Mamba: A Global Illumination Solution for Diffuse and Specular Radiance

Venkataram Edavamadathil Sivaram

Figure 1: The result of Mamba processing an example scene.

1 Introduction

In this project, we explore a method of diffuse global-illumination using a unique storage method (to our knowledge) that allows for higher densities of irradiance probes without significantly sacrificing performance. Our high level idea is to store probes in sparse grids using hash tables; in theory this should allow for fast update and query times, and furthermore, given the spatial hash encoding we use, allows for easy neighbor lookup that makes probe rasterization fast.

Our pipeline can logically be represented as shown in Figure 2. At each frame, we first generate the G-buffer from the current viewing point (either

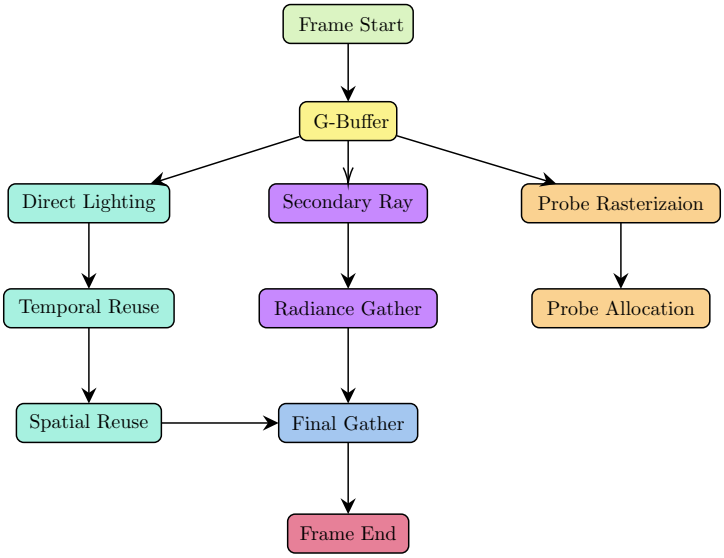


Figure 2: Global Illumination pipeline implemented in MAMBA.

rasterized or raytraced). We then (optionally) compute direct lighting at the computed surfaces and sparsely generate secondary paths. Alongside this, we rasterize (in software) the currently allocated probes in the scene and allocate additional probes to fill the gaps of the rasterized output. Finally, we update the probes with the information received from the secondary paths, and perform a final gather to yield the final color. It should be mentioned that the focus of this project is to experiment with the practicality of implementing this pipeline rather than the accuracy of the results; as a matter of fact the current implementation yields radiance that is wrong but still somewhat visually appealing.

Related concepts: Global illumination, irradiance caching, GPU rendering[1]

2 Raytracing

Even with advances in real-time raytracing, tracing rays into the scene remains computationally expensive, and typical budgets for real-time rendering in games limit the number of rays to around half a ray per pixel plus a shadow ray[2]. As such, we are prompted to heavily reduce the number of rays we cast.

[1] We implement our raytracing pipeline using NVIDIA OptiX.
[2] This is assuming the hardware has some of acceleration for raytracing

2.1 Direct Lighting

Direct lighting in scenes is often of high frequency, and therefore for visually appealing results, it would ideally be computed as accurately as possible. However, it is often also very challenging to solve direct lighting quickly, as sampling lighting in a many-light scene (or scenes with environment lighting) leads to very noisy direct lighting. Recent algorithms like ReSTIR (Bitterli et al., 2020) do indeed solve this issue well, but at the cost of performance, as many additional rays during the temporal and spatial reuse sections.

While we implement a toned-down version of ReSTIR, limiting the number of spatial samples to only two, we mostly disregard direct lighting, and defer a part of it to the radiance caching system in MAMBA. We leave direct lighting in the implementation mostly as an optional feature[3].

2.2 Indirect Lighting

Diffuse indirect lighting is a challenging task because rays scatter in largely random directions as result of the diffuse reflection model. In straightforward implementation of path tracing, this takes it takes many samples to resolve the diffuse inter-reflection phenomenon – contrast this to real-time global illumination, where even just one sample per pixel is too expensive.

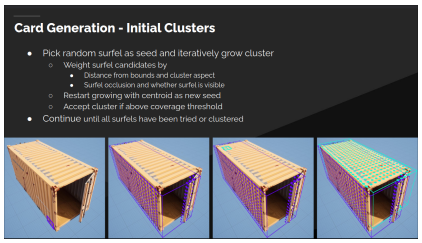
We maintain a low budget during raytracing by sparse tracing secondary paths at a $1/16$th resolution. At each pixel that we choose to path trace, we evaluate the surface reflection model (derived from the G-buffer) and sample the outgoing direction ω_1^o, from the primary vertex v_1 to the secondary vertex v_2. If a valid secondary surface is found, we further compute direct lighting L_d at that vertex (using Next-Event Estimation). From this process, we obtain the radiance L_d coming onto v_1 through ω_1^o as well as the radiance L_e coming onto v_2 through the resulting direction ω_2^o.

3 Irradiance Probes

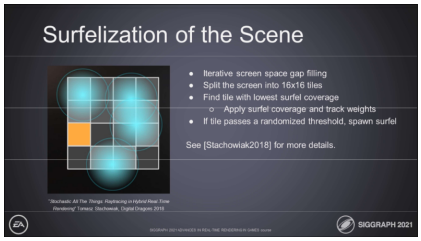
3.1 Motivation

Many modern games have realized that diffuse inter-reflections are computationally expensive to solve, and choose to implement an approximate solution for diffuse irradiance. Of the current approaches, we are interested in surface probe based solutions, similar to the approaches used in Unreal Engine's LUMEN (SKOROBOGATOVA, n.d.) and EA's Global Illumination Based on Surfels (GIBS), see Figure 3. The primary motivation behind using surface-based probes rather than spatially arranged probes is that we can avoid common issues of light leaking, often due to thin geometries.

[3] And furthermore due to some numerical or computational issues during temporal reuse that propagate poorly during spatial reuse.



(a) Lumen's card based system



(b) GIBS' probe spawning algorithm

Figure 3: Irradiance probe management in Lumen and GIBS

3.2 Acceleration Structure

An important decision to make when using surface probes is the choice of acceleration structure used to store the probes. While a typical subdivision structure such as the bounded-volume hierarchy could be used for this, we would quickly run into performance issues due to the repetitive insertions and lookups done during the gathering and rasterization phases (see Figure 2). For this reason, Lumen and GIBS use simple data structures to support constant time (amortized) insertion and access of probes. In particular, Lumen arranges cards, rectangular arrays of probes, throughout the scene, whereas GIBS stores a dense uniform grid of probes (technically they do not use a uniform grid; after a certain extent from the camera, points are translated into the grid using a trapezoidal projection).

In Mamba, we decide to use a sparse uniform grid, in the form of a two-level hash table, to insert and retrieve surface probes. Compared to a dense uniform grid, we can comfortably store an adaptive number of probes without allocating them beforehand, and compared to cards, we can cover smaller and tighter regions of the scene, where the surfaces cannot be effectively represented with cards. The high level structure of the hash table is shown in Figure 4.

The probe allocation algorithm scans 16×16 cells of pixels from the probe rasterization target to determine whether a new probe should be added, and where it should be allocated (e.g. place a new probe furthest away from the current probes). If the number of probe-empty pixels reaches a minimum threshold, we spawn a new probe; in this way we are able to adaptively allocate probes only as they are needed. See Figure 5 for a visualization of the probe coverage in a scene.

3.3 Irradiance Caching

Each probe needs a data structure for storing the incoming radiance. A simple approach involves simply using grids, and associating a mapping from radiance directions and grid indices. Another approach is to use spherical harmonics to encode the incoming radiance. For the sake of simplicity, we have chosen the grid approach. We use a straightforward hemispherical projection to map

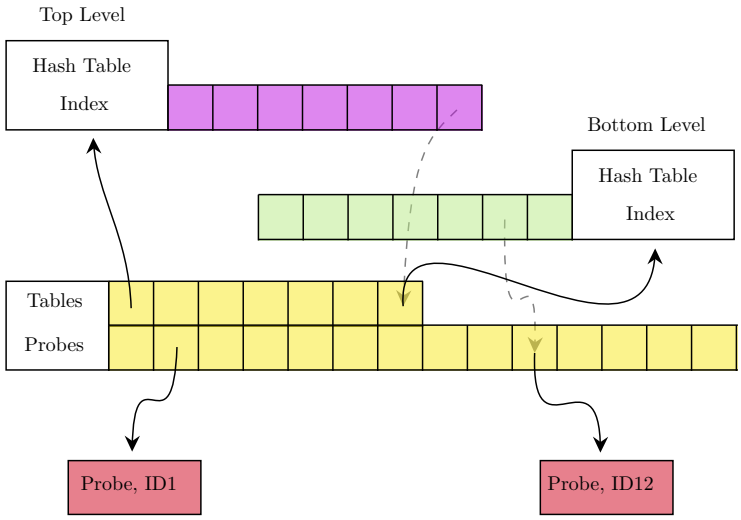


Figure 4: High level overview of the two-level hash table used in MAMBA.

Figure 5: The result of our probe allocation scheme after a couple frames.

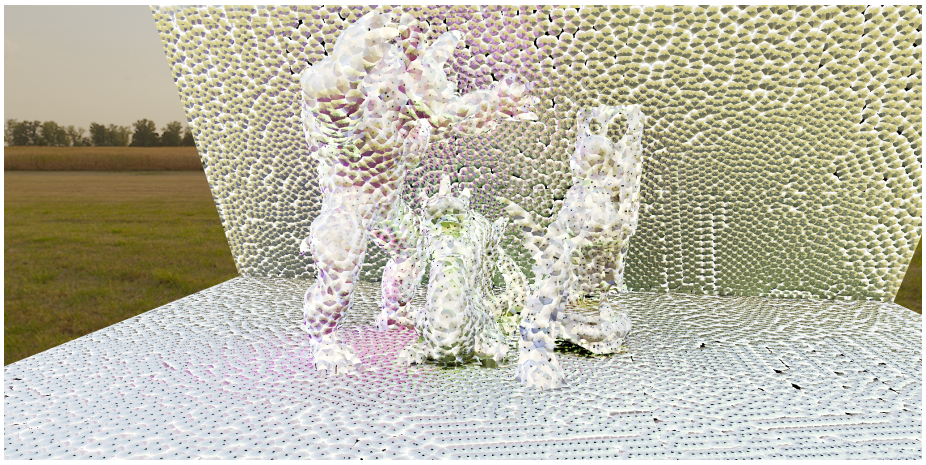


Figure 6: Irradiance probes after caching radiance from a few frames.

the incoming directions onto the grid; other approaches have used the octahedral projection, but since only one hemisphere is being used, its efficacy is theoretically not as great as the hemispherical projection.

After the secondary paths have been generated, we need to update the irradiance probes with their radiance information. We update the nearest probe at the primary vertex v_1 with the radiance pair (L_d, ω_1^o), as well as the secondary probe nearest to v_2, with the pair (L_e, ω_2^o). We also found that a recursive probe update was useful; we can integrate the secondary probe with the corresponding material to retrieve a better approximation of the illumination arriving at v_1. See Figure 6 which shows the result of irradiance caching.

4 Final Gather

The primary objective of the final gather kernel is to compute the final illumination at each visible surface point. At each pixel, we find the nearest N (currently two) probes to the corresponding surface point, and evaluate a weighted average of the radiance integrated with respect to the current viewing direction. While a higher number of closest probes can be found, the cost of integrating each one becomes prohibitively high, so use a low count simply to avoid extreme quantization effects. Figure 7 shows the result of the latest final gather implementation. There are notable issues, such as the arbitrary dark regions – due to the fact that we currently do not allocate separate probes on opposite sides of thin geometry – and the overly smoothed irradiance, with little self shadowing[4] See Figure 8 for more example renderings.

[4]While the image looks excessively bright, it compares well with the ground truth, which is also quite bright for this scene.

Figure 7: Final gather of the current MAMBA pipeline

5 Further Details

Future work By nature of the limited time available during this course, only diffuse inter-reflection has been implemented in MAMBA. However, work will still be continued to extend and improve the pipeline; the following are some of the future aspirations:

- Resampled importance sampling to share radiance amongst nearby probes.
- Spherical harmonics to compress the probe representation.
- Improved filtering to remove quantization effects.
- Path guiding using the radiance information within the probes.
- Radiance field caching for highly specular surfaces.

Submission details MAMBA pipeline described here is implemented in Kobra[5] using `CUDA` and NVIDIA OptiX (Ludvigsen & Elster, 2010). The scope of this project involves only the pipeline, and not the rest of the engine, which has been in development for a few months.

References

Bitterli, B., Wyman, C., Pharr, M., Shirley, P., Lefohn, A., & Jarosz, W. (2020). Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, *39*(4). https://doi.org/10/gg8xc7

[5]See `https://github.com/vedavamadathil/kobra`, where this algorithm is implemented (specifically the sourcesof the form `editor/mamba*`)

(a) Sponza rendered with Mamba.

(b) San Miguel rendered with Mamba.

Figure 8: Additional renderings from the Mamba pipeline.

SKOROBOGATOVA, A. (n.d.). Real-time global illumination in unreal engine 5.

Ludvigsen, H., & Elster, A. C. Real-time ray tracing using nvidia optix. In: In *Eurographics (short papers)*. 2010, 65–68.

Wang, T., & Wu, Z. Importance resampling based on surfels with dynamic lighting. In: In *International conference on image, signal processing, and pattern recognition (ispp 2023). 12707*. SPIE. 2023, 424–431.