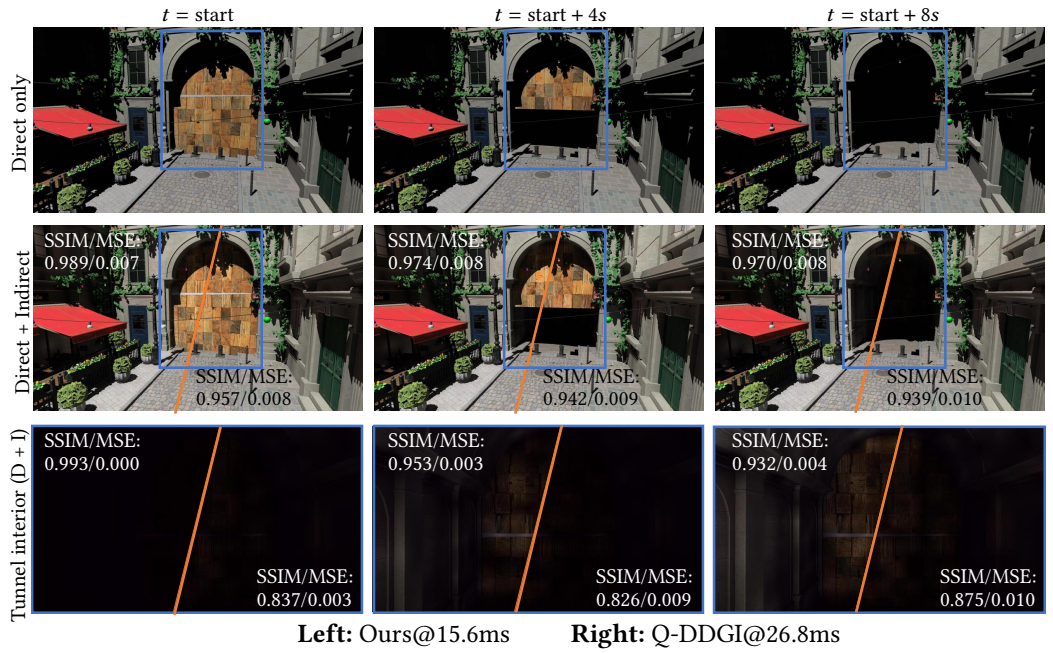
Adaptive Dynamic Global Illumination

SAYANTAN DATTA, McGill University, Canada
NEGAR GOLI, Huawei/AMD, Canada
JERRY ZHANG, Huawei, Canada



Left: Ours@15.6ms **Right:** Q-DDGI@26.8ms

Fig. 1. Our technique demonstrated on a modified Bistro Exterior scene containing $192 \times 64 \times 192$ probes. The third row shows the changes inside the tunnel as the gate opens over time. Our techniques responds faster to a dynamic stimuli and offers 1.7-times higher performance compared to the Q-DDGI implementation even with large probe grid containing excess of 2.3 million probes. Q-DDGI, detailed in section 5, is an extension of vanilla DDGI making it more competitive and comparable against our approach.

We present an adaptive extension of probe based global illumination solution that enhances the response to dynamic changes in the scene while while also enabling an order of magnitude increase in probe count. Our adaptive sampling strategy carefully places samples in regions where we detect time varying changes in radiosity either due to a change in lighting, geometry or both. Even with large number of probes, our technique robustly updates the irradiance and visibility cache to reflect the most up to date changes without stalling the overall algorithm. Our bandwidth aware approach is largely an improvement over the original *Dynamic Diffuse Global Illumination* while also remaining orthogonal to the recent advancements in the technique. [1] [2]

CCS Concepts: • **Computing methodologies** → *Ray tracing*; *Rasterization*.

Additional Key Words and Phrases: Adaptive sampling, irradiance probes, global illumination, real-time

1 INTRODUCTION

Global illumination (GI) strikingly improves the realism of a virtual scene, but its high computational cost has been a long-standing challenge in its application to real-time rendering [22].

[1] Project Page
[2] Poster

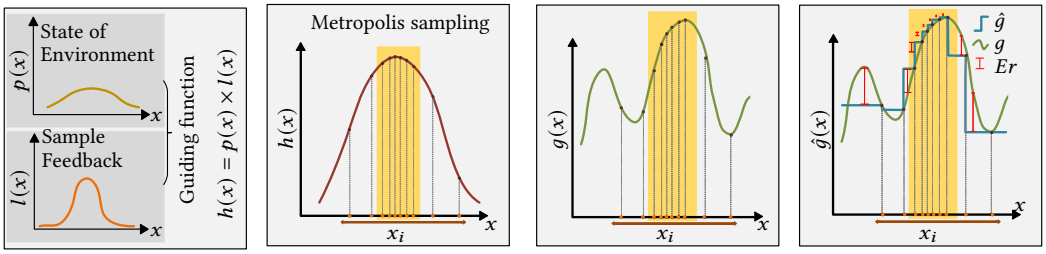
Several real-time GI solutions have been proposed, such as screen space [43] techniques, which support fully dynamic scenes but suffer from quality issues due to the limited availability of information in screen space. On the other hand, baked texture light-maps only support static geometry but remain popular due to their simplicity, low run-time cost, and quality. Precomputed Radiance Transfer [51] combined with light probes [31] and light-maps [15] solved some of the issues plaguing static light maps; in particular, these approaches support semi-dynamic geometry and self-occlusion while adhering to a strict compute budget. The advent of real-time ray-tracing hardware set the stage for modern fully dynamic GI. Dynamic real-time GI methods build upon the decades of research in sampling, and amortization of shading and visibility across space (pixel/world), angle, and time to improve convergence [40]. Adaptations of several offline techniques such as photon mapping [17], many-light rendering [20, 62], and radiosity maps [54] have also been explored in the context of modern [26, 27] ray-tracing capable hardware. However, presence of noise in sampled algorithms require the use of strong denoisers. Machine learning denoisers [6, 66] have demonstrable advantages in terms of quality compared to more traditional frequency [32] or variance [46] based denoisers. However, the prospect of training a neural network, the added complexity of integrating machine learning inference with traditional graphics pipeline, and the proprietary nature of machine learning frameworks have stalled the industry-wide adoption of these techniques. The recent probe-based algorithm, Dynamic Diffuse Global Illumination (DDGI) [28], extending the classic irradiance probes, still remains an excellent choice due to its relative simplicity, quality, and cloud streaming capabilities [14, 53]. However, scaling of DDGI in its original formulation is limited, and approaches such as multi-grid hierarchy and probe rolling [29] are necessary to scale it across large environments. Our adaptive approach focuses on dynamic contents in environments containing millions of probes in a single hierarchy.

We propose Adaptive Dynamic GI (ADGI) algorithm where we trace a few pilot rays per frame to scan the environment and build a coarse representative model of the dynamic events. Using Markov-Chain sampling, we dynamically allocate resources to the critical areas, improving convergence in those regions. While DDGI allocates a fixed number of samples per probe and uniformly distributes samples across directions, ADGI non-uniformly samples the joint spatio-angular domain of the discretized 5D light-field represented by the probes. Our approach essentially decouples resource allocation from the number of probes resulting in a user-controlled performance target (FPS) and improved scaling even with millions of probes. Additionally, our approach results in faster convergence in static and dynamic environments given equal render time. Our approach is drop-in compatible with the original implementation and its several other extensions such as probe rolling and probe volume hierarchies [29].

We achieve these objectives by formulating a *guided function approximation* technique, which is purposefully accurate in specific regions highlighted by our *guiding function* and thus eliminates the need for uniform resource allocation. Furthermore, we develop a sampling methodology based on temporal Markov-chain, which adapts naturally to a dynamic environment while also enabling scaling across large number of probes. Finally, we discuss memory and bandwidth preserving color compression schemes tailored specifically for our purpose.

2 RELATED WORK

Probe-base approaches: Modern games rely extensively on light probes for static and dynamic global illumination due to their ease of integration into the game engine pipeline at low run-time cost. Some advocate a uniform grid probe placement due to their simplicity while others have proposed non-unform probes due to their efficiency. Probe based techniques are usually prone to light leakage. As such, uniform grid approaches [28, 31] use additional information, stored in the probes to determine whether a probe is visible from a shade point. Non-uniform approaches may



(a) Construct the guide $h(x)$. (b) Sample $x_i \sim h(x)$. (c) Evaluate objective $g(x_i)$. (d) $\hat{g}(x)$ - Reconstruct $g(x)$ from $g(x_i)$. Er indicates the reconstruction error.

Fig. 2. Figure showing the steps in our adaptive-sampling strategy. We define a guiding function $h(x)$ that highlights (in yellow) the interesting regions of the domain. The samples x_i obtained from $h(x)$ are used to evaluate the objective $g(x_i)$. Our goal is to obtain an approximate representation of $g(x)$, denoted as $\hat{g}(x)$, from the $(x_i, g(x_i))$ pairs. As more samples are obtained from the highlighted region, the reconstruction error is lower in the yellow area, as shown in sub-figure (d).

use carefully curated probe placement [63] combined with spatial data-structures like octrees to determine the visibility of a probe from a surfel. McGuire et al. [28, 31] stores the depth values of the surrounding geometry from a probe and use a similar idea as Variance-Shadow-Mapping [9] to approximate visibility. However, non-uniform approaches has been mostly limited to static geometry due to their high initial construction cost. Some approaches use rasterization [31, 63] while other may use ray-tracing [28] to compute the probe content. Probe based techniques also differ on how they store the information in the probes. Some use discrete textures [28, 31] while other may use a compressed basis representation such as Spherical Harmonics [14, 55]. Spherical harmonics implicitly pre-filters the content before storage but may cause light and dark ringing issues. Memory bandwidth required for reading and writing from the probes is also a major concern. Texture compression [31, 53] is usually the preferred choice to minimize memory bandwidth. Bandwidth is also crucial for cloud streaming of probe data. In such scenarios, Spherical Harmonics [14] representation may be preferable as they provide excellent compression for low frequency data. At run-time, dynamic probe based [28] GI solutions uniformly distributes rays across probes to update their content; this quickly becomes a bottleneck as the number of probes increases. Our approach on the other hand, focuses on the optimal distribution of resources to maximize visual fidelity. Various extensions have also been proposed to increase scalablity [29] of uniform grid approaches such as multiple-volume hierarchies and probe rolling. Our approach remains largely orthogonal and fully compatible with these extensions.

Adaptive sampling: Adaptive sampling has been used in the context of screen-space ray-traced global illumination where more samples are accumulated in regions with high noise and high frequency [12]. Adaptive sampling is also useful for filtering soft shadows [32], where pilot-rays model the spatial frequency of shadow-penumbra and provide the number of additional samples required at each pixel to improved convergence. Neural versions [13] of adaptive sampling has also been proposed where a neural network generates a sampling-map that is tightly coupled to a post-process neural-denoiser. Conceptually our approach is similar, but our execution is tailored for the problem of temporally coherent sampling of probes. We refer readers to section 8 for an extend related work in irradiance-caching, screen-space GI and MCMC techniques

3 OVERVIEW

We focus on two primary issues with DDGI in its original formulation. First, the technique does not allow for the non-uniform allocation of resources, resulting in unnecessary probe updates in regions that are not crucial for visual fidelity. Seconds, it does not update the probes quick enough to reflect transient changes in the scene environment. Our adaptive strategy involves detecting the changes in the environment and allocating resources driven by the detected changes. While the detection phase requires allocating additional resources, our empirical evaluations suggest our non-uniform adaptive sampling compensates for the lost efficiency in the detection phase. Our detection phase also enables fast probe updates for capturing transient changes in the scene. We model our technique as *guided function approximation* where we approximate a continuous function (e.g. 5D light-field) using a discrete (e.g. probes) representation driven by a *guiding function.*

A naive approach to approximate a continuous function is to discretize the domain and reserve a representative sample for each discretization. The strategy is useful when the domain is relatively small; however, as the domain gets larger or the number of discretizations increases, it is prohibitively expensive to update all discretizations in real-time. This is one of the issues plaguing the original DDGI technique. In many applications, it is not necessary to update the entire domain uniformly; instead, we can tolerate more approximation errors in some regions than others. A simple example is foveated rendering, where errors in the periphery are less intrusive than those near the gaze center. In our case, we need the most accuracy in probes contributing to final shading.

We introduce the notion of *guiding function*, which highlights the regions where a higher reconstruction accuracy is desired. We define the guide using a product of terms - the first term represents the current state of the environment while the second term is a feedback from the sampled cache. We sample the guide using a temporally coherent Markov-chain and use the samples to update our approximate representation using a parallel thread-safe approach. Thus our approach is summarized in three steps - defining a guiding function, sampling the guide, and using the samples to update the approximate representation. We describe these steps in sections 3.2, 3.3 and 3.4 while we discuss various implementation specific details in section 4. See figure 2.

Our approach provides two distinct advantages compared to the original DDGI - approximation quality and scalability. At any time, we concentrate our resources on a potentially challenging area as opposed to the entire domain. Provided our guide correctly identifies the challenging regions, the quality is improved due to a higher concentration of resources in the appropriate region. Since we sample the guide independent of the number of discretizations, the decoupling allows for a high number of statically allocated probes without affecting run-time performance. Increased discretizations improve approximation quality while the independence of sampling from the number of discretizations improves scalability. More specifically, we transparently increase the number of discrete probes without affecting performance. The run-time performance depends on the number of samples we generate; the samples are channeled to the appropriate areas by the guiding function. Our Markov-chain sampling is highly parallel, temporally coherent, and scalable, making it suitable for real-time temporally distributed reconstruction of large probe grids.

3.1 Background

Here we briefly describe the original DDGI algorithm. DDGI consists of a 3D grid of directionally resolved irradiance probes that are updated in real-time through hardware ray-tracing. The probes also contains visibility information to prevent light leakage. The probe representation has many benefits, it performs optimally for diffuse indirect transport and is relatively inexpensive to encode and decode information to and from the probes. The algorithm evenly distributes ray-samples outwards from the probe center at each active probe in a stochastic rotated spiral pattern. DDGI is

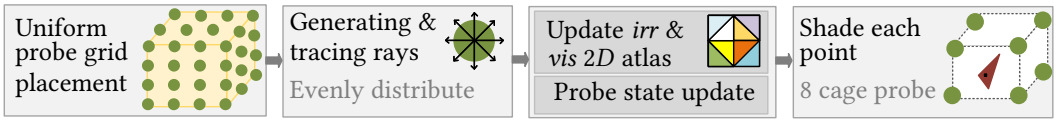


Fig. 3. The figure illustrates the main steps of DDGI algorithm. Algorithm defines a uniform grid of probes and trace uniform-random rays in all direction from each probe. Based on the hit information, we compute the visibility (vis) and irradiance (irr) and update the $2D$ atlas. We also update the probe states based on visibility information (back-face hit ratio). Finally, for each shade-point, we query the eight bounding probes surrounding it and interpolate them to compute incoming indirect illumination.

a two step algorithm. First, it updates the shading on the probe texels. Next a screen-space pass where the up-to-date probe content is used for shading the camera-pixels. The probe texel values are encoded into a spherical-mapped diffuse irradiance-texture with 8×8 resolution. Probes also captures the average ray-hit distance, and squared distances to the nearest geometry at 16×16 resolution. DDGI temporally filters the probe texels by blending in the new values using a fixed hysteresis. The visibility data is used to decide whether a probe is visible at a shade-point and also used to infer whether a probe is inside a geometry and deactivated. The probe's state is not limited to on or off and can vary with scenarios [29]. The world-position of the screen-space pixel is used as a key to the probe-texture lookup. The lookup interpolates the corresponding eight probes of the grid voxel containing the shade-point. The algorithm is illustrated in figure 3. DDGI algorithm is suitable for diffuse and slow changing phenomena in time. Therefore DDGI, combined with our adaptive-sampling strategy is a reasonable real-time GI approximation for dynamic scenes.

3.2 Guiding function

As summarised in section 3 and figure 2, a *guiding function* highlights the important areas in the domain, i.e., challenging regions where more resources are required. These highlighted areas receive more adaptive samples, reducing the approximation error in those regions. Mathematically, the domain of the guiding function $h : R^d \rightarrow R$ is the continuous 5D light field. Upon query, the guide function returns a scalar value indicating the importance of a sampled point. In our case, $d = 5$ as the domain is a 5-dimensional space of world-space positions and directions, and the guide encodes the importance of sampling a direction on a probe (texel's importance).

We model the guiding function (h) as a product of two terms. The first term, we call $f : R^d \rightarrow R$, represents the value in sampling a texel based on our understanding (limited) of whether such a texel would contribute towards the final screen-space shading. The second term is the observed sampled evidence (a.k.a irradiance cache) as they become available. Initially, the irradiance cache is empty but filled progressively through sampling. We define the first term based on some heuristics that describes our understanding of the probe-environment:

- Probes closer to the camera,
- Probes closer to geometric surfaces,
- Directions on the probes facing away from geometric surfaces,
- Directions on the probe with higher incoming irradiance,
- Directions with temporal change in irradiance and visibility

We trace *pilot rays* from the probes to generate the information necessary to quantify the above heuristics. We also call it the *detection phase* where we pre-scan the scene environment for changes. We denote the individual heuristics as $f_i : R^d \rightarrow R$, and compose them into its final form f as shown in equation 1, where ϕ represents a composition function. The composition function is simply a

Algorithm 1: Metropolis algorithm

Input: h: Guide distribution, M : No. of iterations
Input: K: No. of initial samples to reject
Output: x : Sample
Ensure: $M \geq 2$, and $K < M$

```
1 j ← ShaderInvocationIndex()
2 x_0 ← S[j]                                   // Initialize Markov-chain from memory
3 while i ← 0 to M − 1 do
4     x_{i+1} ← RandomWalk (x_i, h(x_i))       // Random walk step, algorithm 5
5     if i > K then
          /* Use sample x_{i+1} for probe updates, see algorithm 2          */
6 S[j] ← x_i + 1                               // Save Markov-chain state
```

recipe to appropriately combine the individual heuristics. We quantify the individual heuristics (f_i) in section 4.1 and the composition (ϕ) in section 4.2.

$$f = \phi(f_0, f_1, ..., f_i). \tag{1}$$

The second term uses the stored irradiance in the probes, denoted by $\hat{g}$, to modulate the first term. We model the second term as - $exp\ (\alpha \cdot \hat{g}(x)/f(x))$, where the scalar $\alpha \in [0, \infty)$ indicates our confidence in the irradiance probe content; a higher value indicating greater confidence. Note that a stored texel with high irradiance value may or may not have a high contribution to the final shading. Example - in a dynamic environment the probe content from the last frame is quickly outdated and thus less useful. The parameter α models this uncertainty. The term $f(x)$ in the denominator ensures that we only trust $\hat{g}(x)$ when $f(x)$ is low. Finally, we define the guiding function as:

$$h(x) = exp\left(\alpha \cdot \frac{\hat{g}(x)}{f(x)}\right) \cdot f(x). \tag{2}$$

3.3 Sampling the guide

Next we sample the guiding function (equation 2). Mathematically, given an unnormalized distribution $h : R^d \rightarrow R$, our goal is to obtain samples x_i from $h(x)$, where $x_i \in R^d$.

Our sampling algorithm is straightforward. We use the Metropolis sampling, as shown in algorithm 1 to sample h. The algorithm randomly initializes a state ($x_0 \in R^d$) and moves the state forward based on the acceptance of a newly proposed state. We generate the proposed states by perturbing the current state with a zero-mean Gaussian noise, also known as *Random-walk* [5].

Parallelism: Note that algorithm 1 runs as a shader invocation, meaning several instances of the chain run in parallel. Each instance is independent with its own memory to load and store the chain state (denoted by S[] in algorithm 1). The instances generate thousands of samples per frame. As an input to our algorithm, we explicitly specify the number of chains that run in parallel, thus controlling the number of adaptive samples and performance. Contrasting with the original DDGI, the number of samples in the original implementation is proportional to the number of probes which increases cubically with scene dimensions. As such, it is difficult to scale up when the scene gets larger or when using a denser probe grid. Our approach is independent of the discretization resolution and scales better to higher probe counts without compromising approximation quality.

Mixing-time: Initially, a Markov chain requires many iterations for the chain to generate samples from the target distribution (here $h(x)$), a phenomenon known as mixing time. We avoid

Table 1. List of symbols

Symbol	Description	Remarks
f	Heuristics model	Section 3.2
h	Guiding function/Target distribution	Section 3.2, 3.3
g	Objective function	Symbolic proxy for g_r, g_c. Section 3.4
$\hat{g}$	Approximation of objective function	Symbolic proxy for $\hat{g_r}$, $\hat{g_c}$. Section 3.4
g_r	5D Light field	Section 3.4
g_c	Chebychev visibility	Section3.4
$\hat{g_r}$	Approximation of 5D light field (Irradiance cache)	Section 3.4, 4.5
$\hat{g_c}$	Approximation of Chebychev visibility (Visibility cache)	Section 3.4, 4.6
x or x_i	Markov-chain samples	Symbolic proxy for p_i, ω_i. Section 3.3
p_i	Positional ($\in R^3$) component of x_i	–
ω_i	Directional ($\in R^2$) component of x_i	–

this problem by bootstrapping the initial chain state from the last frame. As such, we keep the number of iterations per frame small, but over frames, the chain effectively accrues many iterations.

Distribution stationarity: Markov chain sampling requires the target distribution $h(x)$ remain stationary. Due to a dynamic scene environment, the stationarity condition is seemingly violated. This may affect the approximation quality of our technique if the distribution changes rapidly between frames. However, we have several contingencies to deal with the issue. First, we target high frame-rates, which minimizes the change in the target distribution between consecutive frames. As an additional margin of safety, we reject initial K samples per frame as shown in algorithm 1, line 5. This ensures our usable samples are obtained closer to the target distribution. Note that the evaluation time for $h(x)$ negligible and thus rejecting few initial samples per frame does not significantly impact performance. We also smooth out the target distribution (see section 4.1.4) using spatio-temporal convolution to minimize abrupt changes in the target across frames.

Temporal tracking: Since our target distribution may vary with time, we require the samples generated from the Markov-chain to closely follow the distribution to capture the transient changes in the environment. We make some crucial modifications to our sampling algorithm to allow for fast tracking of the target distribution, which we discuss in detail in section 4.9.

3.4 Approximation

With samples obtained from the highlighted (figure 2(b)) parts of the domain, we focus on using the samples to evaluate (figure 2(c)) and reconstruct (figure 2(d)) our objective function. The term *objective function* refers to the quantity we aim to approximate. Mathematically, we denote our objective function as $g : R^d \rightarrow R^c$, and its approximate reconstruction as $\hat{g}$. For ADGI, we have two objective functions - the light field $g_r : R^5 \rightarrow R^3$, and Chebychev-visibility $g_c : R^5 \rightarrow R^2$ surrounding the probes. We denote their approximate reconstructions as the irradiance cache $\hat{g_r}$, and the visibility cache - $\hat{g_c}$ respectively. See section 4.5 and 4.6 for more details.

Updating $\hat{g}$: We evaluate the continuous objective function g at collected sample points x_i and store the evaluations - $g(x_i)$ into $\hat{g}$, as shown in algorithm 2. For ADGI, the evaluation step involves

Algorithm 2: Approximation algorithm

Input: x: Markov-chain samples

function UpdateRepresentation(x):

    $v \leftarrow g(x)$ // Evaluate sample, ray-trace

    AtomicMovingAvg(x, v) // Populate $\hat{g}$, see algorithm 4

tracing a ray to query the local light field and visibility. At each Metropolis iteration, the evaluated samples update the closest entry in the probes ($\hat{g}$) within a critical section construct.

Representing $\hat{g}$: Prior work represent $\hat{g}$ as either as discrete LUTs [28], continuous Spherical Harmonics [14], Neural Networks [36], or any combination. In our case, the choice to use a discrete representation is based on several factors. First, multiple parallel streams of Markov-chain samples may update the same memory location in $\hat{g}$. As such, provisions are necessary to prevent race conditions. We also need a representation that handles temporal accumulation and quickly update itself to reflect any transient changes in the scene. Finally, the representation must be bandwidth efficient to improve the read and write performance. We refer to section 4.5 and 4.9 for details.

3.5 MCMC analysis

In this section, we analyze our adaptive sampling algorithm in the context of MCMC (Markov Chain Monte Carlo). Note that our goal is not variance reduction through importance sampling; rather the focus is guided approximation of the objective function via sampling the target function. As such, unlike importance sampling, the sampling function is not necessarily correlated to the integrand. With this distinction in mind, we first look at the equation driving importance sampling using MCMC and then repurpose it for guided function approximation.

The following equation shows a typical case of importance sampling where the objective is to compute the integral $\int h(x)g(x)dx$ and there exists a strategy to sample from h(x). In many typical scenarios (e.g. full Bayesian inference), the distribution $h(x)$ is a proper distribution ($\int h(x)dx = 1$) but does not have an efficient sampling mechanism. This where Markov Chain MC is useful.

$$\int h(x)g(x)dx \approx \left\{\frac{1}{M}\sum_{i=0}^{M-1} g(x_i)\right\} \int h(x)dx, \quad x_i \sim h(x). \tag{3}$$

In contrast, our choice of Markov Chain (Metropolis) is primarily technical - simplicity, GPU parallelism and temporal sample tracking. Nevertheless, the same equations provide meaningful insight - albeit in a different context of adaptive sampling. In our algorithm, we simply sum the samples obtained from the target distribution without taking into account the sample density. This is equivalent to computing the following:

$$I = \frac{1}{M}\sum_{i=0}^{M-1} g(x_i), \quad x_i \sim h(x). \tag{4}$$

While our goal is to estimate $\int_\Omega g(x)dx$, the expectation of I (rearranging equation 3) is:

$$\mathbb{E}[I] = \frac{\int_\Omega h(x)g(x)dx}{\int_\Omega h(x)dx}, \tag{5}$$

where Ω is the domain of integration. Clearly, the expected value of I does not converge to the correct estimate - $\int_\Omega g(x)dx$. However, there are two factors to consider - size of the domain Ω and

shape of $h(x)$ in the domain. First consider the limit case where $\Omega \rightarrow 0$. In this case, the integrals collapses to a point evaluation and indeed the expected value of I equals the unbiased estimate as shown below.

$$L.H.S. = \lim_{\Omega \to 0} \frac{\int_\Omega h(x)g(x)dx}{\int_\Omega h(x)dx} = \frac{\int_\Omega h(x)g(x)\delta(x-x_0)dx}{\int_\Omega h(x)\delta(x-x_0)dx} = g(x_0). \tag{6}$$

$$R.H.S. = \lim_{\Omega \to 0} \int_\Omega g(x)dx = \int_\Omega g(x)\delta(x-x_0)dx = g(x_0). \tag{7}$$

In the above equation, δ is the Kronecker delta. The result is important as it shows with increasing probe resolution, bias is reduced. However, reducing texel size is not always practical as more rays and memory are required to populate and store a high resolution probe. Notice how the term $h(x)$ is cancelled in equation 6. When the domain of integration is sufficiently small, $h(x)$ is practically constant and the term cancels out in the denominator and numerator.

We now consider the shape of $h(x)$. While the target h(x) varies globally, it is piece-wise constant at a local scale due to its tabular nature. More crucially, the target $h(x)$ is stored at a much lower resolution compared to the irradiance probe $\hat{g}(x)$. This implies $h(x)$ is practically constant across a texel of the irradiance probe. The expected value of I for the k^{th} texel is thus given by:

$$\mathbb{E}\left[I_k\right] = \frac{\int_{T_k} h(x)g(x)dx}{\int_{T_k} h(x)dx} = \frac{\int_{T_k} c_k g(x)dx}{\int_{T_k} c_k dx} = \frac{\int_{T_k} g(x)dx}{\int_{T_k} dx}, \tag{8}$$

where T_k represents the domain of k^{th} texel and c_k represents the piece-wise constant value of $h(x)$ when $x \in T_k$. The area estimate $\int_{T_k} dx$ is fixed for all texels and equivalent to $4\pi/\#resolution$. Thus, due to the tabular nature of our target function, the estimates of irradiance texels remain un-biased. While performing texture filtering over irradiance texels, it is possible to compute an unbiased estimate by weighing the texel values with c_k as follows:

$$I_k^{filter} = \sum_{j \in \mathcal{N}_k} w_{k-j} I_{k-j}, \quad w_i = c_i / \sum_{j \in \mathcal{N}_k} c_j, \tag{9}$$

where $\mathcal{N}_k$ represents the texels in the neighbourhood of texel k. The values c_i are obtained by querying the probes storing $h(x)$. Note that bias is unavoidable as we blend samples temporally in a dynamic environment. In a dynamic environment, the objective is evolving and the bias manifests itself as temporal lag. Practically however, within a small time window, both $h(t)$ and $g(t)$ are assumed constant and the samples can be blended using a windowed moving average. Note that windowed moving average requires storing historical information. A cheaper but biased approximation to windowed moving average is exponential moving.

4 IMPLEMENTATION DETAILS

This section provides the several implementation details with a brief summary in figure 4.

4.1 Heuristics construction

The section describes the construction of f using the heuristics discussed in section 3.2. Our goal is to measure and quantify the heuristics that highlight the probes which actively contribute to the final shading and require additional resources for faster convergence. We represent the heuristics either parametrically (equation 10) or using an explicit LUT representation as shown in figure 5(a). The LUT is constructed such that each probe has eight texels corresponding to an octant. We trace a ray for each octant; the rays return the hit distance and incoming irradiance at the hit-point. From this information, we compute several quantities (equation 11 - 18) and store them in the

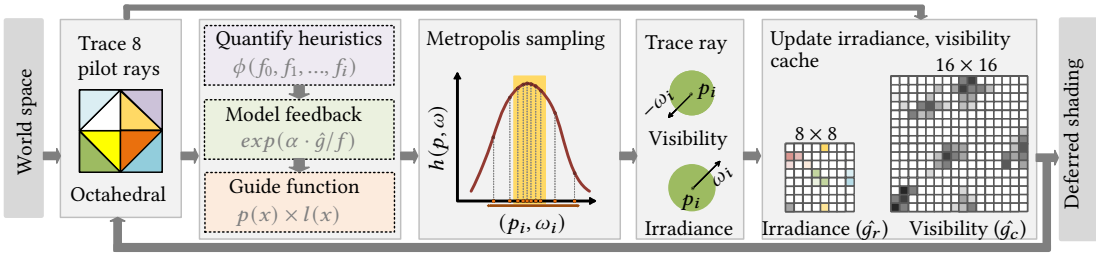


Fig. 4. This figure illustrates our overall algorithm. We trace 8 pilot rays, one from each octant on the probe and approximate the heuristic model $f(p, \omega)$. Using the heuristic and feedback, we define the guide $h(p, \omega)$ and sample it using Metropolis sampling. The sampled (p_i, ω_i) are used to trace more adaptive ray samples, gathering hit-distance and irradiance at the sample points. We update the probe-cache ($\hat{g}$) with adaptive-samples. The cache is used in the next shader and also looped back as feedback to model the target.

LUT/texture mapped to the probe octants. We define and evaluate the following heuristics for a probe at position p and a direction ω.

4.1.1 Distance from camera. A probe far away from the camera is less likely to contribute to the final shading. We represent this parametrically as described in equation 10, where p represents probe position, c camera position and k is a threshold set by the user.

$$f_c(p, \omega) = \begin{cases} 1 & \text{if } ||p - c|| < k\,, \\ e^{-(||p-c||-k)} & \text{otherwise.} \end{cases} \tag{10}$$

4.1.2 Probe visibility. Only the probes encompassing a geometry participates in the deferred shading. Thus, probes closer to a geometric surface are more important. Similarly, texels facing away from the surface are queried more often for shading. We express both quantities together in equation 11, where p represents probe location and $t = trace(p, -\omega)$. The function *trace* returns the distance of the nearest surface hit, and the scalar s is the diagonal distance of a grid voxel.

$$f_v(p, \omega) = e^{-2t/s} \tag{11}$$

4.1.3 Incoming radiance. We consider directions with high incoming radiance as more important. To identify those directions, we query the radiance along each probe octant and use it as a representative for incoming radiance.

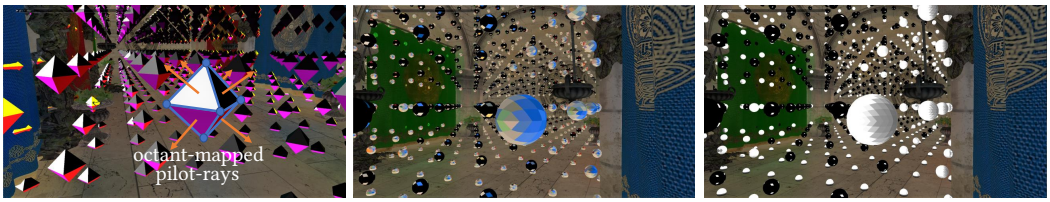
$$f_r(p, \omega) = \frac{min(r, \beta)}{\beta}, \tag{12}$$

where $r = lum(p, \omega)$. The function *lum* returns the incoming luminance using direct illumination at the surface hit point. The parameter β controls the dynamic range and we set $\beta = 5$.

4.1.4 Probe visibility change . Detection of dynamic geometry is crucial for increased resource allocation in regions affected by these changes. We detect dynamic geometry by computing a temporal gradient of probe visibility followed by a spatio-temporal smoothing operation.

$$f_0(p, \omega) = f_v^t(p, \omega) - f_v^{t-1}(p, \omega), \tag{13}$$

where f_v^t, f_v^{t-1} represent visibility in the current and last time step respectively. Equation 13 implicitly states we keep the position and the direction fixed when measuring the time difference across frames to avoid noisy gradients. The gradient is passed through a temporal trigger (Tr) as:



a. Probes storing prior-information (f). b. Irradiance cache ($\hat{g_r}$). c. Visibility cache ($\hat{g_c}$).

Fig. 5. Figure showing various probe-mapped textures and LUT in our technique.

$$f_1(p,\omega) = Tr\left(f_0(p,\omega),\theta\right), \tag{14}$$

where Tr converts a pulse in time to a decaying signal controlled by the parameter θ as shown in figure 6(a). For simplicity, we drop the time axis from the function Tr. The function minimizes temporal discontinuities, thus helping the Markov-chain to closely follow the target distribution (h) across frames. Finally, we perform a spatial convolution as follows:

$$f_{\Delta v}(p,\omega) = \sum_{i,j} f_1(p - p_i, \omega - \omega_j). \tag{15}$$

The convolution step smooths out uncertainties in a single texel and also serves as a weak predictor of possible locations of the dynamic geometry in the next frame. We use a $5 \times 5 \times 5$ and 3×3 convolution in space and direction, respectively.

4.1.5 Probe radiance change. Similar to the previous section, we detect a change in radiosity using a temporal gradient of the probe radiance. We apply the same temporal trigger and spatial convolution operator as in the previous section. The corresponding equations are as follows:

$$f_2(p,\omega) = f_r^t(p,\omega) - f_r^{t-1}(p,\omega), \tag{16}$$

$$f_3(p,\omega) = Tr\left(f_2(p,\omega),\theta\right), \tag{17}$$

$$f_{\Delta r}(p,\omega) = \sum_{i,j} f_3(p - p_i, \omega - \omega_j). \tag{18}$$

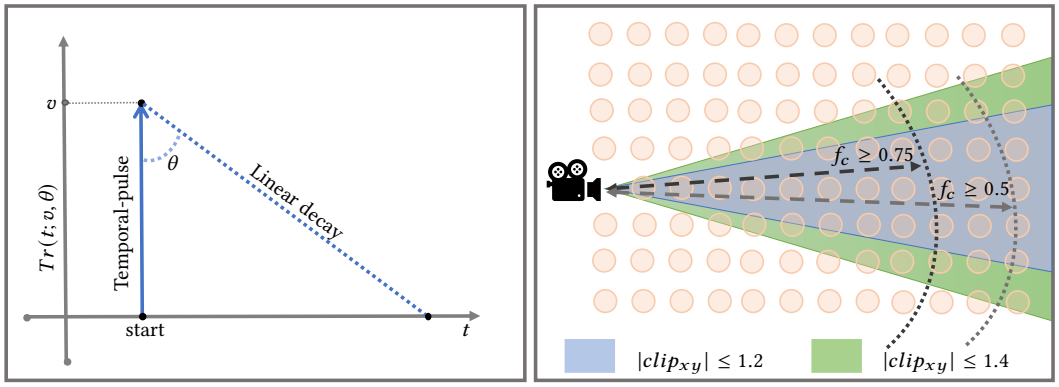
4.2 Heuristics composition

Now that the individual heuristics are defined, as described in equation 1, we compose them for the static and dynamic cases as follows:

$$f_s(p,\omega) = \overbrace{f_c f_v}^{static}, \tag{19}$$

$$f_d(p,\omega) = \overbrace{f_c f_v (f_{\Delta v} + \mu f_{\Delta r})}^{dynamic}. \tag{20}$$

When the environment is static, we sample according to the camera and probe-to-surface distance heuristics denoted by f_c and f_v in equation 19. In the dynamic case represented by equation 20, we modulate the changes in the environment by the static term $f_c f_v$. The modulation indicates we are more interested in changes close to the camera and geometric surfaces. The factor μ weighs the strength of change in geometry versus change in lighting. We use $\mu = 2$ in all our experiments.



a. Transform temporal-pulse to a decaying signal.

b. Defining clip volumes for probes.

Fig. 6. Figure (a) shows the construction of temporal-trigger $Tr(v, \theta)$. In figure (b), we call the volume bounded by the blue frustum and black boundary as inner volume V_{in}. Similarly, outer volume V_{out} is the volume bounded by green frustum and outer grey boundary. All probes in V_{out} participate in the heuristic modelling, as described in section 4.4. Probes inside the blue frustum participate in adaptive sampling as described in section 4.8, 4.9. We set the probe state $N = 16$ for all probes outside V_{in} but inside V_{out}, refer section 4.8.

4.3 Heuristics storage

We store the quantities f_v, f_s, f_d as a 6-10-10 bit encoded 32 bit integer at each octant of the probes. The remaining 6 bits are used for other flags. When querying the LUT/texture, we use a mapping function that maps the continuous position p and direction ω to the corresponding texel in the LUT. We note that f_c is implicitly defined, hence do not require additional storage.

4.4 Improving construction efficiency

The heuristics construction step is a potential bottleneck if we trace 8 rays per probe for all probes in the scene. As such, we restrict the pilot-rays to the probes that are contained within an extended camera frustum as shown in figure 6(b). To maximize the efficiency of our algorithm, we further reuse the samples collected from the 8 pilot-rays to populate the irradiance ($\hat{g_r}$) and visibility ($\hat{g_c}$) caches. We change the ray-directions at alternate frames in an *AABBCCDD...* pattern, improving the detection of temporally varying light-field surrounding the probes. We measure the time-delta (equation 13 and 16) between two frames with identical set of ray-queries, avoiding noisy gradients. However, this effectively halves the detection frequency (*frame-rate / 2*) but improves the spatial awareness. We use a stratified-random ray-direction such that there is always one ray per octant. We update the irradiance and visibility cache at each alternate frame.

4.5 Probe irradiance cache

As shown in figure 5(b), the irradiance cache ($\hat{g_r}$) is represented as a uniform probe grid in space where each probe stores the surrounding diffuse irradiance at a 8×8 texel resolution using a spherical mapping. At each texel, we store the irradiance in a custom RGB encoding with 9-9-8 bits for the three channels. The remaining 6 bits (out of 32bit) store the sample accumulation count (N), used for computing the moving average (see algorithm 3) of a sample stream in time. We take several considerations into account for the choice of our encoding. Our encoding should be bandwidth efficient and must support atomic updates on a commodity GPU. We found both DX12 and GLSL supports atomic operations on 32 bit integers. Finally, our encoding must faithfully

Algorithm 3: Moving Average algorithm

Input: x: Update location, v : New sample, N_{max} : Max sample count
Output: V : Updated value, N: Sample count

function MovingAvgUpdate(x, v, N_{max}):
- $n \leftarrow \hat{g}[x].N$ // Cumulative sample count
- $o \leftarrow \hat{g}[x].V$ // Cumulative value
- $V \leftarrow \frac{v}{n+1} + \frac{n \cdot o}{n+1}$ // Update cumulative value
- $N \leftarrow min(n+1, N_{max})$ // Increment sample count
- **return** V, N

encode intensities beyond the standard definition. We apply a non-linear color compression across the three color channels, $i \in [0..2]$ as shown in the equation below.

$$u_i = \frac{min\left(ln(\gamma \cdot v_i + 1), \beta\right)}{\beta}. \tag{21}$$

We apply an inverse transform $(exp(\beta \cdot u_i) - 1)/\gamma$ while decoding where $\beta = 5$ and $\gamma = 15$. More details regarding our choice of compression scheme is provided in appendix B and figure 11.

4.6 Probe visibility cache

As shown in figure 5(c), texels in the visibility probes store the mean distances and mean squared distances to the nearest geometry at 16x16 texel resolution. We call this $\hat{g}_c$ - our visibility cache. Each texel stores the two channels with 13 bits of precision each while the rest 6 bits are used for sample accumulation count. We normalize the distances with probe cage diagonal length. Similar to irradiance cache, we apply a logarithmic encoding as per equation 21 for efficient use of available precision. We use (β, γ) values of $(5, 15)$ and $(8, 20)$ for the linear and squared channels respectively.

4.7 Temporal sample accumulation mecahnism

We use a moving-average accumulation to store the samples in the irradiance and visibility caches. In the algorithm 3, we have two parameters N and N_{max} to control the moving-average accumulation. As we start accumulating samples, N is incremented and the algorithm performs like a true moving average. However, as N approaches $N_{max} - 1$, the algorithm switches to an exponential moving average form with hysteresis $(N_{max} - 1)/N_{max}$. Also, note that when the value of N is low, the cache updates itself quickly, but the stored values may be noisy. As N increases, the new samples are weighed less in their contribution to the cache. We exploit these parameters to control the learning rate and noise in the static and dynamic cases as discussed in the following sections.

4.8 Adaptive sampling - static

We split our adaptive sampling strategy into two stages - static and dynamic. We have two separate Markov-chain sets, each focusing on different aspects of capturing the surrounding light-field. While the static chain focuses more on the accuracy, the dynamic chain is tuned for capturing the transient responses. We discuss the dynamic chain in detail in the next section.

We set up equation 2 as - $h = exp(min(\hat{g}_r/f_s, 1)) \cdot f_s$. The feedback from irradiance cache $\hat{g}_r$ is obtained from the previous frame and from a higher mip-level (also used in deferred shader). The lowest mip-level $\hat{g}_r$ is continuously updated and thus avoided as feedback due to possible violation of stationarity condition within a frame. We use the Metropolis sampling, algorithm 1, to generate the samples $x_i \equiv (p_i, \omega_i)$. As summarized in the algorithm, 2, we use the samples to evaluate the

Algorithm 4: Atomic moving average algorithm

Input: x: Update location, v : New update value
Output: Update $\hat{g}[x]$

```
function AtomicMovingAvg(x, v):
    current ← ĝ[x]
    /* Repeat until destination value stops changing */
    do
        expected ← current
        next ← MovingAvgUpdate(x, v, 64)
        InterlockedCompareExchange(ĝ[x], expected, next, current)   // Refer HLSL
    while current ≠ expected
```

continuous light field g_r, which involves tracing a ray originating at p_i along the direction ω_i. We trace an additional shadow-ray per sample to compute the visibility in the opposite direction ($-\omega_i$) as the probe queries in the deferred shader for visibility is exactly 180° out of phase w.r.t irradiance. Next we store the irradiance and visibility values in the irradiance ($\hat{g_r}$) and visibility ($\hat{g_c}$) caches using an atomic update rule as presented in the algorithm 4. Atomic updates are required as multiple invocations of the chain may update the same location in the irradiance and visibility caches. Figure 4 summarizes the overall idea.

We set the random walk step size, denoted by $\sigma \in R^5$ in algorithm 5, proportional to the size of discretization in the irradiance and visibility cache. Thus positional step size is proportional to the size of a voxel in the probe grid, while angular step size is roughly $\sqrt{\pi/256}$. Due to the small step size, texels in the cache may accumulate more than one sample per texel, thereby accumulating a large sample count over time. We also note that our cache behaves like a true moving average between sample count $N = 0$ to 64, which also contributes to better accuracy.

The static adaptive samples are useful for improving convergence in a static scene and for slow changes that are undetected during prior construction. For example, slow changes in lighting such as day-night cycles in games. We lower the hysteresis by setting $N = 16$ for all probes in the region $\{V_{out}\} - \{V_{in}\}$ in figure 6(b). This enables the probe to quickly catch-up to the most recent values.

4.9 Adaptive sampling - dynamic

We run a second set of Markov-chain when dynamic content is detected in the scene. When there are dynamic elements, especially moving geometry, we run into two main issues. The generated samples are not well distributed in the region of interest i.e. the areas where time varying changes are present. When the step size is small, the chain cannot track the target distribution fast enough to generate samples from the target, causing the samples to lag the moving target distribution. The second problem is noise due to multi-sampling of the irradiance texel. Potentially, this can be solved by increasing the hysteresis to improve temporal sample reuse. However, the reduced noise comes at the cost of introducing objectionable temporal blur.

We solve the first issue by increasing the chain step size and by coarsening the target function (f_d). Practically, this amounts to grouping the heuristics-probes into virtual proxies. In our case, a virtual proxy represents a group the $3 \times 3 \times 3$ probes. This virtual probe has 8 directions and each direction represents an axis-aligned octant. The value of a texel of the virtual probe is the *max* of all 27 probes it represents along the corresponding direction. We also drop the sampled evidence by setting $\alpha = 0$ in equation 2, as the stale irradiance cache ($\hat{g_r}$) provide little useful information for

Table 2. Table showing probe grid details for various scenes used in our technique.

Scene	Probe Grid	Probe spacing (in meters)	Irradiance ($\hat{g_r}$) Cache Resolution	Visibility ($\hat{g_c}$) Cache Resolution
Bistro - Exterior	$192 \times 64 \times 192$	$0.5 \times 0.5 \times 0.5$	8×8	16×16
Sponza - Diffuse	$192 \times 64 \times 192$	$0.5 \times 0.5 \times 0.5$	8×8	16×16
Sponza - Glossy	$192 \times 64 \times 192$	$0.1 \times 0.1 \times 0.1$	16×16	16×16

Table 3. Table showing probe encoding details for the various techniques we use in our comparison.

Technique	Irradiance Cache Encoding	Visibility Cache Encoding	Temporal Hysteresis
Ours	$\lfloor$R9$\rfloor\lfloor$G9$\rfloor\lfloor$B8$\rfloor$ – N	[R13][G13] – N	Static: 0.98 ($N_{max} = 63$) Dyna: 0.91 ($N_{max} = 10$)
Q-DDGI	$\lfloor$R11$\rfloor\lfloor$G11$\rfloor\lfloor$B10$\rfloor$ – N	[R16][G16] – N	0.94
Reference	RGB32f	RG32f	N/A

sampling a time varying region. The chain step size is 3x, and 6x larger for position and directions, respectively w.r.t the static case.

Since each sample from the coarse chain represents an entire octant, we trace 64 rays for the octant for all underlying 3x3x3 probes in the group. We make the tracing step more efficient by culling probes that are not used in deferred shading. The scheduling of ray-direction is deterministic, passing through the center of a texel in the irradiance cache ($\hat{g_r}$). This solves the problem of sampling noise and also affords the opportunity to simplify the atomic updates. Since the rays are not random, we do not benefit from multiple shader invocations updating the same octant. As such, the first invocation to update the octant marks (atomically) it updated such that other invocations do not repeat the same work move to the next.

We run the dynamic sampling after the static sampling step. During static sampling, if a probe has non-zero dynamic component($f_d > 0$), we quantize the ray directions to go through the irradiance/visibility cache texel center to avoid injecting sampling noise in the texels.

5 RESULTS AND COMPARISONS

We compare our results with Q-DDGI and a reference probe-based implementation in different scenarios - static scene (fig. 7), dynamic geometry (fig. 1, 8, 10), and dynamic lighting (fig. 9).

Q-DDGI: Quantized-DDGI or Q-DDGI is a performance enhanced extension of original DDGI [28], achieved without major modifications to the base algorithm. Q-DDGI is equipped with a more compact irradiance and visibility cache representation that closely resembles ours. See table 3. We also enable camera-frustum culling of probes in Q-DDGI as described in section 4.4 and figure 6. These modifications allow Q-DDGI to have similar performance (table 4) at same probe count (table 2) as ours across different scenes. We believe these modifications make our comparisons more fair. We use 32 rays per probe for a total ray budget of 800-1600k (depending on scene) rays per frame.

Reference: Reference implementation uses a standard *FP32* representation for irradiance and visibility caches as shown in table 3. We also use a higher resolution 32×32 irradiance and visibility cache. Due to memory constraints, we are limited to a smaller probe-grid of size $32 \times 32 \times 32$ using same probe spacing (table 2) as other techniques. For each frame, we discard any previous values in the probes and accumulate samples using a true-average with 64 rays per texel.

Ours: We use 4096 instances of static chain invocations and 1024 instances of dynamic chain invocations. Overall, we use use between 500-900k (depending on scene) rays per frame.

Table 4. Performance breakdown of our technique and Q-DDGI. Our probe sampling stage is divided into three sub-stages - heuristic construction (P), static adaptive sampling (S), and dynamic adaptive sampling (D).

Scene	Ours (in milliseconds)			Q-DDGI (in milliseconds)		
	Probe Sampling (P + S + D)	Deferred	Total	Probe sampling	Deferred	Total
Bistro - Exterior	4.01 + 2.23 + 4.73 = 11.0	4.63	15.6	22.3	4.47	26.8
Sponza - Diffuse	1.21 + 1.85 + 3.18 =6.24	3.62	9.86	9.69	3.51	13.2
Sponza - Glossy	4.83 + 2.11 + 4.33 =11.27	6.44	19.7	24.9	6.71	31.6

Figure 1 and 8 shows a large scene (Bistro Exterior), with the tunnel's entry and exit modified with dynamic gates. The tunnel interior walls are illuminated by indirect illumination alone, controlled by the direct light bouncing off the floor. The direct illumination on the floor is controlled by the dynamic entry gate. The scene tests the tracking capabilities of our algorithm; the dynamic Markov-chain should sample the probes close to the moving door. The scene also tests our color compression scheme under low-light and moving-average accumulation.

Figure 9 shows the Sponza scene under dynamic lighting, testing the detection capabilities of ADGI in the absence of dynamic geometry. Figure 7 shows a static scene without dynamic geometry or lighting, testing the convergence of our static adaptive sampling when no dynamism is detected or the dynamic changes are too slow to detect, such as day-night cycles in games.

Figure 10 shows a dynamic geometry (Stanford Buddha) under glossy indirect illumination with ambient lighting as direct component. The scene is stressful as the camera frustum contains many times more probes compared to other scenes due to the increased probe density required for glossy illumination. This scene tests the transient response of a dynamic geometry on a glossy floor. Thus the scene is less forgiving of spatio-temporal blurring.

We measured the results on a desktop with Nvidia 2080Ti GPU and AMD 5600X CPU at 1920×1080 resolution. The performance numbers cited in table 4 are only for ADGI and Q-DDGI algorithms. The GBuffer and direct-illumination passes require an additional 2ms and 3ms, respectively.

6 LIMITATIONS

We inherit similar limitations as the vanilla DDGI algorithm. The probe visibility from a shade-point is only approximate and requires modifications such as probe movement to minimize light leakage. The probe representation is not efficient in capturing glossy light-transport and requires a dense spatio-angular discretization of irradiance cache to capture glossy reflections.

Accurate detection of transient spatio-temporal changes in a scene are difficult. The accuracy of detecting dynamic geometry reduces with the distance of the dynamic object from a probe. The same is true for dynamic lighting; especially high frequency localized lighting that is far from a probe is difficult to detect. Also, for the Markov chain to track the target distribution, the speed of motion should be capped comparable to the product of Markov-chain step size and average frame-rate. While many game engines keep track of the dynamic objects, facilitating the detection of changing in visibility, we still need ray-tracing to detect dynamic radiosity.

7 CONCLUSION

Our adaptive sampling approach improves upon the efficiency of the original DDGI algorithm. Our approach non-uniformly allocates resources in regions with time varying phenomena and captures

transient localized changes in an environment containing millions of probes. By contrast, DDGI's uniform allocation policy dilutes resource concentration in critical regions, especially when a large number of probes are present. These improvements reduce temporal lag and minimizes reliance on temporal blur to reduce noise. Our probe encoding scheme minimizes memory requirements by 4x (and by extension memory bandwidth) with minimal impact on quality while also enabling millions of probes in a scene. Our adaptive sampling stages have a fixed upper bound on the compute requirement and also decouples sampling from the number of probes, further reducing memory bandwidth requirement. These changes enable improved probe-based rendering while also enabling 1.5-2x performance improvements.

8 RELATED WORK EXTENSION

Irradiance caching Irradiance caching is another line of techniques attempting to overcome the high computation cost of GI. The irradiance caching method assumes that irradiance vary smoothly across the scene, and texture detail can be recovered using albedo modulation [64]. The interpolation and location of the various cache records is a critical, especially when the assumptions on smoothness do not hold. While robust, principled offline solutions exist [16, 24], real-time applications often resort to complex heuristics and impose harsh constraints to achieve online GI. Compression [56], sparse interpolation [49], pre-convolved environment maps [42, 45], spatial hashing [3] and using neural network [37] are instances of advancements in real-time irradiance caching. Although these approaches aim for real-time performance, their complexity and constraints make them challenging to implement and deploy.

Path tracing The flexibility and generality offered by path tracing [18] is highly desirable for real-time rendering. However, path tracing has been out of reach for real-time applications due to its substantial computational requirements. Even with the advent of hardware-accelerated ray tracing [23], it is only possible to trace a few tens of rays at each pixel in real-time. Therefore, effective sampling strategies and high-quality denoising algorithms [38, 46, 47] are essential. Many sampling

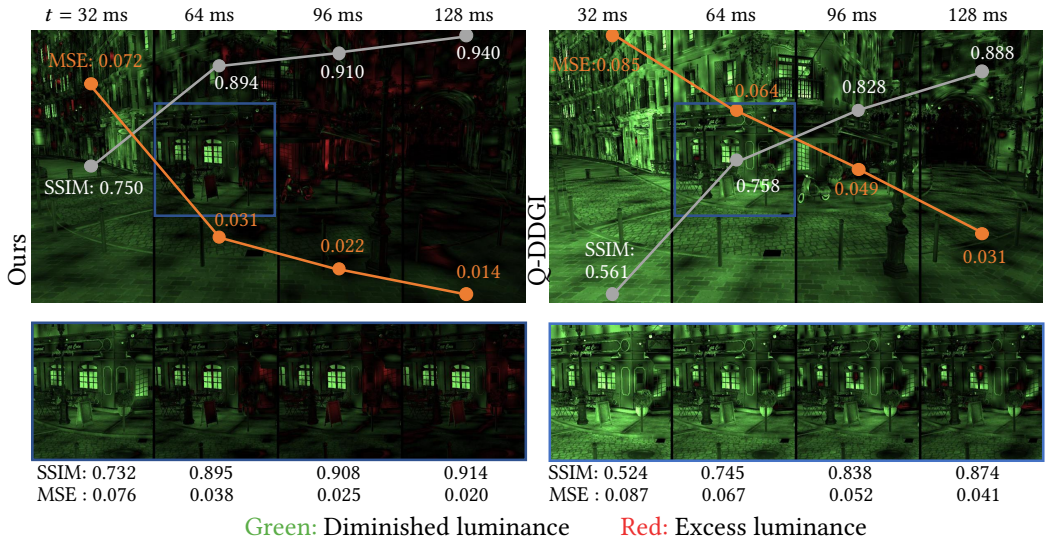


Fig. 7. Comparing the convergence of our technique over time on a static Bistro Exterior scene. The figure demonstrates the effectiveness of our static adaptive sampling step. The two rows measure the difference in luminance w.r.t reference and highlight the error in red and green color.

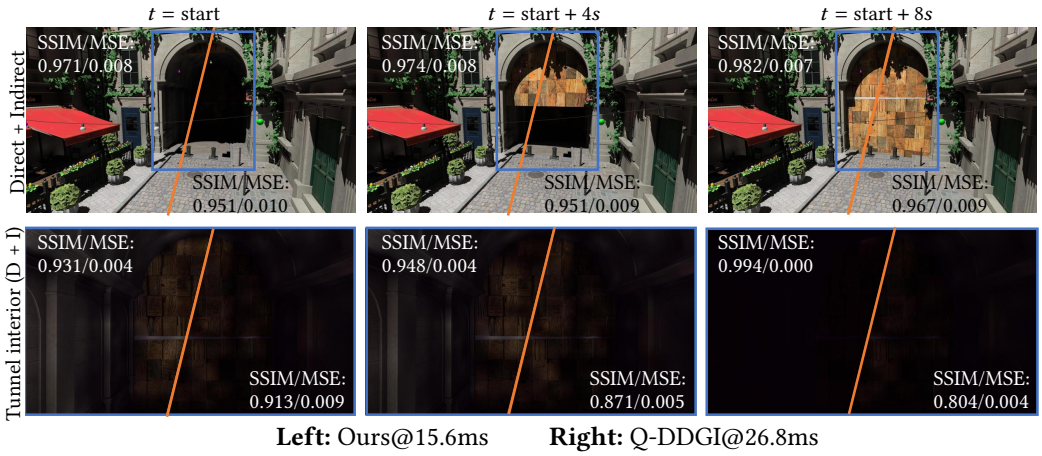


Fig. 8. Our technique compared with Q-DDGI on a modified Bistro Exterior scene augmented with a moving door. The scene has $192 \times 64 \times 192$ probes and shows the convergence of the two techniques near a dynamic area in the scene. The second row shows the changes inside the tunnel as the door closes over time. Our technique is better able to allocate the resources closer to the dynamic areas resulting in faster convergence and higher performance.

methods try to learn the representation of incident illumination during rendering [1, 8, 34, 44, 60]. While these approaches can provide substantial error reduction, constructing these structures in parallel on a GPU incurs a significant overhead that seem unsuitable for real-time applications. Recently proposed ReSTIR GI [41] provides an efficient real-time sampling strategy by reusing the paths spatially and temporally but the algorithm becomes complicated after second bounce and still requires denoising for the final stage. Deep learning has also been applied to path guiding, including work by [35, 36]. These approaches demonstrated a substantial reduction in error due to more effective path sampling, though their performance remain insufficient for real-time applications.

Screen space approaches: Approximating physically plausible illumination at real-time frame rates with screen space methods is popular in games. Screen space methods are fast, GPU-friendly, and simple to implement. Screen space ambient occlusion (SSAO) [2, 33] is part of many real-time rendering engines. Following SSAO, screen Space Directional Occlusion (SSDO) [43] is used for near-field direct and indirect diffuse lighting. Sousa et al. [52] proposed Screen Space Reflections (SSR) using a 2D ray-tracing approach directly in screen space to obtain the indirect specular component. Recently Screen-Space Global Illumination (SSGI) [43, 50, 52] methods offer a viable solution to real-time GI. However, these methods are limited by the information visible from the observer's position, thus making it difficult to engineer a robust solution.

Importance sampling and Bayesian modeling: Importance sampling provides a tool to reduce the cost of brute force integration by selectively evaluating elements of the integrand based on prior knowledge, i.e. an educated guess. Previous works in importance sampling proposed different methods to apply importance sampling to various Monte-Carlo integration existing in rendering equations [21, 48, 57]. Although Markov Chain Monte Carlo(MCMC) methods have been used in Bayesian learning from the early days of neural networks [39], and Stochastic-Gradient MCMC has been proposed [65] with various applications [25], our approach is neither Monte Carlo-based nor Neural-network learning. We exploit Bayesian inference and Markov Chains as our mathematical means to sample the important texels on the probe, by defining our guide function (prior), likelihood, and posterior.

Fig. 9. Figure comparing the convergence of our technique under dynamic lighting controlled by the direct component shown in the first row. The last two rows measure the difference in luminance w.r.t reference and highlight the error in red and green color.

Markov Chain: Markov Chains are used broadly in Monte Carlo path-tracing. For example, Veach and Guibas [58] used Metropolis Sampling to explore the space of all possible paths. Kelemen et al. [19] later applied the exact sampling in the space of random numbers, i.e., in Primary Sample Space. The most recent work by Bitterli et al. [4] combines a simple path tracing integrator with MCMC by using the random seeds of high variance paths as starting points for the Markov Chains. Although Markov Chains are encountered extensively beneficial in solving Monte Carlo sampling, our point of view on sampling and employing the Markov Chain to draw samples from the guide function is distinct.

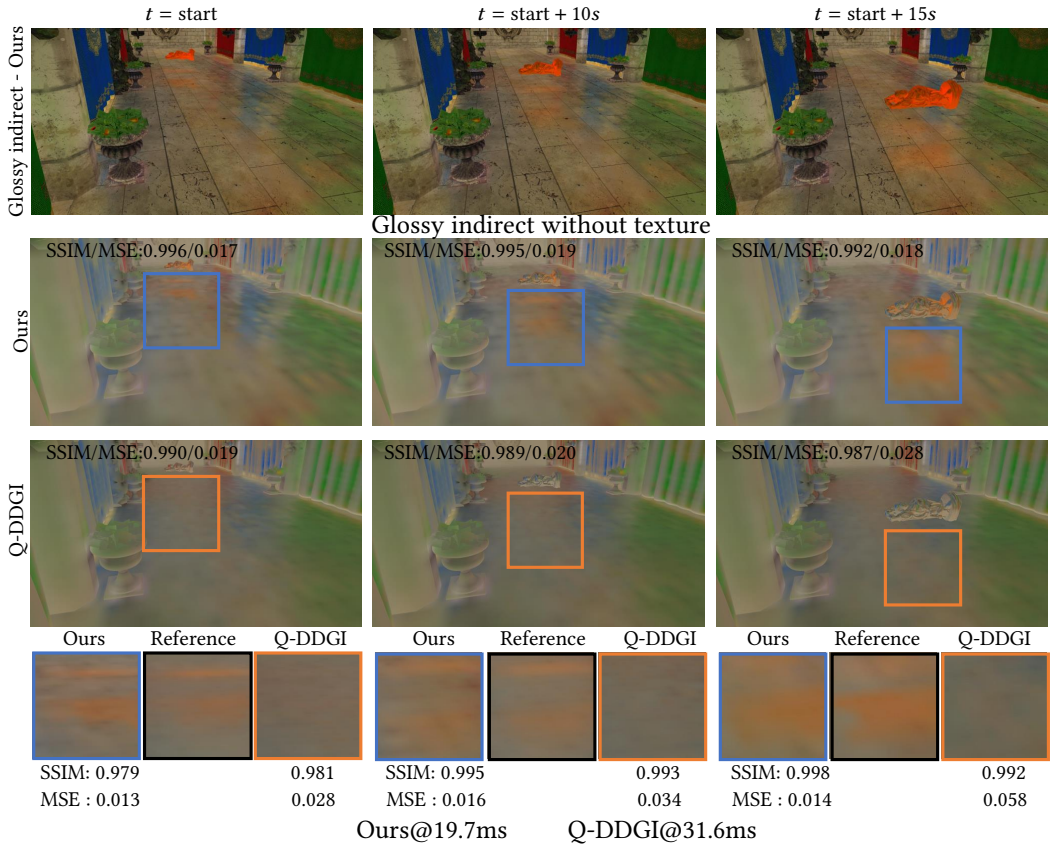


Fig. 10. Figure comparing glossy indirect reflection on a scene lit by ambient lighting. The scene tests transient response due to the moving BUDDHA geometry over a glossy floor.

Bayesian inference: Bayesian modeling is a widespread methodology in computer vision and graphics. Brouillat et al. [5] and Marques et al. [30] pioneered the use of Bayesian Monte Carlo (BMC) [11] in light transport simulation. In contrast, [59] keep the efficient classic, frequentist MC approach and apply Bayesian modeling to optimize their sampling distributions for direct illumination estimates across the scene. Similar approach is used by Vorba et al. [61], who employ a maximum a posteriori (MAP) formulation to regularize training of parametric mixture models for optimized indirect illumination sampling. Our approach uses Bayesian modeling in the context of light-probes to detect important probes and directions based on sampled evidence.

REFERENCES

[1] 2019. SIGGRAPH '19: ACM SIGGRAPH 2019 Production Sessions (Los Angeles, California). Association for Computing Machinery, New York, NY, USA.

[2] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. 2008. Image-Space Horizon-Based Ambient Occlusion. In ACM SIGGRAPH 2008 Talks (Los Angeles, California) (SIGGRAPH '08). Association for Computing Machinery, New York, NY, USA, Article 22, 1 pages. https://doi.org/10.1145/1401032.1401061

[3] Nikolaus Binder, Sascha Fricke, and Alexander Keller. 2018. Fast Path Space Filtering by Jittered Spatial Hashing. In ACM SIGGRAPH 2018 Talks (Vancouver, British Columbia, Canada) (SIGGRAPH '18). Association for Computing Machinery, New York, NY, USA, Article 71, 2 pages. https://doi.org/10.1145/3214745.3214806

[4] Benedikt Bitterli and Wojciech Jarosz. 2019. Selectively metropolised Monte Carlo light transport simulation. ACM Transactions on Graphics (TOG) 38, 6 (2019), 1–10.

[5] Jonathan Brouillat, Christian Bouville, Brad Loos, Charles Hansen, and Kadi Bouatouch. 2009. A Bayesian Monte Carlo approach to global illumination. In Computer Graphics Forum, Vol. 28. Wiley Online Library, USA, 2315–2329.

[6] Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. ACM Trans. Graph. 36, 4, Article 98 (July 2017), 12 pages. https://doi.org/10.1145/3072959.3073601

[7] Siddhartha Chib and Edward Greenberg. 1995. Understanding the metropolis-hastings algorithm. The american statistician 49, 4 (1995), 327–335.

[8] Stavros Diolatzis, Adrien Gruson, Wenzel Jakob, Derek Nowrouzezahrai, and George Drettakis. 2020. Practical Product Path Guiding Using Linearly Transformed Cosines. In Computer Graphics Forum (Proceedings of Eurographics Symposium on Rendering) 39, 4 (July 2020).

[9] William Donnelly and Andrew Lauritzen. 2006. Variance Shadow Maps. In Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games (Redwood City, California) (I3D '06). Association for Computing Machinery, New York, NY, USA, 161–165. https://doi.org/10.1145/1111411.1111440

[10] Charles J Geyer. 1992. Practical markov chain monte carlo. , 473–483 pages.

[11] Zoubin Ghahramani and Carl Rasmussen. 2002. Bayesian Monte Carlo. https://proceedings.neurips.cc/paper/2002/file/24917db15c4e37e421866448c9ab23d8-Paper.pdf

[12] Toshiya Hachisuka, Wojciech Jarosz, Richard Peter Weistroffer, Kevin Dale, Greg Humphreys, Matthias Zwicker, and Henrik Wann Jensen. 2008. Multidimensional Adaptive Sampling and Reconstruction for Ray Tracing. ACM Trans. Graph. 27, 3 (aug 2008), 1–10. https://doi.org/10.1145/1360612.1360632

[13] Jon Hasselgren, J. Munkberg, Marco Salvi, A. Patney, and Aaron Lefohn. 2020. Neural Temporal Adaptive Sampling and Denoising. Computer Graphics Forum 39 (05 2020), 147–155. https://doi.org/10.1111/cgf.13919

[14] Julius Ikkala, Petrus Kivi, Joel Alanko, Markku Mäkitalo, and Pekka Jääskeläinen. 2021. DDISH-GI: Dynamic Distributed Spherical Harmonics Global Illumination. In Advances in Computer Graphics: 38th Computer Graphics International Conference, CGI 2021, Virtual Event, September 6–10, 2021, Proceedings. Springer-Verlag, Berlin, Heidelberg, 433–451. https://doi.org/10.1007/978-3-030-89029-2_34

[15] Michał Iwanicki and Peter-Pike Sloan. 2017. Precomputed lighting in Call of Duty: Infinite Warfare.

[16] Wojciech Jarosz, Craig Donner, Matthias Zwicker, and Henrik Wann Jensen. 2008. Radiance Caching for Participating Media. ACM Trans. Graph. 27, 1, Article 7 (mar 2008), 11 pages. https://doi.org/10.1145/1330511.1330518

[17] Henrik Wann Jensen. 1996. Global illumination using photon maps. , 21–30 pages.

[18] James T. Kajiya. 1986. The Rendering Equation. In Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '86). Association for Computing Machinery, New York, NY, USA, 143–150. https://doi.org/10.1145/15922.15902

[19] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. 2002. A simple and robust mutation strategy for the metropolis light transport algorithm. , 531–540 pages.

[20] Alexander Keller. 1997. Instant Radiosity. In Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97). ACM Press/Addison-Wesley Publishing Co., USA, 49–56. https://doi.org/10.1145/258734.258769

[21] A. Keller, L. Fascione, M. Fajardo, I. Georgiev, P. Christensen, J. Hanika, C. Eisenacher, and G. Nichols. 2015. The Path Tracing Revolution in the Movie Industry. In ACM SIGGRAPH 2015 Courses (Los Angeles, California) (SIGGRAPH '15). Association for Computing Machinery, New York, NY, USA, Article 24, 7 pages. https://doi.org/10.1145/2776880.2792699

[22] Alexander Keller, Timo Viitanen, Colin Barré-Brisebois, Christoph Schied, and Morgan McGuire. 2019. Are We Done with Ray Tracing?. In ACM SIGGRAPH 2019 Courses (Los Angeles, California) (SIGGRAPH '19). Association for Computing Machinery, New York, NY, USA, Article 3, 381 pages. https://doi.org/10.1145/3305366.3329896

[23] Alexander Keller and Carsten Waechter. 2009. Real-time precision ray tracing. https://patents.google.com/patent/US20070024615A1/en US patent US20070024615A1.

[24] Jaroslav Křivánek, Pascal Gautron, Greg Ward, Henrik Wann Jensen, Per H. Christensen, and Eric Tabellion. 2008. Practical Global Illumination with Irradiance Caching. In ACM SIGGRAPH 2008 Classes (Los Angeles, California) (SIGGRAPH '08). Association for Computing Machinery, New York, NY, USA, Article 60, 20 pages. https://doi.org/10.1145/1401132.1401213

[25] Chunyuan Li, Andrew Stevens, Changyou Chen, Yunchen Pu, Zhe Gan, and Lawrence Carin. 2016. Learning Weight Uncertainty with Stochastic Gradient MCMC for Shape Classification. , 5666-5675 pages. https://doi.org/10.1109/CVPR.2016.611

[26] Daqi Lin, Chris Wyman, and Cem Yuksel. 2021. Fast Volume Rendering with Spatiotemporal Reservoir Resampling. ACM Trans. Graph. 40, 6, Article 279 (dec 2021), 18 pages. https://doi.org/10.1145/3478513.3480499

[27] Daqi Lin and Cem Yuksel. 2020. Real-Time Stochastic Lightcuts. Proc. ACM Comput. Graph. Interact. Tech. 3, 1, Article 5 (apr 2020), 18 pages. https://doi.org/10.1145/3384543

[28] Zander Majercik, Jean-Philippe Guertin, Derek Nowrouzezahrai, and Morgan McGuire. 2019. Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields. Journal of Computer Graphics Techniques (JCGT) 8, 2 (5 June 2019), 1–30. http://jcgt.org/published/0008/02/01/

[29] Zander Majercik, Adam Marrs, Josef Spjut, and Morgan McGuire. 2021. Scaling Probe-Based Real-Time Dynamic Global Illumination for Production. Journal of Computer Graphics Techniques (JCGT) 10, 2 (3 May 2021), 1–29. http://jcgt.org/published/0010/02/01/

[30] Ricardo Marques, Christian Bouville, Mickaël Ribardière, Luís Paulo Santos, and Kadi Bouatouch. 2013. A Spherical Gaussian Framework for Bayesian Monte Carlo Rendering of Glossy Surfaces. IEEE Transactions on Visualization and Computer Graphics 19, 10 (2013), 1619–1632. https://doi.org/10.1109/TVCG.2013.79

[31] Morgan McGuire, Mike Mara, Derek Nowrouzezahrai, and David Luebke. 2017. Real-Time Global Illumination Using Precomputed Light Field Probes. In Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (San Francisco, California) (I3D '17). Association for Computing Machinery, New York, NY, USA, Article 2, 11 pages. https://doi.org/10.1145/3023368.3023378

[32] Soham Uday Mehta, Brandon Wang, and Ravi Ramamoorthi. 2012. Axis-Aligned Filtering for Interactive Sampled Soft Shadows. ACM Trans. Graph. 31, 6, Article 163 (Nov. 2012), 10 pages. https://doi.org/10.1145/2366145.2366182

[33] Martin Mittring. 2007. Finding next Gen: CryEngine 2. In ACM SIGGRAPH 2007 Courses (San Diego, California) (SIGGRAPH '07). Association for Computing Machinery, New York, NY, USA, 97–121. https://doi.org/10.1145/1281500.1281671

[34] Thomas Müller, Markus Gross, and Jan Novák. 2017. Practical Path Guiding for Efficient Light-Transport Simulation. Computer Graphics Forum (Proceedings of EGSR) 36, 4 (June 2017), 91–100. https://doi.org/10.1111/cgf.13227

[35] Thomas Müller, Brian Mcwilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural Importance Sampling. ACM Trans. Graph. 38, 5, Article 145 (oct 2019), 19 pages. https://doi.org/10.1145/3341156

[36] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-Time Neural Radiance Caching for Path Tracing. ACM Trans. Graph. 40, 4, Article 36 (jul 2021), 16 pages. https://doi.org/10.1145/3450626.3459812

[37] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-Time Neural Radiance Caching for Path Tracing. ACM Trans. Graph. 40, 4, Article 36 (jul 2021), 16 pages. https://doi.org/10.1145/3450626.3459812

[38] Jacob Munkberg and Jon Hasselgren. 2020. Neural denoising with layer embeddings. In Computer Graphics Forum, Vol. 39. Wiley Online Library, 1–12.

[39] Radford M Neal. 2012. Bayesian learning for neural networks. Vol. 118. Springer Science & Business Media, USA.

[40] Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and J. Pantaleoni. 2021. ReSTIR GI: Path Resampling for Real-Time Path Tracing. Computer Graphics Forum 40, 8 (2021), 17–29. https://doi.org/10.1111/cgf.14378 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14378

[41] Yaobin Ouyang, Shiqiu Liu, Markus Kettunen, Matt Pharr, and Jacopo Pantaleoni. 2021. ReSTIR GI: Path Resampling for Real-Time Path Tracing. In Computer Graphics Forum, Vol. 40. Wiley Online Library, 17–29.

[42] Hauke Rehfeld, Tobias Zirr, and Carsten Dachsbacher. 2014. Clustered Pre-Convolved Radiance Caching. In Proceedings of the 14th Eurographics Symposium on Parallel Graphics and Visualization (Swansea, Wales, United Kingdom) (PGV '14). Eurographics Association, Goslar, DEU, 25–32.

[43] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. 2009. Approximating Dynamic Global Illumination in Image Space. In Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (Boston, Massachusetts) (I3D '09). Association for Computing Machinery, New York, NY, USA, 75–82. https://doi.org/10.1145/1507149.1507161

[44] Lukas Ruppert, Sebastian Herholz, and Hendrik P. A. Lensch. 2020. Robust Fitting of Parallax-Aware Mixtures for Path Guiding. ACM Trans. Graph. 39, 4, Article 147 (jul 2020), 15 pages. https://doi.org/10.1145/3386569.3392421

[45] Daniel Scherzer, Chuong Nguyen, Tobias Ritschel, and Hans-Peter Seidel. 2012. Pre-convolved Radiance Caching. Computer Graphics Forum (2012). https://doi.org/10.1111/j.1467-8659.2012.03134.x

[46] Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R. Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination. In Proceedings of High Performance Graphics (Los Angeles, California) (HPG '17). Association for Computing Machinery, New York, NY, USA, Article 2, 12 pages. https://doi.org/10.1145/3105762.3105770

[47] Christoph Schied, Christoph Peters, and Carsten Dachsbacher. 2018. Gradient estimation for real-time adaptive temporal filtering. Proceedings of the ACM on Computer Graphics and Interactive Techniques 1, 2 (2018), 1–16.

[48] Pradeep Sen and Soheil Darabi. 2012. On filtering the noise from the random parameters in Monte Carlo rendering. ACM Trans. Graph. 31, 3 (2012), 18–1.

[49] Ari Silvennoinen and Jaakko Lehtinen. 2017. Real-Time Global Illumination by Precomputed Local Reconstruction from Sparse Radiance Probes. ACM Trans. Graph. 36, 6, Article 230 (nov 2017), 13 pages. https://doi.org/10.1145/

3130800.3130852
[50] Ari Silvennoinen and Ville Timonen. 2015. Multi-scale global illumination in quantum break.
[51] Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. In Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (San Antonio, Texas) (SIGGRAPH '02). Association for Computing Machinery, New York, NY, USA, 527–536. https://doi.org/10.1145/566570.566612
[52] Tiago Sousa, Nick Kasyan, and Nicolas Schulz. 2011. Secrets of CryENGINE 3 graphics technology.
[53] Michael Stengel, Zander Majercik, Benjamin Boudaoud, and Morgan McGuire. 2021. A Distributed, Decoupled System for Losslessly Streaming Dynamic Light Probes to Thin Clients. In Proceedings of the 12th ACM Multimedia Systems Conference (Istanbul, Turkey) (MMSys '21). Association for Computing Machinery, New York, NY, USA, 159–172. https://doi.org/10.1145/3458305.3463379
[54] Eric Tabellion and Arnauld Lamorlette. 2004. An Approximate Global Illumination System for Computer Generated Films. In ACM SIGGRAPH 2004 Papers (Los Angeles, California) (SIGGRAPH '04). Association for Computing Machinery, New York, NY, USA, 469–476. https://doi.org/10.1145/1186562.1015748
[55] Natalya Tatarchuk. 2005. Irradiance volumes for games. , 0 pages.
[56] Kostas Vardis, Georgios Papaioannou, and Anastasios Gkaravelis. 2014. Real-time Radiance Caching using Chrominance Compression. Journal of Computer Graphics Techniques (JCGT) 3, 4 (16 December 2014), 111–131. http://jcgt.org/published/0003/04/06/
[57] Eric Veach and Leonidas J. Guibas. 1995. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95). Association for Computing Machinery, New York, NY, USA, 419–428. https://doi.org/10.1145/218380.218498
[58] Eric Veach and Leonidas J. Guibas. 1997. Metropolis Light Transport. In Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97). ACM Press/Addison-Wesley Publishing Co., USA, 65–76. https://doi.org/10.1145/258734.258775
[59] Petr Vévoda, Ivo Kondapaneni, and Jaroslav Křivánek. 2018. Bayesian online regression for adaptive direct illumination sampling. ACM Transactions on Graphics (TOG) 37, 4 (2018), 1–12.
[60] Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. 2014. On-Line Learning of Parametric Mixture Models for Light Transport Simulation. ACM Trans. Graph. 33, 4, Article 101 (jul 2014), 11 pages. https://doi.org/10.1145/2601097.2601203
[61] Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. 2014. On-Line Learning of Parametric Mixture Models for Light Transport Simulation. ACM Trans. Graph. 33, 4, Article 101 (jul 2014), 11 pages. https://doi.org/10.1145/2601097.2601203
[62] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. 2005. Lightcuts: A Scalable Approach to Illumination. ACM Trans. Graph. 24, 3 (jul 2005), 1098–1107. https://doi.org/10.1145/1073204.1073318
[63] Yue Wang, Soufiane Khiat, Paul G. Kry, and Derek Nowrouzezahrai. 2019. Fast Non-Uniform Radiance Probe Placement and Tracing. In Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (Montreal, Quebec, Canada) (I3D '19). Association for Computing Machinery, New York, NY, USA, Article 5, 9 pages. https://doi.org/10.1145/3306131.3317024
[64] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. 1988. A Ray Tracing Solution for Diffuse Interreflection. SIGGRAPH Comput. Graph. 22, 4 (jun 1988), 85–92. https://doi.org/10.1145/378456.378490
[65] Max Welling and Yee Whye Teh. 2011. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In Proceedings of the 28th International Conference on International Conference on Machine Learning (Bellevue, Washington, USA) (ICML'11). Omnipress, Madison, WI, USA, 681–688.
[66] Lei Xiao, Salah Nouri, Matt Chapman, Alexander Fix, Douglas Lanman, and Anton Kaplanyan. 2020. Neural Supersampling for Real-Time Rendering. ACM Trans. Graph. 39, 4, Article 142 (July 2020), 12 pages. https://doi.org/10.1145/3386569.3392376

A METROPOLIS-HASTINGS

Markov Chain Monte Carlo (MCMC) allows sampling from the posterior without computing the marginal. [10]. Metropolis-Hastings (Metropolis), which we exploit in this work, is a specific implementation of MCMC [7]. The Metropolis–Hastings algorithm can draw samples from any probability distribution with probability density $P(x)$, provided a function $h(x)$ proportional to the density $P(x)$. The Metropolis algorithm works by generating a sequence of sample values so that, as more samples are produced, the distribution of samples more closely approximates the

desired distribution. These sample values are produced iteratively, meaning the next sample being dependent on the current sample (thus making the sequence of samples into a chain). Let $h(x)$ be a function that is proportional to the desired probability density function $P(x)$ (a.k.a. a target distribution). The Metropolis Markov Chain algorithm with random walk is defined as follows:

Algorithm 5: Random-walk algorithm

Input: x^i: Current state, y^i : Probability of current state
Input: σ : Step size or std-dev of Gaussian noise
Output: x^{i+1}: Next state, y^{i+1} : Probability of next state

function RandomWalk(x^i, y^i)**:**
  $x^{i+1} \leftarrow x^i + \mathcal{N}(\sigma)$ // Propose a new state
  $y^{i+1} \leftarrow h(x^{i+1})$
  $\mu \leftarrow \min\left\{\frac{y^{i+1}}{y^i}, 1\right\}$ // Compute acceptance ratio
  $\epsilon \sim U(0,1)$ // Sample uniform distribution
  if $\epsilon > \mu$ **then**
    /* Reject proposed state */
    $x^{i+1} \leftarrow x^i$
    $y^{i+1} \leftarrow y^i$
  return x^{i+1}, y^{i+1}

Initialization: Choose an arbitrary point x^{i-1} as the initial observation in the sample-space and choose an arbitrary probability density $\mathcal{N}(x^i \mid x^{i-1})$ that suggests the next sample candidate x^i, given the previous sample value x^{i-1}. In our work, $\mathcal{N}$ is assumed to be symmetric. A usual choice is to let $\mathcal{N}(x^i \mid x^{i-1})$ be a Gaussian distribution centered at x^{i-1}, so that points closer to x^{i-1} are more likely to be visited next, making the sequence of samples resemble a *random walk* [7]. The random walk algorithm is described in algorithm 5.

B PROBE COMPRESSION

We tested several 26-bit encoding and settled on a non-linear RGB encoding represented by ⌊R9⌋⌊G9⌋⌊B8⌋ – N in figure 11. In this encoding, the RGB color is first passed through a logarithmic non-linearity as per equation 21 such that the quantization errors are distributed evenly across intensities. We perform a *round-to-lowest-integer* (⌊⌋) quantization for all channels, although *round-to-nearest-integer* ([ ]) is more accurate. Our quantization scheme ensures the moving-average updates produce dark colors when the intensity of new samples are low. In a *round-to-nearest* setting, due to a *round-up* error, the colors may never go to zero. Interestingly, YCbCr encoding allows *round-to-lowest* for the Y channel and round *round-to-nearest* for Cb and Cr channels, however, they perform poorly in both luminance and color preservation metrics as shown in figure 11.

The parameters in equation 21 are obtained by performing a grid search minimizing the reconstruction error w.r.t RGB32f reference across various color and intensity combinations as shown in figure 11. Luminance error is the r.m.s. value of the difference between the two color-maps. Color accuracy is measured using a normalized dot product between the two flattened color-maps.

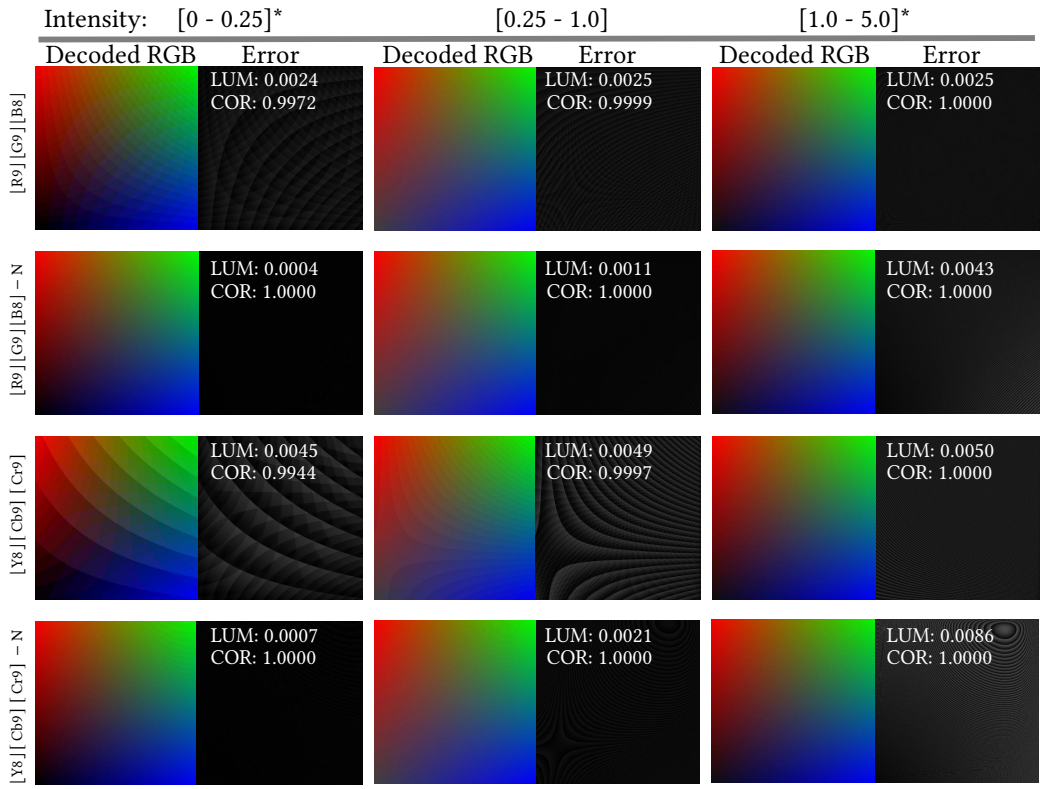


Fig. 11. Figure comparing 26-bit color encodings on slices of the 3D color-space with dynamic range. We compare the reconstruction error measured in Luminance and Color Correlation with RGB32f reference. The log-non-linear encodings marked with - N suffix shifts the bit error from lower to higher intensities - which are less frequent in indirect illumination. ⌊⌋ and [ ] denotes round-low and round-nearest quantizations respectively. * Color map visualizations are normalized.