Dynamic Diffuse Global Illumination: Implementation and Evaluation

Michal Hvezda
Supervised by: Jiri Bittner[*]

Department of Computer Graphics and Interaction
Czech Technical University
Prague / Czech Republic

Abstract

We describe our implementation of a recent real-time global illumination method based on irradiance fields, which uses ray tracing to approximate multiple bounces of light transmission in a virtual scene. Furthermore, we present qualitative comparisons with a brute force path tracer and the method's performance results. Lastly, we discuss possible extensions, such as adaptive ray generation and probe placement.

Keywords: Global Illumination, Real-time graphics, Ray-tracing, Irradiance Probes

1 Introduction

Accurate illumination in a virtual scene is often the most crucial visual effect in computer graphics. Correct light transfer simulation blurs the line between a real image and an artificial one. Furthermore, the effect can yield stunning images even in stylized graphical applications. However, the computation of correct light propagation is costly, and thus global illumination is often only approximated for real-time applications. With the rise of the ray-tracing GPUs in recent years, a lot of development into algorithms that try to leverage the potential of the hardware has spurred up. We implemented one such algorithm from a recent paper by Majercik et al. [11] which complements traditional rasterization with a ray-traced irradiance field to approximate a diffuse global illumination for dynamic illumination and geometry. We present our implementation of the Dynamic Diffuse Global Illumination (DDGI) method and show our performance results together with quality comparisons with path-traced ground truth.

The paper is structured as follows. Section 2 gives an overview of existing global illumination methods used in real-time CG. Then, Section 3 details our implementation of the DDGI [11] method to approximate diffuse global illumination for a fully dynamic scene. Our results are described in Section 4, where we present our performance findings and compare the quality of the produced images to a path-traced reference. Last but not least, in Section 5 we discuss the drawbacks of the method and its possible improvements.

2 Related Work

Since global illumination is costly to compute, GI methods usually resorted to some trade-off to calculate the effect. The most notable is the use of a fully static scene. For scenes where only the camera is dynamic, we can use any of the well-established offline methods, e.g. *path-tracing* [8]. Using such techniques, we would precompute the light transfer and store it in textures beforehand for later rendering. This process is often called light baking.

However, for dynamic or at least partially dynamic scenes, the real-time global illumination problem becomes more difficult. We give a rough overview of three categories of global illumination methods for interactive applications [14].

2.1 Finite Elements

The first category would be Finite Elements (Radiosity) methods [4], whereas the name implies the scene is discretized into a finite number of surface elements. The approach completely omits the camera view from its calculations and only considers the light transfer between the surface patches.

The original method is suitable for diffuse light transfer and later was extended [6] by accounting for glossy materials. The quality of the global illumination is dependent on the number of patches which introduces the problem of scalability. Thus, due to the quadratic nature of the light transfer calculation, the original approach does not scale well. However, P. Hanrahan et al. [5] introduced a hierarchical approach to solving the light transfer, reducing the complexity of the problem.

2.2 Photon mapping

Another approach would be *photon mapping* [7]. The idea behind photon mapping is to emit a large number of pho-

[*]bittner@fel.cvut.cz

tons from the light source and let the photons bounce inside the scene. At each hit point, we can store the photon inside a photon map. During rendering, we then use the photon map to determine the irradiance of each fragment based on the photon density in the area.

The method can be adapted to achieve interactive rates under some conditions. Purcell et al. [13], and Ma and McCool [10] described an approach where they used spatial hashing instead of the nearest neighbor search required in density estimation, which better fits GPUs.

2.3 Monte Carlo Ray Tracing

A possible approach to solving GI is the use of Monte Carlo techniques [8]. These techniques produce A high number of directional samples, which are evaluated for incoming light, and the average of the results converges to the correct solution. The evaluation of each sample is usually done using *Ray-tracing*.

The Monte Carlo integration is a complex problem for millions of dynamic lights. The paper by Bitterli et al. [2] introduces an algorithm called ReSTIR that renders such lights interactively. They achieve this by repeatedly resampling a set of candidate light samples and leveraging information from relevant nearby samples by further spatial and temporal resampling. Later Majercit et al. [12] proposed to combine ReSTIR's shadowing with DDGI. By combining these two algorithms, they outperform hardware accelerated path tracing in both runtime and noise.

Silvennoinen et al. [16] presented a radiance field method for mostly static scenes with dynamic lights, cameras, and diffuse and emissive materials. Their approach features minor light leaking due to their algorithm for faithfully interpolating incidence radiance captured at a sparse set of low-frequency radiance probes to nearby receiver points.

3 Dynamic Diffuse Global Illumination

To approximate global illumination for a dynamic virtual scene, Majercik et al. [11] proposed to compute the light transfer by recurrently updating the irradiance field every frame. This is achieved by using information gathered by rays cast from each irradiance probe placed in the virtual scene. Since the ray casting is independent of the primary rendering, it avoids denoising or prefiltering high-resolution spherical textures. Our probe placement strategy and probe representation are detailed in Section 3.1. The geometry data and material attributes gathered by ray casting are saved into a structure similar to the G-buffer called a surfel buffer. The surfel buffer is then used for ray shading by direct and indirect light. Lastly, we use the computed illumination contributions to update data inside the probes. To summarize, the algorithm executes four steps in each frame:

1. Generate m rays from each probe, creating $m \times n$ rays in total, where n is the number of probes in the scene. The ray generation is further detailed in Section 3.2.

2. Cast and trace $m \times n$ rays into the scene. Each ray gathers attributes from the scene's geometry into a G-buffer-like structure of surfels. Section 3.3 gives a more in-depth description of the ray casting and the surfel buffer.

3. Shade rays by direct and indirect illumination using data in surfel buffer. The ray shading method is detailed in Section 3.4.

4. Update irradiance and distance probe data for each of m probes using the shaded rays and their hit distances. The update procedure is detailed in Section 3.5.

The resulting irradiance probe field is then used to compute the indirect illumination contribution for the final image visible from the camera. This is done in the same way as in ray shading by indirect illumination, but instead of probe rays we use view rays from the camera.

3.1 Probe placement and representation

We place the irradiance probes in the scene's bounding volume at vertices of a uniform 3D grid since it provides fast probe queries and effective interpolation between probes. To have a bit more flexibility, at the cost of slightly lower performance, we opted for arbitrary grid resolution instead of an always a power of two resolution as was described in the original paper.

As shown in Figure 1 the probes' data is stored as two texture atlases, where the first holds irradiance maps of each probe and the second contains distance maps used for visibility testings during indirect illumination computation. Since neither of these maps do not require high resolution, they are stored as octahedral maps instead of the standard cube maps. The implementation allows for separate settings of maps' side lengths for each atlas for scenes where we need higher precision.

3.2 Generating rays

For each of the n probes, m rays are generated, yielding $m \times n$ rays in total. Rays share a common origin with their probe's center position. Directions of the m rays are uniformly sampled spherical directions. To achieve the uniform distribution of spherical ray directions we use a Fibonacci spherical mapping [9]. In the implementation we store origins and directions as 32-bit 4-element float vectors in two separate textures. The textures are then passed to a ray-tracer to be cast as one batch, see Figure 2.

3.3 Probe ray casting

From each of the n probes, we cast m rays. In order to avoid visibility errors when a probe is inside geometry,

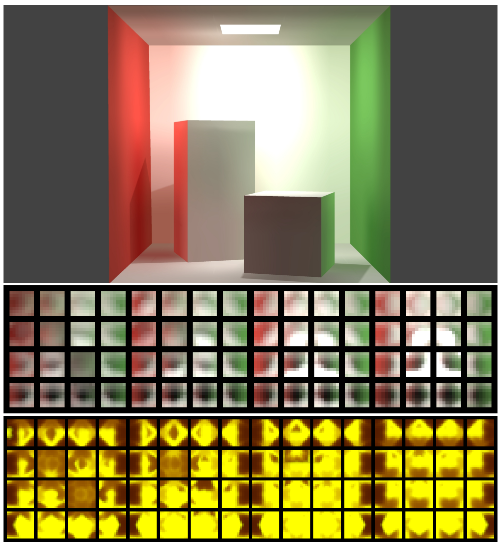

Figure 1: On the first image is a standard Cornell box scene where we've placed a 4x4x4 probe grid. The second image is our atlas of octahedral irradiance maps for each of the probes placed in the scene, where each map has a resolution of 8x8. The final image shows the distance atlas where each map has a resolution of 16x16. Each entry in the distance atlas contains distance and a squared distance.

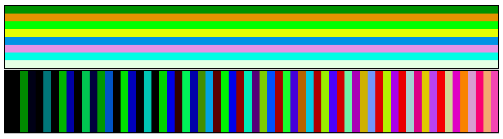

Figure 2: Images demonstrate the generated ray textures for 8 probes with 64 rays per probe. Top texture stores ray origins. Each entry in the texture contains the corresponding probe center and a minimum ray distance. The bottom texture contains ray directions sampled using Fibonacci spherical mapping. The alpha channel contains the maximum distance a ray can travel, which is set to infinity.

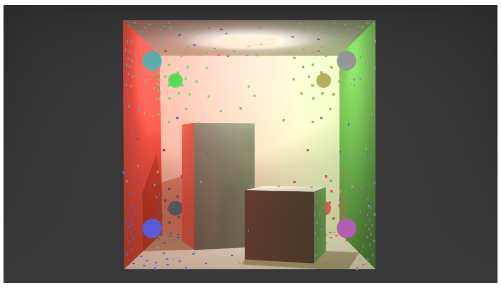

Figure 3: Sampled points from each probe in the Cornell box scene. Each sampled point color corresponds to its probe.

backface culling is ignored. On a hit, each ray then gathers data about the scene's geometry into a G-buffer-like structure of surfels. If a ray misses the scene we store the miss as a normal vector with zero length. The structure contains a set of $m \times n$ textures where each texture represents data about surfels' attributes. In the implementation, the buffer contains explicit world-space position, normal and surfels' material attributes: Lambertian, emissive, glossy, and transmissive terms. Each value in these textures is a 32-bit 4-element float vector.

3.4 Ray shading

For clarity, we separate the ray shading into two passes. First, we shade the rays by diffuse indirect illumination leveraging the probe data computed in the previous frame. Section 3.4.1 describes the indirect illumination pass and the weights used to handle visibility errors such as light leaks and shadow leaks.

In the second pass, we compute the direct illumination for each ray from the light sources in the scene and add indirect illumination contributions from the second pass. The Direct illumination is described in Section 3.4.2.

3.4.1 Diffuse Indirect illumination

Once the probes collect their data from the scene, we can shade the rays by diffuse indirect illumination. We sample each ray's hit location from the surfel buffer. If the hit surfel exists, we find eight closest probes to that surfel forming a grid cage around it (see Figure 4). This way, we can encapsulate every point in the scene in a grid cage because of the grid structure of the irradiance field.

After the probe cage is formed, we iterate over every probe in the cage. We sample the light that the probe sees in the direction of the surfel point. More specifically, we take the normal of the surfel point, encode it into octahedral texture coordinates, and sample the corresponding irradiance texture. The sampled irradiance is then added to the total irradiance sum over every probe in the cage.

The sampled irradiance from probes on its own would not yield correct results since it does not account for visibility. To ensure that the indirect light appears continuous and accounts for dynamic geometry and lighting Majercik et al. [11] describes various methods to smooth it and cull unwanted contributions. The authors use the following weights to blend the irradiance from the closest eight probes:

Smooth backface weight - Wrap shading Wrap shading [17] is commonly used as a cheap approximation to subsurface scattering or as a more expressive base-shading model. Furthermore, it can be used as a heuristic to cull indirect contributions from probes that are not mutually visible to the surfel.

Chebyshev moment visibility test - Variance shadows In CG Variance-biased Chebyshev interpolant [3] is used

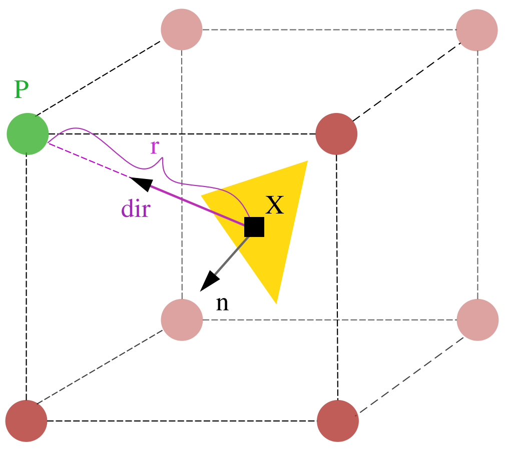

Figure 4: Depiction of the eight probe grid cage around a surfel X. Each probe P is sampled using the surfel's normal n in world space. Each probe's contribution is then weighted based on its visibility using direction *dir* from the sampled point X to the current probe P. r represents the mean distance from X to P.

to counter shadow aliasing by approximating how shadows soften on their edges. DDGI makes use of the variance shadow maps as an effective heuristic to counter light leaks by filtering out probe contributions which are occluded by geometry.

Log perception weight The weight accounts for human perception, specifically how eyes are sensitive to contrast in low-light conditions. Specifically, the weight crushes small weights but keeps the curve continuous.

Trilinear weight The usual trilinear interpolation to smooth the contributions of probes based on their distance from the surfel point.

3.4.2 Direct illumination

To calculate direct illumination for each of the samples, we use the surfel buffer as a standard G-buffer as if we were rendering the final image. More specifically, we compute direct light contributions from each light source in the scene for each surfel stored in the buffer. For visibility queries, we project surfel's world space coordinates from the surfel buffer to light space to check if the point is in shadow. Once the contribution of the direct light is computed, we sample the indirect illumination texture from the previous pass and add its contribution.

3.5 Updating probes

The final step is to update the irradiance and distance maps. Using the now shaded surfels, for each probe texel, we gather the irradiance E_e from all sampled points and update them using linear interpolation between the new and already existing irradiance value stored in the map, where the linear interpolation is controlled by a *hysteresis* parameter α:

$$E_{e_{new}}(\omega) = \text{lerp}(E_{e_{old}}(\omega), \sum_{probeRays} \max(0, \omega \cdot r) * L_e, \alpha) \quad (1)$$

where E_e is irradiance, ω is the probe's texel direction, r is the rays direction, and L_e is the rays radiance.

The distance map is updated in the same way. For each texel in the distance map, we sum the distances between the probe center and hit locations of the rays. Then the old distance, and the squared distance, are updated using the hysteresis parameter:

$$D_{new}(\omega) = \text{lerp}(D_{old}(\omega), \sum_{probeRays} \max(0, \omega \cdot r) * l, \alpha) \quad (2)$$

where D is the distance value, ω is the probe's texel direction, r is a ray direction, and l is the rays length.

The hysteresis values close to 1 change the texture map very slowly, which improves stability at the cost of lower accuracy when objects move in the scene. On the other hand, values close to 0.9 (and lower) lead to rapid reactions to changes in the scene. However, it also leads to noticeable flickering. Furthermore, the flickering can occur even with the hysteresis parameter being close to 1. This is due to the low amount of sampling rays, which leads to rapid changes in irradiance maps if there is an exceptionally bright surface in the scene.

4 Results

We evaluated our implementation on multiple scenes in terms of performance and quality of produced images. This Section is structured as follows. We detail our quality comparisons in Section 4.1. The produced images by the DDGI method are compared to a reference image from a brute-force path-tracer. We also show a perceptual error between the reference image and the image produced by our implementation. The error was measured by Nvidia's FLIP algorithm [1] which shows the perceptual difference between two images. The perceptual error is shown as color values from perceptually uniform magma color map. Run-time and ray-tracing throughput performance evaluations are detailed in Section 4.2.

4.1 Qualitative results

In order to evaluate how the method handles shadow and light leaks, we prepared an outdoor scene with a closed Section with a door opening. The scene is enclosed in a 16x8x16 grid, where each probe has a resolution 8x8 and 16x16 distance map. The scene is then illuminated by a spotlight placed high in the scene simulating sunlight. Then we gradually applied visibility weights in order to negate said leaks, see Figure 7.

To see how the light propagates we created a closed scene with a ceiling opening from which the light floods the scene, see Figure 5. The scene is separated by a wall into two parts to create an occluded section in the scene. See how the DDGI method correctly creates a shadowed area in the corner of the scene occluded by the wall. The scene was discretized by a 16x16x15 probe grid, with 8x8 probe resolution and 16x16 distance map resolution.

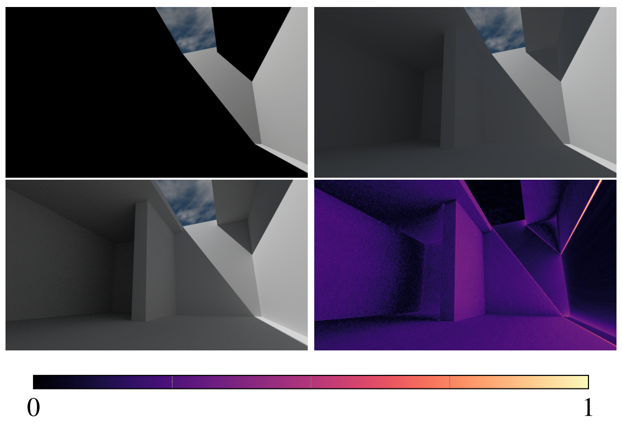

Figure 5: Images (left to right) show diffuse light propagation in a closed scene flooded by light. The first image shows the scene only with direct illumination. On the second image is global illumination produced by our implementation of DDGI. The third image is our path-traced ground truth. The last image depicts the perceptual error between the second and third images.

The color bleeding effect introduced by the diffuse light transfer is demonstrated in Figure 6. The scene is a closed box scene with two colored objects and a dynamically translating spotlight. We set the probe density to 2x2x2, where each probe had an 8x8 irradiance map and 16x16 distance map. Notice that the approximated color bleeding matches almost perfectly the ground truth.

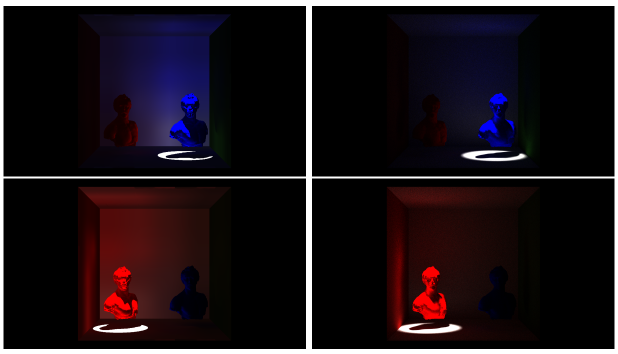

Figure 6: Images illustrate color bleeding effect comparison in a closed scene between the DDGI method (left) and path-traced reference (right).

Figures 8 and 9 show the DDGI method on scenes with more elaborate geometry. Notice the diffuse light on the left wall reflected from the floor in Figure 8. In Figure 9,

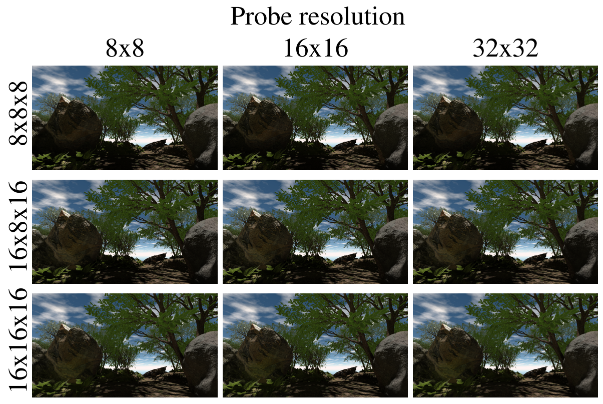

Table 1: Global illumination based on Probe density and resolution comparison. The table shows the quality of indirect illumination based on probe density and probe resolution in a forest scene.

we can see noticeable light leaks on the leaves of the tree. This is due to the dense foliage geometry, which would require a much denser irradiance field. Also, notice how the scenes tend to be overilluminated by the indirect illumination produced by the method.

Lastly Table 1 demonstrates quality comparisons across multiple probe densities and probe resolution selections.

4.2 Performance results

We measured the performance on the following machine. OS: Windows 10; Processor: AMD Ryzen 2700X, 4100 MHz, 8 cores, 16 Logical processors; RAM: 32 GB; GPU: RTX 3080. Tables 2 and 4 show the ray-tracing throughput of the irradiance field update based on the number of rays per probe and probe density. We also include times of each step of the irradiance field update in Table 3.

Rays	16x8x16	32x8x32	32x16x32	32x32x32
32	15.1	57.4	101.1	155.2
64	29.5	103.8	160.2	216.8
128	56.8	161.7	225.1	277.3
256	100.7	224.1	283.8	316.1

Table 2: Ray-trace throughput [MRays/s] of the irradiance field update with respect to probe density and number of rays per probe. The throughput was measured on the Sibenik scene (72862 triangles), where we set the probe resolution to 8x8.

5 Discussion

As mentioned in the methods overview and results the method has some drawbacks which we further discuss in Section 5.1. Two possible extensions to improve the performance of the method are discussed in Section 5.2.

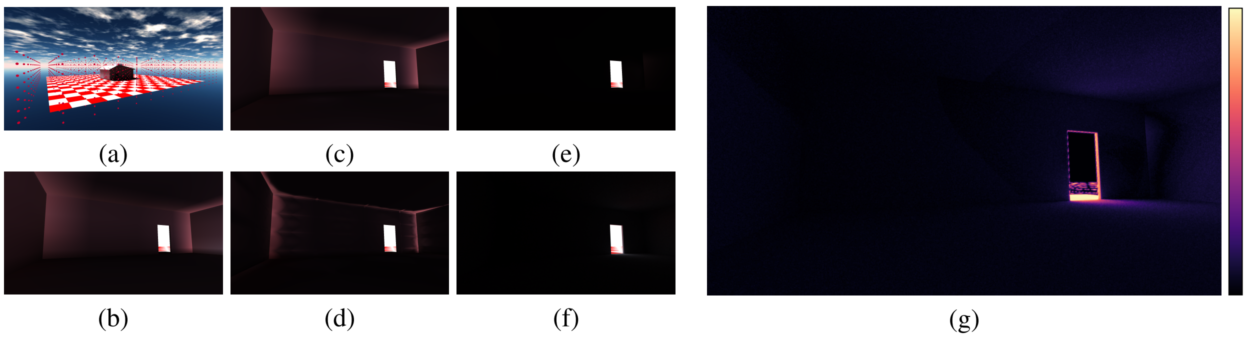

Figure 7: Irradiance visibility sampling comparison. Scene (a) is a closed room, where light enters through a small door opening. Images (b) to (e) show the interior view of the room, illustrating the gradual reduction of light-leaking by added sampling weights. On the image (b) are no weights applied, (c) backface weight, (d) Chebishew weight, (e) normal bias. Image (f) shows our path-traced reference, and (g) illustrates the perceptual error between (e) and (f).

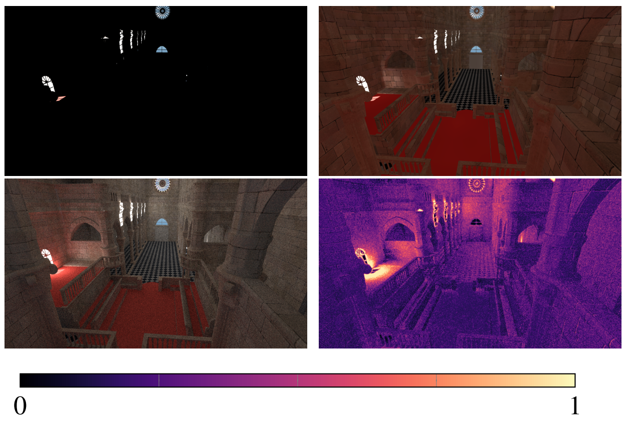

Figure 8: Global illumination quality comparison. Images (left to right) show the Sibenik scene, where the first image shows the scene with only the direct light contribution. The second image shows global illumination produced by DDGI, the third is our reference, and the fourth is the perceptual error between DDGI and the reference image. We set the grid resolution to 16x8x12 with 16x16 resolution for irradiance and distance maps.

Irradiance field pass	time[ms]
Ray generation	0.03
Ray cast	1.88
Indirect shade	0.22
Direct shade	0.18
Probe irradiance update	0.22
Probe depth update	0.46

Table 3: Table of timings [ms] of the irradiance field update in a single frame. The timings were taken on the Sibenik scene, where we placed 32x8x32 probes with 8x8 probe resolution.

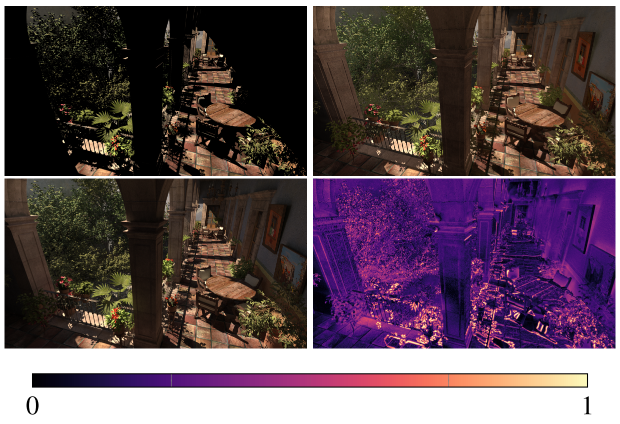

Figure 9: Global illumination quality comparison. Images (left to right) show the San Miguel scene, where the first image shows the scene with only the direct light contribution. The second image shows global illumination produced by DDGI, the third is our reference, and the fourth is the perceptual error between DDGI and the reference image. The grid density was 16x8x15 with 64x64 probe resolution and 16x16 distance maps.

Rays	16x8x16	32x8x32	32x16x32	32x32x32
32	12.3	42.3	68.2	94.2
64	23.8	68.9	96.0	119.2
128	42.1	98.4	123.7	140.9
256	68.6	125.8	146.3	156.9

Table 4: Ray-trace throughput [MRays/s] of the irradiance field update with respect to probe density and number of rays per probe. The throughput was measured on the Forest scene, our most complex scene (21.6M triangles). The probe resolution was set to 8x8.

5.1 Drawbacks

As mentioned in Section 3.5 the method can suffer from noticeable flickering. This problem can be either caused by a low hysteresis parameter or an insufficient amount of sampling rays in a scene with bright objects. Unfortunately, the only problem that can be directly managed by the method is the low amount of sampling rays and that is by increasing the amount of rays per probe. Unfortunately this could potentially lead to infeasible amounts of rays cast in each frame.

Another drawback of the method is a noticeable illumination delay caused by the recurrent computation of indirect light between frames. This problem is most noticeable in static scenes, however, it is not really noticeable in dynamic scenes and can be compensated for by lowering the hysteresis parameter.

As was shown in Figures 8 and 9 the method can struggle with light leaks in scenes with dense geometry, such as foliage, and can suffer from over-illumination. These problems however can be easily resolved by the use of ambient occlusion methods such as SSAO. This would resolve the light leaks on dense geometry such as foliage and dim the over-illuminated areas.

5.2 Proposed extensions

Even though the approximation of diffuse transfer produces images that are sometimes almost indistinguishable from the ground truth, the probe update in each frame is still relatively costly. This is because we update every probe in each frame with an initially set ray budget, which leads to an infeasible amount of ray casts and slow probe map updates. An adaptive approach would be more appealing, where we would change the budget of each probe based on its surroundings. A probe with a low contribution to the overall global illumination, such as probes placed in geometry or dark areas, does not need the same ray budget as a probe in a well-illuminated area.

We could take inspiration from the paper of K. Vardis et al. [18], which describes an illumination driven technique to optimize automatic probe placement methods for light baking, e.g., uniform 3D grids or tetrahedral grids. More specifically, their use of YCoCg color space to determine which probes to disable. To adaptively reduce the number of rays a probe emits, we would compute the cost based on absolute percentage errors, e.g., SMAPE. Since the DDGI method already computes illumination between frames we would compute the errors between the current and previous frame. Probes with low error, or probes placed inside geometry, would have their budget gradually reduced to a set minimum.

Another possible extension would be to use more elaborate probe placement, which would reduce artifacts caused by probes inside geometry and increase performance by reducing the required probes in the scene. However, this approach would possibly lessen the dynamic aspect of the method. Inspiration could be taken from the paper by Wang et al. [19] or Sedlacek's approach [15] for placing sparse radiance probes [16]. The solution by Sedlacek uses a voxelization of a scene together with a few simple rules to avoid placing irradiance probes inside geometry and to avoid probe overlap. However, as already mentioned, this approach might limit the method to static, or at least partially dynamic scenes due to the costly voxelization of the scene.

6 Conclusions

We presented our implementation of one of the recent methods for approximating global illumination for completely dynamic scenes. We compared the effects produced by the algorithm to the ground truth and evaluated the method's performance on multiple scenes. We also discussed the drawbacks of the method and how they are managed by the method or how they could be compensated for by other methods. Lastly, we proposed possible extensions such as adaptive scaling of probes' ray budget and dynamic probe placement tactic to increase the performance of the method.

References

[1] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D Fairchild. Flip: A difference evaluator for alternating images. *Proc. ACM Comput. Graph. Interact. Tech.*, 3(2):15–1, 2020.

[2] Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (TOG)*, 39(4):148–1, 2020.

[3] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 161–165, 2006.

[4] Cindy M Goral, Kenneth E Torrance, Donald P Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *ACM SIGGRAPH computer graphics*, 18(3):213–222, 1984.

[5] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 197–206, 1991.

[6] David S Immel, Michael F Cohen, and Donald P Greenberg. A radiosity method for non-diffuse environments. *Acm Siggraph Computer Graphics*, 20(4):133–142, 1986.

[7] Henrik Wann Jensen. Global illumination using photon maps. In *Eurographics workshop on Rendering techniques*, pages 21–30. Springer, 1996.

[8] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer*

graphics and interactive techniques, pages 143–150, 1986.

[9] Benjamin Keinert, Matthias Innmann, Michael Sänger, and Marc Stamminger. Spherical fibonacci mapping. *ACM Transactions on Graphics (TOG)*, 34(6):1–7, 2015.

[10] Vincent CH Ma and Michael D McCool. Low latency photon mapping using block hashing. In *Graphics Hardware*, pages 89–98. Citeseer, 2002.

[11] Zander Majercik, Jean-Philippe Guertin, Derek Nowrouzezahrai, and Morgan McGuire. Dynamic diffuse global illumination with ray-traced irradiance fields. *Journal of Computer Graphics Techniques Vol*, 8(2), 2019.

[12] Zander Majercik, Thomas Mueller, Alexander Keller, Derek Nowrouzezahrai, and Morgan McGuire. Dynamic diffuse global illumination resampling. In *ACM SIGGRAPH 2021 Talks*, pages 1–2. 2021.

[13] Timothy J Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *ACM SIGGRAPH 2005 Courses*, pages 258–es. 2005.

[14] Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. The state of the art in interactive global illumination. In *Computer graphics forum*, volume 31, pages 160–188. Wiley Online Library, 2012.

[15] Šimon Sedláček. Real-time global illumination using irradiance probes. 2019.

[16] Ari Silvennoinen and Jaakko Lehtinen. Real-time global illumination by precomputed local reconstruction from sparse radiance probes. *ACM Transactions on Graphics (TOG)*, 36(6):1–13, 2017.

[17] Peter-Pike Sloan, Derek Nowrouzezahrai, and Hong Yuan. Wrap shading. *Journal of Graphics, GPU, and Game Tools*, 15(4):252–259, 2011.

[18] Konstantinos Vardis, Andreas Alexandros Vasilakis, and Georgios Papaioannou. Illumination-driven light probe placement. 2021.

[19] Yue Wang, Soufiane Khiat, Paul G Kry, and Derek Nowrouzezahrai. Fast non-uniform radiance probe placement and tracing. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 1–9, 2019.