

Forward Light Cuts: A Scalable Approach to Real-Time Global Illumination

Gilles LAURENT^{+,*}

Cyril DELALANDRE⁺

Grégoire de LA RIVIÈRE⁺

Tamy BOUBEKEUR^{*}

⁺ Dassault Systèmes

^{*} LTCI, CNRS, Telecom ParisTech, Université Paris-Saclay

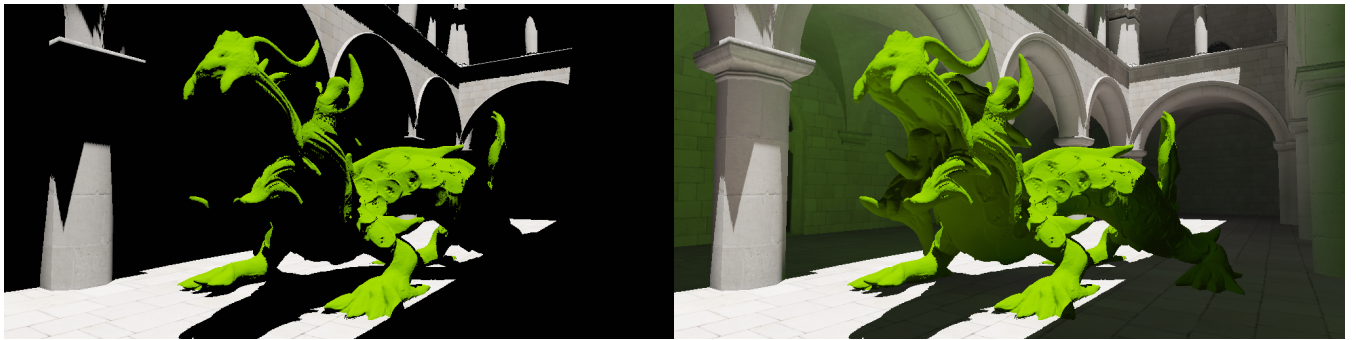


Figure 1: Real-time rendering with the first bounce of indirect lighting using Forward Light Cuts. The scene is composed of 7M triangles and is rendered in 16 ms at 1024×512 pixels resolution, without any precomputation and accounting for occluded yet contributing surfaces and long range light bounces.

Abstract

We present *Forward Light Cuts*, a novel approach to real-time global illumination using forward rendering techniques. We focus on unshadowed diffuse interactions for the first indirect light bounce in the context of large models such as the complex scenes usually encountered in CAD application scenarios. Our approach efficiently generates and uses a multiscale radiance cache by exploiting the geometry-specific stages of the graphics pipeline, namely the tessellator unit and the geometry shader. To do so, we assimilate virtual point lights to the scene's triangles and design a stochastic decimation process chained with a partitioning strategy that accounts for both close-by strong light reflections, and distant regions from which numerous virtual point lights collectively contribute strongly to the end pixel. Our probabilistic solution is supported by a mathematical analysis and a number of experiments covering a wide range of application scenarios. As a result, our algorithm requires no precomputation of any kind, is compatible with dynamic view points, lighting condition, geometry and materials, and scales to tens of millions of polygons on current graphics hardware.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Radiosity

1. Introduction

Light transport simulation is an important component of realistic image synthesis. Beyond direct lighting, *global illumination*, despite its well known physics laws, remains a challenging problem due to its high computational cost, with even more critical consequences for fully dynamic real-time scenarios involving large objects. Hidden behind the recursive nature of the rendering equation [Kaj86], global illumination simulation has been addressed in a number of approaches with applications ranging from visual special effects to scientific visualization, through animated pictures and video games. Currently, we can distinguish two lines of research: *offline rendering*, which targets a solution as close as possible to physics and *interactive rendering*, which aims at quickly

providing a visually convincing approximation of global illumination. While significant progress have been recently made for the former using the Monte Carlo rendering framework, we focus on the latter and the set of constraints induced by real-time scenarios.

Due to the low-pass filtering nature of diffuse material reflection [RH01, BJ03], most real-time global illumination methods exploit the reasonable assumption that indirect radiance can be described by a low frequency function, especially when the emitter is far from the receiver (Sec. 2). In particular, numerous *radiance caching* methods [WFA*05, REG*09] compute a hierarchical spatial structure over the geometry of the scene and use it to model a multiscale radiance function, later queried to illuminate the pixels of the final image. The leaves of this tree structure are typically

formed by so-called *virtual point lights* (or VPLs), which are sampled on the scene surfaces that are visible from the primary light emitters at caching time. Every internal node of this tree is then set with a representative response that approximates the radiance of its related subtree. At shading time, a set of nodes, called *light cut*, is adaptively gathered from the tree to evaluate the incoming radiance at a given point. Although it may contain leaves (i.e., original VPLs) for close-by elements, it is typically mostly formed of internal nodes that act as economic substitutes to represent the incoming radiance from distant locations, saving both time and memory.

For this category of methods, a fully dynamic scenario, involving large objects, induces at least two limitations. First, the caching structure needs to be recomputed at each frame, as detecting changes under fully dynamic conditions ends up being just as costly as recomputing the whole cache, in particular regarding the limitation of the fine-grained parallel nature of modern graphics hardware. Second, the initial set of VPLs may be too large to cope with real time constraints, in particular when their generation cannot be amortized over time.

In this paper, we adopt a forward strategy to address these problems (Sec. 3). First, we observe that the scene polygons themselves can trigger the VPL generation process and propose a tessellation/decimation GPU pipeline that uses both the geometry shader and the tessellator unit to generate an initial set of VPLs (Sec. 4), refining the geometry of the scene wherever it is too coarse to accurately capture the radiance, and simplifying it where the polygon distribution is too dense. Second, we propose a stochastic clustering scheme that associates subsets of the resulting VPLs to bounded regions of influence for which they act as radiance representatives to shade points. This yields a multiscale representation of indirect lighting free from any explicit tree structure used to efficiently shade receivers (Sec. 5). Moreover, we designed both the caching and shading stages to efficiently map on modern graphics architectures (Sec. 6). As a result, our entire algorithm runs from scratch at every frame and preserves real-time performance even for large scenes, capturing diffuse unshadowed indirect illumination under dynamic conditions, while naturally accounting for long range indirect illumination and hidden geometry (Sec. 7).

2. Previous Work

A full survey of real-time global illumination methods is beyond the scope of this paper and we refer the reader to the recent manuscripts by Ritschel et al. [RDGK12] for an up-to-date overview of real-time global illumination methods and Dachsbacher et al. [DKH*14] for a complete overview of the many-lights framework. As we target diffuse GI for large dynamic scenes, we focus on the most relevant prior art in the following, namely *screen-space* and *object-space* diffuse global illumination solutions.

Mittring et al. [Mit07] introduced an ambient occlusion approximation method using the depth-buffer as an economic, random-accessible substitute to the actual (potentially large) geometry of the scene, and parameterizing the light cache in screen-space. Later, Ritschel et al. [RGS09] extended this approach to simulate diffuse color bleeding in a similar setting. A large variety of other methods [RDGK12] exploit screen-space approximations to lower the

computational complexity of some lighting effects. However, despite their real-time and dynamic performances, such approaches rely on depth peeling and multiple views rendering to account for the full geometry of the scene i.e., beyond the first depth layer and outside the view frustum, which quickly impacts negatively their native speed. Recently, Mara et al. [MMNL14] proposed to take advantage of temporal coherency to build a multi-layered sampling strategy and remove most of hidden surface issues – e.g. view dependent ghosting artifacts. This approach is effective in a number of cases, but still suffers from grazing angle geometry undersampling issues.

In contrast to screen-space approaches, solving for indirect illumination in object-space avoids such view-dependent artifacts, at the cost of less GPU-friendly light caches. For instance, Instant Radiosity (IR) methods [Kel97] work with VPLs, a set of secondary point light sources, directly generated on the geometry illuminated by the primary sources. Thus, a VPL set acts as a discrete representation of the scene's indirect lighting and allows to reduce computations drastically when approximating light bounces. *Reflective Shadow Maps* (RSM) [DS05, DS06] provide an efficient VPL generation mechanism by sampling the scene in light-view space. This method has been improved by adding a clustering strategy over the RSM pixels which allows to reduce the number of VPLs by keeping the relevant ones only [PKD12]. However, scaling up to massive data requires huge amounts of VPLs. This is challenging as both generation and shading costs of so many VPLs is prohibitive in dynamic scenes. This problem is typically addressed with hierarchical methods such as *Lightcuts* [WFA*05] or *Point-Based Global Illumination* [Chr08, REG*09], which aim at managing massive sets of VPLs using multiple level-of-details of the point-sampled light field. In this context, *ManyLODs* [HREB11] reached interactive rendering time by computing in parallel many coherent cuts in the VPL tree. To reach real-time performances on complex scenes with up to millions of lights, Olsson et al. [OBA12] chose to address fill-rate issues by clustering the GBuffer pixels using an extension of tiled shading [OA11]. However, these techniques are based on a tree structure which requires amortizing its construction over time, preventing full dynamism in the scene.

A challenging issue while simulating indirect illumination with IR is to compute visibility between VPLs and pixels. Numerous approaches have been developed to this end, such as the *Imperfect Shadow Maps* (or ISMs) [RGK*08] which are generated quickly and in droves from on a point sampling of the scene, and queried during shading to approximate indirect visibility. Virtual Area Lights (or VALs) [DGR*09] allow to reduce the number of visibility queries by clustering light emitters into small surfaces, with the visibility being approximated by computing soft shadows from VALs shadow maps. In order to amortize VPLs shadow computation, Laine et al. [LSK*07] proposed to select at each frame a subset of VPLs for which shadow maps are computed or updated. To improve temporal coherency, Barák et al. [BBH13] used a RSM to sample preferably the scene in region with high radiance. Recently, Hedman et al. [HKL16] developed a technique which generates a temporally coherent VPL sampling of large scenes by memorizing their position from frame to frame and only invalidating them when they no longer influence framebuffer pixels. Indirect visibility is then implicitly solved by using ray tracing to sample VPLs.

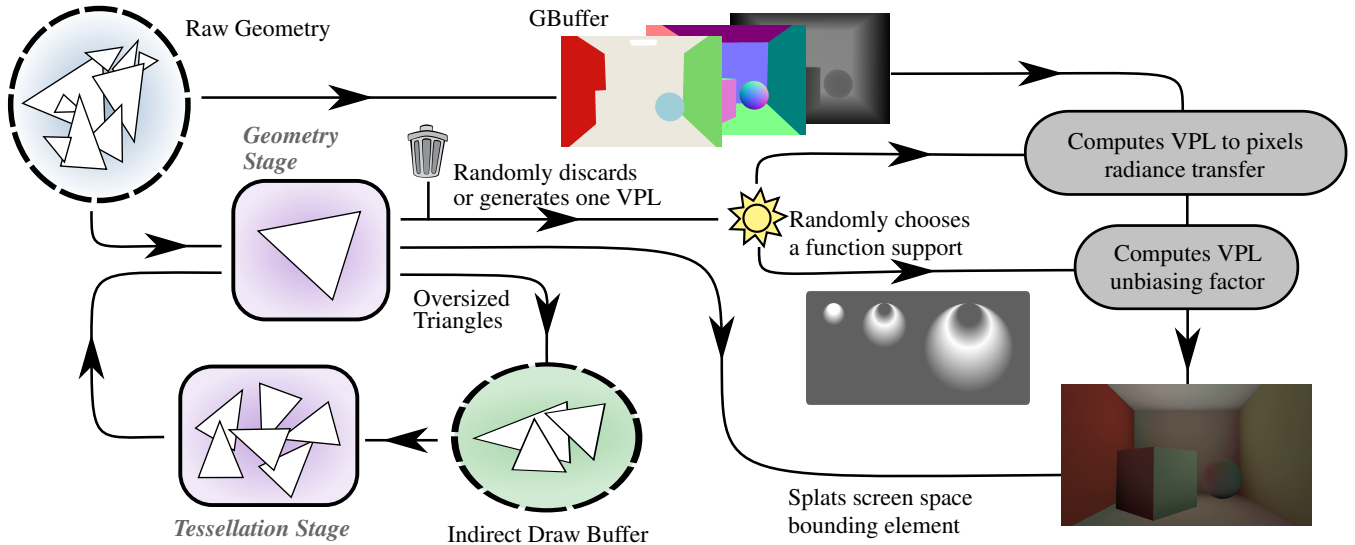


Figure 2: **Forward Light Cuts**. The raw geometry is sent to the geometry shader where it is split into regular and divergent triangles. Divergent triangles – *i.e.* with an area greater than a certain threshold – are used to fill an indirect draw buffer and then subdivided such that every new triangle may be considered as regular. Regular and subdivided triangles are then randomly distributed over $N + 2$ subsets including one which is discarded. The surviving triangles are classified among subsets according to a probability distribution depending on their size. Finally, for each of these triangles, a single VPL is created whose power depends on the subset it belongs to and its support function is computed accordingly.

Note that our proposed algorithm does not address indirect visibility, as it is an orthogonal problem to the one we target: efficiently balancing VPL distribution to focus computations where needed.

Finally, the Deep Screen Space (DSS) approach [NRS14] proposes to exploit the advantages of both screen-space and object-space radiance caching. The same way as object-space strategies, this method generates on-surface VPLs, even on occluded geometry that may still impact the image; and similarly to screen-space approaches, it benefits from a native GPU support, with the tessellator unit – instead of the rasterizer – being used as a surface sampler to generate the VPLs. Still, although DSS can successfully be used for rendering small to medium size scenes, it cannot cope with larger ones, where the real-time constraint imposes decimating geometry rather than refining it. Our technique makes a step forward in this direction, by proposing a diffuse GI pipeline which can both refine and simplify the set of geometry-driven VPLs in a two-pass strategy. Exploiting both the tessellator unit and the geometry shader to adjust the resolution of an object-space radiance cache in the context of scenes with a massive number of triangles. In particular, we also reach real-time performance by using a multi-scale representation of the light field but, contrary to the aforementioned techniques, our method is fully dynamic and does not resort to any tree structure, nor imposes to maintain any data structure among frames.

3. Algorithm Overview

Our algorithm, illustrated in Fig. 2, has the structure of typical VPL based pipelines, composed of three main stages: VPL generation, indirect light caching and lighting with VPLs. Our contributions

mainly focus on VPL generation and their usage during lighting, and can be summarized as follows:

- at loading time, we associate a single random integer to each vertex; this number will be used at rendering time to generate per-triangle random numbers consistently, even under dynamic geometry transformations (see Sec. 6);
- the full set of triangles \mathcal{L} is then randomly partitioned according to a probability distribution (Sec. 4, Fig. 3); for each triangle, a VPL is stochastically generated and its outgoing radiance is computed based solely on its related partition ($\mathcal{L}_0 \dots \mathcal{L}_N$); in order to balance computations, we distribute many samples with small influence distance in \mathcal{L}_0 and decrease the number of samples while increasing the influence distance in \mathcal{L}_k when k grows; this leads to a functional equation on the VPLs radiance that we derived in Sec. 5;
- in addition, to significantly reduce the number of triangles to manage while optimizing the amount of sampling information, a special set – mainly composed of small triangles – is completely discarded;
- furthermore, to properly cope with hardware restrictions when it comes to dynamic data amplification, we create exactly one VPL per triangle; as such, large triangles – that we call “divergent” (Sec. 4) – are not well sampled by the aforementioned strategy and may introduce important lighting artifacts in the final image; consequently, we reroute them through the tessellation unit, where they are subdivided to reach the proper resolution (see Sec. 6);
- last, we splat indirect illumination in a typical deferred shading process, with the splatted function supports depending on the VPL partition, reserving powerful VPLs to carry on distant lighting using crescent-shaped support (Sec. 5, Fig. 3).

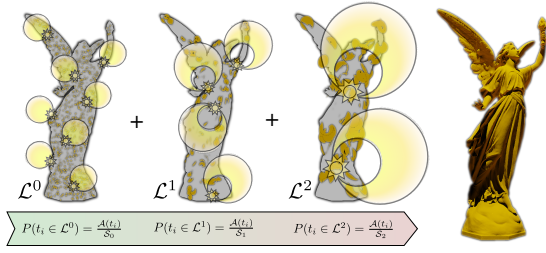


Figure 3: Multi-scale radiance cache. In our approach, VPLs are distributed among different subsets, each of which having a bounded region of influence (in yellow). The final illumination computed at a given point uses samples from the different clusters to reconstruct the incoming radiance at shading time.

4. VPLs Generation

Our VPL generation method is based on the standard hardware rasterization pipeline and is designed to exploit the GPU fine-grained parallelism by generating each VPL independently from the others. Moreover, our algorithm only implies a single draw pass of the geometry, enabling its use with scenes featuring a high polygon count. To do so, we distinguish regular triangles from divergent ones, *i.e.* the set of triangles $\{t_i\}$ with surface area $\mathcal{A}(t_i)$ greater than a certain threshold \mathcal{S}_0 . We set

$$\mathcal{S}_0 = 4\pi \frac{D_{near}^2}{N_{avg}}, \quad (1)$$

which is a heuristic aiming at lighting pixels with approximately N_{avg} VPLs at least D_{near} far from them. Let R_{scene} be the scene radius, we typically set $D_{near} = 0.2 \times R_{scene}$ and N_{avg} between 64 and 1024 depending on the desired quality/speed tradeoff. We discuss how we handle divergent triangles in Sec. 6 and assume triangles to be regular in the remaining of this section.

Triangle decimation Small triangles contribute weakly to the final rendering for diffuse indirect lighting [DKH*14] and we tend to favor their removal in our pipeline. However, we cannot just discard every triangle smaller than a given threshold, since *groups* of small triangles may collectively have an important impact in the light transport reaching a distant point. We address this problem by adopting a stochastic decimation approach: we retain the contribution of heavily tessellated geometry by computing, for each triangle, a uniform random value u_i lying between 0 and 1. We then keep this triangle if $\mathcal{A}(t_i) > u_i \mathcal{S}_0$. Because every triangle is assumed to be regular, the probability for a triangle to be kept boils down to:

$$\forall t_i \in \mathcal{L}, P(t_i \in \mathcal{L}^*) = \frac{\mathcal{A}(t_i)}{\mathcal{S}_0}, \quad (2)$$

where \mathcal{L} denotes the set of all triangles and \mathcal{L}^* the set of kept triangles. This means that the smaller a triangle is, the greater its chance to be discarded becomes. At the same time, this partitioning translates into a uniform distribution of samples over the entire scene surface, such that the expectation of the surviving triangle count is $\mathbb{E}[N_{sample}] = \frac{\mathcal{A}_{scene}}{\mathcal{S}_0}$, with \mathcal{A}_{scene} the total scene area.

Triangle multiscale partitioning Once small triangles have been discarded, we randomly dispatch the remaining ones (\mathcal{L}^*) in a par-

ition ($\mathcal{L}^0, \dots, \mathcal{L}^N$) such that \mathcal{L}^N contains a few triangles and \mathcal{L}^k is more and more populated when k gets closer to 0. To do so, we introduce the sequence of $N+1$ increasing values $\{\mathcal{S}_0 < \dots < \mathcal{S}_N\}$ representing the desired VPL partitioning. By further defining in Alg. 1:

$$\forall k \in [0 \dots N], \quad \bar{\mathcal{S}}_k = \frac{1}{\sum_{j=0}^k \frac{1}{\mathcal{S}_j}},$$

a multiscale partitioning of representative scene triangles emerges from our decimation strategy (Fig. 3), with the probability for a triangle to lie in the subset \mathcal{L}^k being:

$$\forall k \in [0 \dots N], \quad P(t_i \in \mathcal{L}^k) = \frac{\mathcal{A}(t_i)}{\bar{\mathcal{S}}_k}. \quad (3)$$

Note that, with this definition, a triangle is considered as divergent if its area is greater than $\bar{\mathcal{S}}_N$.

In such a way, the key property of our approach at this stage is that we **do not** generate, maintain or manage any kind of explicit hierarchy – because we affect a triangle to a certain subset independently from the choice made for any other – while still being able to later gather an adaptive multiscale light cut. This clearly favors parallel execution, however, this also means that a given triangle, located in a given subset, does not capture any coarse-grained information carried by finer triangles. This issue is discussed and partially addressed in Sec. 5.

Algorithm 1 Multiscale Partition

```

1: procedure COMPUTELEVEL( $u_i, t_i$ )
2:   for  $k \leftarrow 0 \dots N$  do
3:     if  $u_i < \frac{\mathcal{A}(t_i)}{\bar{\mathcal{S}}_k}$  then
4:       return  $k$ 
5:     end if
6:   end for
7:   DISCARDTRIANGLE()
8: end procedure
    
```

5. Lighting with VPLs

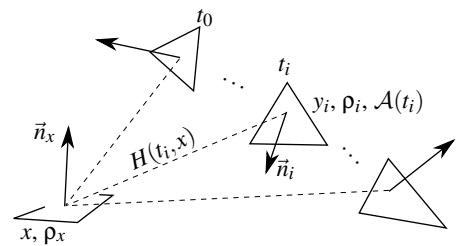


Figure 4: VPL illumination at point x

In the many-lights framework, the indirect outgoing radiance $L(x, \vec{n}_x)$ of a point x with normal \vec{n}_x is approximated by $L^{ML}(x, \vec{n}_x)$ which is defined as the discrete sum of the radiance coming from a set of VPLs:

$$L^{ML}(x, \vec{n}_x) = \sum_{t_i \in \mathcal{L}} H(t_i, x, \vec{n}_x) \mathcal{A}(t_i), \quad (4)$$

where \mathcal{L} represents the set of all triangles in the scene and $H(t_i, x, \vec{n}_x)$ stands for the incoming radiance transfer function starting from t_i toward the receiver x oriented by \vec{n}_x (Fig. 4). For a diffuse receiver with albedo ρ_x , this function is defined as:

$$H(t_i, x, \vec{n}_x) = L(t_i, y_i x) \frac{\rho_x \langle \vec{n}_x, x\vec{y}_i \rangle^+ \langle \vec{n}_i, y_i x \rangle^+}{\pi d_i^2}, \quad (5)$$

where $\vec{u} = \frac{\vec{u}}{\|\vec{u}\|}$, $\langle \vec{u}, \vec{v} \rangle^+ = \max(0, \langle \vec{u}, \vec{v} \rangle)$, $L(t_i, y_i x)$ is the radiance leaving the VPL centered at $y_i \in t_i$ toward the direction $y_i x$ and $d_i = \max(\epsilon, \|x\vec{y}_i\|)$ is the distance between y_i and x clamped to a user parameter ϵ to avoid singularities. We model the first diffuse bounce of light with the following VPL outgoing radiance expression:

$$L(t_i, y_i x) = \rho_i E(t_i) \frac{3}{2\pi} \langle \vec{n}_i, y_i x \rangle^+, \quad (6)$$

where $E(t_i)$ is the direct irradiance falling to the triangle t_i . Note that because of the term $\langle \vec{n}_i, y_i x \rangle^+$, these reflectors cannot be considered as perfectly lambertian anymore. Although light is therefore preferably reflected in the direction of the geometric normal, our experiments show that no important changes appear, while this greatly alleviates upcoming computations. The term $\frac{3}{2\pi}$ comes to ensure energy conservation, with the radiosity $B(t_i)$ and the irradiance $E(t_i)$ being related by:

$$B(t_i) = \int_{\Omega} L(t_i, \omega) \langle \vec{n}_i, \omega \rangle^+ d\omega = \rho_i E(t_i)$$

Approximating VPL lighting We propose to approximate the computation of $L^{ML}(x, \vec{n}_x)$ by summing the contribution of a subset of VPLs (Fig. 3), *i.e.* the ones lying in $(\mathcal{L}^0, \dots, \mathcal{L}^N)$. Thus, we define $K(x, \vec{n}_x)$ an estimator of $L^{ML}(x, \vec{n}_x)$ as follows:

$$K(x, \vec{n}_x) = \sum_{k=0}^N \sum_{t_i \in \mathcal{L}^k} H(t_i, x, \vec{n}_x) F^k(t_i, x), \quad (7)$$

$F^k(t_i, x)$ is an unknown function of the position x , the emitter t_i and the index k . Its equation is derived below.

By computing the expectation of $K(x, \vec{n}_x)$ over the set of every possible partition $(\mathcal{L}^0, \dots, \mathcal{L}^N)$, we get:

$$\begin{aligned} \mathbb{E}[K(x, \vec{n}_x)] &= \mathbb{E} \left[\sum_{k=0}^N \sum_{t_i \in \mathcal{L}^k} H(t_i^k, x, \vec{n}_x) F^k(t_i^k, x) \right] \\ &= \sum_{t_i \in \mathcal{L}} H(t_i, x, \vec{n}_x) \mathbb{E} \left[\sum_{k=0}^N F^k(t_i, x) \mathbb{1}_{[t_i \in \mathcal{L}^k]} \right] \\ &= \sum_{t_i \in \mathcal{L}} H(t_i, x, \vec{n}_x) \sum_{k=0}^N F^k(t_i, x) P(t_i \in \mathcal{L}^k), \end{aligned} \quad (8)$$

where $\mathbb{1}_{[t_i \in \mathcal{L}^k]}$ is the indicator function, that equals to 1 if $t_i \in \mathcal{L}^k$ and 0 otherwise. If we want $K(x, \vec{n}_x)$ to represent an unbiased estimator of the incoming radiance $L^{ML}(x, \vec{n}_x)$, we have to verify the following functional equation on F^k :

$$\forall x, \sum_k F^k(t_i, x) P(t_i \in \mathcal{L}^k) = \mathcal{A}(t_i). \quad (9)$$

According to our VPL partitioning strategy (see Eqn. 3), we define

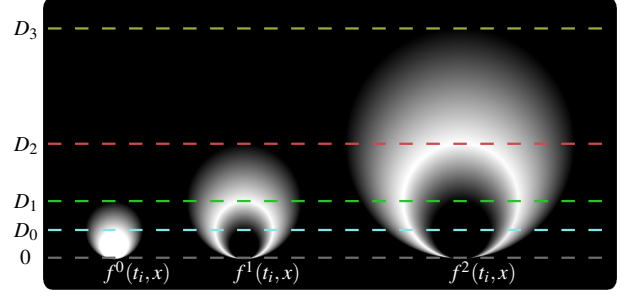


Figure 5: Visualization of our support functions $f^k(t_i, x)$ on a planar section, for values ranging from 0 (black) to 1 (white). $\mathcal{B}_h(t_i)$ are the isolevels of these functions.

F^k as:

$$F^k(t_i, x) = \mathcal{S}_k f^k(t_i, x), \forall [0 \dots N], \quad (10)$$

which translates the unbiased condition (Eqn. 9) to a partition of unity problem, seeking for a set of functions $(f^k)_k$ such that:

$$\forall x, \sum_k f^k(t_i, x) = 1. \quad (11)$$

Choice of partition of unity Inspired from PBGI tree cuts strategies [Chr08], we introduce a family of nested balls $\mathcal{B}_h(t_i)$ characterized by $h > 0$. For a given h , $\mathcal{B}_h(t_i)$ represents the set of points for which the contribution of the VPL is significant. This means that for each point x outside of $\mathcal{B}_h(t_i)$, the function $H(t_i, x, \vec{n}_x)$ has a smaller value than h , whatever the orientation of the receiver \vec{n}_x :

$$\forall h \in \mathbb{R}^*, \mathcal{B}_h(t_i) = \left\{ x \in \mathbb{R}^3 \text{ s.t. } \max_{\vec{n}} H(t_i, x, \vec{n}) \geq h \right\}. \quad (12)$$

Furthermore, $H(t_i, x, \vec{n})$ (Eqn. 5) is maximal when the receiver is front facing the emitter, *i.e.* $\vec{n} = x\vec{y}_i$. Thus, with our VPL radiance distribution model, we can write:

$$\mathcal{B}_h(t_i) = \left\{ x \in \mathbb{R}^3 \text{ s.t. } \frac{\|x - y_i\|}{\langle \vec{n}_i, x\vec{y}_i \rangle^+} \leq D(h) \right\}, \quad (13)$$

where

$$D(h) = \frac{1}{\pi} \sqrt{\frac{3\rho_x \rho_i E(t_i)}{2h}}.$$

Hence, as $D(h)$ does not depend on x , $(\mathcal{B}_h(t_i))_{h \in \mathbb{R}^*}$ is a family of nested balls whose frontier owns y_i and center lies on the line (y_i, n_i) (Fig. 5). We impose that our 3D unit partition $(f^k)_k$ is constant over the spheres being the frontier of a $\mathcal{B}_h(t_i)$. Then, by defining the following mapping from \mathbb{R}^3 to \mathbb{R} :

$$\forall x \in \mathbb{R}^3, d(t_i, x) = \frac{\|x - y_i\|}{\langle \vec{n}_i, x\vec{y}_i \rangle^+}, \quad (14)$$

our problem boils down to a 1D partition of unity $(\tilde{f}^k)_k$:

$$\forall x \in \mathbb{R}^3, f^k(t_i, x) = \tilde{f}^k(d(t_i, x)). \quad (15)$$

Since $(f^k)_k$ will be used as splatting functions during rendering, we aim at making them as smooth as possible while keeping them

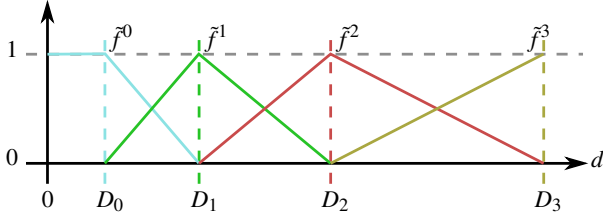


Figure 6: One dimensional partition of unity

easy to compute and define them as the following set of continuous piecewise affine functions with compact support:

$$\forall d \in \mathbb{R}, \tilde{f}^k(d) = \begin{cases} 1 & \text{if } k = 0 \text{ and } d \in]0, D_1] \\ \frac{d - D_{k-1}}{D_k - D_{k-1}} & \text{if } k > 0 \text{ and } d \in]D_{k-1}, D_k] \\ \frac{D_{k+1} - d}{D_{k+1} - D_k} & \text{if } k > 0 \text{ and } d \in]D_k, D_{k+1}] \\ 0 & \text{otherwise} \end{cases}. \quad (16)$$

Where $\{D_k\}$ allow to specify the influence distance of each level (Fig. 5, 6).

Parameters setting In order to mimic the traditional hierarchical representations used with light fields, we generate our partition with subsets size expectation that decreases geometrically. Furthermore, while \mathcal{S}_k may be understood as the average surface of triangles lying in the level \mathcal{L}^k , we define them by:

$$\mathcal{S}_k = \mathcal{S}_0 \mu^k, \quad (17)$$

where $\mu > 1$ is a user-defined real number. Depending on the scene and the number of levels, we typically set μ between 1.4 and 5. In addition, still by mimicking the hierarchical approaches, we propose to define the distance of VPLs influence such that each point in space is reached by a controlled number of VPLs, which may be translated into:

$$\begin{cases} D_k = \sqrt{\mathcal{S}_0 \mu^k}, \quad \forall k \in [0 \dots N] \\ D_{N+1} = D_N \end{cases}, \quad (18)$$

6. Implementation details

We implemented our method with the OpenGL 4.4 API.

Pipeline description As depicted in Fig. 2, our pipeline only contains two geometry draw passes of the entire scene: one to generate the GBuffer and one to generate and splat the VPLs. This moderate use of the raw geometry is an important metric for our application scenarios because we aim at managing scenes with a large number of polygons. In fact, a third geometry draw pass occurs, but involves only a fraction of the scene: the divergent triangles. As the divergence criterion is set such that the number of divergent triangles remains small, this last pass is not computationally prohibitive. Our algorithm exploits intensively the geometry shader stage to perform computations on a per-triangle basis rather than a per-fragment or a per-vertex one. Fortunately, for recent GPU architecture, the formally prohibitive overhead of the geometry shader stage has been greatly reduced, enabling polygon-wise computations for large streams.

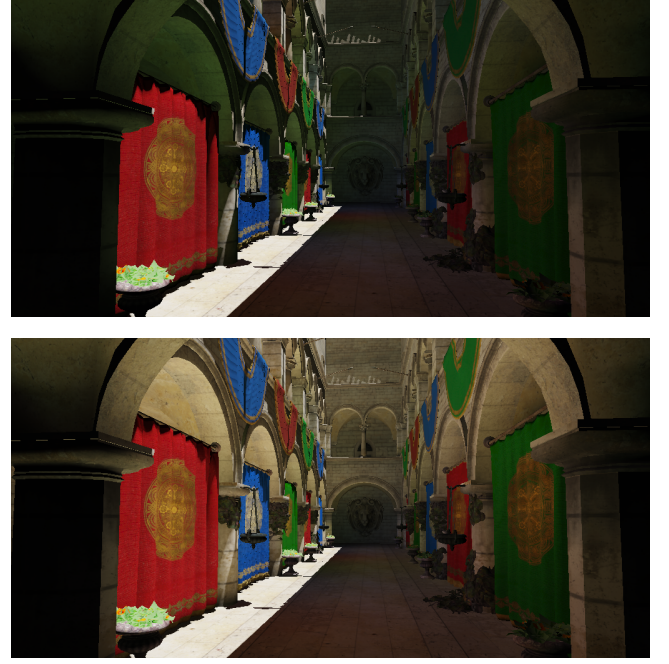


Figure 7: FLC on the Crytek Sponza. Top: without the divergent pipeline, only the heavy tessellated geometry (green flowers and arc hessian) cast indirect lighting. Bottom: With the full pipeline, the ground (4 triangles) light bounce reveals much of the scene.

Divergent triangle management Current hardware tessellation units are not designed to manage massive triangle sets as input, inducing a noticeable overhead while processing a triangle even if this triangle does not require any subdivision. At the same time, the geometry stage of these architectures are extremely efficient at discarding or letting polygons pass trough, *i.e.* when no geometry amplification is mandatory. This motivates us to design our indirect lighting pipeline with the two following passes. In the first pass, the entire scene geometry is processed but the tessellation stage is disabled. Divergent triangles are detected at the geometry stage and stored in a separate buffer, while regular ones are stochastically sampled. In order to manage scenes with numerous materials and textures, we also store a per-triangle material index, used in the following pass to fetch information from a material or texture atlas.

Using the OpenGL “glDrawArrayIndirect” feature, the divergent buffer is subsequently directly used as input geometry for the second pass without any CPU synchronization. The tessellation stage is solely activated for this particular pass and used to subdivide the triangles such that their area becomes small enough to be processed by our regular pipeline. In general, the number of large triangles is relatively small compared to the total triangle count. Consequently, the overhead induced by the divergent buffer filling and vertex processing is negligible compared to the visual effect improvement (see Fig. 7 for instance).

Per-triangle random number generation To evaluate Alg. 1 for each triangle, we use a pseudo-random value for the variable u_i . The generation of this value only requires the mesh to own an additional per-vertex uint32 attribute – `v_rand`. This attribute is initial-

Algorithm 2 Per-triangle random value computation

```

1: uniform uint32 u_rand
2: uniform texture2D noise
3: function REGULARRAND(ivec3 v_rand)
4:   return u_rand xor v_rand.x xor v_rand.y xor v_rand.z
5: end function
6: function DIVERGENTRAND(ivec3 v_rand, vec2
  tess_coord[3])
7:   ivec3 v_tess_rand
8:   v_rand_tess.x = TEXTURE(noise, tess_coord[0])
9:   v_rand_tess.y = TEXTURE(noise, tess_coord[1])
10:  v_rand_tess.z = TEXTURE(noise, tess_coord[2])
11:  return REGULARRAND(v_rand xor v_tess_rand)
12: end function

```

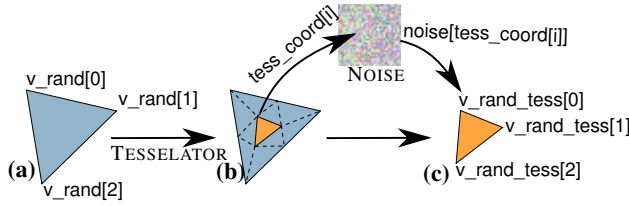


Figure 8: Generation of per-triangle uniform random values in the divergent pipeline. Each divergent triangle (a) is first subdivided. For each sub-vertex (b), its barycentric coordinates $tess_coord$ are used to fetch a random value in the noise texture. Finally the fetched value is used to initialize the subtriangle (c) random attribute.

ized when loading the mesh with random values uniformly chosen between 0 and $2^{32} - 1$. In the regular pipeline, at the geometry shading stage, u_{t_i} is computed as a xor between the random attributes of the three vertices of t_i . Note that the choice of the xor operator avoids correlation among two or three triangles. Moreover, new random values can be generated at any time by using u_rand , a global uniform random value that may be updated at most once per frame. Being xor'd with the original per-vertex values, it allows to get whole new random distributions over the mesh (Alg. 2).

For the divergent pipeline, the construction of u_{t_i} is quite more subtle. Indeed, this value remains unaltered, even under camera motion or mesh deformation, *i.e.* as long as the mesh topology remains the same. To preserve these properties, we perform the tessellation in the model space and exploit the fact that the tessellation pattern only depends on the input triangle shape. Indeed, in our use cases, the tessellation parameters are only determined by the original triangle area. Therefore, during the tessellation evaluation stage, we use the barycentric coordinates of the generated vertices to fetch a uint32 value, named v_rand_tess , from a precomputed noise texture. Thus, to generate u_{t_i} for a triangle sprung from the subdivision, we compute a xor between the three v_rand_tess , the three base triangle v_rand and finally the global uniform random value u_rand (Fig. 8, Alg. 2).

Progressive Rendering Since the VPL support functions defined in Eqn. 16 are smooth, our algorithm produces at each frame a visually plausible rendering without high frequency artifacts. Nevertheless, with the aforementioned global uniform variable u_rand , it is straightforward to generate many independent renderings of

	Cornell Box Tiling $\times 64$	Power Plant Tiling $\times 64$
GBuffer + SM	0.3	3.6
Indirect	3.5	8.5
Regular Pipeline	0.9	6.5
Divergent Pipeline	1.0	0.8
Upscaling + Blur	1.2	1.2
Full frame	3.8	12.1

Table 1: Rendering time break down (ms) at 1280×720 resolution, for the Cornell Box (1K tri.) and the PowerPlant (12M tri.)

the same scene with our approach. In addition, for each triangle, we can easily derive two new independent random values from u_rand . These values are used to jitter the VPL center y_i defined in Eqn. 5 over the triangle t_i . Then, by averaging all these renderings with jittered VPLs, Eqn. 8 provides a result that is close to the true solution of our problem, *i.e.* the computation of the first bounce outgoing radiance on surface composed of diffuse reflectors and ignoring indirect visibility. While this means that indirect lighting is computed by integrating over every scene triangle, this progressive version of our technique is able to manage any kind of disturbed geometry (*e.g.* normal mapping, alpha tested) and could also be extended to manage emissive textured geometry (Fig. 11).

Splatting indirect illumination The way the VPL contributions are summed to simulate the indirect lighting is orthogonal to the previous discussion. Hence, to keep a simple pipeline, we use a splatting strategy similar to deferred lighting [ST90]. In particular, this allows to manage geometry decimation, VPL generation and lighting in a single shader program. Indeed, besides determining whether a triangle is divergent or not and which is the level of generated VPLs, we use the geometry shader to transform input triangles in *sized* point primitives which encompasses the underlying VPL screen space function support. Although the major drawback of such a single-pass lighting pipeline is the fillrate consumption, our algorithm aims at simulating a low frequency phenomenon. Therefore, undersampling the resulting signal appears as a reasonable optimization. To do so, we partition the viewport in $4^{N_{\text{tiling_level}}}$ tiles and assign to each pixel in the viewport a unique tile pixel at the same relative location – this technique is often referred as *interleaved sampling* [KH01, LSK*07]. Next, at splatting time, a tile is randomly chosen for each VPL thus dividing the number of touched pixels by $4^{N_{\text{tiling_level}}}$. We finally recombine the image by untiling the buffer and blurring it to remove the generated noise.

7. Results

All our experiments are performed on a standard PC equipped with a Quadro M6000 GPU. Tab. 1 gathers timings of our FLC algorithm on two scenes: one very simple and one containing a much more complex geometry. We observe that the overall performance of our algorithm mainly depends on two factors: the framebuffer resolution and the number of triangles composing the scene (Fig. 13). Note also that the divergent pipeline costs roughly the same in both cases.

When geometry is not the bottleneck, the framerate heavily depends on the number of lower resolution subtiles used when splat-

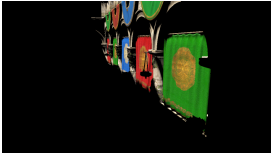
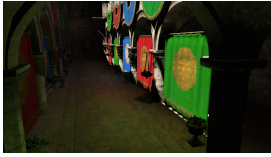
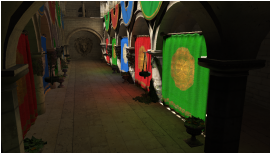













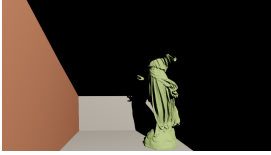



Model	Direct lighting only	With SSDO	With DSS	With FLC (ours)
Sponza 270k tri.	 4ms	 7ms	 19ms	 8ms
Oil Platform 700k tri.	 1ms	 5ms	 22ms	 14ms
Car 3M tri.	 2ms	 6ms	 28ms	 20ms
San Miguel 12M tri.	 9ms	 13ms	 35ms	 22ms
Lucy 28M tri.	 18ms	 20ms	 80ms	 35ms

Table 2: Comparisons performed at a 1280x720 resolution. All high resolution images are provided as supplemental material.

ting indirect illumination. However, even if no information is lost from the viewport sampling, reducing subtile resolution also comes with drawbacks such as removing geometry details in shadowed area (Fig. 9), due to the indirect lighting interpolation. The number of required VPLs depends on the scene (Fig. 10) and induces fillrate, bandwidth and shading costs which obviously influences the FLC framerate. However, although the number of used VPLs is proportional to rendering time, the overall visual quality does not map linearly to this value. In fact, we observe that the degradation brought with our downscale strategy is well compensated by increasing the number of VPLs. This allows trading VPLs for subtiles to adjust a visually pleasant real-time framerate.

Comparison We evaluate our FLC approach against two algorithms that we reimplemented in our framework: the popular SSDO method [RGS09] and the more recent DSS approach [NRS14]. These two algorithms share the same properties as our FLC method: (i) they run in real time (ii) with no preprocessing (i.e., on fully dynamic scenes) and they reproduce one-bounce, unshadowed diffuse indirect lighting. We perform our comparison using 5 scenes that exhibit different structures, polygon counts and lighting

conditions. Table 2 provides resulting images obtained by the three different approaches and reports the complete frame generation time in each case. First, we can observe that our method succeeds at producing similar or better result than DSS, while being significantly faster. This behavior is emphasized when the polygon count grows and can be explained by our rerouting strategy, which reduces significantly the tessellator work load for large models. Second, when comparing to SSDO, we can observe that this method often fails at revealing the scene regions which are not directly lit, but should receive a significant amount of first bounce indirect lighting. Of course, being fully screen-space, the SSDO framerate remains high even with large scenes, but the visual improvement appears to be small compared to direct lighting only, while our approach, just such as DSS, reproduces better and more consistent long range indirect lighting. In addition, Fig. 11 illustrates the execution of our approach when enabling progressive rendering. We compute the average of many renderings for which we change the random seed at each frame. Each rendering is done with a high number of subtiles and a few number of VPLs which allows to improve the rendering time until convergence. We can visually assess

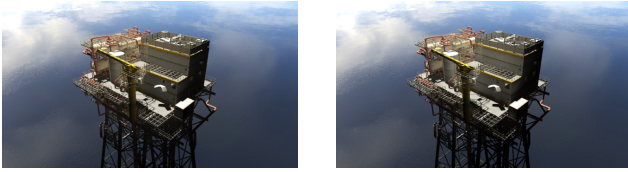


Figure 11: (Left) Progressive rendering (250ms) by averaging 50 renderings. (Right) Real time rendering (14 ms).

that, at the cost of a longer rendering time, this strategy allows to completely remove any kind of artifact (like spikes).

Limitations and future work Our algorithm does not produce indirect shadow casting (Fig. 12) nor multiple light bounces since splatting indirect illumination in screen space does not provide any direct mean to simulate such phenomena. A future approach would be to incorporate object space strategies [OBA12] to address these issues, while still preserving the scalability and full dynamic-compatibility of our method. The second limitation of our approach is its current restriction to diffuse indirect lighting: incorporating glossy reflectors in our mathematical framework is an important research direction for managing scenes which are complex both from a geometric and a material point-of-view.

8. Conclusion

We have proposed *Forward Light Cuts*, a novel approach to compute diffuse indirect global illumination in real-time. Our geometric method generates VPLs using a stochastic decimation process of the input triangles within a two-stages pipeline that either simplifies or refines the scene's geometry to reach a suitable radiance caching resolution. Our approach does not imply any complex preprocessing nor requires carrying complex data structures over frames. It is compatible with large fully dynamic scenes, including light, view point and geometry animations. Last, in terms of integration, our approach naturally fits modern graphics pipelines and does not make any strong assumption on the complementary rendering techniques employed by the host application. In addition to the mathematical basis of our method, we evaluated it on a number of scenes with high polygon counts, ranging from CAD models to scans, and reported interactive performances in each case.

References

- [BBH13] BARÁK T., BITTNER J., HAVRAN V.: Temporally coherent adaptive sampling for imperfect shadow maps. In *Comp. Graph. Forum* (2013), vol. 32, pp. 87–96. 2
- [BJ03] BASRI R., JACOBS D. W.: Lambertian reflectance and linear subspaces. *IEEE Trans. PAMI* 25, 2 (2003), 218–233. 1
- [Chr08] CHRISTENSEN P.: Point-based approximate color bleeding. *Pixar Technical Notes* 2, 5 (2008), 6. 2, 5
- [DGR*09] DONG Z., GROSCH T., RITSCHER T., KAUTZ J., SEIDEL H.-P.: Real-time indirect illumination with clustered visibility. In *Proc. VMV* (2009), pp. 187–196. 2
- [DKH*14] DACHSBACHER C., KRIVÁNEK J., HAŠAN M., ARBREE A., WALTER B., NOVÁK J.: Scalable realistic rendering with many-light methods. In *Comp. Graph. Forum* (2014), vol. 33, pp. 88–104. 2, 4

- [DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *Proc. I3D* (2005), pp. 203–231. 2
- [DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *Proc. I3D* (2006), pp. 93–100. 2
- [HKL16] HEDMAN P., KARRAS T., LEHTINEN J.: Sequential monte carlo instant radiosity. In *Proc. I3D* (2016), pp. 121–128. 2
- [HREB11] HOLLANDER M., RITSCHER T., EISEMANN E., BOUBEKEUR T.: Manylods: Parallel many-view level-of-detail selection for real-time global illumination. In *Comp. Graph. Forum* (2011), vol. 30, pp. 1233–1240. 2
- [Kaj86] KAJIYA J. T.: The rendering equation. In *Proc. SIGGRAPH* (1986), vol. 20, pp. 143–150. 1
- [Kel97] KELLER A.: Instant radiosity. In *Proc. SIGGRAPH* (1997), pp. 49–56. 2
- [KH01] KELLER A., HEIDRICH W.: *Interleaved sampling*. Springer, 2001. 7
- [LSK*07] LAINE S., SARANSAARI H., KONTKANEN J., LEHTINEN J., AILA T.: Incremental instant radiosity for real-time indirect illumination. In *Proc. EGSR* (2007), pp. 277–286. 2, 7
- [Mit07] MITTRING M.: Finding next gen: Cryengine 2. In *SIGGRAPH 2007 courses* (2007), pp. 97–121. 2
- [MMNL14] MARA M., MCGUIRE M., NOWROUZEZAHRAI D., LUEBKE D.: *Fast global illumination approximations on deep G-buffers*. Tech. rep., Tech. Rep. NVR-2014-001, NVIDIA Corporation., 2014. 2
- [NRS14] NALBACH O., RITSCHER T., SEIDEL H.-P.: Deep screen space. In *Proc. I3D* (2014), pp. 79–86. 3, 8
- [OA11] OLSSON O., ASSARSSON U.: Tiled shading. *Journal of Graphics, GPU, and Game Tools* 15, 4 (2011), 235–251. 2
- [OBA12] OLSSON O., BILLETTER M., ASSARSSON U.: Clustered deferred and forward shading. In *Proc. HPG* (2012), pp. 87–96. 2, 9
- [PKD12] PRUTKIN R., KAPLANYAN A., DACHSBACHER C.: Reflective shadow map clustering for real-time global illumination. In *Proc. EUROGRAPHICS Short Papers* (2012), pp. 9–12. 2
- [RDGK12] RITSCHER T., DACHSBACHER C., GROSCH T., KAUTZ J.: The state of the art in interactive global illumination. In *Comp. Graph. Forum* (2012), vol. 31, pp. 160–188. 2
- [REG*09] RITSCHER T., ENGELHARDT T., GROSCH T., SEIDEL H.-P., KAUTZ J., DACHSBACHER C.: Micro-rendering for scalable, parallel final gathering. In *ACM Trans. Graph.* (2009), vol. 28, p. 132. 1, 2
- [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. In *ACM Trans. Graph.* (2008), vol. 27, p. 129. 2
- [RGS09] RITSCHER T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *Proc. I3D* (2009), pp. 75–82. 2, 8
- [RH01] RAMAMOORTHY R., HANRAHAN P.: On the relationship between radiance and irradiance: determining the illumination from images of a convex lambertian object. *JOSA A* 18, 10 (2001), 2448–2459. 1
- [ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-d shapes. In *Proc. SIGGRAPH* (1990), vol. 24, pp. 197–206. 7
- [WFA*05] WALTER B., FERNANDEZ S., ARBREE A., BALA K., DONIKIAN M., GREENBERG D. P.: Lightcuts: a scalable approach to illumination. In *ACM Trans. Graph.* (2005), vol. 24, pp. 1098–1107. 1, 2

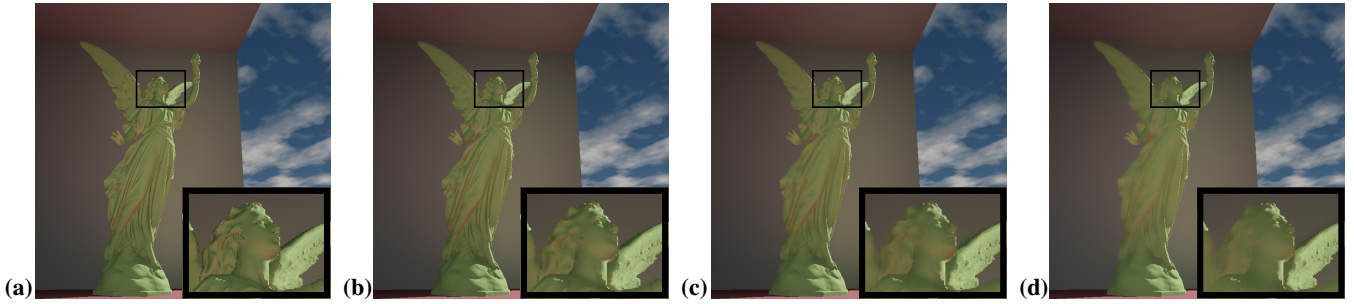


Figure 9: Influence of the tiling resolution parameter on the Lucy statue (28M triangles) in the Cornell Box model. The viewport resolution is 1024×1024 pixels and we vary the tiling level from 0 (a) to 3 (d). Because of indirect light interpolation, Lucy’s face details are more and more blurred while reducing resolution. At the same time, the indirect splatting time is reduced from 93 ms (a), to 48 ms (b) and 33 ms (c, d). (c) and (d) rendering times do not vary because our algorithm is no more fillrate bottlenecked at this level.

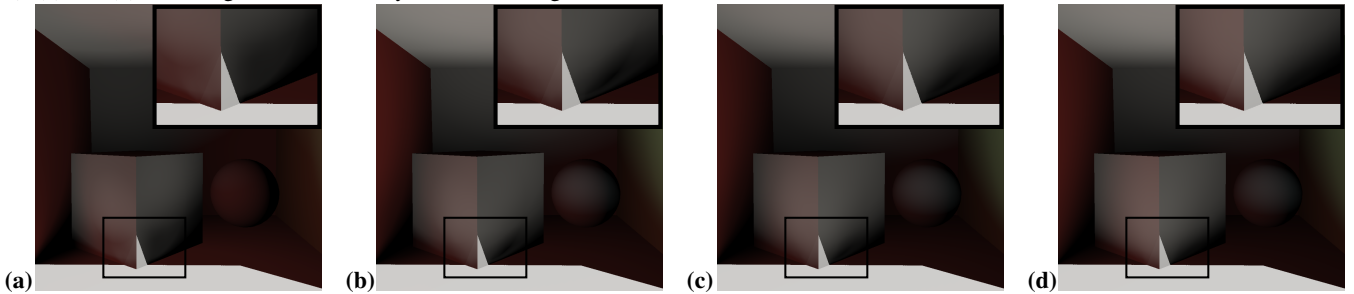


Figure 10: Indirect illumination computed with our algorithm on the Cornell Box model at 1024×1024 resolution. We fixed the number of partitions level to $N = 5$ and we compare the results by varying N_{avg} , *i.e.* the approximate number of VPLs influencing every pixel. Rendering times are directly proportional to this number – (a) $N_{avg} = 64$ is rendered in less than 1ms, (b) $N_{avg} = 128$ in 3ms, (c) $N_{avg} = 256$ in 7ms, (d) $N_{avg} = 512$ in 14ms. The close-up shows the typical artifacts appearing when N is not high enough.

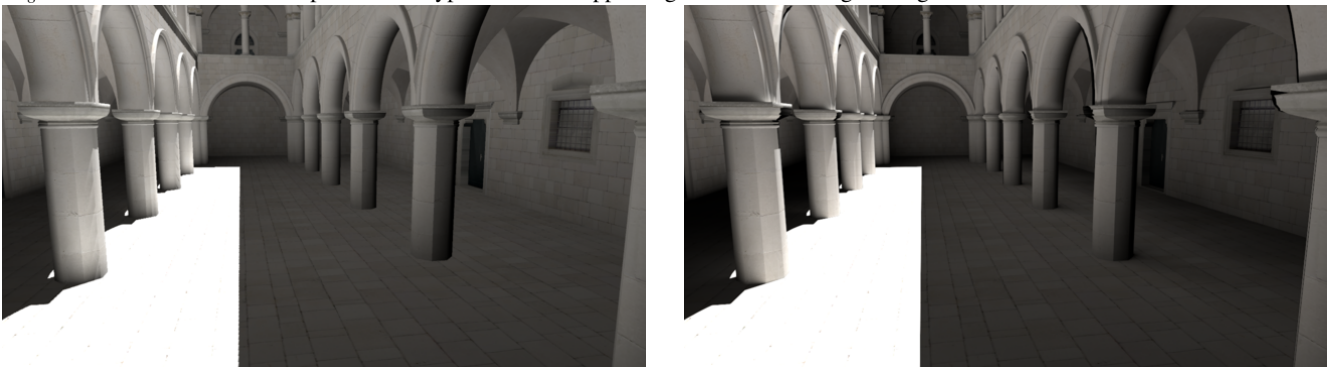


Figure 12: **Limitation.** Comparison between our technique (left) and a ground truth solution (right) for the first indirect light bounce. The ground truth is rendered with path tracing using 8196 samples per pixels. We can observe the missing contact indirect shadow in our solution, mainly visible at the pillar bases on the right. Underneath the right archs also appears much brighter than the reference in our solution, which is due to the fact that the entire left facade illuminates this region without being occluded by the first floor.

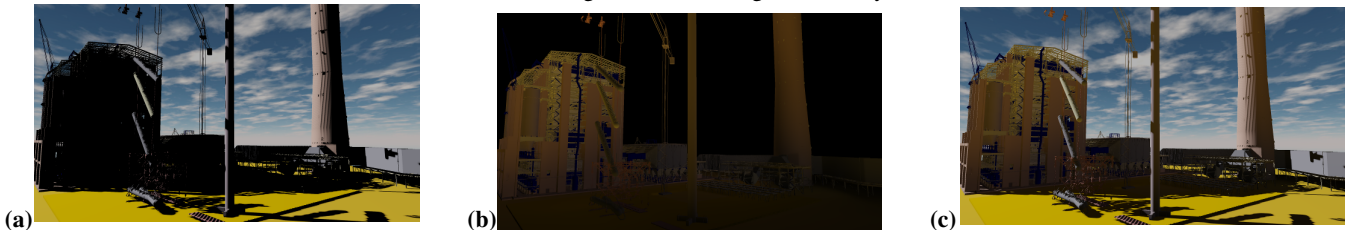


Figure 13: FLC on the Power Plant model (12M triangles). The scene is rendered at 2560×1440 pixels resolution in 8 ms for the direct lighting (a) and 25 ms for the direct and indirect lighting (c) per frame.