Improving Probes in Dynamic Diffuse Global Illumination

Dominik Roháček*
Supervised by: Tomáš Iser†

Computer Graphics and Game Development
Department of Software and Computer Science Education
Charles University
Prague / Czech republic

Abstract

For a long time, real-time renderers typically only supported direct illumination. With recent technological advances, such as much faster GPU computations or the RTX platform, simulating more accurate global illumination in real time is now possible. This is especially important for rendering indoor scenes in the context of architectural visualization as users can now add and modify illuminants in real time without waiting for a fully path-traced render.

In this paper, we briefly describe the existing real-time solutions and investigate the Dynamic Diffuse Global Illumination technique in detail. We implement the solution to an existing real-time renderer with RTX support. We specifically describe several problems and artefacts that the method has and present our solutions to those. Mainly, we show an improved approach to probe placing and investigate the improvement it provides. We note that our implementation achieves a better visual quality as it avoids highly noticeable artefacts.

Keywords: global illumination, real-time rendering, indirect lighting

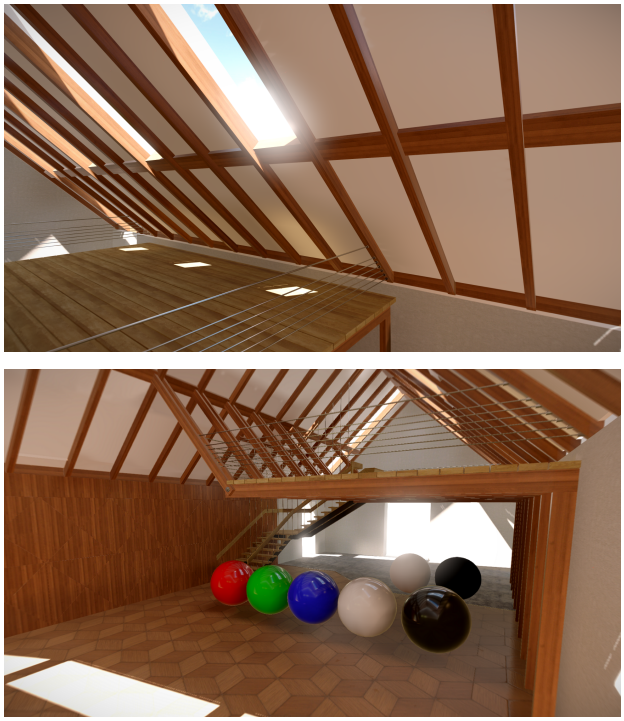


Figure 1: Examples of scenes rendered by our approach. Both scenes had visible artifacts with default probe placement.

1 Introduction

In real-time computer graphics, one of the major contemporary topics is simulating global illumination, which adds much needed details and realism into real-time scenes. Unfortunately, techniques that we rely on in offline rendering are not feasible or fast enough to employ in real time. The goal is to synthesize as physically plausible and immersive images as possible while still retaining the frame rate needed for a smooth user experience.

In this paper we present our updates to the Dynamic Diffuse Global Illumination (DDGI) (Section 2) method, results can be seen in Figure 1. In Section 3, we describe our suggested improvements for DDGI, mainly a way for improving the probe placement, which results in a higher visual quality by avoiding highly noticeable artefacts. This section also contains short implementation explanation of details. Results of our improvements are then presented in Section 4, where we show that the performance overhead of our additional steps is very low, and we also show comparison images to the original method. Finally, we dedicate Section 5 to the conclusion and discussion on possible future improvements we are considering to our work.

2 Previous work

In this section, we briefly introduce the previous work on global illumination techniques that had appeared in the last years. A new platform by NVidia called RTX allowed us to bring ray tracing approaches into the real time. This

*RohacekD@gmail.com
†tomas@cgg.mff.cuni.cz

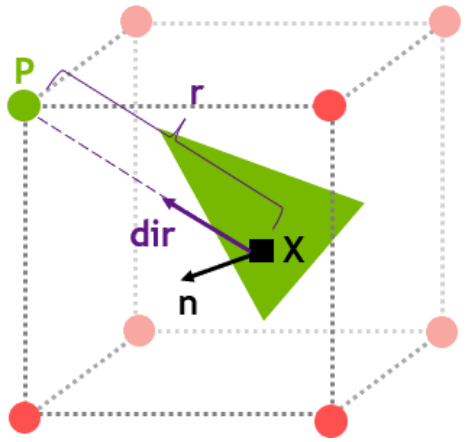


Figure 2: Shading of a fragment X. This fragment has a normal facing in the direction n, and we can compute a direction dir to each probe P belonging to this cage and distance to this probe P denoted by r. (Image source: [8])

opened new ways to calculate indirect lighting in real-time renderers.

2.1 Dynamic Diffuse Global Illumination (DDGI)

DDGI [8] is based on a uniform grid of irradiance probes. The technique heavily utilises the RTX technology by NVidia in order to dynamically update probe data in each frame. Each probe stores its irradiance and depth data in small textures inside a texture atlas. To minimize the memory footprint of the technique, the original paper [8] proposes storing irradiance and depth data in very small textures of just 8x8 and 16x16 pixels respectively.

To capture geometry detail in such a small depth map, variance shadow maps [4] are used to store depth information, and later Chebyshev inequality [1] is used to determine occlusion.

As we rely on the ability to update probes, we need some representation that is easily updated. Unfortunately, spherical harmonics that are usually used in irradiance probes [9] are not easily updatable. Instead, the we use octahedral mapping of a UV space onto a sphere [2]. The mapping function is simple and fast. All the textures are stored inside two texture atlases [7] — one for depth and one for irradiance textures.

A final sampling of indirect lighting is done in the fragment shader. Each fragment lies within the probe cages defined by eight probes forming cube as seen in Figure 2. We iterate over all eight probes and evaluate each probe's contribution to the indirect lighting of a given fragment.

First part of the contribution weight is backface culling. The original paper [8] proposes smooth backface culling with an offset added to reduce the "going to zero" impact as shown in Figure 3. However, we have not found this useful, so we completely avoided smooth backface culling and used only a dot product between the surface normal and the direction to the probe, with negative values clamped to zero. The problem is that the offset tends to cause light bleeding through geometry.

The next part of the contribution weight is a probability of a pixel being occluded from the probe, which is calculated by Chebyshev inequality [1].

Finally, trilinear filtering is used as the final weight. This provides us with a smooth transition between each probe cage without visible artifacts.

2.2 Signed Distance Fields Dynamic Diffuse Global Illumination (SDFGI)

SDFGI [6] is based on Signed Distance Field (SDF) that are constructed from scene geometry and later used during the ray tracing phase. This technique is using probes to cache the irradiance data and update them each frame. SDF is also used as a way to update probe position when geometry moves.

Unfortunately, the build of SDF on generic geometry is expensive and dynamic geometry requires rebuilding every time the geometry changes.

2.3 Global Illumination Based on Surfels (GIBS)

This technique by Stachowiak [10] and further improved by Halen et al. [5] is based on so-called surfels. Surfels discretizes surface adaptively to screen space coverage. Surfels are used to cache irradiance in the scene. Each surfel holds irradiance distribution in a hemisphere around the surface normal and depth information. The depth information is stored similarly as in DDGI in form of variance shadow maps [4].

The technique is trying to optimise surfels density to uniformly cover screen space. This leads to finer coverage of the surface near the camera and lower detail in the distance.

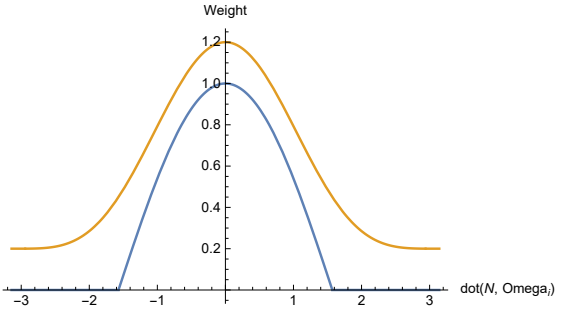


Figure 3: A comparison between smooth (orange) and sharp (blue) backface culling.

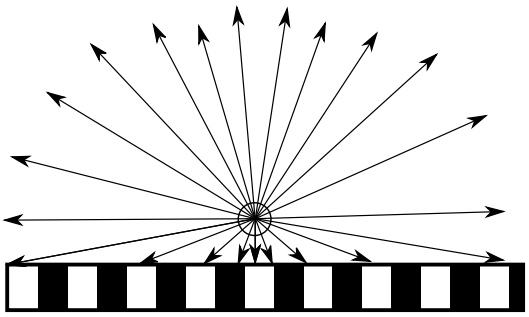


Figure 4: When the probe is placed right next to the geometry, a large number of rays ends up covering only a small geometry surface. Increasing the distance of the probe results in a better texture utilization.

3 Our work

The DDGI method is grid-based, which brings many advantages but also disadvantages. One of the inherent problems that grid-based methods introduce is the level of control over the placement of the probes. Mainly, it is clear that all the probes that happen to be inside a geometry have to be ignored. Moreover, because the grid is uniform, the probability of certain probes being placed in a bad position is very high, especially in architectural scenes with many flat surfaces, often even aligned along coordinate axes.

Ideally, we would like to find a place where each probe is separated from its nearest surface at least by an user-defined threshold. Moving the probe only to the surface could not only lead to a depth fight at a given place, but would also waste the already small texture space we have available for each probe. The reason is that a probe placed too close to the surface uses almost half of its texture space to cover only a small part of geometry (shown in Figure 4) and leads to different artifacts.

3.1 Dead probe detection

In order to minimize calculations needed to detect dead probes, we re-use information that we already have from the ray-tracing step in the DDGI method. We introduce a new step in which we count the number of rays that were shot in the last frame and that encountered the backface of any geometry as a first event. Now, in an ideal world, where we would be able to make such constrain as forcing all the geometry to be manifold and not intersecting other geometry, we would simply check one ray and would be able to tell whether the whole probe is submerged inside geometry. Unfortunately, in the real-world 3D production and level design, geometry placed within another geometry is often present and our algorithm needs to be robust enough to handle it.

So instead, we mark back-face hits in the ray tracer with a negative distance. In a new pipeline step, we are counting rays for each probe that hit geometry in a distance less than a user-defined threshold. When the count exceeds half of the number of total rays, we consider the probe being "dead".

We use this result to update the texture that marks "dead probes". We update this texture with an adjustable hysteresis. The hysteresis is used to define the amount of certainty we want to remain from the previous frame. The hysteresis in this case avoids the too aggressive movement of probes.

3.2 Moving probes

Once we have our "dead probes" marked, we need to move them from their current position. There are two approaches to this problem — geometry-aware and unaware. We have decided to use the approach that is not aware of the probe surroundings as we want to promote responsivity of the algorithm. However, exploring also the geometry-aware approach could be subject of future work.

We propose to use a spiral pattern that extends away from the probe's original position. This is done through a new pipeline step that increases an integer counter for a given probe whenever it is detected as being dead and the resulting texture from the previous subsection exceeds the user-defined threshold. The threshold is defined as a percentage of the probe cage side and has to be lower than 50%.

This counter corresponds to a position on a spiral. Points on the spiral are defined only on cardinal axes with origin in the original probe position for simplicity. The way in which we translate the counter value to a spiral position follows: We take a modulo and integral division by six. Six here stands for six directions that we want the probe to move in. The result of the modulo operation chooses the direction, and the division result defines the distance from the original position.

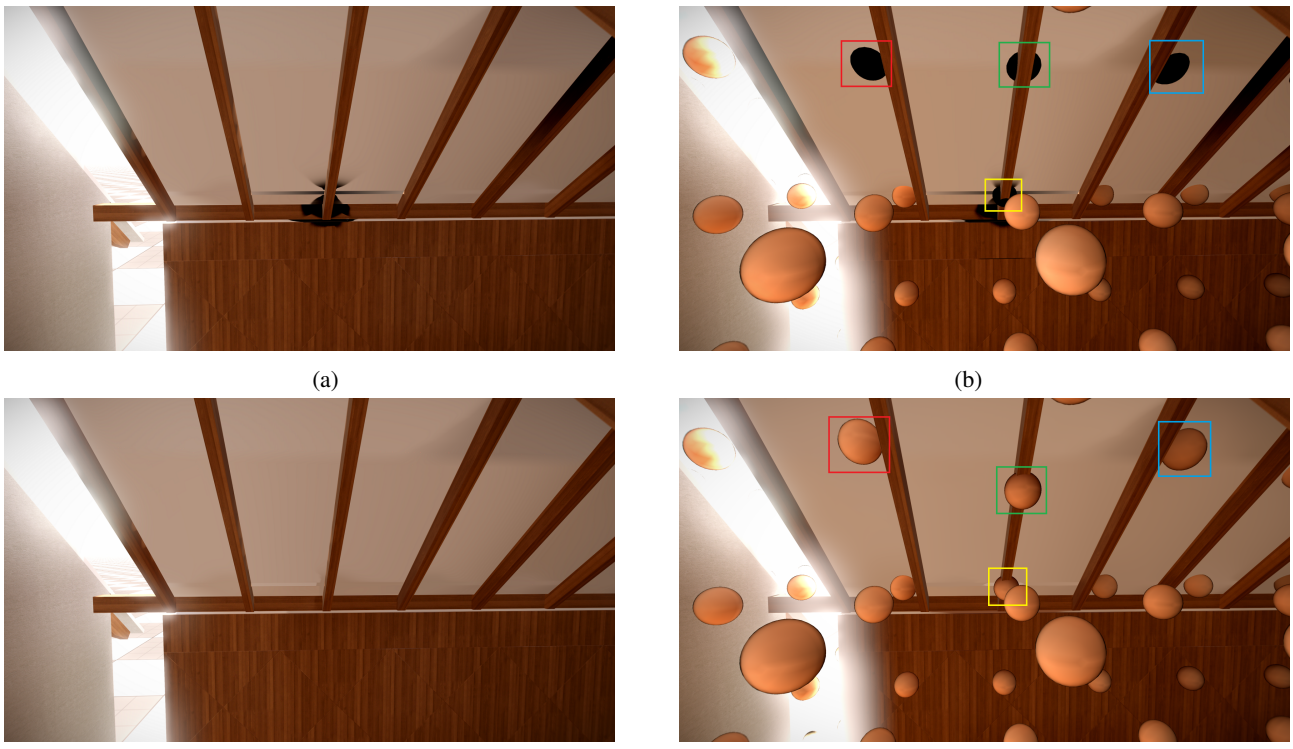
Figure 5 shows the result of the probe movement. We can see that four probes have been moved away from "dead" positions inside the gometry, and as a result, the visual quality of the image has significantly improved.

3.3 Correct filtering

By moving probes from their original uniform grid, trilinear filtering is no longer valid. The probe after the move does not satisfy an important condition. In order to achieve correct results from the indirect light sampling, the sum of weights need to be one, and each weight needs to remain on the $< 0, 1 >$ interval.

Violating the first invariant leads to visible boundaries of probe cages in a final image. On the other hand, when we violate the second one, it can lead to oversaturation of part of the cage or subtraction of light by some probe as shown in Figure 6.

Instead, we propose filtering that accounts on probe offsets. Similarly to the original paper [8], we first calculate fragment coordinates normalized into the probe cage space



(a) (b)

(c) (d)

Figure 5: An example of visual artefacts and probe positioning with the original DDGI method (subfigures a, b) and our modified approach (subfigures c, d). The orange and black spheres in subfigures b, d indicate correctly working and "dead" probes, respectively. Notice how the four previously dead probes in subfigure b were automatically moved to a better location in subfigure d. An error visualisation of the same view is shown later in Figure 9.

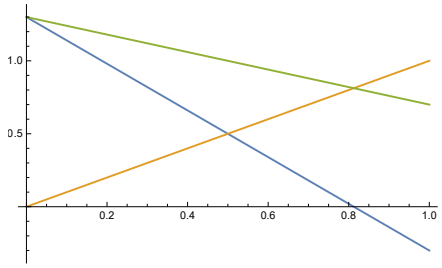
defined as $(x,y,z) \in < 0,1 >^3$ as shown in Equation 1. Then we calculate the current probe position within the cage with the account to the probes offset in Equation 2. In the last step, we use linear interpolation which gives us trilinear weights (Equation 3) and to get the final weight we simply need to multiply individual axial weights in Equation 4.

$$\text{fragNorm} = \frac{\text{withinCageCoord}}{\text{probeCageSize}} \tag{1}$$

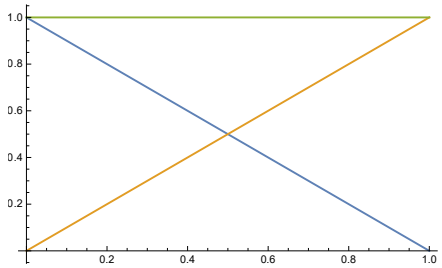
$$\text{offset} = \frac{\text{probeCoordNorm} - \text{probeOffsetNorm}}{(1,1,1) - \text{probeOffsetNorm}} \tag{2}$$

$$\text{trilinear} = \text{lerp}(1 - \text{fragNorm}, \text{fragNorm}, \text{offset}) \tag{3}$$

$$\text{weight} = \text{trilinear.x} * \text{trilinear.y} * \text{trilinear.z} \tag{4}$$



(a) Original filtering

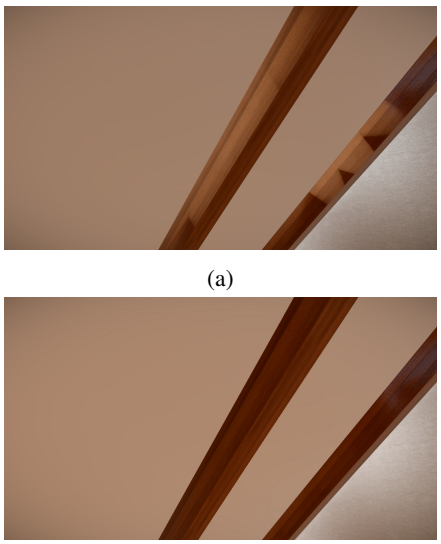


(b) Our filtering

Figure 6: Graphs show a simplified situation with only two probes on a straight line. The blue line shows the weight of the left probe, the orange line corresponds with the right probe and the green line is a sum of both.

3.4 Depth samples rejection

During the implementation, we have faced multiple additional artifacts caused by light leaking through the geometry as shown in Figure 7. The source of such artifacts



(a)

(b)

Figure 7: (a) Example of light bleeding through the roof of a house. The roof is being highly lit by the sun. The depth samples are skewed by depth samples from directions almost parallel to the roof. (b) The second image shows the same view with depth sample rejection.

is better explained in Figure 8. Because DDGI uses such a small resolution for depth maps, depth information is filtered into a sphere. It is done in a similar way as we integrate irradiance from incoming radiance with the difference of not filtering depth into the full hemisphere. A sharpening term is used in the distribution of depth samples. This can cause samples originally from neighbouring pixels to carry a local extreme in depth into its surroundings.

To avoid such behaviour, we have decided to discard any depth samples that are longer than the main diagonal of the probe cage. We can do this modification without loss of generality as we never check the distance from the fragment to the probe that lies outside the probe cage itself.

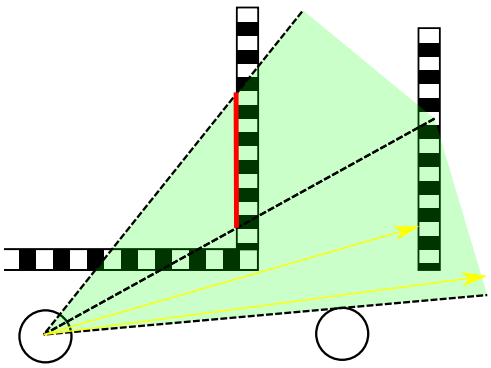


Figure 8: The green area represents projection of the depth map pixel into the world. Yellow lines represent depth samples. The red wall could be lit by the left probe when not using depth sample rejection.

With our dead probe detection and movement, we have to extend the threshold for the sample discard to the maximum size of the probe cage. We need to enlarge the size of the cage by the twice percentage threshold of the maximal probe movement.

3.5 Implementation

We implemented DDGI in an existing production real-time engine called Fibix[1], which also contains an implementation of an offline path tracer. This allowed us to also measure our solution in comparison to reference path-traced images. This way, we were also able to try different scenes provided by the Fibix Studio and verify that our implementation can be used with different geometries without bigger difficulties.

In our implementation, we use the DXR pipeline to gather radiance information from the scene. Afterwards we integrate irradiance into the probe texture through series of compute shaders and use same data for probe offseting. Finally, we sample results in the final fragment shader as we described in subsection 2.1.

4 Results

The main goal of our improved technique was to avoid highly noticable and visually unpleasant artefacts, or in other words, to make sure that any error is not concentrated in a small area, but is better spread across a frame. To show that we have achieved the goal, we are presenting error heatmaps in Figure 9. The view presented in this figure is identical to the one shown in Figure 5. It is clear that the most visible hotspot has disappeared with probes moved out of the geometry to the surface. With probes pushed even further to the 15 cm distance from the surface, the maximum error is brought down. With the improved filtering, the isle of error is barely noticable.

Table 1 shows time for each step of the DDGI pipeline measured as average over multiple frames. You can see that the main part of the total time is spent in the RTX pipeline and our addition to the pipeline takes only about 0.15 ms. Such a small difference in frame time in comparison to the improvement of visual quality is easily justifiable. The table was measured with 64 rays per probe per frame. It is needless to say that the number of samples had a negligible impact on the render pass length. Also, we were not able to measure any change in the duration of those steps with an increasing resolution of depth maps, which is understandable as the size of computing space for those shaders is only dependent on the number of rays.

[1] https://www.fibix.eu/

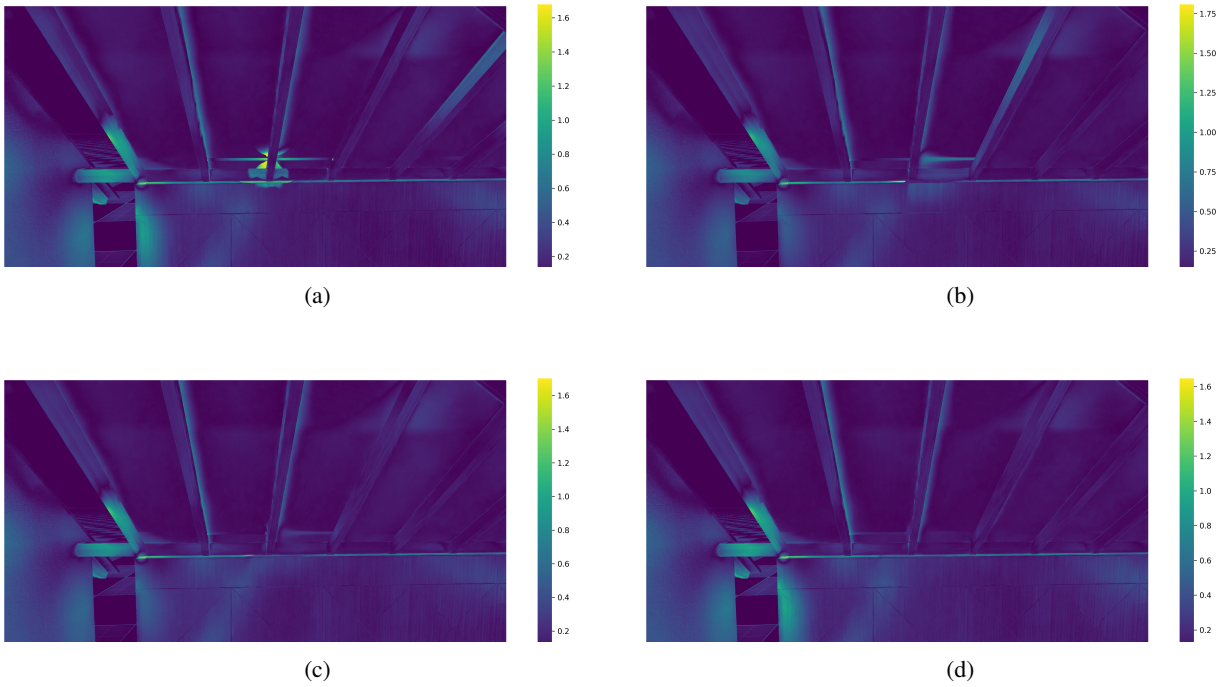


Figure 9: Heatmaps show the spread of the error across the image. The same view as in Figure 5 is shown. Please note the scale of each subfigure. (a) Original implementation of DDGI algorithm. (b) All probes offset to the surface. (c) All probes offset at least 15 cm from nearest surface. (d) Added correct trilinear filtering described in subsection 3.3.

Render pass	Time (ms)
Ray cast (RTX)	2.71
Probe update (irradiance and depth)	0.02
Dead probe detection	0.05
Probe offset	0.10
Indirect light sampling	0.48
Total	3.36

Table 1: Timings for separate passes of DDGI. For these measurements, we used 11x10x15 probes. Each probe produced 64 rays per frame. The render passes written in bold are ones we have added.

5 Discussion

5.1 Comparison to Other Solutions

In comparison to the original DDGI, we have shown that we can avoid situations with most or all probes of a given cage submerged into the scene geometry. This problem lead to dark sections of the scene and broke the immersion of the user into the virtual world we were trying to model. In the parts of the scene, where we experienced the most noticable artifacts, we were able to spread the error more evenly across the space and we lowered the visibility of such errors.

In comparison to the Global Illumination Based on Surfels (GIBS) [5], our solution has problems with specific scenes where GIBS has better strategies of cache placements. On the other hand, DDGI is simpler for implementation and its integration into the existing engine has fewer requirements.

The solution presented by **Hu et al.** [6] places probes in better places due to geometry aware approach to probe offsetting. On the other hand, this approach requires the construction of Signed Distance Fields that are hard to construct in runtime for dynamic geometry.

5.2 Limitations and Future Work

Probe cascades Even in our improved implementation, the probe grid space is still relatively uniform. This is convenient for world-space to grid-space normalized mapping. But it also means that for huge probe grids, we have just the same level of detail for irradiance even a few kilometres away from the camera, which may not be a good use of resources. The possible future improvement would be probe cascades, which comes from the same idea as cascade shadow maps [3] that we need finer detail in the immediate surrounding of the camera but not further in the scene.

Ray budgeting Right now, our solution uses a uniform distribution of a ray budget for each frame between

probes. This works well, but we could do better. We can come up with some heuristics for rays distribution. One solution could be manually feeding data about changes in the scene from the engine code. A better solution could be to think a little about where we want finer sampling. Those are such parts of a scene that encountered the biggest change of lighting in the previous frames. We can use a short term mean-variance estimator of the irradiance and increase ray budgeting accordingly as proposed by [5].

Dynamic hysteresis We can also change the speed of updates by using a more informed way to set hysteresis. Right now, we use a user-defined constant hysteresis. We could use the same approach as with ray budgeting and change the hysteresis based on a variance of a short term mean-variance estimator.

5.3 Conclusion

We have presented an adaptive approach to probe positioning for DDGI that is suitable for an improved global illumination of dynamic scenes. We have shown better error distribution across the scene and lower visual artifacts from the synthetized images. We have verified that our approach can be implemented into an existing engine and mainly enables architectural users or level designers to iterate over the level design faster.

References

[1] Pafnutii Lvovich Chebyshev. Des valeurs moyennes. *J. Math. Pures Appl*, 12(2):177–184, 1867.

[2] Zina H. Cigolle, Sam Donow, Daniel Evangelakos, Michael Mara, Morgan McGuire, and Quirin Meyer. A survey of efficient representations for independent unit vectors. *Journal of Computer Graphics Techniques (JCGT)*, 3(2):1–30, April 2014.

[3] Rouslan Dimitrov. Cascaded shadow maps. *Developer Documentation, NVIDIA Corp*, 2007.

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

[5] Henrik Halen, Andreas Brinck, Kyle Hayward, and Xiangshun Bei. Siggraph 21: Global illumination based on surfels. *SIGGRAPH course*, 2021.

[6] Jinkai Hu, Milo K. Yip, Guillermo Elias Alonso, Shihao Gu, Xiangjun Tang, and Xiaogang Jin. Signed distance fields dynamic diffuse global illumination. *CoRR*, abs/2007.14394, 2020.

[7] Jérôme Maillot, Hussein Yahia, and Anne Verroust. Interactive texture mapping. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 27–34, 1993.

[8] Zander Majercik, Jean-Philippe Guertin, Derek Nowrouzezahrai, and Morgan McGuire. Dynamic diffuse global illumination with ray-traced irradiance fields. *Journal of Computer Graphics Techniques (JCGT)*, 8(2):1–30, June 2019.

[9] Ravi Ramamoorthi and Pat Hanrahan. An efficient representation for irradiance environment maps. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 497–500, New York, NY, USA, 2001. Association for Computing Machinery.

[10] Tomasz Stachowiak. Stochastic all the things: Raytracing in hybrid real-time rendering. *SEED, Digital Dragons*, 2018.