **Find Depth Value** step, the $L_p = (z_0, z_1, z_2, \dots)$ sequence is searched to find the depth value, $z_x$, which corresponds to $x$. The key to the



(a) Top view. (b) Side view. (c) Forward view.

Figure 3: Point cloud visualizations of LDMs. Each subfigure uses a different view direction. The scene is the Crytek sponza. The point cloud rendering (white points) is overlaid the flat-shaded geometry.

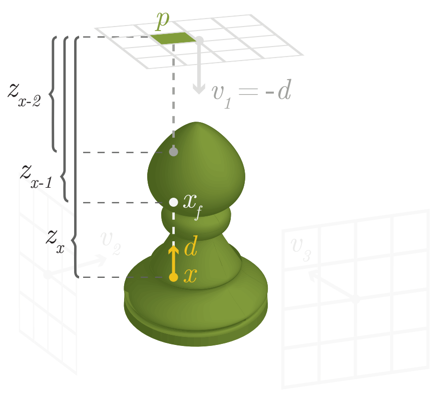


Figure 4: Querying the multi-view LDMs along the ray from $x$ in direction $d$. First, the LDM corresponding to $d$ is found ($v_1 = -d$). Second, $x$ is projected into the LDM's view to find $p$. Third, the $(\ldots, z_{x-2}, z_{x-1}, z_x, \ldots)$ sequence is searched to find $z_x$. Fourth, the first intersection along $d$ is at depth $z_{x-1}$ from which $x_f$ can readily be constructed.

**Compute Intersection** step is to note that the previous depth value, $z_{x-1}$, belongs to the first intersection with the scene geometry. From $z_{x-1}$ and the LDM's orientation, $x_f$ can be reconstructed (Figure 4).

Note that a single LDM oriented in direction $v$ can actually be queried both in the $v$ and $-v$ direction. The small extra step is to also find $z_{x+1}$ which will correspond to $x_f$ in the $-v$ direction. Algorithmically, only a single additional step has to be added. Thus both directions can be tested simultaneously with practically no overhead.

## 3.2. Deterministic Directions

The ray directions are restricted to the available LDM orientations. As such, it is important to construct representative directions. We use a deterministic approach to avoid temporal flickering.

In the case of the unit hemisphere, one way to divide the domain is by equal area. The sample directions are then the centers of each subdomain. Equal-area subdivision, however, can be achieved using ring slices (and a spherical cap). With such a subdivision, it is impossible to choose a proper center for each subdomain (apart from the cap). Therefore, it is additionally required that each subdomain must have a small diameter. The latter is defined as the maximum Euclidean distance between two points in the domain. Combined, the equal-area and small-diameter requirements restrict the subdivison to well-distributed patches from which representative centers can be easily chosen.

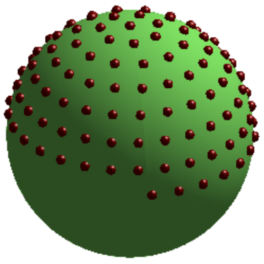


Figure 5: Visualization of 128 samples generated using the recursive zonal equal-area partition [Leo06]. Note that half of the samples in the bottom ring have intentionally been culled because parallel sample directions exist on the other side of the sphere.

Such a subdivision has been achieved for the sphere through what is known as the recursive zonal equal-area partition [Leo06]. Said algorithm recursively divides the domain into equal-area small-diameter subdomains starting with the entire sphere. It returns both the subdomains and their centers (which we interpret as directions). Please refer to Figure 5 for an example.

## 3.3. Ambient Occlusion

The formal definition of AO is

$$AO(x) = \frac{1}{\pi} \int_{\mathcal{H}} V(x, \omega_i) \cos\theta_i \, d\omega_i, \tag{1}$$

where $V$ is the visbility term and $\mathcal{H}$ is the unit hemisphere. Using deterministic equal-area sample directions, the above integral can be approximated with a sum

$$AO(x) \approx \frac{2}{N} \sum_{i=0}^{N} V(x, \omega_i) \cos\theta_i, \tag{2}$$

where $\omega_i$ is the $i$th sample direction out of the $N$ total directions (one for each LDM). Using the *trace* function described above, the visibility term, $V$, can be trivially computed

$$V(x, \omega_i) = \begin{cases} 0 & \text{trace}(x, \omega_i) \text{ returned a position} \\ 1 & \text{Otherwise.} \end{cases} \tag{3}$$

Ambient Occlusion can be generalized to Ambient Obscurance [ZIK*98] where $V(x, \omega_i, d)$ depends on $d$; the distance to the first occluder.

### 3.4. Indirect Lighting

The indirect lighting method is explained in two parts. First, we explain how photon differentials are traced. Second, we explain how to split and diffusely reflect photon differentials.

**Photon Differentials.** We base our indirect lighting method on photon differentials [SFES07]. We limit the method to point light sources for which photons can be traced as camera rays. As such, each pixel on the image plane corresponds to a photon emission. Two differentially offset rays are traced along with the primary ray

$$\begin{aligned} D_u r &= (D_u x, D_u d) \\ D_v r &= (D_v x, D_v d), \end{aligned}$$

where $D$ is the differential operator taken with respect to the screen-space $uv$-coordinates (Figure 6a). The ray differentials are calculated using the derivative tracing functions by Igehy [Ige99]. The positional differential spans a parallelogram which is a measure of the ray footprint

$$A_r = |D_u x_p \times D_v x_p|.$$

The area of the photon footprint, $A_p$, is the max-area ellipse inscribed in the parallelogram [FSES14]

$$A_p = \frac{\pi}{4} A_r = \frac{\pi}{4} |D_u x_p \times D_v x_p|.$$

Using this, the irradiance estimate for a single photon, $E_p$, can be found directly as

$$E_p = \frac{\Phi_p}{A_p}, \tag{4}$$

where $\Phi_p$ is the radiant flux carried by the photon $p$. In contrast, a real-time estimate is usually found indirectly by gathering nearby photons in a compute shader [MML13]. Like in photon mapping [JC95], the irradiance can be directly used to estimate the reflected radiance. The outgoing radiance becomes

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \sum_{p=0}^{N} f_s(x, \omega_o, \omega_p) E_p \pi K(\|M_p(x - x_p)\|), \tag{5}$$

where the sum is over the $N$ photons whose footprint overlaps $x$. The kernel function, $K$, can be any kernel that applies to a unit circle. We use Silvermann's second order kernel (as done in [FSES14])

$$K(l) = \begin{cases} \frac{3}{\pi}\left(1 - l^2\right)^2 & l < 1 \\ 0 & \text{Otherwise.} \end{cases} \tag{6}$$

$M$ is a 3×3 matrix that maps from world coordinates into filter coordinates (the domain of the kernel)

$$M_p = \frac{2}{D_u x_p \cdot (D_v x_p \times n_p)} \begin{bmatrix} D_v x_p \times n_p \\ n_p \times D_u x_p \\ a n_p \end{bmatrix}, \tag{7}$$

where $n$ is the normal and $a$ is a parameter which controls topological bias due to differences in normal orientation [FSES14].

Conventionally, the position, $x$, would be used to index into a map of photon differentials to find the overlapping footprints [SFES07]. The sum in Equation 5 can then be computed directly. This is the standard ray-tracing approach (pixel→photon differentials). Another approach is to splat the photon differential directly onto the image plane (photon differential→pixels) [FSES14]. This has the added benefit that no photon map needs to be stored and thus no costly lookups into said map. With a rasterization-based pipeline, primitive→pixels is the natural order. Therefore splatting is an ideal approach in our use case.

**Photon Splitting.** The LDMs are used for the photon tracing itself. Since the sample directions are chosen deterministically, we use photon splitting (Figure 6b) instead of Russian roulette. Specifically, a primary photon is split into $N$ secondary photons for each of the $N$ LDMs. To limit the number of generated photons, the photons are absorbed after the first bounce. Thus our method produces LDDE paths (single bounce indirect diffuse lighting).

The photon differential must be updated accordingly. To find a derivative diffuse reflection function, we first describe regular diffuse reflection in terms of the above approach. Let $\omega_i$ be the incoming direction of the photon and let $\omega_o$ be one of the $N$ outgoing directions in which a new photon is traced. Upon diffuse reflection, the ray $r = (x, d)$ results in the $r^* = (x^*, d^*)$ where

$$\begin{aligned} x^* &= x \\ d^* &= \alpha(\omega_i, \omega_o) \cdot d \cdot \bar{\alpha}(\omega_i, \omega_o). \end{aligned}$$

The $\alpha(\omega_i, \omega_o)$ term is the rotation quaternion which represents the rotation of vector $\omega_i$ to vector $\omega_o$. $\bar{\alpha}$ is the conjugate of $\alpha$. Note that $d$ is implicitly converted to a pure quaternion (and back). The $\cdot$ operator is the Hamilton product. Informally, the expression $qd\bar{q}$ denotes the rotation of $d$ by the rotation quaternion $q$. Note that $\alpha$ does not depend on neither $x$ or $d$. Thus $\alpha$ is also independent of the corresponding $uv$-coordinates. This leads to the following straight-forward derivative diffuse reflection functions

$$\begin{aligned} D_u x^* &= D_u x \\ D_u d^* &= \alpha(\omega_i, \omega_o) \cdot D_u d \cdot \bar{\alpha}(\omega_i, \omega_o). \end{aligned}$$

The positional differential is unchanged and the directional differential is rotated according to $\alpha$. Analogous expressions can be derived for the $v$-coordinate.

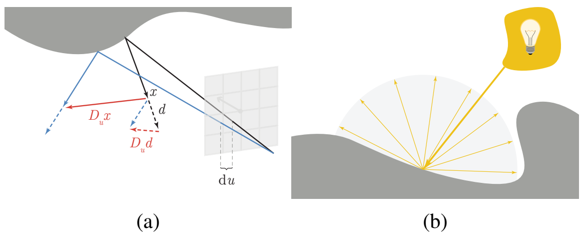


Figure 6: (a) Ray differential. The black ray is the main ray which we are currently tracing. The blue ray is the offset ray (which is not actually traced). The solid and dashed red vectors are the positional and directional ray differentials, respectively. (b) Photon splitting. The primary photon is split into multiple secondary photons. Each secondary photon is weighed according to the BRDF at the surface intersection.

## 4. Implementation

The two methods share the previously-mentioned LDM-based ray tracing. In the following, we explain the steps that are unique to each method.

### 4.1. Ambient Occlusion

Solving Equation 2 requires the following steps:

1. **Compute AO (screen-aligned quad).** Choose a sample direction $w_i$.
   a. Compute trace$(x, \omega_i)$ where $x$ is the position in world coordinates corresponding to the current pixel.
   b. $V$ is calculated using Equation 3.
   c. Weigh $V$ by the cosine term using the normal from the G-buffer and accumulate the result.

A sample direction, $\omega_i$ is chosen for each of the $N$ LDMs. In practice, step (a) is computed by iterating over all the nodes in the PSPPSLL corresponding to the current pixel position. The search can be terminated as soon as a geometric intersection is found since the lists are pre-sorted (an optimzation over previous approaches).

### 4.2. Indirect Lighting

The indirect lighting is computed in two steps. First, the photons are traced from the light and stored in a buffer. Second, the photons are splatted to the frame buffer. Please refer to Figure 7 for an overview of the implementation.

**Photon Tracing.** This pass is rendered over the scene geometry from each light's point of view. The tracing pass is divided into the following steps

1. **Calculate Radiant Flux.** Each photon's radiant flux, $\Phi_p$, is based on the light's total radiant flux, $\Phi_{\text{light}}$.
2. **Initialize Photon Differential.** This is done using the pixel's $uv$-coordinates and the light's orientation. Let $x_0$ be the photon's position on the light source and let $d_0$ be its initial direction.
3. **Transfer Photon Differential.** Transfer from $x_0$ to the first intersected surface, $x_1$. The direction is unchanged, so $d_1 = d_0$.
4. **Split Photon.** Let $N$ be the number of LDMs. Then a photon is traced in both directions of each LDM (totaling in $2N$ photons). For each corresponding direction $d_2$:
   a. Compute $x_2 = \text{trace}(x_1, d_2)$; the intersection with the first diffuse surface.
   b. Project $x_2$ into the user's view. Discard the photon if it is not visible.
   c. Diffusely reflect the photon differential from $d_1$ to $d_2$.
   d. Transfer the photon differential from $x_1$ to $x_2$.
   e. Store the photon differential in the photon buffer.

In the **Split Photon** step, the photon is first traced (a) before the differential is updated (c,d). This is done so that occluded photons can be rejected early (b). Note that the primary photons (from the light source) are not stored. Only indirect photons are stored. As such, the splats will only contribute with indirect lighting. A third pass is needed to compute direct lighting. This can be done using conventional rasterization.

**Photon Splatting.** The photon buffer generated in the previous pass is sent through the rasterization pipeline in the next pass. This is simply a matter of rebinding the underlying buffer object as a vertex array (instead of an SSBO) and issuing a draw call (as points). The properties stored through the previous SSBO binding can then be accessed as vertex attributes directly in the vertex shader. This pass is rendered over a full-screen quad. The scene information is available through the G-buffer. The splatting itself is best explained in terms of the three shader stages:

1. **Vertex Shader.** The irradiance estimate is made here (Equation 4).
2. **Geometry Shader.** The positional differential is used to expand the point into a quad corresponding to the ray footprint.
3. **Fragment Shader.** The kernel function, $K$, is applied after transformation to filter space using $M_p$ and the result is written to the frame buffer. Additive blending is used so that the sum in Equation 5 is computed.

## 5. Results

We have performed all tests on an Nvidia GeForce GTX 780 Ti at 800×800 resolution. The reference images are generated using an offline path-tracing renderer.

### 5.1. Ambient Occlusion

In Figure 8, we have rendered the AO term for the Crytek Sponza with an attenuated visibility function. The latter de-

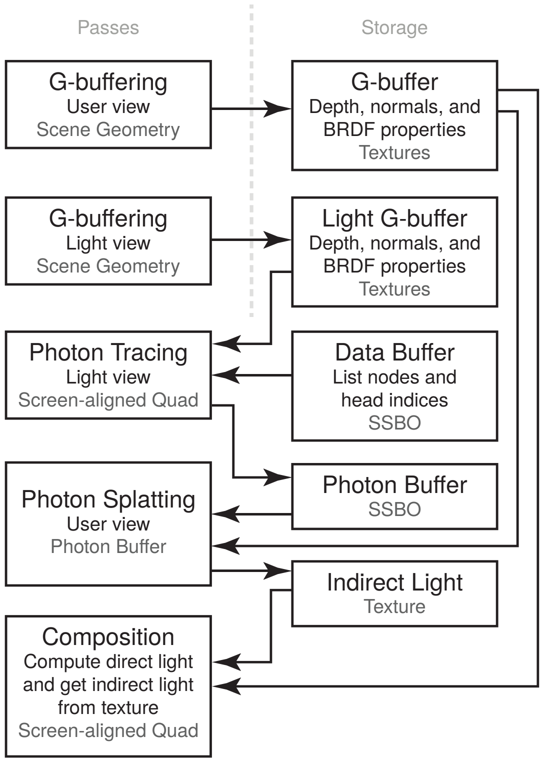
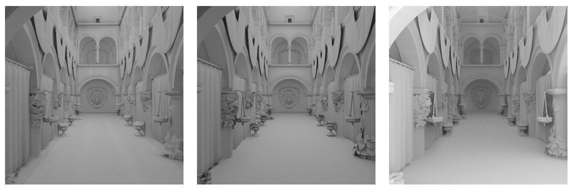


Figure 7: Overview of photon tracing and splatting. The data buffer contains the PSPPSLL of all LDMs. This buffer is generated in an unlisted pass prior to the indirect lighting routine.

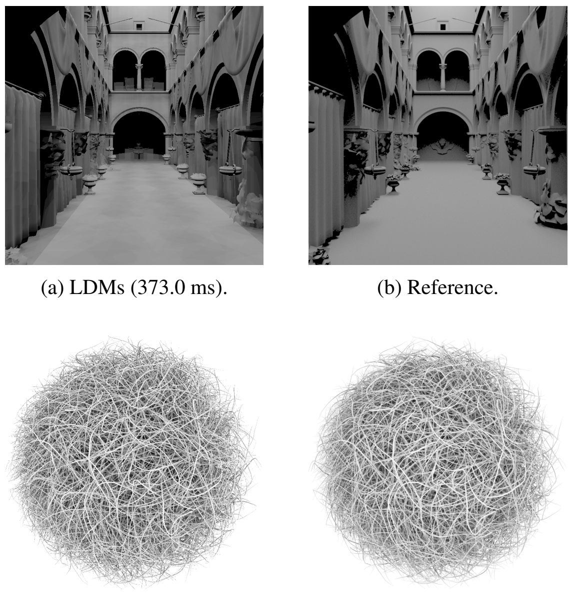
pends on an emperical parameter, $d_{max}$, which is the maximum tracing distance. This parameter is necessary in screen-space approaches such as HBAO to limit the kernel size of the AO filter. Our approach based on LDMs does not impose such restrictions. From the figure, it is evident that for large $d_{max}$ the HBAO method fails to resemble the reference. This is due to the limited geometric information available in the depth map. With LDMs, however, global geometric information is available which is why our method better resembles the reference. Performance-wise, LDMs are also faster than HBAO for large $d_{max}$. Specifically, the performance of HBAO drops due to the large kernel size. Note that these conclusions are only valid for large $d_{max}$. As $d_{max} \to 0$, the reverse conclusion can be made. That is, for small $d_{max}$ the construction and tracing of LDMs takes much more time than evaluating the filter kernel in HBAO.

It is possible to trade quality for performance by using fewer low-resolution LDMs (Figure 9). The optimal configuration must be found emperically. Figure 9 also shows how our LDM-based method can be implemented with an unattenuated visibility function; something that is not possible with the screen-space approaches.



(a) LDMs (512.4 ms). (b) Reference. (c) HBAO (903.6 ms).

Figure 8: Ambient occlusion for $d_{max} = 1280m$. Our method (a) resembles the reference (b) better than the screen-space method (c). This is because the LDMs provide global scene information whereas the screen-space method is limited by the information available in the depth map without layers. 512 LDMs are used each with resolution 200×200 totalling in 805.5 MB of memory.



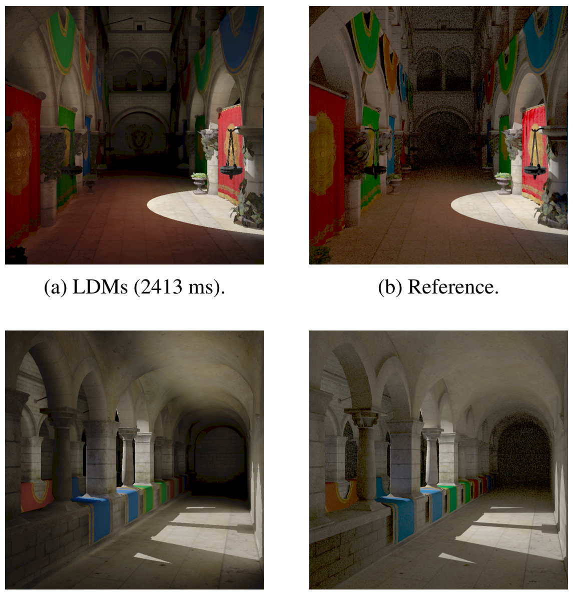
(a) LDMs (373.0 ms). (b) Reference.

(c) LDMs (1174 ms). (d) Reference.

Figure 9: Ambient occlusion. These images are generated using the unattenuated visbility function. Furthermore, 512 low-resolution LDMs (50×50) are used for improved performance. The LDMs takes up 204.1 MB and 295.5 MB for the sponza and the hairball, respectively.

## 5.2. Indirect Lighting

In Figure 10, we have rendered the Crytek Sponza using our indirect lighting method. Note that the render times are much longer compared to the simpler AO method. Somewhat surprisingly, the bottleneck is the splatting pass which accounts for a majority of the render time. The LDM tracing itself is relatively cheap in comparison. The splatting step depends on the additive blending operators of OpenGL's


(a) LDMs (2413 ms). (b) Reference.

(c) LDMs (1716 ms). (d) Reference.

Figure 10: Indirect Lighting. The Crytek Sponza rendered using our approach based on photon differentials traced with LDMs. 512 LDMs of 200×200 resolution are used totalling in 805.5 MB of memory. Note that the total render time (in parenthesis) also includes pipeline overhead. Photon tracing takes 2062 ms and 1434 ms in (a) and (c), respectively.

fixed-function pipeline which we cannot change. We did experiment with an implementation that performed the tracing and splatting in a single fragment shader pass using atomic image load/store operations. Unfortunately, this turned out to perform even worse than the fixed-function blending. Quality-wise, our LDM-based method resembles the reference image but there are some key differences. The large photon splats blur the indirect shadows in places where there should be hard shadows (as seen in the reference). These artifacts can be reduced by using a smaller splat size and increasing the photon count accordingly. The optimal tradeoff between quality and performance must be found empericially.

In Figure 10, we render all direct light using conventional rasterization. In principle, direct light could also be modelled with photons and rendered in the same way as the indirect light. That is, our LDM-based method can be used as a unified lighting solution. In Figure 11, we show how LDMs can be used to compute both direct environment lighting and indirect lighting.

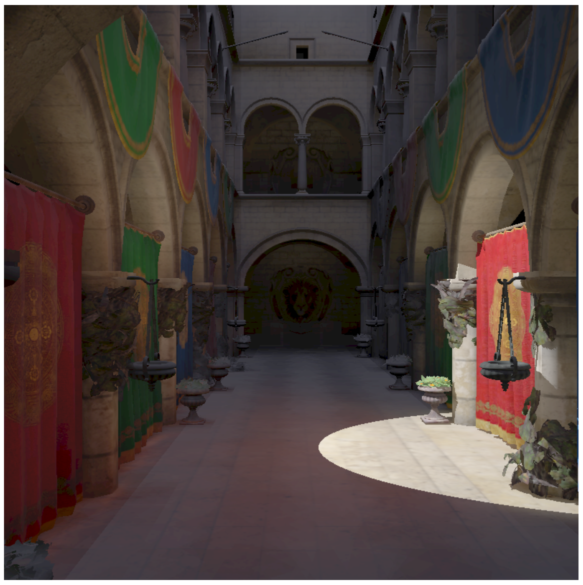


Figure 11: Direct environment light, direct spot light, and indirect spot light. The direct environment light and indirect spot light is traced using LDMs. The direct light is compued using conventional rasterization.

## 6. Discussion

In this section, we present various extensions to our LDM-based indirect lighting method.

**Multiple Bounces.** Our method can be trivially extended to support multiple light bounces. As it is now, we split the primary photon into multiple secondary photons. The very same approach can be used to split the secondary photons into tertiary photons and so on.

The first practical difficulty is to stop the recursion. We propose to either fix the light bounces globally (like we did with one bounce) or to use Russian roulette. The problem with the latter is that it introduces temporal flickering. The second practical difficulty is to control the splitting. A naive extension of our approach would split photons exponentially based on the number of light bounces. Alternatively, one can choose a subset of the sample directions for the secondary bounces. This subset can then be reduced further for the tertiary bounces and so on. This scheme can be used to balance out the exponential growth. Another alternative is to choose sample directions randomly. Again, we chose not to do so for temporal coherence.

**Non-Lambertian BRDF.** The problem with our current approach is that the outgoing directions are fixed. Therefore, perfect specular reflections are impossible. An approximate solution is to instead choose the outgoing direction which is closest to the perfect specular reflection. For large $N$, the difference would be negligible. The same approach can be used to implement glossy reflections.

In principle, any BRDF can be approximated this way. Of course, the quality of the result heavily depends on how well the hemisphere is sampled. That is, whether a truly representative direction can be chosen. Assuming this is possible, then the LDMs could in principle also be used to implement path tracing. Though in this extreme, it is arguably more practical to use one of the aforementioned ray-marching approaches.

**Arbitrary Light Sources.** Currently, photon emission is restricted to point light sources. This is so that the photon differentials can be easily traced via basic ray differential theory. It is possible to emit photon differentials from arbitrary light sources [FSES14]. Beyond the emission, the tracing itself is identical to tracing ray differentials. Therefore, the **Photon Tracing** step can be replaced with a compute shader that emits and traces photon differentials from arbitrary light sources. The **Photon Splatting** step would not change. Note that this not only enables area light sources such as disks and squares but also arbitrary geometry light sources. As mentioned earlier, the tracing step is currently not a bottleneck. Therefore, a more complex compute shader can easily be afforded.

**Progressive Rendering.** The scene can be rendered over multiple frames to improve visual quality (inspired by [HHZ*14]). This is mostly relevant for interactive purposes. A simple approach is to randomly rotate the LDMs each frame so that new directions are sampled. The result is then averaged over several frames to produce a more convincing image. This is similar to the approach used by many path tracers. Of course, the random rotation would introduce noise in the result and temporal coherence is lost.

## 7. Conclusion

We have presented two global illumination methods using an auxiliary data structure based on LDMs. Specifically, we presented an interactive AO method and an interactive single-bounce indirect lighting method which both convincingly resemble the path traced reference. Compared to HBAO, our LDM-based AO method is more efficient for large $d_{max}$ both in terms of quality and performance. The indirect lighting method is novel in the way it handles the diffuse reflection of photon differentials. The pre-sorted LDMs allow us to quickly find the first occluder in both directions with a novel trace algorithm.

The LDMs are constructed independently of one another. In that sense, our auxiliary data structure is actually a collection of individual data structures that each stores a scene representation. Contrast this to, say, a voxel grid which stores a scene representation in a single data structure. The problem with a collection of individual data structures is that scene information may be duplicated. That is, the same surface point may be stored in multiple LDMs. This wastes space. There has been research on optimizing the data storage by sharing information between list nodes [KWBG13]. It would be interesting to see if entire list nodes could be shared between the LDMs in order to reduce data duplication.

We have striven to stay physically correct and only introduced bias to gain reasonable performance. The bias can be reduced by increasing the sampling density. We envision that our LDM-based approach can ultimately be used as a unified lighting solution.

## References

[AMHH08] Akenine-Möller T., Haines E., Hoffman N.: *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008. 1, 3

[BCL*07] Bavoil L., Callahan S. P., Lefohn A., Comba J. a. L. D., Silva C. T.: Multi-fragment effects on the GPU using the *k*-buffer. In *Proceedings of i3D 2007* (2007), pp. 97–104. 2

[BHKW07] Bürger K., Hertel S., Krüger J., Westermann R.: GPU rendering of secondary effects. In *Vision, Modeling and Visualization 2007* (2007). 2, 4

[BSD08] Bavoil L., Sainz M., Dimitrov R.: Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH Talks 2008* (2008), pp. 22:1–22:1. 3

[Car84] Carpenter L.: The a-buffer, an antialiased hidden surface method. *Computers and Graphics 18*, 3 (1984), 103–108. 2

[CICS05] Callahan S., Ikits M., Comba J., Silva C.: Hardware-assisted visibility sorting for unstructured volume rendering. *IEEE Transactions on Visualization and Computer Graphics* (2005), 285–295. 2

[CNS*11] Crassin C., Neyret F., Sainz M., Green S., Eisemann E.: Interactive indirect illumination using voxel cone tracing. In *Proceedings of i3D 2011* (2011). 1, 3

[Cra10a] Crassin C.: Fast and accurate single-pass A-Buffer using OpenGL 4.0+. `http://blog.icare3d.org/2010/06/fast-and-accurate-single-pass-buffer.html` (2010). 2

[Cra10b] Crassin C.: OpenGL 4.0+ ABuffer v2.0: Linked lists of fragment pages. `http://blog.icare3d.org/2010/07/opengl-40-abuffer-v20-linked-lists-of.html` (2010). 3

[CTBM12] Comba J. L. D., Torchelsen R., Bastos R., Maule M.: Memory-efficient order-independent transparency with dynamic fragment buffer. In *Proceedings of SIBGRAPI 2012* (2012), pp. 134–141. 3

[DS05] Dachsbacher C., Stamminger M.: Reflective shadow maps. In *Proceedings of i3D 2005* (2005), pp. 203–208. 3

[Eve01] Everitt C.: Interactive order-independent transparency. *Technical Report, NVIDIA Corporation* (2001). 2

[FM08] Filion D., McNaughton R.: Effects & techniques. In *ACM SIGGRAPH 2008 Games* (2008), ACM, pp. 133–164. 3

[FSES14] Frisvad J. R., Schjøth L., Erleben K., Sporring J.: Photon differential splatting for rendering caustics. *Computer Graphics Forum 33*, 6 (2014), 252–263. 6, 10

[GT10] Gruen H., Thibieroz N.: OIT and indirect illumination using DX11 linked lists. In *Proceedings of the 2010 Game Developers Conference* (March 2010). 3

[Hec90] Heckbert P. S.: Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics (Proceedings of ACM SIGGRAPH 1990) 24*, 4 (1990), 145–154. 3

[HHGM10] Hermes J., Henrich N., Grosch T., Mueller S.: Global illumination using parallel global ray-bundles. *Vision, Modeling and Visualization Workshop 2010* (2010), 65–72. 4

[HHZ*14] Hu W., Huang Y., Zhang F., Yuan G., Li W.: Ray tracing via GPU rasterization. *Visual Computer 30*, 6-8 (2014), 697–706. 2, 4, 10

[Ige99] Igehy H.: Tracing ray differentials. In *Proceedings of ACM SIGGRAPH 1999* (1999), pp. 179–186. 6

[JC95] Jensen H. W., Christensen N. J.: Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computers and Graphics 19*, 2 (1995), 215–224. 6

[JC99] Jouppi N. P., Chang C.-F.: Z3: an economical hardware technique for high-quality antialiasing and transparency. *SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (1999), 85–93. 2

[KBW06] Krüger J., Bürger K., Westermann R.: Interactive screen-space accurate photon tracing on GPUs. In *Proceedings EGSR 2006* (2006), pp. 319–329. 2, 4

[Kub14] Kubisch C.: Order independent transparency in OpenGL 4.x. In *Proceedings of the 2014 GPU Technology Conference* (2014). 3

[KWBG13] Kerzner E., Wyman C., Butler L., Gribble C.: Toward efficient and accurate order-independent transparency. *ACM SIGGRAPH 2013 Posters* (2013). 10

[Lan02] Landis H.: Production-Ready Global Illumination. In *ACM SIGGRAPH 2002 Course Notes* (2002), vol. 16. 3

[Leo06] Leopardi P.: A partition of the unit sphere into regions of equal area and small diameter. *Electronic Transactions on Numerical Analysis 25* (2006), 309–327. 5

[LHL13] Lefebvre S., Hornus S., Lasram A.: *HA-Buffer: Coherent Hashing for single-pass A-buffer*. Rapport de recherche RR-8282, INRIA, Apr. 2013. 3

[LHL14] Lefebvre S., Hornus S., Lasram A.: Per-pixel lists for single pass a-buffer. In *GPU Pro 5: Advanced Rendering Techniques*, Engel W., (Ed.). CRC Press, 2014, pp. 3–23. 3, 4

[LHLW09] Liu F., Huang M.-C., Liu X.-H., Wu E.-H.: Single pass depth peeling via cuda rasterizer. *ACM SIGGRAPH 2009 Talks* (2009). 2, 3

[LHLW10] Liu F., Huang M.-C., Liu X.-H., Wu E.-H.: Freepipe: A programmable parallel rendering architecture for efficient multi-fragment effects. In *Proceedings of i3D 2010* (2010), pp. 75–82. 2, 3

[Lip10] Lipowski J. K.: Multi-layered framebuffer condensation: The l-buffer concept. *Computer Vision And Graphics, Part 2 6375* (2010), 89–97. 3

[Lip13] Lipowski J. K.: D-buffer: irregular image data storage made practical. *Opto-Electronics Review 21*, 1 (2013), 103–125. 3

[LR98] Lischinski D., Rappoport A.: Image-based rendering for non-diffuse synthetic scenes. *Rendering Techniques '98 (Proceedings of EGWR 1998)* (1998), 301–314. 4

[LS10] Loos B. J., Sloan P.-P.: Volumetric obscurance. In *Proceedings of i3D 2010* (New York, NY, USA, 2010), ACM, pp. 151–156. 3

[MB07] Myers K., Bavoil L.: Stencil routed a-buffer. In *ACM SIGGRAPH 2007 Sketches* (2007), ACM. 2

[MCTB11] Maule M., Comba J. L. D., Torchelsen R. P., Bastos R.: A survey of raster-based transparency techniques. *Computers & Graphics 35*, 6 (2011), 1023–1034. 2

[Mit07] Mittring M.: Finding next gen: Cryengine 2 (course notes). In *ACM SIGGRAPH 2007 Courses* (2007), pp. 97–121. 3

[Mit12] Mittring M.: The technology behind the unreal engine 4 elemental demo. In *ACM SIGGRAPH 2012 Talks* (2012). 3

[MML13] Mara M., McGuire M., Luebke D.: Toward practical real-time photon mapping: Efficient gpu density estimation. In *Interactive 3D Graphics and Games 2013* (2013). 6

[MMNL14] Mara M., McGuire M., Nowrouzezahrai D., Luebke D.: *Fast Global Illumination Approximations on Deep G-Buffers*. Tech. Rep. NVR-2014-001, NVIDIA Corporation, June 2014. 4

[MOBH11] McGuire M., Osman B., Bukowski M., Hennessy P.: The alchemy screen-space ambient obscurance algorithm. In *Proceedings of HPG 2011* (2011), pp. 25–32. 3

[MP01] Mark W., Proudfoot K.: The F-buffer: A rasterization-order FIFO buffer for multi-pass rendering. In *Proceedings of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2001), pp. 57–63. 2

[NRS14] Nalbach O., Ritschel T., Seidel H.-P.: Deep screen space. In *Proceedings of i3D 2014* (2014), pp. 79–86. 1, 4

[NSS10] Niessner M., Schaefer H., Stamminger M.: Fast indirect illumination using layered depth images. *Visual Computer 26*, 6–8 (2010), 679–686. 2, 4

[PH04] Pharr M., Humphreys G.: *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. 3

[RGK*08] Ritschel T., Grosch T., Kim M. H., Seidel H.-P., Dachsbacher C., Kautz J.: Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2008) 27*, 5 (2008), 129. 3

[RGS09] Ritschel T., Grosch T., Seidel H. P.: Approximating dynamic global illumination in image space. In *Proceedings of i3D 2009* (2009), pp. 75–82. 1

[SA07] Shanmugam P., Arikan O.: Hardware accelerated ambient occlusion techniques on GPUs. In *Proceedings of i3D 2007* (2007), pp. 73–80. 3

[SAF*13] Segal M., Akeley K., Frazier C., Leech J., Brown P.: The OpenGL graphics system: A specification (version 4.3 core profile), 2013. 2

[SFES07] Schjøth L., Frisvad J. R., Erleben K., Sporring J.: Photon differentials. In *Proceedings of GRAPHITE 2007* (2007), pp. 179–186. 6

[SKP98] Szirmay-Kalos L., Purgathofer W.: Global ray-bundle tracing with hardware acceleration. In *Rendering Techniques '98 (Proceedings of EGWR 1998)* (1998), pp. 247–258. 4

[SRS14] Sugihara M., Rauwendaal R., Salvi M.: Layered reflective shadow maps for voxel-based indirect illumination. *Proceedings of HPG 2014* (2014), 117–125. 3

[SS96] SBERT M., SÀNDEZ X.: The use of global random directions to compute radiosity. global monte carlo techniques. *Ph.D. Thesis. Universitat Politècnica de Catalunya* (1996). 4

[Thi11] THIBIEROZ N.: Order-independent transparency using per-pixel linked lists. In *GPU Pro 2: Advanced Rendering Techniques* (2011), Engel W., (Ed.), A K Peters, pp. 409–431. 3

[TO12a] TOKUYOSHI Y., OGAKI S.: Imperfect ray-bundle tracing for interactive multi-bounce global illumination. *High Performance Graphics 2012 Posters* (2012). 4

[TO12b] TOKUYOSHI Y., OGAKI S.: Real-time bidirectional path tracing via rasterization. In *Proceedings of i3D 2012* (2012), pp. 183–190. 4

[UPSK07] UMENHOFFER T., PATOW G., SZIRMAY-KALOS L.: Robust multiple specular reflections and refractions. In *GPU Gems 3*, Nguyen H., (Ed.). 2007, pp. 387–407. 4

[VF12] VASILAKIS A., FUDOS I.: S-buffer: Sparsity-aware multi-fragment rendering. In *Eurographics 2012 - Short Papers Proceedings* (2012), pp. 101–104. 3

[VPG13] VARDIS K., PAPAIOANNOU G., GAITATZES A.: Multi-view ambient occlusion with importance sampling. *Proceedings of i3D 2013* (2013), 111–118. 3

[Wit01] WITTENBRINK C.: R-buffer: A pointerless A-buffer hardware architecture. In *Proceedings of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2001), pp. 73–80. 2

[YHGT10] YANG J. C., HENSLEY J., GRUEN H., THIBIEROZ N.: Real-time concurrent linked list construction on the GPU. *Computer Graphics Forum 29*, 4 (2010), 1297–1304. 3

[YM10] YANG J., MCKEE J.: Real-time order independent transparency and indirect illumination using Direct3D 11. In *ACM SIGGRAPH Courses* (2010). 3

[ZHS08] ZHANG C., HSIEH H.-H., SHEN H.-W.: Real-time reflections on curved objects using layered depth textures. In *Proceedings of MCCSIS'08* (2008), pp. 276–281. 2

[ZIK*98] ZHUKOV S., IONES A., KRONIN G., DRETTAKIS G., MAX N.: An ambient light illumination model. *Rendering Techniques '98 (Proceedings of EGWR 1998)* (1998), 45–55. 3, 6

[ZRD14] ZIRR T., REHFELD H., DACHSBACHER C.: Object-order ray tracing for fully dynamic scenes. In *GPU Pro 5: Advanced Rendering Techniques*, Engel W., (Ed.). CRC Press, 2014, pp. 419–438. 1