# SparseDR: Differentiable Rendering of Sparse Signed Distance Fields

**Alexey Budak**[1] , **Albert Garifullin**[1] , **Vladimir Frolov**[2]
[1]Lomonosov Moscow State University
[2]Institute for Artificial Intelligence, Lomonosov Moscow State University
{ alexey.budak, albert.garifullin, vladimir.frolov }@graphics.cs.msu.ru

## Abstract

We present SparseDR, a novel differentiable rendering algorithm designed for sparse representations based on Signed Distance Fields (SDF). We leverage the Sparse Brick Set representation and propose an adaptation of redistancing for sparse SDFs. This enables SparseDR to surpass existing works in accuracy of surface reconstruction by increasing the effective resolution of the SDF representation. SparseDR is efficiently implemented in C++ and Vulkan, achieving several times shorter reconstruction time than other methods.

## 1 Introduction

The signed distance field (SDF) provides a continuous implicit representation of geometry, defining the shortest Euclidean distance from any point in space to the nearest surface, with the sign indicating interior and exterior regions. It has proven to be a convenient shape representation for differentiable rendering, allowing robust and relatively simple gradient-based optimization [Wang *et al.*, 2024].

In an SDF-based differentiable rendering, each gradient descent step updates the distance values at grid cells, so the grid no longer strictly satisfies the SDF property. To address this issue, existing methods typically apply *redistancing* – a process that recomputes grid values so that each cell stores the correct distance to the surface [Detrixhe *et al.*, 2013]. This makes SDF-based differentiable rendering on sparse data structures non-trivial, so all existing methods rely on regular grids. Unfortunately, this approach results in high memory and computational costs, while the reconstruction accuracy remains limited by the maximum grid resolution. Our work proposes a solution to this problem, alleviating limitations through better data structures and sampling techniques:

- We propose an adaptation of SDF-based differentiable rendering to a sparse data structure, which increases the effective resolution of the SDF representation and thereby improves the surface reconstruction accuracy.
- We adapt redistancing to sparse data structures by separating it into two distinct algorithms: local and global redistancing.

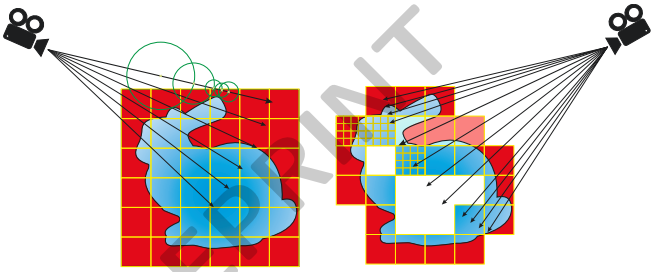

Figure 1: Existing methods (left) use a dense SDF grid to represent the surface, sphere tracing for ray-scene intersection, and uniform image sampling when casting rays. SparseDR (right) uses Sparse Brick Set, where a ray intersects each brick on its way until it hits the surface. It also uses adaptive sampling for a boundary integral, placing more rays in the relaxed boundary area. The red and the blue colors depict positive and negative SDF values, respectively.

## 2 Related Work

The representation of SDF on a regular grid in existing works [Vicini *et al.*, 2022; Wang *et al.*, 2024; Zhang *et al.*, 2025] has three major limitations: (1) significant memory consumption, especially when using automatic differentiation frameworks such as DrJit [Jakob *et al.*, 2022] or PyTorch [Jason and Yang, 2024]; (2) slow ray-surface intersection based on sphere tracing; (3) an expensive redistancing algorithm that propagates modified SDF values across the entire grid. Our work shows how these components – data structure, ray intersection, and redistancing – can be joined together in an efficient manner.

## 3 Proposed Method

SparseDR uses the Sparse Brick Set (SBS) proposed in [Söderlund *et al.*, 2022], a hybrid of hierarchical and regular representations (Figure 1). The entire scene is encoded as a Bounding Volume Hierarchy (BVH); its leaves store small regular SDF grids, for example, 4x4x4 voxels. For gradient computation, we adapt the relaxed boundary method from [Wang *et al.*, 2024], which we refer to as our baseline.

The proposed optimization process consists of several epochs with multiple gradient descent steps in each. At the beginning, the distance field is initialized with a sphere defined on a low-resolution grid (e.g., $32^3$). At each iteration, we estimate the gradients and update the distance values using a gradient descent step followed by a regularization and

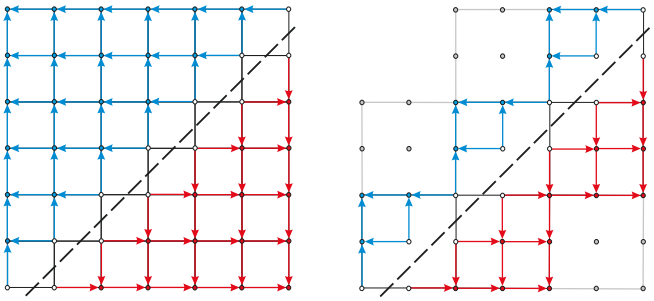

Figure 2: Comparison of the global redistancing algorithm on a dense grid (on the left) with the local redistancing (on the right) for SBS; bricks are 3x3 distances in size. The dots represent the distance values. Red and blue dots are the distances outside and inside the object, respectively. The dashed line represents the surface, set by distances colored white. Arrows show the order of updating the values. Local redistancing ignores the bricks without the surface (grey).

local redistancing (see Figure 2). At the end of each epoch, two steps are executed: upsampling and global redistancing. After the last epoch, only global redistancing is performed.

**Gradient Evaluation.** The key feature of SparseDR is the transition from a regular grid to a Sparse Brick Set. A BVH tree is constructed for the SBS, with each leaf node storing a single brick. Instead of sphere tracing, we have adapted a faster Newton method for ray-SDF intersection [Söderlund *et al.*, 2022]. A ray first traverses the BVH; if an intersection with a leaf node is found, it then intersects the voxels of the brick. The ray traverses each brick along its path until it hits the surface or there are no bricks left. The Newton method was modified to search for the relaxed boundary points: within the voxel, a potential relaxed boundary point is found as the root of a quadratic equation, after which the SDF value at the point is checked to be below the relaxed epsilon. The case when the local minimum occurs on the voxel boundary is handled separately.

SparseDR is implemented in C++ and Vulkan and does not use automatic differentiation, i.e., all derivatives were obtained analytically. In the shader, we accumulate the pixel color in batches of 16 samples, storing the information about voxels the samples hit in a fixed-size array. Every 16 samples, or after the last one, the gradients are accumulated using atomic operations.

SparseDR optionally uses an additional pass that only samples pixels containing borders and evaluates only the boundary integral, eliminating the need for color computation. This allows for cheaper boundary samples, which helps reduce variance.

**Local Redistancing.** We implemented two versions of redistancing: local and global (Figure 2). For the proposed method, the distances in the bricks without a surface are not required to strictly satisfy the SDF property during optimization. Therefore, the local version is applied at each iteration, recomputing distances only within the bricks containing the surface. In practice, the local redistancing is sufficient, since our ray-surface intersection method relies solely on the distances within the traversed bricks. However, this approach

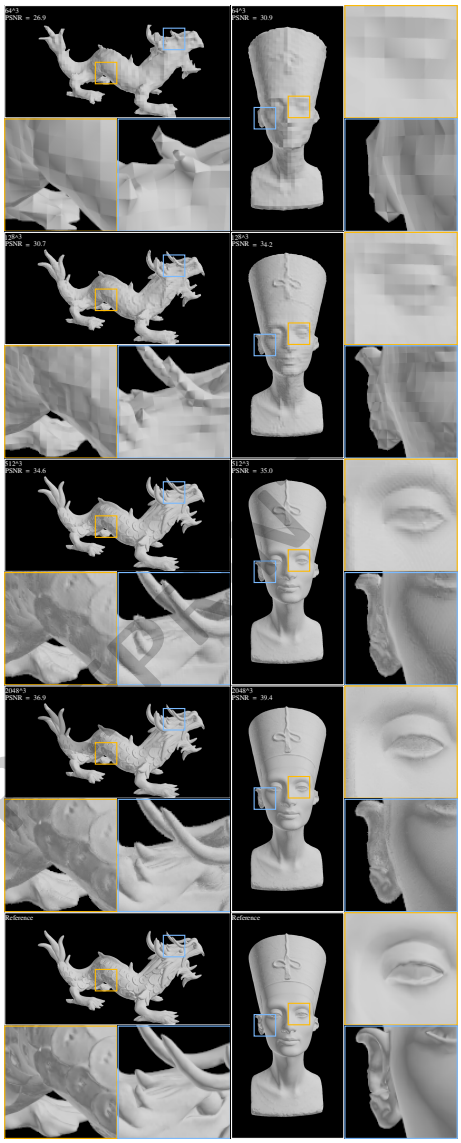

Figure 3: Reconstruction examples for different epochs.

may cause distance inconsistencies between adjacent bricks, so we perform global redistancing at the end of each epoch.

**Upsampling and Global Redistancing.** The global redistancing step updates distances across empty regions between bricks by extrapolating values between brick faces. For each face without a direct neighbor, the nearest neighboring bricks are found, and one of their faces is selected as the extrapolation source. The neighbors are found once, after which extrapolation is performed following each fast-sweeping iteration. At each update, the value with the smaller absolute distance between the extrapolated and existing values is retained. The redistancing ends with a fast-sweeping pass to propagate the extrapolated distances throughout the domain.

The upsampling step begins with SBS sparsification. At this stage, only the bricks containing the surface are preserved, along with all neighboring bricks within a radius of

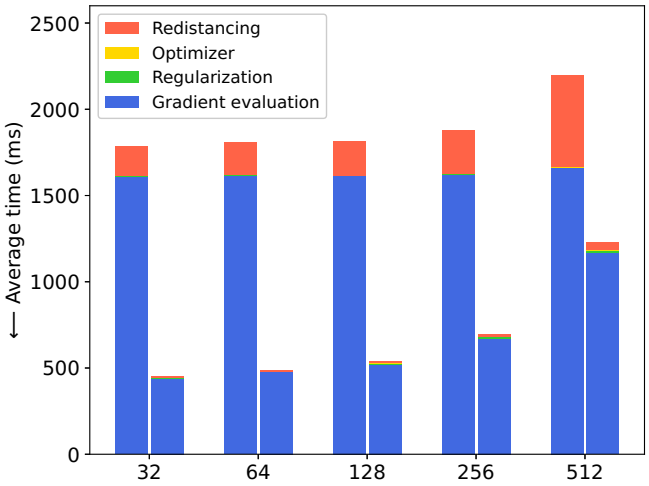


Figure 4: The average execution time per iteration for the first five epochs. For each grid size, the left bar depicts the baseline, and the right bar shows SparseDR for the SBS with equivalent resolution.

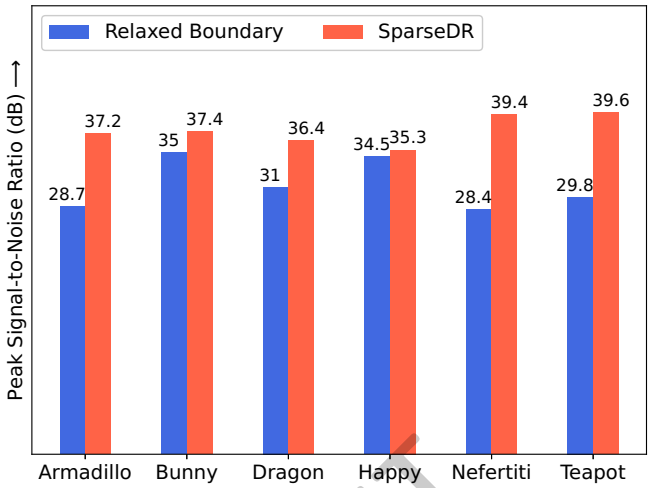


Figure 5: Average PSNR values across viewpoints for models.

| Representation | $64^3$ | $128^3$ | $512^3$ | $1024^3$ | $2048^3$ |
|---|---|---|---|---|---|
| Baseline | **1** | **8** | 512 | 4096* | 32768* |
| Ours(Armadillo) | 3.1 | 14.1 | **126.2** | **514.2** | **1074.2** |
| Baseline | **1** | **8** | 512 | 4096* | 32768* |
| Ours(Bunny) | 2.9 | 11.4 | **195.1** | **775.3** | **3093.8** |
| Baseline | **1** | 8 | 512 | 4096* | 32768* |
| Ours(Dragon) | 1.8 | **4.8** | **67.9** | **275.6** | **1154.6** |
| Baseline | **1** | 8 | 512 | 4096* | 32768* |
| Ours(Happy) | 1.5 | **4.9** | **88.5** | **373.9** | **1562.8** |
| Baseline | **1** | 8 | 512 | 4096* | 32768* |
| Ours(Nefertiti) | 2.2 | **7.3** | **114.1** | **459.0** | **1845.3** |
| Baseline | **1** | 8 | 512 | 4096* | 32768* |
| Ours(Teapot) | 1.7 | **6.3** | **96.4** | **392.1** | **1597.7** |

Table 1: The model sizes for the Relaxed Boundary and SparseDR methods. All values are in MBs. Asterisks denote theoretical values, as we could not acquire them on our hardware (OOM in Dr. JIT).

1 brick (diagonal neighbors included). If any of the adjacent bricks are missing, they are added now. This step is necessary, as the regions corresponding to the surface might otherwise lack the necessary brick coverage. Thus, this procedure not only reduces the overall model size but also removes a potential limitation compared to the baseline. Following sparsification, the upsampling step proceeds by doubling the resolution of each brick, after which it is subdivided into eight smaller bricks of the original size. New distance values are computed using trilinear interpolation. After upsampling, a global redistancing step recomputes all distance values, ensuring that the resulting representation remains a valid SDF.

## 4 Experiments

Experiments with the baseline were conducted on the official implementation's original setup [Wang, 2024]. The only modifications are the number of viewpoints, the batch size, and the epoch sizes. For the consistency across experiments, an identical lighting setup was used: two directional light sources, (1,1,1) and (-1,-1,-1), and the ambient light. The experiments were carried out on six models. 4 models are from the Stanford 3D scanning repository: Armadillo, Asian Dragon, Stanford Bunny, and Happy Buddha. The other two are Nefertiti scan [Nelles and Al-Badri, 2015] and the Utah Teapot. Reconstruction used 16 viewpoints, uniformly distributed over a sphere around the scene, generated by the algorithm from [Wang, 2024]. The batch size is 4 viewpoints; the image resolution is 1024x1024.

The full optimization process comprised 1000 steps, with upsampling applied at steps 200, 400, 600, 700, 800, and 900. Hardware configuraiton is AMD Ryzen 9 7950X CPU and NVIDIA GeForce RTX 4090 GPU. Some reconstruction examples are shown in Figure 3.

Figure 4 shows the average reconstruction time across all models per epoch for SparseDR and the baseline, epochs labeled by the SDF grid of the corresponding resolution. "Gradient evaluation" refers to the consecutive rendering of all views from the current batch. Figure 5 presents the mean reconstruction quality across viewpoints for SparseDR and the baseline. Table 1 illustrates the model sizes per epoch. For the baseline, it only shows the grid size; for SparseDR, it shows both the size of the SBS and its BVH. The results show increased reconstruction speed and quality compared to the baseline, while the growth of the SBS size per epoch is an order of magnitude lower than that of the SDF grid.

## 5 Conclusion

In this work, we introduce SparseDR, an SDF-based differentiable rendering method that leverages sparse SDF representations. Building on the [Wang *et al.*, 2024], we exploit the strengths of Sparse Brick Sets to surpass the baseline while reducing memory consumption by an order of magnitude. Future work will focus on improving performance. We believe that the current approach holds the potential for further efficiency gains, particularly through more accurate gradient estimation and a reduction in per-pixel sampling.

## Acknowledgments

Alexey Budak and Albert Garifullin were supported by the The Ministry of Economic Development of the Russian Federation in accordance with the subsidy agreement (agreement identifier 000000C313925P4H0002; grant No 139-15-2025-012).

## References

[Detrixhe *et al.*, 2013] Miles Detrixhe, Frédéric Gibou, and Chohong Min. A parallel fast sweeping method for the eikonal equation. *Journal of Computational Physics*, 237:46–55, 2013.

[Jakob *et al.*, 2022] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. Dr.jit: a just-in-time compiler for differentiable rendering. 41(4), 2022.

[Jason and Yang, 2024] Jason and Edward He et al. Yang. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 929–947, New York, NY, USA, 2024. Association for Computing Machinery.

[Nelles and Al-Badri, 2015] Jan Nikolai Nelles and Nora Al-Badri. Nefertiti scan, 2015.

[Söderlund *et al.*, 2022] Herman Hansson Söderlund, Alex Evans, and Tomas Akenine-Möller. Ray tracing of signed distance function grids. *Journal of Computer Graphics Techniques Vol*, 11(3), 2022.

[Vicini *et al.*, 2022] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Differentiable signed distance function rendering. *ACM Trans. Graph.*, 41(4), July 2022.

[Wang *et al.*, 2024] Zichen Wang, Xi Deng, Ziyi Zhang, Wenzel Jakob, and Steve Marschner. A simple approach to differentiable rendering of sdfs. In *SIGGRAPH Asia 2024 Conference Papers*, New York, NY, USA, 2024. Association for Computing Machinery.

[Wang, 2024] Zichen Wang. A simple approach to differentiable rendering of sdfs repository, 2024.

[Zhang *et al.*, 2025] Ziyi Zhang, Nicolas Roussel, and Wenzel Jakob. Many-worlds inverse rendering. *ACM Trans. Graph.*, 45(1), October 2025.