Scaling Probe-Based Real-Time Dynamic Global Illumination for Production

Zander Majercik   Adam Marrs   Josef Spjut   Morgan McGuire

NVIDIA

Figure 1. Image rendered in a pre-release version of Unity with our global illumination technique. Most of the indirect lighting in this scene comes from emissives (the orange monitor screens) which are integrated automatically by our technique.

Abstract

We contribute several practical extensions to the probe-based irradiance-field-with-visibility representation [Majercik et al. 2019] [McGuire et al. 2017] to improve image quality, constant and asymptotic performance, memory efficiency, and artist control. We developed these extensions in the process of incorporating the previous work into the global illumination solutions of the NVIDIA RTXGI SDK, the Unity and Unreal Engine 4 game engines, and proprietary engines for several commercial games. These extensions include: an intuitive tuning parameter (the "self-shadow" bias); heuristics to speed transitions in the global illumination; reuse of irradiance data as prefiltered *radiance* for recursive glossy reflection; a probe state machine to prune work that will not affect the final image; and multiresolution cascaded volumes for large worlds.

1. Introduction

This paper discusses an algorithm to accelerate the evaluation of global illumination. The acceleration happens in two parts. The main part creates and maintains a data structure that allows a query of the form *irradiance(location, orientation)* $(E(X, \omega))$, which replaces a potentially expensive computation of diffuse global illumination with a O(1) lookup into a data structure for locations anywhere in space. The second part reuses that data structure to sample weighted average of incident radiance for glossy global illumination $(\int_{\Gamma} L(X, \omega) \cdot W(X, \omega) d\omega)$ and combines the result with filtered screen-space and geometric glossy ray tracing.

We refine a previous version of the diffuse portion of this method [Majercik et al. 2019]. This refinement is the union of what we learned when incorporating that algorithm into several products, including the Unity game engine, the Unreal Engine 4 game engine, the NVIDIA RTXGI SDK, and several unannounced commercial games. These refinements include changes to the underlying algorithm to improve quality and performance, advice on tuning the algorithm and content, expansion of the algorithm to a complete solution that also accelerates glossy reflections, and system integration best practices for these methods. This was driven by constraints from various platforms, requests from game developers and game artists, and new research on the problem. Because these improvements were developed across several different productization efforts with different vendors, we believe that they are fairly universal

| Term	Definition
Probe	A probe stores data at a point with values for directions on the sphere.
Probe Query	Trilinear interpolation (bilinear filtering and direction) and a visibility and angle weighted interpolation between multiple probes. The net result is an irradiance value that estimates the irradiance field at a point relative to a normal.
Irradiance	Incident power per unit area; the cosine-weighted integral of radiance over the hemisphere of the sample direction.
Weighted sum of distance	Weighted sum (a weighted average in our implementation) of the distance to the nearest surface seen from a 3D point in a particular direction. We weight our average with a cosine raised to a power.
Direct lighting	Light that is emitted from a light source, reflects from one surface, and then reaches the viewer.
Indirect lighting	Light that reflects off two or more surfaces before reaching the viewer (all lighting that is not direct)
Global illumination (GI)	Light that includes both direct and indirect lighting.

Table 1. Terms and definitions.

and robust, but they should not be construed as describing the features or performance of any one in particular.

A key element of our algorithm is a probe that stores directional information at a point. Environment maps are a type of probe—they store distant radiance as seen from any point in the scene. Our probes store irradiance, weighted averages of distance, and weighted averages of squared distance for a 3D grid-like structure of points in the scene. See Table 1 for terms we use in relation to probes.

Our algorithm has several components related to the organization, computation, and querying of probes. The new information described in this paper is indicated in Table 2, where we indicate what is new relative to descriptions of previous versions of this algorithm. In the following sections, we give a complete description of the full algorithm so that readers will not need to consult descriptions of previous versions to understand the algorithm.

2.   Overview of the Algorithm

At the core of the algorithm are probes that store weighted sums of color, distance, and squared distance. A 2D version of a probe storing a weighted average of distance to nearest object is shown in Figure 2. This probe shows one "cell" (texel) as the bold segment of the circle. The bold arrow is the direction associated with the cell. The cell stores the weighted average of the hit distances of each of the sample directions. Note that this weighted average includes directions "outside" the center cell. The weighting function is larger for directions near the cell center, and the resulting weighted average is thus influenced more by the longer directions in this particular example. The bold dotted line represents the average "distance" stored across multiple cells. Note that a direction can contribute to more than one cell, and we loop over directions updating any cell to which a direction contributes.
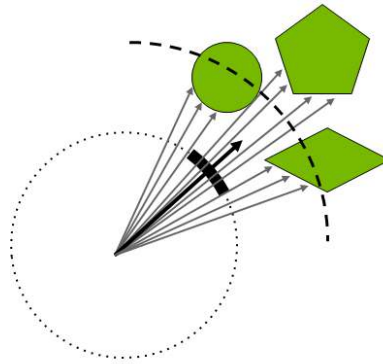
Figure 2. A 2D probe for illustration.

2.1.   Build and Initialization

Start by building a 3D grid. From that grid, optimize probe positions by moving them outside of static geometry (Section 5). Then, classify all probes into "Off," "Sleeping," "Newly Awake," "Newly Vigilant," "Awake," or "Vigilant" (Section 6). At the end of this stage, all probes are in their final positions and initial states.

2.2.   Probe Query

Take a 3D point (within the probe volume) and normal direction. For every point within the volume, there are eight probes (corners of a 3D box) that surround it. Loop over those eight probes. For each one, compute a probe weight from a combination of

- the trilinear weight from probe position;

- the backface weight (is the probe behind the point relative to the normal?);

- the visibility evaluation (can the probe see the point?). This includes a "self-shadow bias" term for robust occlusion queries (Section 4.1).

Read the value from each probe in the direction of the normal and sum those using the computed weights—this is the sampled irradiance value.

For multiple volumes, do this for each volume, and then weight between the volumes as described in Section 7.4. Volume blending with tracking windows is discussed in Section 7.3.

2.3.   Probe Update

For each probe that is "Awake" or "Vigilant" (Section 6), trace rays in a spherical Fibonacci pattern, [Marques et al. 2013] rotating the pattern randomly every frame. Shade these ray hits using the normal deferred-shading algorithm, including sampling the probe volume to include the irradiance from the probes. The update then proceeds for both irradiance and mean distance values as follows.

Irradiance.   Compute a cosine-weighted average of the radiance values of these shaded ray hits relative to the direction of each probe texel. Then, for each probe texel, blend these newly computed values into the probe texel at a rate of $(1 - \alpha)$—we refer to this alpha term as *hysteresis*. We adjust this hysteresis per probe and per texel based on our convergence heuristics, described in Section 4.3.

Mean Distance and Mean Distance-Squared.   Compute a *power*-cosine weighted average of the distance values for each ray relative to the direction of each probe texel. For each probe texel, blend these values as with irradiance above. We adjust the

hysteresis for mean distance separately from irradiance—details are provided in Section 4.3.

As with the irradiance update, we update the probe texels by alpha blending in the new shading results at a rate of $1 - \alpha$, where α is the hysteresis parameter (see Equation (2)). The update equation for irradiance is as follows:

$$E'[n] = \alpha E[n] + (1 - \alpha) \sum_{\text{ProbeRays}} \max(0, n \cdot \omega) \cdot L(\omega), \tag{1}$$

where E is the old irradiance/visibility texel in direction n, E' is the new texel value, ω is the direction of the ray, and $L(\omega)$ is the radiance transported along the ray. The update equation for mean distance/distance-squared weights with a power-cosine as opposed to the clamped cosine show in Equation (1) but is otherwise the same.

3.   Related Work

Interactive global illumination has been an active area of research for years. We review the areas most relevant to our work.

Interactive Global Illumination with Light Probes.   Image-based lighting solutions are ubiquitous in modern video games [Martin and Einarsson 2010; Ritschel et al. 2009; McAuley 2012; Hooker 2016]. A common workflow for such solutions involves placing light probes densely inside the volume of a scene, each of which encodes some form of a spherical (ir)radiance map. Prefiltered versions of these maps can also be stored to accelerate diffuse and glossy runtime shading queries.

Variants of traditional light probes allow artists to manually place box or sphere proxies in a scene. These proxies are used to warp probe queries at runtime in a manner that better approximates spatially-localized reflection variations [Lagarde and Zanuttini 2012]. Similarly, manually-placed convex proxy geometry sets are also used to bound blending weights when querying and interpolating between many light probes at runtime in order to reduce the light leaking artifacts common to probe-based methods.

Practitioners agree that eliminating manual probe and proxy placement remains an important open problem in production [Hooker 2016]. Without manual adjustment of traditional probes, it is impossible to automatically avoid probe placements that lead to light and dark (i.e., shadow) leaks or displaced reflection artifacts. Majercik et al.'s [2019] light probes avoid light and dark leaking with ray-traced visibility information, but placing these probes in a uniform grid still leads to suboptimal probe locations (e.g., probes stuck in walls). To avoid these issues for glossy GI, some engines rely instead on screen-space ray tracing [Valient 2013] for pixel-accurate reflections. These methods, however, fail when a reflected object is not visible from the camera's point of view, leading to inconsistent lighting and view-dependent (and so temporally unstable) reflection effects.

	Previous Approaches [Martin and Einarsson 2010] [Ritschel et al. 2009] [McAuley 2012] [Hooker 2016] [Stefanov 2016]	Light-field probes [McGuire et al. 2017] [Wang et al. 2019]	DDGI [Majercik et al. 2019]	This work
Spatial Organization	3D grid, with manually placed probes and box proxies, algorithmically precomputed probe locations	3D grid, $4 \times 4 \times 4$ probes [2017] Non-uniform automatic placement over static geometry [2019]	3D grid, varying resolutions	3D grid with offsets, multiple volumes, tracking windows
Encoding	Cube maps	Octahedral encoding [Cigolle et al. 2014], 1024×1024	Octahedral, varying resolutions	Octahedral, 8×8 irradiance, 16×16 visibility
Initialization	Precomputed	Static, precomputed	Uniform initialization to 0, value converges with update	Classified into states based on update rate, converge "live" probes
Update	Static. Dynamic lighting, static geo [Stefanov 2016]	Static, precomputed	Ray trace with alpha blending, pixel shader with stencil buffer	Ray trace with dynamic alpha blending (convergence and perception), Optimized convolution compute shader
Query	Shading weights based on manually placed proxy geometry	Light field ray tracing using probes	Raster for direct lighting. Use world-space positions to query 8 probe cage with variance bias, Chebyshev bias, loads of bias terms	Previous probe sampling + single bias term (self-shadow bias), multivolume blending, primary hit glossy raycast + second-order glossy reflection sampled from probes

Table 2. Evolution of probe-based GI showing spatial organization, encoding, initialization, update, and query for the GI computation.

Light Field Probes [McGuire et al. 2017] automatically resolve many light/dark leaking issues (in scenes with static geometry and lighting) by encoding additional information about the scene geometry into spherical probes. A solution for dynamic lighting is presented in Silvennoinen et al. [2017], but this solution only supports coarse dynamic occluders and requires complex probe placement based on static geometry. As mentioned above, the irradiance probes of Majercik et al. [2019] avoid most light/dark leaks in scenes with dynamic lighting and geometry, but probe placement is still suboptimal. Suboptimal placement can lead to lighting results that, while believable, are inferior to the correctly sampled result, and they sometimes exhibit shadow leaking in sharp corners.

Interactive Ray Tracing and Shading. Correct shading with probe-based lighting methods relies on point-to-point visibility queries. At a high level, one can interpret our ray-tracing technique as tracing rays against a voxelized representation of the scene (as in voxel cone tracing), but with a spherical voxelization instead of an octree. Two important differences that contribute to many of the practical advantages of our representation are that (1) we *explicitly* encode geometric scene information (i.e., radial depth and depth squared) instead of relying on the *implicit* octree structure to resolve local and global visibility details, and (2) that neither our spatial parameterization nor our filtering relies on scene geometry. This prevents light (and dark) leaking artifacts and allows us to resolve *centimeter-scale* geometry at about the same cost (in space and time) as a voxel cone tracer that operates at *meter-scale*. As we target *true world-space* ray-tracing in a pixel shader, and not just screen-space ray tracing, our technique can be seen as a generalization of many previous, e.g., real-time environment map Monte Carlo integration methods[Stachowiak and Uludag 2015; Wyman 2005; Toth et al. 2015; Jendersie et al. 2016] .

Probe Representation. As in the work by Majercik et. al [2019], we apply Cigolle et al.'s [2014] octahedral mapping from the sphere to the unit square to store and query our spherical distributions. This parameterization has slightly less distortion than cube maps and provides easier methods for managing seams. In this work, we select resolutions for octahedral irradiance and mean distance/distance squared for quality and performance.

GI in Production: A Motivating Example. In both offline and real-time rendering, significant previous work has been devoted to adapting existing global illumination algorithms for production. Path tracing in the film industry, which radically changed both artist workflow and render-farm computation load, is a good example. The core path-tracing algorithm has remained largely unchanged, but practical considerations of the particular hardware and software systems required specialized updates to the technique [Keller et al. 2015].

Similarly, our extensions to the previously published DDGI algorithm are a guide for adapting it and other probe-based techniques to a production setting. We report real changes that we made to the base algorithm to fit production constraints.

4. Qualitative Image Improvements

4.1. Self-shadow Bias for Correct Visibility

When querying the probe volume at a surface, variance in the visibility estimate will be highest around the mean of the distribution—in other words, *at* the surface. To avoid the shadow leaking that results from this, an additional bias away from the mean of the distribution is added to the sample point during probe query. The previous technique [2019] used a combination of scene-tuned biases on the mean of the distribution, the variance of the distribution, and the Chebyshev statistical test to move the visibility query to a point of lower variance in the distribution. Intuitively, "a point of lower variance in the distribution" can be thought of as a point slightly offset from the surface (in world space). Based on this insight, we unify these statistical bias parameters into a single *self-shadow* bias term. The self-shadow bias is a world-space vector pointing away from the initial sample point on the surface and is computed as follows:

$$\mathrm{BiasVector} = (\mathbf{n} * 0.2 + \omega_\mathrm{o} * 0.8)$$
$$* \, (0.75 * \mathrm{minDistanceBetweenProbes}) * \mathrm{TunableShadowBias}, \tag{2}$$

where $\mathbf{n}$ is the normal vector at the sample point, ω_o is the direction from the sample point to the camera, 0.2 and 0.8 are empirically determined constants, $\mathrm{minDistance}$ $\mathrm{BetweenProbes}$ is the minimum axial distance between probes, and $\mathrm{TunableShadow}$ Bias is a user-tunable parameter (with a default value of 0.3). We add this bias vector to the initial sample point to yield a new point which we use for the visibility test.

Our self-shadow bias is usually more robust than the previous biases: the default value of the $\mathrm{TunableShadowBias}$ worked well for most scenes, whereas the previous combination of biases each had to be specifically tuned for each scene. In cases where scene-specific tuning is necessary, the self-shadow bias is easier to tune because it presents a single tunable parameter instead of three. Generally, a higher self-shadow bias is necessary when there is increased variance in the depth estimate, as would be the case when lower ray counts are used to update the probes (which might be done to improve performance).

To further decrease light leaking, probe-update rays that hit backfaces record a value of 0 for irradiance and shorten their depth values by 80%. Shortening depth values ensures that the probe will see backface surfaces as shadowed and not light them. We set irradiance to 0 to ensure that any lighting that does come from that probe does not cause light to leak where it should not. We do not set depth values

Figure 3. A night scene from our prototype. The wall entering the alley in the left image shows light leaking due to overly high self-shadow bias. The correct self-shadow bias in the right image computes proper occlusion.

to 0 for two reasons: (1) it would drive the computed Chebyshev weight towards 0, which might be driven higher when the weights are normalized and (2) probes that see some backfaces but are not stuck in walls (due to modeling idiosyncrasies) could have overly skewed average depths. To minimize the number of probes stuck in walls as much as possible, we offset probe positions using an iterative adjustment algorithm, as described in Section 5.

4.2. Perception-based Exponential Encoding

If the irradiance probes are slow to converge, abrupt lighting changes in a scene can create noticeable lag in the diffuse indirect illumination. The lag is most salient in light-to-dark transitions. To combat this, we accelerate convergence by applying a perception-based exponential gamma encoding to probe irradiance values. This encoding interpolates *perceptually linearly* during lighting changes—faster light-to-dark convergence reads perceptually as a linear drop in brightness. We determined experimentally that an exponent of 5.0 leads to best results (lower does not converge as fast, higher does not converge any faster). See Listings 1 and 2 for pseudocode as well as our video supplement available at xxxxxxx for results.

```
float irradianceGamma = 5.0;

// Perception encoding during probe update

// Passed in or computed earlier in the shader
in vec3 sumOfCosineWeightedRayContributions;
in vec3 oldValue;
in float hysteresis;

vec3 newIrradiance =
    pow(sumOfCosineWeightedRayContributions, 1.0 / irradianceGamma);

return lerp(newIrradiance, oldValue, hysteresis);
```

Listing 1. Perceptual *encoding* of probe irradiance during update and sampling.

9

```
float irradianceGamma = 5.0;
in float probeWeight;

// Perception decoding during probe sampling
vec3  irradiance = vec3(0);
// For the 8 probes in the surrounding cage
for (int i = 0; i < 8; ++i):
    vec3 probeIrradiance = texture(irradianceTexture, texCoord).rgb;
    // Decode the tone curve, but leave a gamma = 2 curve
    // to approximate sRGB blending for the trilinear
    probeIrradiance = pow(probeIrradiance,
    vec3(irradianceGamma * 0.5));

    irradiance += probeWeight * probeIrradiance;

// Go back to linear irradiance
irradiance = square(irradiance);

return irradiance;
```

Listing 2. Perceptual *decoding* of probe irradiance during update and sampling.

This perception-based encoding has the additional effect of reducing low frequency flicker due to fireflies—bright flashes in the diffuse GI caused by an update ray hitting a small, bright irradiance source.

4.3. Fast Convergence Heuristics

We further accelerate convergence with a new heuristic based on per-texel thresholding for irradiance data. Our lower threshold detects changes with magnitude above 25% of maximum value and lowers the hysteresis by 0.15. Our higher threshold detects changes with magnitude above 80% and lowers the hysteresis to 0.0—we assume in this case that the distribution that the probe is sampling has changed completely. These thresholds are active only for irradiance updates—we found them to be too unstable when updating visibility.

We also implement scene-dependent, per-probe heuristics that adjust the hysteresis based on lighting or geometry changes (see Listing 3). These are as follows:

- Small lighting change (e.g., player-held flashlight turns on): reduce irradiance hysteresis by 15% for four frames;

- Large lighting change (e.g., abrupt time of day shift): reduce irradiance hysteresis by 50% for 10 frames;

- Large object change (e.g., ceiling caves in): reduce irradiance hysteresis by 50% for 10 frames and visibility hysteresis by 50% for seven frames.

```
// Irradiance Probe Update With Per-Texel Hysteresis Adjustment

// Sum ray contributions
in vec3 sumOfCosineWeightedRayContributions;
in vec3 oldValue;

in float hysteresis;

float significantChangeThreshold = 0.25;
float newDistributionChangeThreshold = 0.8;


float changeMagnitude =
maxComponent(sumOfCosineWeightedRayContributions.rgb - oldValue.xyz);

// Lower the hysteresis when a large change is detected
if (abs(changeMagnitude) > significantChangeThreshold)
    hysteresis = max(0, hysteresis - 0.15);

if (abs(changeMagnitude) > newDistributionChangeThreshold)  {
    hysteresis = 0.0;
}

return lerp(sumOfCosineWeightedRayContributions, oldValue, h);
```

Listing 3. Pseudocode for probe update with per-texel hysteresis adjustment.

In all our heuristics, we try to avoid low hysteresis for visibility updates as much as possible to achieve the most stable result. In each of the scene-dependent heuristics, hysteresis for *all* probes (not just the probes local to the change) is reduced.

Many effective heuristics exist for adjusting probe hysteresis per-texel and per-probe on a scene-dependent basis—we have not explored this space in depth. For example, it would probably be more effective to reduce hysteresis only for probes affected by a lighting or object change rather than for all probes in the scene. While exploring more specific and sensitive heuristics remains a fruitful subject for future work, the heuristics presented here worked well enough for us as we integrated the technique into multiple engines. We never came across content that forced us to adapt them, but our survey was not exhaustive.

Note that temporal anti-aliasing (TAA) applies its own hysteresis, so the base hysteresis for our technique can be lower if TAA is applied. In this case, the TAA hysteresis should be adjusted according to scene heuristics just like the probe hysteresis, or else it will always add a large cost to convergence even on a dramatic lighting or object change.

4.4.   Second-order Glossy

We compute glossy reflections with a half-screen resolution wavefront ray trace. These shaded ray hits are then blurred according to surface roughness and distance from the camera before being integrated into the indirect radiance computation during the deferred shading pass. These ray-traced reflections are more realistic than screen-space reflections, but tracing rays for second through nth-order reflections is infeasible on most scenes. We improve reflections by reusing the filtered radiance data in the probes to shade second through nth-order glossy reflections, resulting in better image quality with minimal performance overhead (see Figure 4).
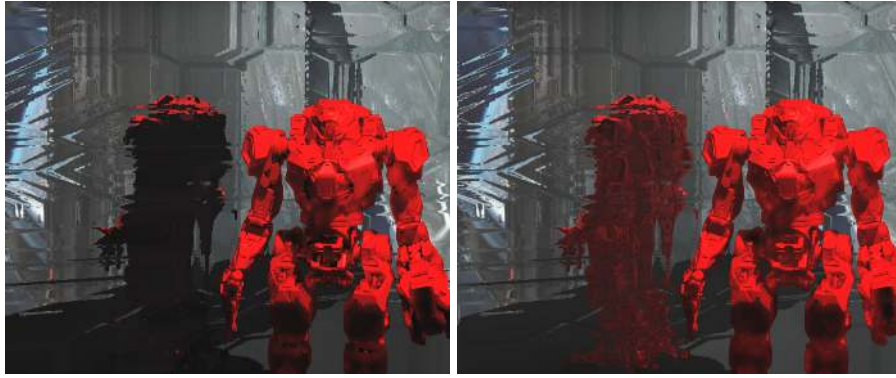
Figure 4. A shiny robot against a mirror background. Both the mirror background and the robot have high glossy reflectance. The left image shows no second-order glossy reflections, while the right image shows second-order glossy reflections sampled from probes.

It is common practice in production path-tracing to reduce noise by roughening surfaces (or otherwise truncating the BSDF evaluation) on recursive bounces [Fascione et al. 2019]. Reusing the irradiance probes for second-order reflections is a similar approximation, which here avoids noise by taking advantage of a data structure already available to us. Note, however, that the probe data structure stores the cosine-weighted *integral* of radiance over the hemisphere—not cosine-weighted radiance which is the correct measure for reflectance. These two quantities are equivalent to a factor of 2π, but the units are different: irradiance (Wm^{-2}) vs. radiance ($\mathrm{Ws}^{-1}\mathrm{m}^{-2}$).

5.   Probe-position Adjustment

The probe visibility information prevents light and shadow leaks from occluded probes, but leaves some probes in total occlusion such that they never contribute to shading. We present a simple, fast optimizer that iteratively shifts probes around static geometry to maximize the number of useful probes and generate good viewpoints. During initialization, our optimizer adjusts each probe through the closest backface it can see,
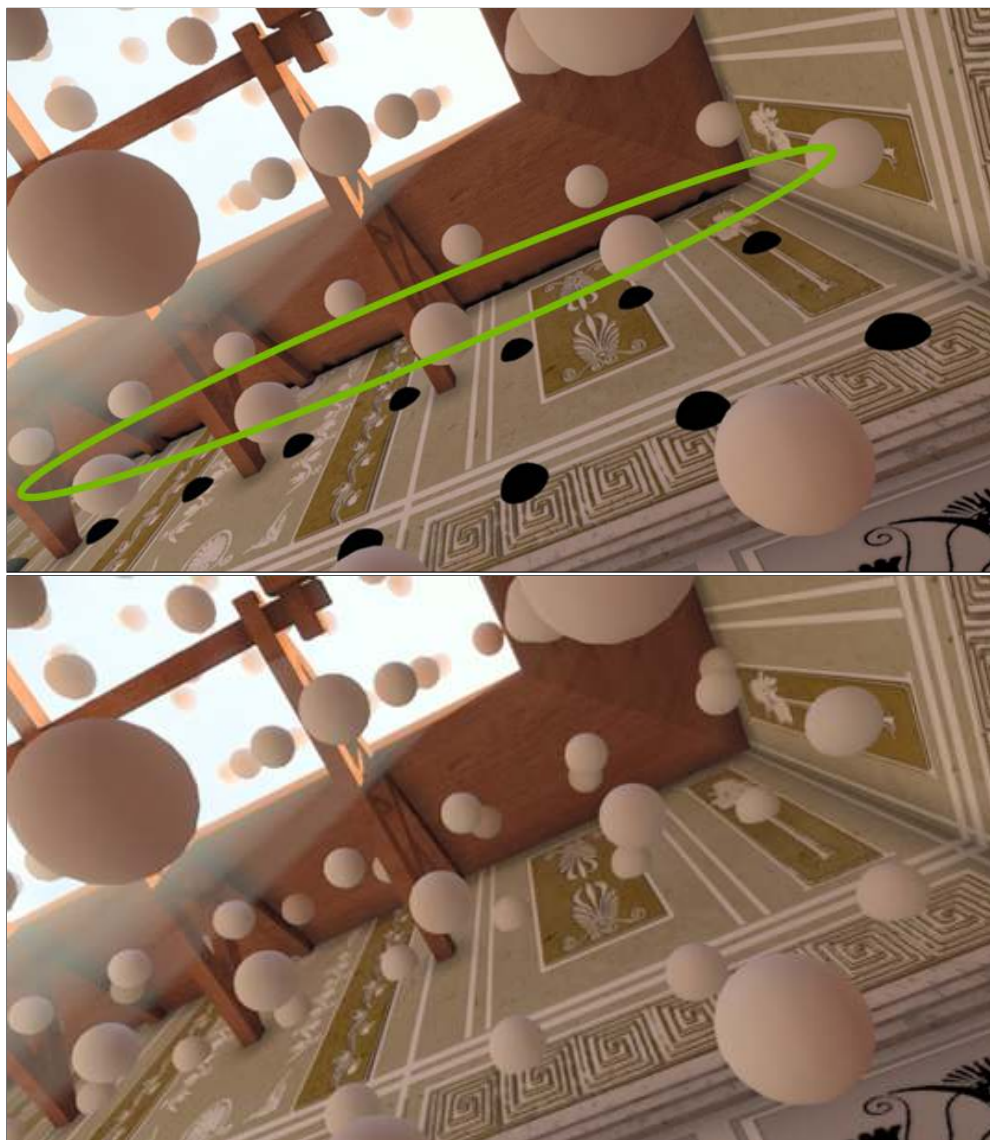
Figure 5. A view of the ceiling on our Greek Villa scene. Spheres are a visualization of the irradiance probes. The black probes are correctly dark, but are not contributing to the final image. The acute corner leads to shadow leaking (labeled with a green ellipse) with a default probe grid (top). Our optimizer adjusts probes out of the wall and ceiling to remove the leak (bottom).

then further adjusts probes *away* from close front faces to maximize surface visibility (see Figure 5). We do not move probes around dynamic geometry because this causes instability—a stable result is preferable to an unstable result with lower average error.

To correctly light dynamic objects, we leverage the fact that a *uniformly* sampled probe is an approximation of the full irradiance field at its sample location. If

a probe passes through a dynamic object, our backface heuristics (described at the end of Section 4.1) will prevent shadow leaking. When the probe emerges, our convergence heuristics (Section 4.3) will quickly converge its value. Out of a desire to maintain a uniformly sampled irradiance-field representation, we also did not implement more complex probe-sampling techniques, such as importance sampling, which might speed probe convergence at the cost of stability and on-the-fly generalization to moving geometry. Exploring these update techniques in detail is promising for future work.

The purpose of the optimizer is to increase the number of probes that can contribute to the final image. The following scenario, however, demonstrates that our optimizer can sometimes add additional computation without increasing image quality. Consider an eight-probe cage surrounding a flat wall. The optimizer can cause probes to "double cover" a surface if the four probes within the surface are adjusted outside it. This causes the full probe cage to turn on and shade the surface, increasing the number of actively tracing probes without appreciably affecting the image quality (Figure 6). For our test scenes, this slight inefficiency was worth the added benefit of optimizing probe positions globally.

Listing 4 shows code for our probe-position optimizer. To preserve the indexing properties of the 3D grid, probes never move more than one half the minimum probe spacing (relative to a grid axis) during optimization.
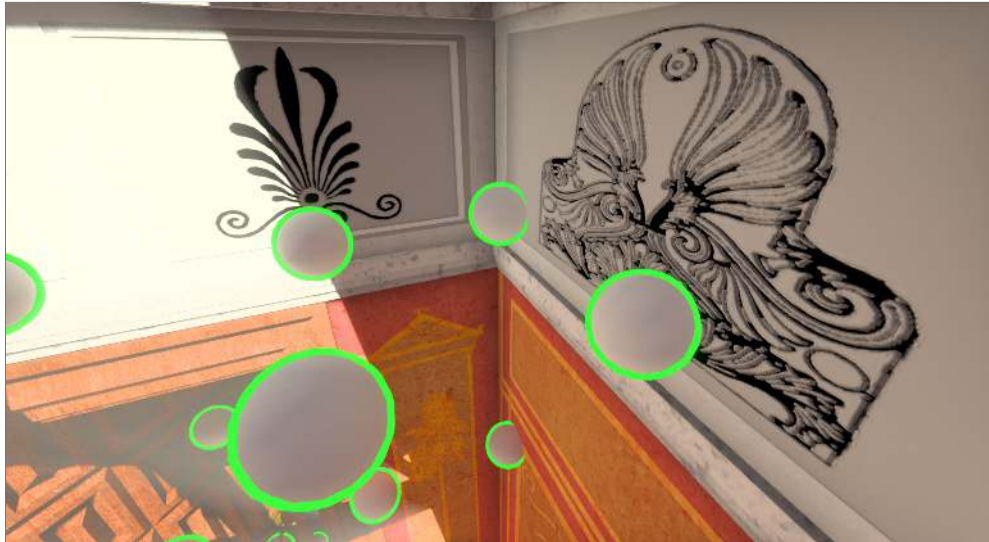
Figure 6. A corner of the Greek Villa scene. Spheres are visualizations of the probes, encircled in green to denote the "Vigilant" state. Probes are marked "Vigilant" when the optimizer adjusts them out of surfaces, leading to double coverage of surfaces when all eight probes of a cage can see the front face of the point being shaded.

```
// Single iteration of probe position optimizer code

in float backfaceCount;            // number of rays that hit backfaces
in vec3 closestBackfaceVector;     // direction to closest backface scaled by distance
in vec3 farthestFrontfaceVector;   // direction to farthest frontface scaled by distance
in vec3 closestFrontfaceVector;    // direction to closest frontface scaled by distance

in vec3 offsetlimit; // A maximum offset computed from the probe grid spacing

// Current probe offset from the grid
inout vec3 currentOffset;

Vector3 fullOffset = vec3(inf);

// If there's a close backface AND you see more than 25% backfaces, assume you're inside something.
if (backfaceCount / RAYS_PER_PROBE) > 0.25) {
    float scaleFactor = 2.0;
    // Slowly ramp down the scale factor to move the maximum allowed distance through the closest backface.
    // Give explicit bounds so the compiler can unroll the loop.
    for (int i = 1; i <= 100; ++i) {
        if ( !all(lessThan(abs(fullOffset), offsetLimit))) {
            fullOffset = currentOffset.xyz + closestBackfaceVector * (scaleFactor - (i * 0.01));
        }
    }
} else if
(!(dot(farthestFrontfaceVector, closestFrontfaceVector) > 0.5)) {
    // The farthest frontface may also be the closest if the probe can only see one surface.
    // If this is the case, don't move the probe.

    // Ensure that we never move through the farthest frontface by moving minimum distance possible.
    vec3 farthestDirection = min(0.2, farthestFrontfaceDistance) * farthestFrontfaceVector;

    fullOffset = currentOffset.xyz + farthestDirection;
}

if (all(lessThan(abs(fullOffset), offsetLimit))) {
    currentOffset = fullOffset;
}
```

Listing 4. Pseudocode for an iteration of the probe position optimizer operating on a single probe.

The probe position optimizer runs for five iterations during probe-state classification, which is enough for almost all probes to converge their locations. We cap the number of iterations at five to prevent probes from moving back and forth (infinitely) through tangent backfaces.

More work is needed to determine the best position-optimizer algorithm, and many investigations in this vein exist (see, for example, Wang et al. [2019]). Our optimizer worked well for multiple engines, but is almost certainly not optimal.

6. Probe States

In any scene with significant open space, even after adjustment, many probes in a uniform 3D grid will not contribute to the final image. We introduce a robust set of probe states to avoid tracing or updating from such probes to increase performance with the same visual result. Our probe states separate probes that should not update from probes that must, with an additional intermediate state to identify probes that have just appeared (either at scene initialization or with a moving volume—see Section 7.3) and adjust their hysteresis accordingly. The full set of states is shown in Figure 7 and discussed in the following sections.
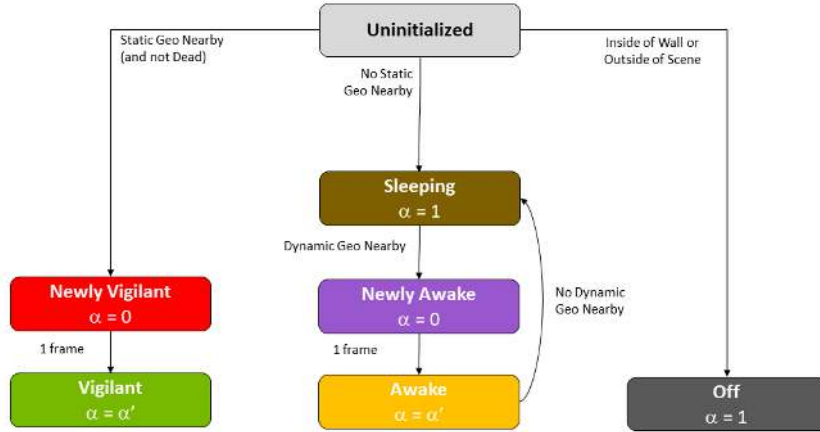
Figure 7. Probe states with transitions between each state; α is the hysteresis for the current frame; α' is the default hysteresis for the scene.

6.1. Deactivated Probes

As noted above, the constraints on probe movement imposed by the 3D grid indexing make it impossible to move all probes out of walls (some probes are too constrained by the grid structure). We identify probes that remain inside static geometry and turn them "off" (never trace or update). As the optimizer only considers static geome-

try, probes that happen to spawn inside dynamic geometry are unaffected and will correctly turn on when appropriate.

6.2. Probe-update States

Even probes that are outside static geometry are not used for shading at every frame: when no geometry is within $\mathrm{probeSpacing}$ of a probe, that probe's value is not used. We set these probes to "Sleeping" and wake them up when a surface is about to use them for shading. Note that a probe needs to be "Awake" *if and only if* it is shading a surface or about to shade one. Lighting changes and camera proximity do not matter if the probe is not shading a surface. The same is true for setting probes to "Sleeping": when the camera can't see a probe, it still needs to be "Awake" if it is shading a surface because it is propagating diffuse irradiance (with second through nth-order visibility). Thus, probes that shade static geometry should be "Vigilant" (they should *always* trace and update). Though probes near geometry must trace to propagate GI, the grid resolution need not be as fine in regions that are far from the camera.

6.3. Full Probe-initialization Algorithm

Probe positions and states are computed in four steps (see Listing 5):

- For all uninitialized probes, trace rays for five frames to determine optimal positioning and initial state. At the end of this pass, all previously uninitialized probes are "Newly Vigilant," "Off," or "Sleeping."

- Extend AABBs for all dynamic objects by a probe grid cell + the self-shadow bias for a conservative estimate. Set all "Sleeping" probes inside the extended AABB of a dynamic object to "Newly Awake."

- Optionally trace a large number of rays for "Newly Vigilant" and "Newly Awake" probes to converge them in a frame, setting hysteresis to 0. Set their states to "Vigilant" and "Awake," respectively.

- Trace rays from "Vigilant" and "Awake" probes to update their values with the normal hysteresis value for the scene. This step can also be used to converge "Newly Vigilant" and "Newly Awake" probe values if the previous step was omitted.

The first step of the algorithm can be greatly accelerated with static geometry bounding boxes, as a probe can be directly adjusted against those bounding boxes rather than relying on distance and backface information from the spherical ray cast. Many probes could be immediately classified "Newly Vigilant" with this approach, though ray tracing would still be necessary to correctly determine which probes should be set to "Off."

```
for each uninitialized probe:
    Trace rays (distance only, no shading)
    Position optimizer iteration
    if (still in wall):
        OFF
    if (frontfaceDistance < probeSpacing):
        NEWLY VIGILANT
    else
        SLEEPING

for all dynamic geo:
    Extend bounding boxes grid cell size + self shadow bias
    for all SLEEPING probes:
        if (probe inside bounding box):
            NEWLY AWAKE

// Optionally converge probes in this frame...
for all NEWLY AWAKE and NEWLY VIGILANT probes:
    Trace rays to converge value
    NEWLY AWAKE -> AWAKE
    NEWLY VIGILANT -> VIGILANT

// ...or let them converge in the update pass.
for all VIGILANT or AWAKE probes:
    Trace rays and update value.
```

Listing 5. Pseudocode for probe state computation.

Participating media and probe states.   The probe data structure encodes a 3D irradiance field that is queryable at any point within its volume. Thus, it might be queried at positions in empty space to provide global illumination in participating media. In this case, even probes not shading a surface would need to be "Awake" if they are within the participating medium.

Though the passes described in Listing 5 potentially run every frame, for the majority of frames the first step will not run because no probes will be uninitialized. If the optional convergence pass is omitted, then only the final update step will run for most frames.

7. Quantitative Performance Improvements

7.1.   Probe Sleeping Performance

Probe sleeping using our probe-state scheme leads to a 30–50% average performance improvement (Figure 8). In addition to the performance improvement (shown in the middle column), we also show corresponding increases in rays cast per probe for the same performance. Casting more rays per probe makes new probe values more stable and allows for a lower global hysteresis, which makes the GI converge faster.

| Scene	Baseline (time ms)	Equal Quality	Time saved (ms)	Better Quality (rays) (Equal Time)
Rungholt 12M tris	2.05	32%	0.65	+ 416,512
Game Asset 1.4M tris	3.7	38%	1.4	+ 1,565,952
Greek Temple 1M tris	2.7	54%	1.45	+ 1,703,936
Terrain 2.8M tris	0.75	28%	0.3	+ 151,424

Figure 8. Performance data for probe sleeping. The "Baseline" column shows the time for probe trace and update without probe sleeping (all probes are marked "Vigilant"). The "Equal Quality" and "Time Saved" columns show savings of probe sleeping as a percentage of time and as absolute time, respectively. Finally, the "Better Quality" column shows the absolute ray increase achievable by tracing more rays from active probes to match the baseline time.

7.2. Probe Update Shader Optimization

The approach of Majercik et al. [2019] updated probe texels using a pixel shader with a stencil buffer (to avoid processing border texels in the update pass). Border texels were updated in a separate pixel shader pass for correct bilinear interpolation. This approach leverages the graphics hardware for alpha blending results. Despite this, faster update can be achieved by using a general-purpose GPU (GPGPU) compute operation optimized with GPU compute best practices. We give background and details of this approach below.

Modern GPU architectures dispatch thread groups to cover user-specified compute grid dimensions. All threads in a group execute the same code in parallel, so ensuring that threads do not take different control paths in the code (coherent execution) is vital for performance. By ensuring coherent execution, we achieve a 3x performance improvement in the update pass over the pixel-shader approach with careful indexing over thread blocks consisting of an integer number of groups. In addition, we store incoming shaded sample ray hits in shared memory buffers so that all threads can read it in parallel when computing a new probe texel value.

Previous work showed the effect of probe resolution on image quality and performance. We maintain image quality while selecting probe resolution (8×8 irradiance, 16×16 visibility) for a combination of bandwidth, memory footprint, fast convolution, efficient index computation, and most important: mapping to SIMD instructions (thread lanes on a GPU) for peak occupancy on our target hardware. At powers of two, a probe can be updated by an integer number of 32 or 64 thread groups (common hardware-defined minimum sizes) for maximum possible occupation and coherence. Arbitrary resolution values offer the highest flexibility at the cost of efficiency.
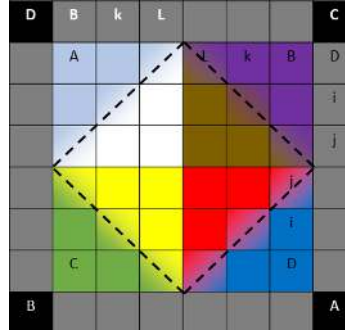
(a) Octahedral representation and border-copy texels. Colors denote faces on the collapsed octahedron. Letters in border cells denote copy destinations for cells inside the border labeled with the same letter.
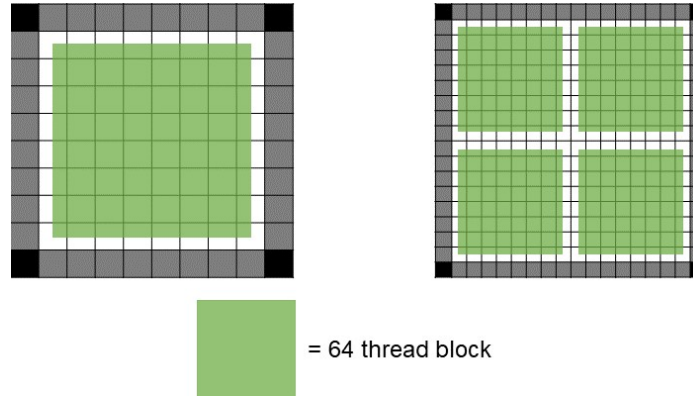
(b) Thread-block alignment for probe update on an 8×8 irradiance probe (left) and a 16×16 visibility probe (right).
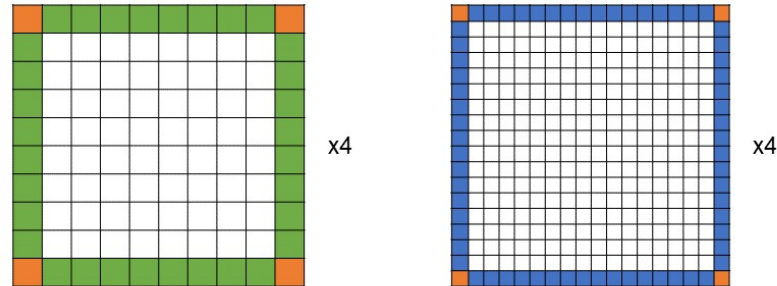
(c) Thread-block alignment for probe-border copy. One block of 32 threads copies corners for four irradiance and four visibility probes (orange). Four blocks copy edges for four irradiance probes (green). Eight blocks copy edges for four visibility probes (blue).

Figure 9. Octahedral probe layout and probe update thread indexing.

Figure 9 shows details of our compute shader indexing, including an example octahedral probe encoding to illustrate border-texel copy for correct hardware bilinear interpolation. Our optimized compute shader is included alongside the update shader of the previous technique [2019] in the supplemental material.

7.3. Tracking Windows

Conceptually, a probe grid covers all space in the scene. In practice, however, we do not have the compute or ray-tracing budget to update and trace a level-sized, high-resolution probe grid as it may contain tens of thousands of probes. To maintain high probe resolution where it is most necessary, we implement a 3D tracking window of probes. We used this window to track the camera, though any object can be tracked with the same strategy. Our window begins centered on the camera. As the camera moves, if it moves further from the center than the distance between two probes in a cage (along any axis), a new plane of probes spawns in front of it (relative to its direction of motion), and the plane furthest behind it disappears. We implement this behavior using a 3D fixed-length circular buffer. When a new probe plane appears and is initialized, its new values are written to the memory of the plane in the last row behind the camera: the probes "leapfrog" over the camera in discrete steps (Figure 10). A discretely stepping probe window necessitates careful interpolation between multiple probe volumes—our strategy for this is discussed in Section 7.4.

(a) Default grid   (b) Offset grid
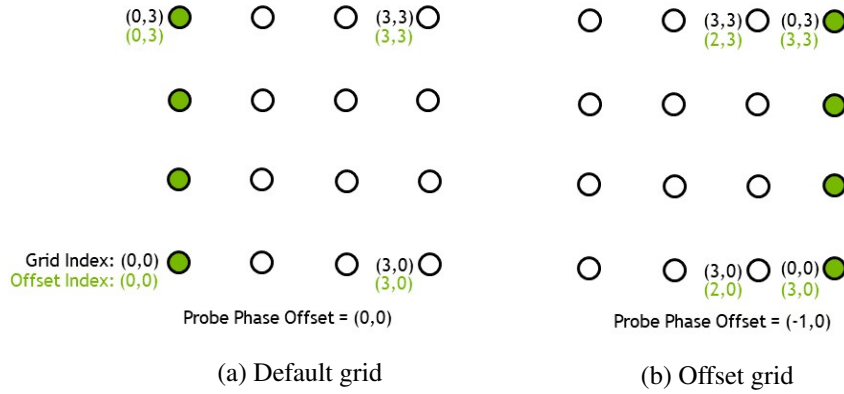
Figure 10. Conceptual layout of the camera tracking window indexing with phase offset in 2D. The row of probes that moves is colored in green. When the camera passes the center bounding threshold moving in the +X direction, the leftmost row of probes leapfrogs to the +X face of the volume. The newly computed grid index is shown in green. The corresponding phase offset change is shown on the right.

7.4. Multiple Probe Volumes

Multiple probe volumes at differing resolutions can be used to efficiently implement progressively decreasing grid resolutions that cascade out from the camera, thus sav-

Figure 11. Shaded ray data for multiple volumes packed into a single texture. This texture is irradiance data taken from our multivolume scene in the supplemental video. The texture includes shaded update rays for the camera-locked volume, the city-scale volume, and the level-scale volume—these are labeled in the figure and delineated within the texture by the red lines (which are not part of the irradiance data).

ing performance without effecting image quality. The same approach is used in geoclipmaps [Losasso and Hoppe 2004], light-propagation volumes [Kaplanyan and Dachsbacher 2010], and voxel cone volumes [Crassin et al. 2011]. Additional high-resolution volumes can also be used to efficiently cover hero assets with complex geometry that require higher resolution diffuse irradiance. Multiple volumes can be efficiently updated in one trace by packing all update rays into a single texture (see Figure 11).

(a) Multiple probe volumes   (b) Transition start (marked in green)   (c) Dense volume hidden
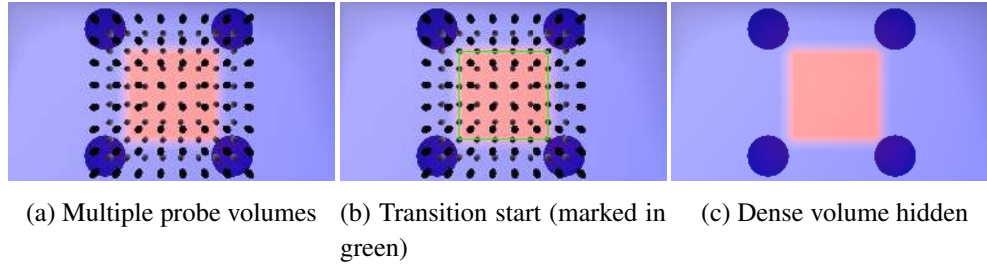
Figure 12. A dense volume (smaller spheres) within a sparse volume (larger spheres). The spheres are sized based on the probe spacing within each volume. On the far left, the pink region shows the area fully shaded by the dense volume, which gradually falls off to blue, the area shaded by the sparse volume. The center image marks the start of this transition. The rightmost image hides the dense probes to clearly visualize the transition.

We blend between volumes by linearly falling off from 1.0–0.0 at the last grid cell (starting at the second-to-last plane of probes) along each axis of the 3D grid (Figure 12). In the deferred shader, a weight is computed for each volume starting from most to least dense. We sample in density order because the densest volume will have the best approximation of the local light field at a point it contains. Volume weights are accumulated at each volume sample. After the weight total reaches 1.0, further volumes are skipped.
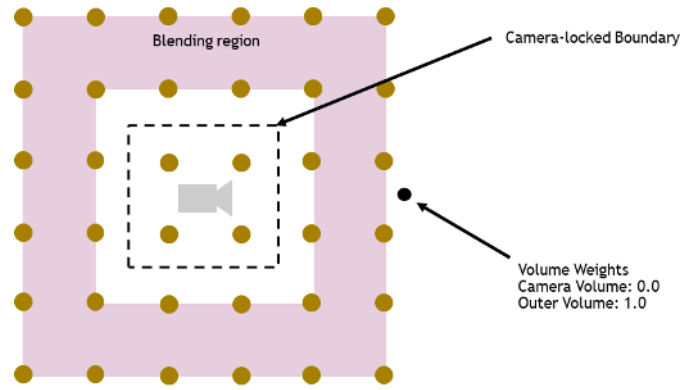
(a) Initial camera position. Labels show the blending region for the camera tracking window, the camera boundary that will cause the volume to move, and the volume weights for a point being shaded by the camera volume (brown circles) and a surrounding volume (not visualized).
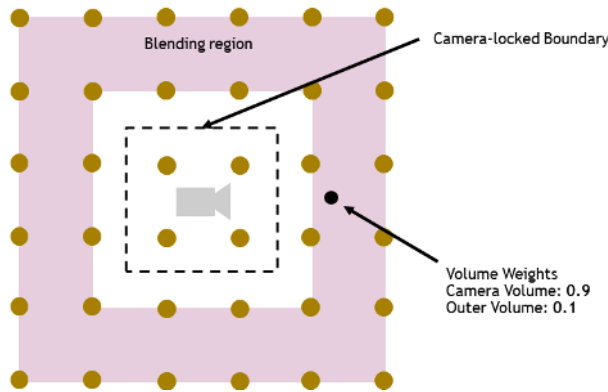
(b) Camera moves. Without camera-aware blending, volume weights on the point change dramatically in one frame.
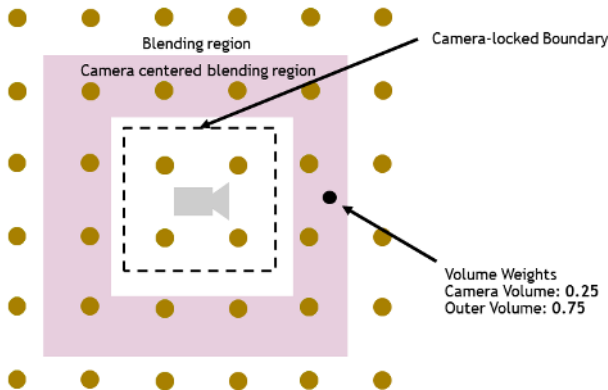
(c) Camera moves. With camera-aware blending, the volume weights change slowly over the course of multiple frames, leading to smoother transitions.

Figure 13. 2D illustration of volume blending using the static volume method vs. a camera-aware volume blending.

The weighted volume blending described above yields smooth transitions for static volumes, but can cause popping in the GI when applied to camera-locked volumes. When a plane of probes leapfrogs in front of the camera, some points can go from being fully shaded by a sparse cascade to being heavily shaded by the camera cascade (Figure 13). When computing blending weights for camera-locked volumes, we address this by tightening the transition region by one grid cell (along each axis) then centering it on the camera. When the new plane appears at the front of a volume, points that are newly within that volume will not immediately be shaded by it because of the tightened transition region. Instead, those points will gradually transition between volumes as the camera moves towards them. Results are shown in our supplemental video.

The prototype multivolume code passes all probe volumes to the deferred shader, and then per-pixel iterates through them to figure out which ones contain the point being shaded. Though not the optimal approach for performance, this provides the highest flexibility in tweaking the blending algorithm to evaluate image quality. For a production implementation, the usual solutions for the deferred shading light-loop issue (considering the volumes as lights) are available:

- Do the full brute-force light loop—for fewer than 10 volumes, the point-in-OBB test to determine which volumes contain the shaded point is fast to evaluate;

- Rasterize each volume's bounds to find the covered pixels;

- Make a spatial data structure (e.g., octree, BVH) over the volumes and then traverse that at runtime in the pixel shader to find in which volumes the pixel is located. This method requires more bookkeeping and potentially costly data-dependent fetches;

- Use tiles [Olsson et al. 2012] set up on the CPU or with a GPU pass to conservatively approximate one of the previous methods.

For the pure cascaded method, these optimizations are not necessary because volumes are axis-aligned in world space and nested in a regular pattern.

7.5. Inline Shading

Previous probe schemes required an extra shader pass to gather the indirect contribution over the frame. We present a simpler framework that optimizes the global-illumination gather step to directly sample the probe data structure during shading, yielding reduced bandwidth requirements. Our code is included in the supplemental material in `GIRenderer_deferredShade.pix`.

8.　**Conclusion and Discussion**

We present multiple extensions to the dynamic diffuse global-illumination algorithm [Majercik et al. 2019] to improve image quality, performance, and ease of deployment in a production setting. These extensions were developed in response to production constraints encountered when integrating the technique into the NVIDIA RTXGI SDK, the Unity game engine, the Unreal Engine 4 game engine, and several commercial games.

The base algorithm of Majercik et al. [2019] is inherently practical due to its image quality and performance. This paper covers the gap between a practical algorithm and one that is ready for production deployment. Extensions like our "self-shadow bias" make the algorithm easier to tune, and our performance optimizations to the update pass make it feasible for the render budget of production games. For all of our extensions, we sought solutions that were robust, easy to understand, and easy to tune without fundamentally changing the algorithm.

8.1.　Limitations and Future Work

Though our proposed convergence heuristics increase convergence over the previous approach, there is still some ghosting in the indirect illumination for small, bright light sources (like flashlights—see our video supplement at 7:05). This lag could be addressed by intensifying our specific hysteresis-reduction heuristics on small lights known to cause ghosting, though doing this globally may cause instability in other regions of the image. While more specialized methods like reflective shadow maps yield less ghosting [Dachsbacher and Stamminger 2005], an advantage of our method is that all light sources can be handled generically to produce global illumination—we trade some quality for generality.

In addition to our performance improvements, a per-frame ray budget could be implemented to allow more control over the render budget of the technique. For our applications, we found that controlling (a) the rays per probe and (b) the number of probes in a volume was enough to hit our performance targets. A more sophisticated treatment of ray budget would trace different ray amounts on a per-probe basis, adding a lot of complexity to the implementation. We chose simplicity over a more optimized ray budget, but a study of optimal ray apportioning between probes (taking into account lighting and geometry changes, the camera position, etc.) is interesting future work.

Our algorithm covers a large space of rendered effects and thus suggests many possible directions for future work. For instance, our technique forces second-order glossy reflections to maximum roughness in order to reuse the irradiance values as cosine-filtered radiance. Second-order glossy reflections could be improved by using multiple higher-resolution filtered radiance textures with different cosine power weighting—like the weighting for visibility probes, but with multiple octahedral rep-

resentations per sample point instead of one. These could be used to render second-order glossy reflections of varying roughness.

Acknowledgements

References

CIGOLLE, Z. H., DONOW, S., EVANGELAKOS, D., MARA, M., MCGUIRE, M., AND MEYER, Q. 2014. A Survey of Efficient Representations for Independent Unit Vectors. *Journal of Computer Graphics Techniques (JCGT) 3*, 2 (April), 1–30. URL: http://jcgt.org/published/0003/02/01/. 6, 7

CRASSIN, C., NEYRET, F., SAINZ, M., GREEN, S., AND EISEMANN, E. 2011. Interactive Indirect Illumination Using Voxel Cone Tracing. *Computer Graphics Forum 30*, 7, 1921–1930. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2011.02063.x. 22

DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, Association for Computing Machinery, New York, NY, USA, I3D '05, 203–231. URL: https://doi.org/10.1145/1053427.1053460. 25

FASCIONE, L., HANIKA, J., HECKENBERG, D., KULLA, C., DROSKE, M., AND SCHWARZHAUPT, J. 2019. Path Tracing in Production: Part 1: Modern Path Tracing. In *ACM SIGGRAPH 2019 Courses*, Association for Computing Machinery, New York, NY, USA, SIGGRAPH 19. URL: https://doi.org/10.1145/3305366.3328079. 12

HOOKER, J. 2016. Volumetric Global Illumination at Treyarch. In *Advances in Real-Time Rendering 2016*, SIGGRAPH 2016. ACM, New York, NY, USA. URL: https://www.activision.com/cdn/research/Volumetric_Global_Illumination_at_Treyarch.pdf. 5, 6

JENDERSIE, J., KURI, D., AND GROSCH, T. 2016. Real-Time Global Illumination Using Precomputed Illuminance Composition with Chrominance Compression. *Journal of Computer Graphics Techniques (JCGT) 5*, 4 (December), 8–35. URL: http://jcgt.org/published/0005/04/02/. 7

KAPLANYAN, A., AND DACHSBACHER, C. 2010. Cascaded Light Propagation Volumes for Real-Time Indirect Illumination. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, Association for Computing Machinery, New

York, NY, USA, I3D 10, 99107. URL: https://doi.org/10.1145/1730804.1730821. 22

KELLER, A., FASCIONE, L., FAJARDO, M., GEORGIEV, I., CHRISTENSEN, P., HANIKA, J., EISENACHER, C., AND NICHOLS, G. 2015. The path tracing revolution in the movie industry. In *ACM SIGGRAPH 2015 Courses*. ACM, New York, NY, USA, 1–7. URL: https://dl.acm.org/doi/10.1145/2776880.2792699. 7

LAGARDE, S., AND ZANUTTINI, A. 2012. Local Image-based Lighting With Parallax-corrected Cubemap. SIGGRAPH 2012. ACM, New York, NY, USA. URL: https://seblagarde.wordpress.com/2012/11/28/siggraph-2012-talk/. 5

LOSASSO, F., AND HOPPE, H. 2004. Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids. *ACM Trans. Graph. 23*, 3 (Aug.), 769776. URL: https://doi.org/10.1145/1015706.1015799. 22

MAJERCIK, Z., GUERTIN, J.-P., NOWROUZEZAHRAI, D., AND MCGUIRE, M. 2019. Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields. *Journal of Computer Graphics Techniques (JCGT) 8*, 2 (June), 1–30. URL: http://jcgt.org/published/0008/02/01/. 1, 2, 5, 6, 7, 8, 19, 21, 25, 28

MARQUES, R., BOUVILLE, C., RIBARDIÈRE, M., SANTOS, L. P., AND BOUATOUCH, K. 2013. Spherical Fibonacci Point Sets for Illumination Integrals. *Computer Graphics Forum 32*, 8, 134–143. URL: https://hal.inria.fr/hal-01143347. 4

MARTIN, S., AND EINARSSON, P. 2010. A Real Time Radiosity Architecture for Video Games. In *Advances in Real-Time Rendering 2010*, SIGGRAPH 2010. ACM, New York, NY, USA. URL: https://www.ea.com/frostbite/news/a-real-time-radiosity-architecture. 5, 6

MCAULEY, S. 2012. Calibrating Lighting and Materials in Far Cry 3. In *Practical Physically Based Shading in Film and Game Production*, SIGGRAPH 2012. ACM, New York, NY, USA. URL: https://blog.selfshadow.com/publications/s2012-shading-course/. 5, 6

MCGUIRE, M., MARA, M., NOWROUZEZAHRAI, D., AND LUEBKE, D. 2017. Real-Time Global Illumination using Precomputed Light Field Probes. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, New York, NY, USA, February, 11. URL: http://casual-effects.com/research/McGuire2017LightField/index.html. 1, 6, 7

OLSSON, O., BILLETER, M., AND ASSARSSON, U. 2012. Tiled and Clustered Forward Shading: Supporting Transparency and MSAA. In *ACM SIGGRAPH 2012 Talks*, Association for Computing Machinery, New York, NY, USA, SIGGRAPH 12. URL: https://doi.org/10.1145/2343045.2343095, doi:10.1145/2343045.2343095. 24

RITSCHEL, T., GROSCH, T., AND SEIDEL, H.-P. 2009. Approximating Dynamic Global Illumination in Image Space. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '09, 75–82. URL: http://doi.acm.org/10.1145/1507149.1507161. 5, 6

SILVENNOINEN, A., AND LEHTINEN, J. 2017. Real-time Global Illumination by Pre-computed Local Reconstruction from Sparse Radiance Probes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia) 36*, 6 (11), 230:1–230:13. URL: https://doi.org/10.1145/3130800.3130852. 7

STACHOWIAK, T., AND ULUDAG, Y. 2015. Stochastic Screen-Space Reflections. In *Advances in Real-Time Rendering 2015*, ACM, New York, NY, USA, SIGGRAPH 2015, EA DICE. URL: https://www.ea.com/frostbite/news/stochastic-screen-space-reflections. 7

STEFANOV, N. 2016. Global Illumination in Tom Clancy's *The Division*. Presented at Game Developers Conference, 2016. URL: https://www.youtube.com/watch?v=04YUZ3bWAyg&feature=youtu.be&t=657. 6

TOTH, R., HASSELGREN, J., AND AKENINE-MÖLLER, T. 2015. Perception of Highlight Disparity at a Distance in Consumer Head-mounted Displays. In *Proceedings of the 7th Conference on High-Performance Graphics*, ACM, New York, NY, USA, HPG '15, 61–66. URL: http://doi.acm.org/10.1145/2790060.2790062. 7

VALIENT, M. 2013. Killzone Shadow Fall Demo Postmortem. Sony Devstation 2013, Guerilla Games. URL: https://www.guerrilla-games.com/read/killzone-shadow-fall-demo-postmortem. 5

WANG, Y., KHIAT, S., KRY, P. G., AND NOWROUZEZAHRAI, D. 2019. Fast Non-Uniform Radiance Probe Placement and Tracing. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, Association for Computing Machinery, New York, NY, USA, I3D 19. URL: https://doi.org/10.1145/3306131.3317024. 6, 16

WYMAN, C. 2005. An Approximate Image-space Approach for Interactive Refraction. *ACM Trans. Graph. 24*, 3 (July), 1050–1053. URL: http://doi.acm.org/10.1145/1073204.1073310. 7

Index of Supplemental Materials

The supplemental material contains video results for each of our extensions. as well as relevant C++ and shader code and is available at http://jcgt.org/published/0010/02/01/supplement.zip. Where appropriate, we have included code from Majerick et al. [2019] for comparison.

Author Contact Information

Zander Majercik

NVIDIA Corporation

2788 San Tomas Expy

Santa Clara, CA 95051

amajercik@nvidia.com

Adam Marrs,

NVIDIA Corporation

2788 San Tomas Expy

Santa Clara, CA 95051

amarrs@nvidia.com

Josef Spjut

NVIDIA Corporation

2788 San Tomas Expy

Santa Clara, CA 95051

jspjut@nvidia.com

Morgan McGuire

NVIDIA Corporation

2788 San Tomas Expy

Santa Clara, CA 95051

mcguire@nvidia.com