

# Signed Distance Fields Dynamic Diffuse Global Illumination

Jinkai Hu Milo Yip G. Elias Alonso  
Shihao Gu Xiangjun Tang Xiaogang Jin

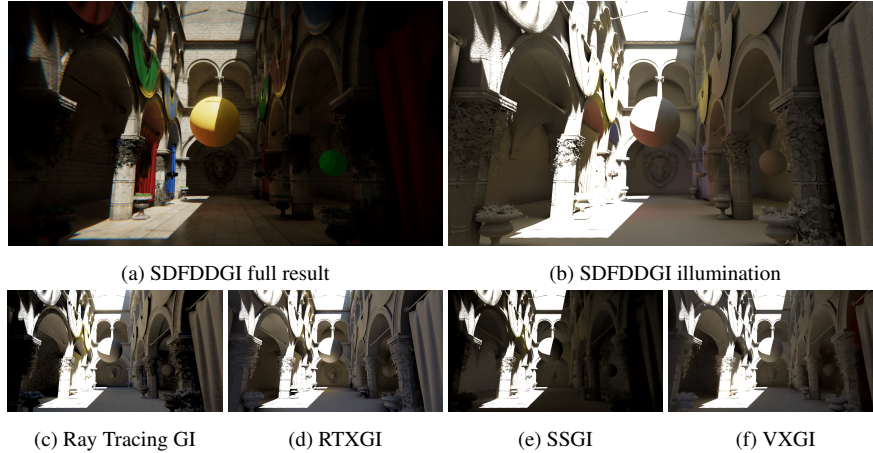


Figure 1: The figure above provides an example of our SDFDDGI on the upper row (a)(b) as well as a comparison to other real-time GI methods in the row below. Our approach achieved less than 5 ms per frame on GTX 970M hardware with the lowest acceptable quality, while on RTX 2080Ti even achieved a performance within 1 ms per frame. Besides, it solves other method’s deficiencies such as real-time Ray Tracing’s extra noise and lack of multi-bounce lighting (a), light leaking issues on the dynamic yellow sphere on RTXGI (b), mere use of screen space information in Screen Space GI (c), and rougher detailing on Voxel-Based GI (d).

## Abstract

*Global Illumination (GI) is of utmost importance in the field of photo-realistic rendering. However, its computation has always been very complex, especially diffuse GI. State of the art real-time GI methods have limitations of different nature, such as light leaking, performance issues, special hardware requirements, noise corruption, bounce number limitations, among others. To overcome these limitations, we propose a novel approach of computing dynamic diffuse GI with a signed distance fields approximation of the scene and discretizing the space domain of the irradiance function. With this approach, we are able to estimate real-time diffuse GI for dynamic lighting and geometry, without any precomputations and supporting multi-bounce GI, providing good quality lighting and high performance at the same time. Our algorithm is also able to achieve better scalability, and manage both large open scenes and indoor high-detailed scenes without being corrupted by noise.*

## CCS Concepts

• *Computing methodologies* → *Real-Time Rendering; Global Illumination; Signed Distance Fields;*

## 1. Introduction

Global illumination allows us to improve strikingly the realism of a virtual scene. However, its real-time computational cost is far too expensive for most applications, such as in the video game industry.

To solve this problem, many approaches have been proposed. For example, baked Light Maps [McT04] allow us to compute global illumination for static lights and scenes by baking the lighting of the scene. Precomputed Radiance Transfer [SKS02] solved their limitations for dynamic light sources. [SSS\*20] proposed an ex-

tension for the baking system of Light Maps that supports dynamic changes. However, all these approaches pose limitations on the dynamic changes, thus affecting the artistic design of the scenes. Besides, precomputations add an extra production cost.

Of course, there also exist some methods that can manage completely dynamic GI, for example: Voxel-Based GI [CNS\*11], Ray Tracing GI [OS19], Screen Space GI [RGS09] [ST15] [SKS11], RTXGI [MGNM19] and so on. These techniques fill the gap left by the area of real-time global illumination, making it possible to use

GI for real-time applications, like in the video game industry. However, these methods have limitations of different nature, such as light leaking, performance issues, special hardware requirements, noise corruption, bounce number limitations and lack of scene information, among others.

Inspired by RTXGI [MGNM19], we propose SDFDDGI (Signed-Distance-Field Dynamic Diffuse Global Illumination). Compared to RTXGI, our approach doesn't need the use of specific hardware, made improvements in its performance and result, increased dynamic response speed and completely solved light leaking problems.

This method uses Signed Distance Fields (SDF) [Har96] to make a slim representation of the scene. We sample the irradiance function of the space domain and interpolate these samples to estimate its global illumination. To improve the resemblance of the original irradiance function and its discrete form, we employ SDF to lead the distribution of sampling points. Then, when interpolating samples, we use SDF for the visibility tests to prevent the occurrence of light leaking artifacts. We also use Contact GI to enhance details on SDFDDGI.

Our paper makes the following contributions:

- We use SDF primitives and clusters to compress the information in the scene, reduce data size, increase memory use and speed up computations.
- SDF allows us to query very fast the nearest distance and spatial gradient to optimize the position of discretized sampling points in order to better suit the spatial distribution of the irradiance function.
- SDF is able to compute soft shadows efficiently [Aal18], optimize the interpolation strategy of the discretized points, and completely avoid light leaking artifacts, as well as provide soft indirect shadowing effects.

## 2. Related work

Since the appearance of the rendering equation [Kaj86], global illumination has historically shaped the whole development of graphics theory.

$$L_o(P, D_o) = L_e(P, D_o) + \int_{\Omega} f_r(P, D_i, D_o) L_i(P, D_i) (n \cdot D_i) dD_i. \quad (1)$$

In the field of real-time photorealistic rendering, the speed and quality of the computation of global illumination has all along been a hot topic of research. In this section, we present a general overview of some of the most concerning GI techniques and research.

Virtual Point Lights (VPL) [LSK\*07] is one of the earliest real-time GI technique. Its main contribution is the substitution of GI computations by adding virtual point light sources at the areas illuminated by direct lighting. This approach has many limitations but its result is satisfactory for local indirect illumination of spotlight light sources or other narrow-ranged ones.

Reflective Shadow Maps (RSM) [DS05] is also an early method for real-time global illumination. It is based on the same idea as

shadow maps, not only store depth information but also store light sources direction and radiant flux as well and use this reflective shadow map as global illumination. However, it does not take into account indirect occlusion, thus producing severe indirect illumination misestimations in some scenes. It also only provides one-bounce GI and is not able to manage area lights and skylight.

RSM and VPL ultimately evolved into Light Propagation Volumes (LPV) [KD10]. LPV introduced the concept of volume in VPL and transfers illumination data across space. This method solved many of the problems of VPL and RSM, but light leaking remained a serious concern as well as some accuracy issues.

Apart from this, there are voxelization approaches [CNS\*11], which first voxelize the scene into a sparse voxel tree and then inject lighting data. This allows us to estimate global illumination at real-time frame rates but it also produces light leaking. Besides, in scenes with highly varying dynamic geometry, the computational cost of voxelization is too high.

High-end global illumination approaches like Ray Tracing GI [OS19] reconstruct world coordinates and normals out of G-buffers, then sample the hemisphere to compute global illumination. The number of computations of this kind of approaches is very large, so it is only viable for a small amount of samples and high-end hardware, thus needing a final denoising stage [SKW\*17] [KIM\*19] for an acceptable result. The main disadvantage of this method is its performance, being almost impossible to calculate multi-bounce illumination, apart from the cost of additional denoising.

At the moment of writing this, performance-wise the best choice is Screen Space GI, like screen space diffuse GI [RGS09] [ST15], as well as screen space specular reflections (SSR) [SKS11]. Comparatively speaking, necessary information for specular reflections normally resides inside screen space, while diffuse GI often lacks most needed lighting information, thus not being able to provide an optimal result.

Last year, NVIDIA proposed a new approach RTXGI [MGNM19] using its ray tracing accelerated hardware, by means of discretizing the spatial distribution of the irradiance function. Compared to common probe-based GI [McA15], its main contribution is the use of depth information and Variance Shadow Maps (VSM) [DL06] as well, in order to prevent light leaking artifacts that arise from the discretization of irradiance. However, its effect on GI of details at real-time frame rates is not optimal. Besides, light leaking artifacts can also appear with very thin objects and it depends severely on RTX-accelerated hardware, which affects its use extent.

On that basis, we realised that SDF [Har96] can be used to simplify the scene representation for low-frequency global illumination like diffuse GI. SDF is a scalar field in the space domain, which represents the distance from a point in space to the nearest surface in the scene. A positive value is assigned if the point is in the outer region of the nearest surface and negative if it is inside, thus producing a compact representation of the geometry information of a scene.

Inspired by RTXGI, we proposed a novel approach SDFDDGI,

which overcomes aforementioned limitations and has the following advantages:

- It does not need any precomputations.
- It can manage both dynamic geometry and dynamic lighting, as well as animations and skylight.
- It provides interframe stability and low delay response for dynamic changes.
- It completely eradicates light leaking problems.
- Our technique is not limited to specific hardware, it can also be used in lower-end hardware.

### 3. Approach

SDFDDGI has mainly 4 stages:

1. **SDF Cluster**: generate the scene’s SDF representation and its clustering acceleration structure.
2. **Probe choosing**: choose suitable sampling points in space.
3. **Probe update**: update the irradiance on a sphere of directions around the sampling points.
4. **Per pixel GI shading**: interpolate different probes to calculate global illumination for all pixels in the screen.

#### 3.1. SDF Cluster

SDF is usually stored in volume textures. However, with this approach we need to make a trade-off between the resolution and detail of the scene and the data size. Therefore, we describe our scene with a distance field composed of different SDF primitives, then parse the global SDF value in space instead of performing the voxelization of the scene. For example, we only need 4 KB to store an SDF representation of the Sponza Palace scene.

Our scene objects are represented by a series of basic SDF primitives. Each SDF primitive includes a primitive type and its respective transform. Example primitive types are rectangular blocks, planes, cylinders, or any other whose analytical form is simple enough to easily compute its SDF value. In every frame, we use CPU to perform culling on SDF primitives on the surroundings of the camera and perform Level Of Detail (LOD) for further away primitives, i.e. use less primitives to express a larger rougher shape. Afterwards, we generate a cluster structure to speed up the SDF query.

We use a clustering approach to pack near SDF primitives into a cluster, as shown is Figure 2. When performing a query, SDF primitives in a cluster will first be treated as one and be jointly rejected or not, what refrains us from querying every single SDF primitive, thus accelerating the process by 20% to 100% depending on the scene. This has proven to have a better performance than BVH [WBS07] for smaller scale data. Since the number of primitives is not much, we use the CPU to better distribute the strain of the GPU. Besides its time expenditure is negligible in comparison to the other stages.

In today’s hardware, memory access cost is far greater than the cost of computations. Unordered access of memory, which is characteristic of diffuse GI, has an even more negative impact on performance on cache. In ray tracing algorithms we often need to

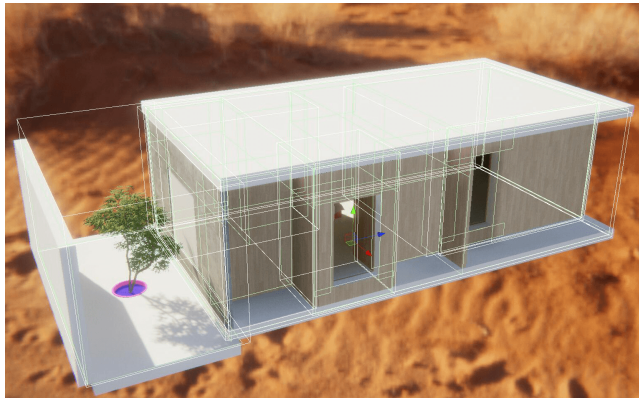


Figure 2: For the computation of the diffuse global illumination, the scene is represented as a set of different SDF primitives, marked as green lines in the example picture. Then, near SDF primitives are packed into a cluster, marked as white lines above, to accelerate the query of the scene SDF value at one point.

use Ray Binning [Ben19] and other techniques to improve cache hit ratio. However, in our case the primitives are relatively small, so much as to being able to put everything inside the L1 cache, making SDF query even more advantageous than reading volume textures. Moreover, this implies that we can easily support dynamic scenes, not like voxelization approaches that

At the same time, since the data we need is mainly on the L1 cache, global memory is available for use for other parts that have lower requirements on memory access. In this manner, processes that have high requirements on memory but are computationally inexpensive, like G-Buffer or Shadow Map generation, can run on the GPU, thus increasing GPU general utilization rate, which is an important factor on the general frame renderization time.

#### 3.2. Probe choosing

Since Irradiance is a  $\mathbb{R}^5$  function, we split its domain in two parts: the space coordinates  $\mathbb{R}^3$  and the direction  $\mathbb{R}^2$  as Equation 2.

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, D = \begin{bmatrix} \omega \\ \theta \end{bmatrix}, \quad (2)$$

$$Irradiance = E(P, D).$$

Our method discretizes the spatial domain of the irradiance function  $E$  into many sampling points  $P$ . Every sampling point is referred to as probe. Every probe stores the irradiance on a sphere of directions around its position. We create a spatially arranged probe volume around the camera and use these probes to interpolate global illumination.

Irradiance is not a continuous function in its spatial domain. For example, in walls or at occluded points there are evident discontinuities, which are the main cause of light leaking in most real-time global illumination algorithms. If we want an appropriate represen-

tation of the irradiance distribution in space, we need to carefully place the probes.

Because of this reason, we first calculate the SDF value at probe’s location. If it is smaller than a threshold value, this means it is too near to other objects or even inside one, what would negatively impact the quality of the sampling. Then, we query the gradient of SDF and use gradient descent method to obtain an acceptable sampling point near the original position. If the displacement between the position of last and current frame surpasses a threshold, the irradiance at last frame will need to be rejected, so we allocate more rays to this probe to ensure a more stable result. In order to decrease the number of probe updates, in each frame we

```

1: procedure UPDATEPROBEPOS
2:   lastPos ← position of probe in last frame
3:   pos ← position of probe in the uniform grid
4:   if querySdf(pos) < threshold1 then
5:     pos ← gradientDescent(SDF, pos)
6:   if distance(pos, lastPos) > threshold2 then
7:     markRejectHistory(probe)
8:   return pos

```

Figure 3: Pseudocode for the algorithm to find a suitable position for a probe. We move our probe from the grid if it is too near or inside objects and we check if the displacement of the probe from last frame’s position is too large to consider GI information of last frame.

first choose which probes need to be updated and divide the probe updates between different frames so as to increase performance.

Different weights are assigned to the probes according to their distance to the camera and direction to the camera in order to decide which probes should be updated.

Due to the probe updates, this method may cause jitter between frames. Nevertheless, this phenomenon is not as evident as in the original RTXGI since the performance of our approach allows us to perform a more extensive sampling, what partly mitigates the effect of jittering.

In contrast to RTXGI, our method does not need human intervention in order to have a better probe distribution and it can rapidly and accurately respond to scene changes. This also reduces the leaking caused by dynamic objects.

### 3.3. Probe update

To obtain the irradiance function  $E$ , radiance  $L$  needs to be computed first. We use compute shaders to sample the radiance over a sphere of directions at each probe:

$$E(P, D) = \int_{4\pi} \max(0, \cos(D, D')) \cdot L(P, D') dD'. \quad (3)$$

In this phase, we use a 8x8 thread block to update each chosen probe. Before sampling, all threads in a block need to cooperate to

move all or part of the SDF primitives into the L1 cache, since they will be queried frequently.

In order to speed up the tracing process, we employ clusters to perform culling on SDF primitives. Taking point  $A$  in Figure 4 as the starting point of tracing (a), the algorithm to query SDF can be described as follows:

1. Compute the smallest distance  $d1$  to the bounding box of the first cluster  $C1$ .
2. Since  $d1$  is smaller than initial distance  $D$  (initialized here as infinity), traverse all primitives in cluster  $C1$ , find the nearest distance  $d2$  to them and update distance  $D = \min(D, d2)$ .
3. Calculate the smallest distance  $d3$  to cluster  $C2$ .
4. Since  $d3$  is smaller than  $D$ , traverse all primitives inside cluster  $C2$ , find the nearest distance  $d4$  to them and update distance  $D = \min(D, d4)$ .
5. Since the distance to the bounding box of cluster  $C3$  is larger than  $D$ , we skip this cluster, so SDF query result is  $D = d4$ .

This algorithm targets a single SDF query but it can be employed together for a series of queries in a ray for further taking advantage of this culling process. Take point  $B$  as example (b), we can initialize our minimum distance  $D$  to twice the previous SDF value  $D = 2 \cdot \text{lastSDF}$ , since the SDF value of point  $B$  needs to be smaller than twice the previous SDF. So the process of querying the SDF at point  $B$  can be described as follows:

1. Get the smallest distance  $d1$  to cluster  $C1$ , and since it is greater than  $D = 2 \cdot \text{lastSDF}$ , skip it.
2. Compute the smallest distance  $d2$  to the bounding box of cluster  $C2$ .
3. Since distance  $d2$  is smaller than 0, get the smallest distance  $d3$  to the primitives inside the cluster and update distance  $D$ , as it is smaller.
4. Then, since distance  $d4$  to cluster  $C3$  is greater than  $D$ , we also skip this cluster, so SDF query result is  $D = d3$ .

As aforementioned, many clusters can be rejected using twice the previous SDF query result as initial distance, especially when SDF value is smaller. This works notably good for sphere tracing, where query density is very high near object boundaries, i.e. where SDF value is small. Thus, algorithm on Figure 4 is able to greatly boost the performance of SDF sphere tracing, especially for complex scenes. Besides, this algorithm is stack free and GPU-friendly.

We sample the irradiances at random low-discrepancy [Kel13] directions and store them in the shared memory. To calculate the intersections with the scene, we perform an accelerated version of sphere tracing [KS14] [BV18] for our circumstances.

After intersection point is calculated, Reflective Shadow Map (RSM) is used to obtain its flux. Additionally, the emission value stored inside the primitive is used in order to support area lights and self emission.

By reusing probe’s GI data of last frame, we can achieve multi-bounce global illumination. Since multi-bounce diffuse GI is generally lower frequency than first bounce diffuse GI, we limit its sample number to a lower level in order to decrease computational cost. Apart from this, we can speed up GI response by multiplying multi-bounce GI by a coefficient between 0 and 1 and, thus, decrease loop gain effects.

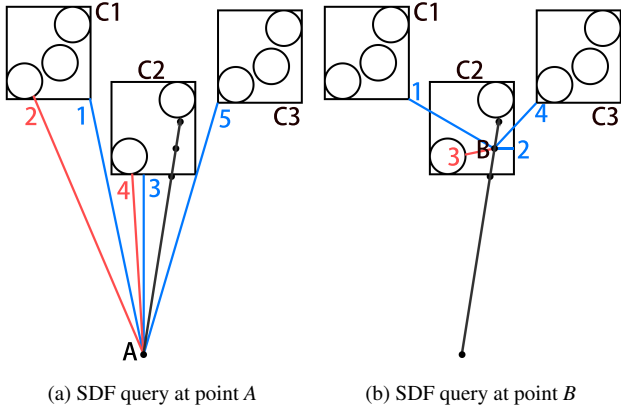


Figure 4: This figure provides a 2D example of SDF querying for two consecutive points  $A$  and  $B$  for a scene with three clusters  $C1$ ,  $C2$  and  $C3$  with their respective SDF primitives inside. The blue lines represent the smaller distance to the bounding box of each cluster while the red lines represent the smaller distance to each cluster’s SDF primitive. The numbers enumerate each of the steps of the algorithm.

After synchronizing all threads in a block, each thread calculates the irradiance in their own direction  $D$  using Equation 3, and use octahedral mapping [CDE\*14] to write irradiance data into a probe atlas texture. We keep a balance between the sample amount and the degree of temporal mixing. If there are too few samples, we apply a stronger temporal mixing in order to prevent jitter. By storing the radiance samples to the group shared memory, we can reuse them across threads to compute the irradiance, thus stabilizing the result.

### 3.4. Per pixel GI shading

Probe Visibility tests take an important role to prevent light leaking triggered by the discretization of irradiance in spatial domain.

RTXGI used probe depth buffers and VSM [DL06] to perform these tests, which is limited to the resolution of the depth buffer, thus unable to completely remove the light leaking effect, especially the leaking caused by thin objects. Inspired by the easy generation of soft shadows using SDF [Aal18], we use the SDF shadow trace to make visibility test, which is able to naturally produce soft indirect shadows and transitions.

Each pixel needs to interpolate 8 probes, what implies to carry sphere tracing 8 times, making it unacceptable as it is too expensive to compute. Fortunately, we only need to perform probe visibility test instead of sampling radiance using sphere trace. For this reason, we can do an extensive down sampling at this stage.

First we down sample the screen depth buffer according to a min/max checkerboard [Bau19] to obtain a half resolution buffer. For checkerboard black pixels, we obtain the maximum value in an area of 4 pixels in the full resolution depth buffer, otherwise for white pixels we take the minimum. This is made to assure we will have valid samples that can cover the whole depth range. This algorithm was proposed in 2019 by the Red Dead Redemption team and

it has proven to be very effective for downsampling in the domain of low frequency rendering.

After that, in the min/max checkerboard down sampled depth buffer, out of every  $2 \times 2$  pixel block we choose one to perform probe visibility test. This choosing process follows the next principles:

- For different frames, choose a different pixel.
- Assure as much as possible that we cover the whole depth extent.

Before performing visibility test on the chosen pixel, we first perform a duplicate removal on each  $2 \times 2$  block unit. It is obvious that close starting points have the same visibility test result for a specific probe. Therefore, in every  $2 \times 2$  block it is necessary to assure that the rest of the probes are not repeated or the distance between starting points is not too large for the visibility test. We use shared memory for the pixels to exchange information, then after performing duplicate removal 4 threads are assigned on average to each pixel inside the block, making every thread’s visibility task diminish from 8 to 4 on average. In our experiments, we use a  $4 \times 4$  size block to obtain an optimal accelerated result, because an oversized block can decrease the merging of visibility test tasks, thus decreasing the effectiveness of the weighting algorithm.

After completing the visibility test tasks, we write the result back into shared memory and distribute it to every corresponding pixel in order to employ this for probe interpolation. Since our approach is able to accurately obtain the visibility of the probe, we do not need to perform an extra cosine weighting or any other additional weighting terms like RTXGI to further avoid leaking artifacts.

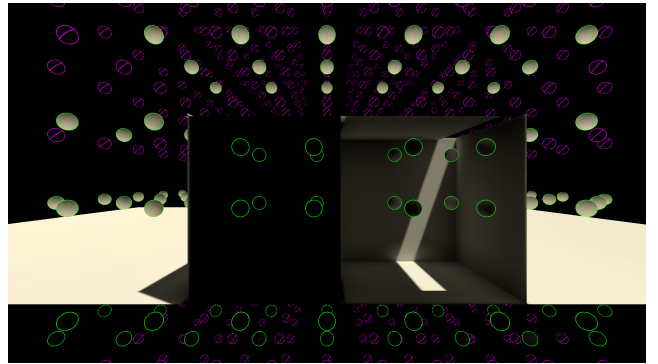


Figure 5: Dark room without light leaking even with thin walls. Our approach is independent of voxel resolution, so objects of any thickness are not able to produce light leaking.

In the upsampling stage, if a pixel does not have a valid result, we assign it a value according to surrounding pixels and last frame’s reprojection result using motion vector. What affects more in the whole process to the final quality of the result is finding representative pixels in the second phase, but we can reuse past results and Neighbourhood Clipping [Kar14] to reduce its influence until it is negligible.

This method greatly compensates the overwhelming strain produced by the per pixel visibility tests. This technique reduces 8 tests per pixel to 0.25 tests per pixel or lower, greatly increasing performance and almost without a negative impact on the end result.



(a) SDFDDGI



(b) RTXGI



(c) Ray Tracing GI

(d) VXGI

Figure 6: Our approach (a) is able to retain the reflections on the pillar, like for RTXGI (b) and VXGI (d). Ray Tracing GI (c), however, lacks multi-bounce reflections and needs extra denoising, which may blur details.

RTXGI uses a low resolution depth buffer to perform these visibility tests, whereas we employ a per pixel probe visibility test to completely eradicate light leaking as Figure 5 shows and optimizing the process with an effective down sampling to extra reduce the computational cost.

### 3.5. Contact GI

We can greatly diminish the computational cost of GI by discretizing the irradiance on the space domain. However, this leads to loss of detail. We created a new technique called Contact GI based on Ground Truth Ambient Occlusion (GTAO) [JWPJ16] to enhance the details of diffuse GI.

Ambient occlusion represents how much ambient lighting receives an object surface according to the occlusion received by surrounding objects. In the per pixel GI shading stage, the position of probes and that of the actual pixel is not the same triggers the loss of global illumination detail. By adding AO to the GI shading result we can to some extent enhance its details. However, this practice does not take into account the effect of multi-bounce indirect illumination among other additional problems. For example, in directly lighted regions the effect of diffuse GI over AO should be greater, but in RTXGI this wrongly weakens the indirect illumination. By using contact GI we corrected this problem.

At the same time of computing ambient occlusion, we sample the lighting at occluded points and merge it with the probe GI so that we can take into account the effect of multi-bounce lighting, therefore vastly improving the level of detail of GI.

### 3.6. Cascade volume

By employing a varying-density probe volume, we provide a stratified Cascade Probe Volume, that is, assign a sparsely populated probe volume for further away regions from the camera, making the algorithm feasible for a greater extension of the scene, even more than 1 km away. This way this method can not only be applied for small indoor scenes, but for mid-ranged building scenes and large open worlds as well. Besides, this process of reduction of more distant probes can be adapted to the actual cost of global illumination in order to further extend the effect of global illumination in the scene. We employ Mean Value Coordinates [FKR05] instead of trilinear interpolation for the interpolation between different density volumes so that we can have a softer transition.

For the furthest strata of the probe volume we do not perform any visibility test since at this scale it seemed unnecessary for global illumination.

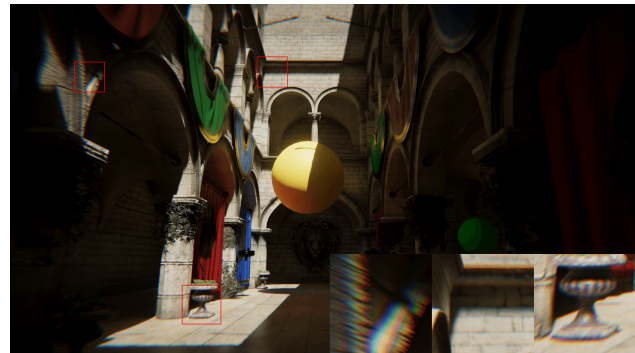
## 4. Results

We tested the performance and the quality of GI of our algorithm with different hardware and scenes. Most of the experiments were performed in comparison to other state-of-the-art real-time global illumination approaches.



(a) Contact GI on

(b) Contact GI off



(c) Contact GI on

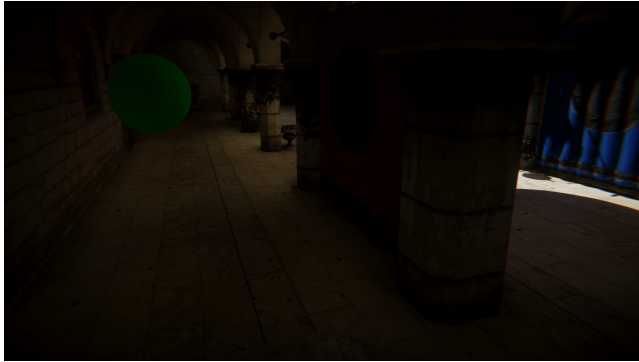
(d) Contact GI off

Figure 7: Some examples of the effect of Contact GI on a simple lighted room (a) (b) and the Sponza Palace (c) (d). Contact GI greatly compensates the loss of detail produced by the discretization of the irradiance by enhancing the realism of details.

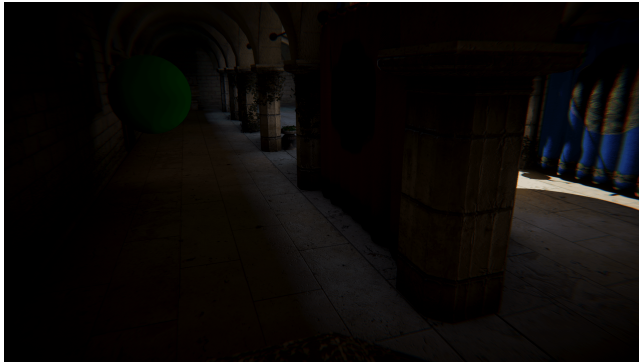
### 4.1. Probe volume resolution

Since the position of the probes is assigned at runtime according to SDF gradient descent, this method does not need any previous human intervention for an ideal distribution of the probes. We first observed the effect of the resolution of the probe volume on the result GI and then we compare it to an offline path tracer as Ground Truth.

In Figure 9 we can observe that if probe volume resolution is very low it will not produce light leaking problems, although the result GI will, of course, not be very accurate.



(a) SDFDDGI



(b) RTXGI

Figure 8: Comparison of light leaking artifacts on SDFDDGI and RTXGI. Our method (a) is able to eradicate light leaking. However, on thin objects such as the curtain on the Sponza scene RTXGI (b) may face light leaking issues.

#### 4.2. Effect of Contact GI

Contact GI is able to compensate the loss of detail of diffuse GI produced by the sampling of the irradiance function. In Figure 7 we can observe the difference of using Contact GI on a basic lighted room and on the Sponza scene.

Contact GI brings about significant improvements to the effect of minor traits of the scene to the diffuse global illumination. It solves some of the loss of detail caused by the disparity of the position of the probes and the real position taken into account for shading.

#### 4.3. GI comparison to other methods

We compared our method with other real-time GI techniques using the Sponza scene. This scene has 5,000 triangles, 26 different materials and 48 different resolution textures. We employed 43 SDF primitives for the dynamic representation of the structure of the scene and fall it apart into 8 clusters. The probe volume of SDFDDGI, as well as RTXGI, has a resolution of 22x14x32 probes. VXGI voxel resolution is 64x64x64.

In Figure 6 we can see that with the same probe volume, our approach fits better the subtleties of the scene. In comparison to Ray Tracing GI, our method is able to support multiple bounce il-

lumination and at the same time it does not need a denoiser, which could blur some details. VXGI with a higher resolution is not able to achieve the same level of detail and, at the same time, it already needs longer computation time.

As seen in Figure 8, for low resolution probe volume there is no light leaking, which is still a problem for RTXGI for, for example, thin objects as the curtains. Instead our method is able to correctly manage the occlusion of curtains.

#### 4.4. Performance

We tested the performance of the algorithm with Sponza as test scene under the same configuration as in previous section, analyzing the time consumed for every stage of the algorithm, also in comparison to the other aforementioned methods.

	Consumed time	
	Total	Per stage
SDFDDGI	1.67	<i>Probe Update: 0.70</i> <i>Contact GI: 0.41</i> <i>Shading: 0.56</i>
SSGI (diffuse only)	1.17	<i>Depth Mipmap: 0.06</i> <i>HiZ trace: 0.69</i> <i>Denoise: 0.42</i>
RTXGI	3.98	<i>Probe Update: 3.28</i> <i>Shading: 0.70</i>
Ray Tracing GI (diffuse only)	4.13	<i>Trace ray: 2.23</i> <i>Denoising: 1.90</i>
VXGI (diffuse only)	5.24	<i>Voxelize: 3.31</i> <i>Cone trace: 1.93</i>

Table 1: Stage by stage performance comparison of state of the art real-time GI approaches with our method, all running on RTX 2080Ti and I7 9700k, with a render resolution of 1920x1080 on the Sponza scene.

As we see in Table 1, our method achieves the best performance for the same or better quality. Time per frame is only shorter for Screen Space GI, which makes a lot of quality compromises in order to achieve this performance.

#### 4.5. Conclusions

We have proposed a novel approach to calculate real-time global illumination using SDF. Apart from saving all work flow for baking, and supporting fully dynamic lighting and geometry for a scene, it also provides better quality GI and has higher performance than other similar approaches, thus having some potential applications. However, our method still has room for improvement. For example, according to the relative placement of the camera and the probes, we could use importance sampling around this direction in order to further stabilize global illumination because only the normal facing to camera can be seen by the camera. Our research interest also focuses on dynamic GI, so for specular GI we still have to rely on a mixed approach using other methods such as SSR or Ray Tracing, but using SSR on top does not add any extra cost to achieve diffuse-specular path. Last of all, our approach uses simplified SDF primitives to represent the scene, until now we manually provide its



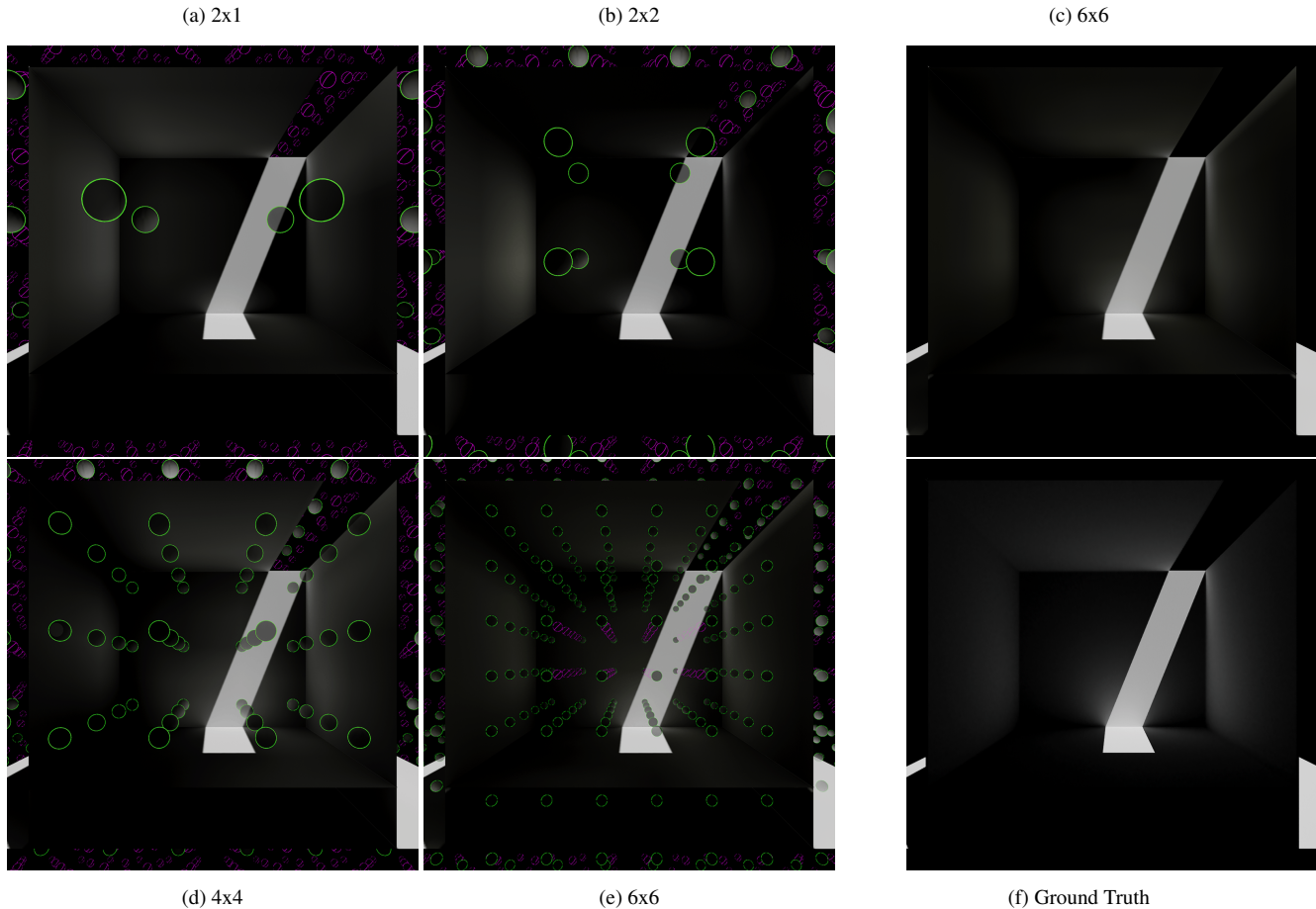


Figure 9: SDFDDGI results with different probe volume resolutions. Path Tracing Ground Truth as reference. With too few probes, as we see in the room with 2x1 probes (a), we are not able to produce satisfying global illumination but light leaking is still unprevent. Besides, with a denser, but still relatively rough, probe volume grid (c), we can get similar global illumination as in a Path Tracing reference (f).

simplified SDF representation, what requires an enormous amount of work for large and complex scenes. For the future, it would be necessary to research on the automatization of this process.

## References

- [Aal18] AALTONEN S.: Gpu-based clay simulation and ray-tracing tech in claybook. *San Francisco, CA* (2018). 2, 5
- [Bau19] BAUER F.: Creating the atmospheric world of red dead redemption 2: A complete and integrated solution. *SIGGRAPH'19 ACM SIGGRAPH 2019 Courses* (2019). 5
- [Ben19] BENYOUB A.: Leveraging real-time ray tracing to build a hybrid game engine. *SIGGRAPH'19 ACM SIGGRAPH 2019 Courses* (2019). 3
- [BV18] BÁLINT C., VALASEK G.: Accelerating sphere tracing. In *Eurographics (Short Papers)* (2018), pp. 29–32. 4
- [CDE\*14] CIGOLLE Z. H., DONOW S., EVANGELAKOS D., MARA M., MCGUIRE M., MEYER Q.: A survey of efficient representations for independent unit vectors. *Journal of Computer Graphics Techniques* 3, 2 (2014). 5
- [CNS\*11] CRASSIN C., NEYRET F., SAINZ M., GREEN S., EISEMANN E.: Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 1921–1930. 1, 2
- [DL06] DONNELLY W., LAURITZEN A.: Variance shadow maps. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games* (2006), pp. 161–165. 2, 5
- [DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2005), I3D '05, Association for Computing Machinery, p. 203. URL: <https://doi.org/10.1145/1053427.1053460>, doi:10.1145/1053427.1053460. 2
- [FKR05] FLOATER M. S., KÓS G., REIMERS M.: Mean value coordinates in 3d. *Computer Aided Geometric Design* 22, 7 (2005), 623–631. 7
- [Har96] HART J. C.: Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 10 (Dec. 1996), 527–545. URL: <https://doi.org/10.1007/s003710050084>, doi:10.1007/s003710050084. 2
- [JWPJ16] JIMÉNEZ J., WU X., PESCE A., JARABO A.: Practical real-time strategies for accurate indirect occlusion. *SIGGRAPH 2016 Courses: Physically Based Shading in Theory and Practice* (2016). 7
- [Kaj86] KAJIYA J. T.: The rendering equation. In *Proceedings of the 13th*

- annual conference on Computer graphics and interactive techniques (1986), pp. 143–150. [2](#)
- [Kar14] KARIS B.: High-quality temporal supersampling. *Advances in Real-Time Rendering in Games, SIGGRAPH Courses 1*, 10.1145 (2014), 2614028–2615455. [5](#)
- [KD10] KAPLANYAN A., DACHSBACHER C.: Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (2010), pp. 99–107. [2](#)
- [Kel13] KELLER A.: Quasi-monte carlo image synthesis in a nutshell. In *Monte Carlo and Quasi-Monte Carlo Methods 2012*. Springer, 2013, pp. 213–249. [4](#)
- [KIM\*19] KOSKELA M., IMMONEN K., MÄKITALO M., FOI A., VITANEN T., JÄÄSKELÄINEN P., KULTALA H., TAKALA J.: Block-wise multi-order feature regression for real-time path tracing reconstruction. *ACM Transactions on Graphics (TOG)* 38, 5 (June 2019). [doi:10.1145/3269978](https://doi.org/10.1145/3269978). [2](#)
- [KS14] KORNDÖRFER B. K. H. S. J., STAMMINGER U. G. M.: Enhanced sphere tracing. *STAG: Smart Tools & Apps for Graphics* (2014), 8. [4](#)
- [LSK\*07] LAINE S., SARANSAARI H., KONTKANEN J., LEHTINEN J., AILA T.: Incremental instant radiosity for real-time indirect illumination. In *Rendering Techniques* (2007), pp. 277–286. [2](#)
- [McA15] MCAULEY S.: Rendering the world of far cry 4. In *Game Developers Conference* (2015), pp. 143–146. [2](#)
- [McT04] MCTAGGART G.: Half-life 2/valve source shading. In *Game Developer's Conference* (2004), vol. 1. [1](#)
- [MGNM19] MAJERICZ Z., GUERTIN J.-P., NOWROUZEZAHRAI D., MCGUIRE M.: Dynamic diffuse global illumination with ray-traced irradiance fields. *Journal of Computer Graphics Techniques Vol 8*, 2 (2019). [1](#), [2](#)
- [OS19] OLES SHYSHKOVTSOV SERGEI KARMALSKY B. A. D. Z.: Exploring raytraced future in metro exodus. In *Game Developer's Conference* (2019). [1](#), [2](#)
- [RGS09] RITSCHER T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games* (2009), pp. 75–82. [1](#), [2](#)
- [SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 527–536. [1](#)
- [SKS11] SOUSA T., KASYAN N., SCHULZ N.: Secrets of cryengine 3 graphics technology. In *ACM SIGGRAPH* (2011). [1](#), [2](#)
- [SKW\*17] SCHIED C., KAPLANYAN A., WYMAN C., PATNEY A., CHAITANYA C. R. A., BURGESS J., LIU S., DACHSBACHER C., LEFOHN A., SALVI M.: Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*. 2017, pp. 1–12. [2](#)
- [SSS\*20] SEYB D., SLOAN P.-P., SILVENNOINEN A., IWANICKI M., JAROSZ W.: The design and evolution of the uberbake light baking system. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 39, 4 (2020). [1](#)
- [ST15] SILVENNOINEN A., TIMONEN V.: Multi-scale global illumination in quantum break. *SIGGRAPH'15 ACM SIGGRAPH 2015 Courses* (2015). [1](#), [2](#)
- [WBS07] WALD I., BOULOS S., SHIRLEY P.: Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics (TOG)* 26, 1 (2007), 6–es. [3](#)