# SparseDR: Differentiable Rendering of Sparse Signed Distance Fields

**First Author**[1] , **Second Author**[2] , **Third Author**[2,3] and **Fourth Author**[4]
[1]First Affiliation
[2]Second Affiliation
[3]Third Affiliation
[4]Fourth Affiliation
{first, second}@example.com, third@other.example.com, fourth@example.com

## Abstract

We present SparseDR, a differentiable rendering implementation designed for sparse representations based on Signed Distance Fields (SDF). We leverage the Sparse Brick Set (SBS) representation and propose an adaptation of redistancing and regularization of the SDF defined on SBS. This enables our method to surpass existing differentiable rendering implementations in accuracy at limited memory budgets by increasing the effective resolution of the SDF representation.

## 1 Introduction

Signed distance functions (SDFs) provide 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 or exterior regions. This enables gradient-based optimization by supporting differentiable rendering, where ray marching samples distances to compute density and color, facilitating inverse rendering tasks such as shape and material recovery from images.

SDF-based differentiable rendering produces high-fidelity reconstructions with smooth surfaces and topological consistency, outperforming many existing methods in handling complex topologies. Unfortunately, existing SDF-based differentiable rendering implementations are constrained in reconstruction accuracy due to high memory consumption and substantial computational demands. 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 provide two novel algorithms of redistancing: a full version for SBS in general and its accelerated version specifically for the differentiable rendering of SBS.

## 2 Related Work

[Vicini *et al.*, 2022] introduces differentiable rendering operating directly on an SDF representation. The authors modified sphere tracing so that, in addition to computing the surface intersection, it evaluates a special reparameterization of rays, enabling correct gradients with respect to shape parameters, including the visibility contribution of individual surface regions. This work also first proposed the use of a *redistancing* — a process of recomputing values on a grid such that all grid cells store correct distances to the surface. In differentiable rendering this recomputation is required because, after an optimization step and updating the distances at grid nodes, the grid no longer strictly represents an SDF. [Wang *et al.*, 2024] exploits properties of distance functions and estimate the boundary integral by an integral over a relaxed boundary, i.e., a thin band around the object silhouette. [Zhang *et al.*, 2025] propose for computing gradients in the presence of multiple surfaces along a ray path: instead of locally differentiating a single surface, the authors consider a volumetric, non-local perturbation of the surface.

All of the above methods represent the distance function volumetrically on a regular grid, which leads to three fundamental issues: (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.

Hash-table-based methods such as InstantNGP [Müller *et al.*, 2022] could mitigate the first issue, but by themselves cannot address the second and third. The key difficulty is that the data structure used to store the SDF directly constrains how differentiable ray–surface intersections are computed and how redistancing is performed, and implementing redistancing on hash tables is, in our view, highly nontrivial. Our work shows how these three components – data structure, ray intersection, and redistancing – can be joined together in an efficient manner.

## 3 System Implementation

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

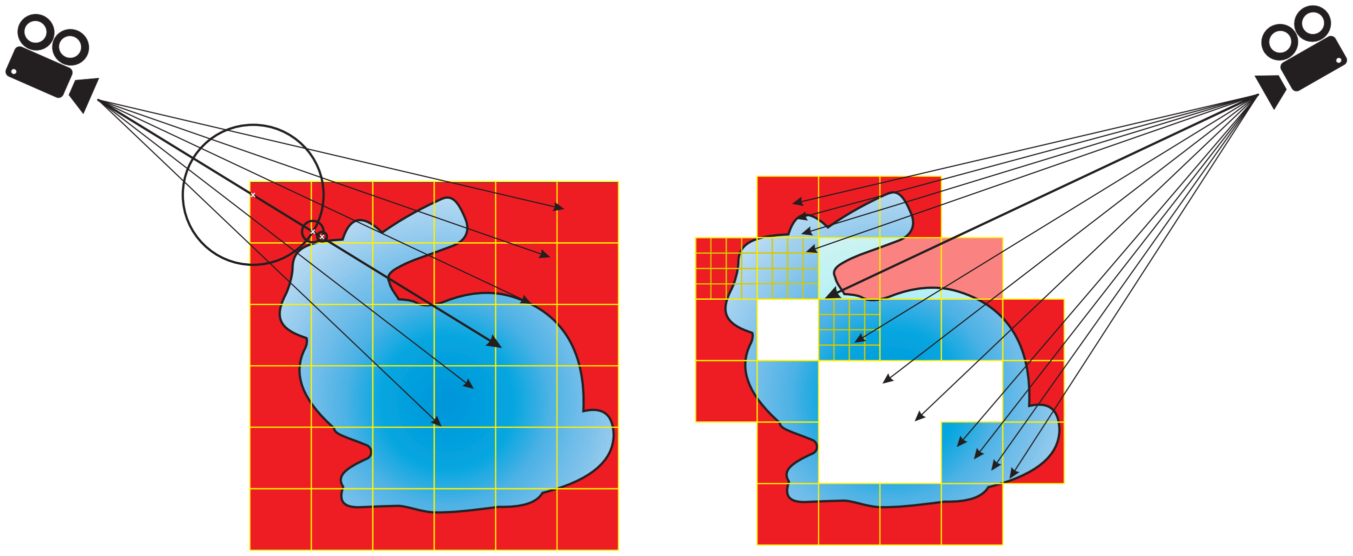

Figure 1: Comparison of SparseDR algorithm (on the right) with the baseline. The key differences are using sparse SDF representation, SBS, and intersecting voxels along the ray instead of sphere tracing.

beginning of the optimization process, the distance function is initialized with a sphere defined on a low-resolution grid (32x32x32). At each iteration, we estimate the gradient and update the distance values using a gradient descent step, followed by a regularization and accelerated redistancing (see Figure 2). On every iteration when upsampling is required, which is defined by the user, three steps are executed: sparsification step, full redistancing and upsampling. Sparsification and full redistancing are also applied before the resulting model is saved.

## 3.1 Gradient evaluation

The key distinction of the method underlying SparseDR lies in the transition from an SDF grid to a sparse brick set. In our case, each brick has a fixed size of 4×4×4 voxels. For the SBS, a BVH tree is constructed, with each leaf node storing a single brick. In addition to sphere tracing, alternative ray–SDF intersection methods have been implemented: Newton and analytic, specifically designed to find intersections inside the voxel. 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 proceeds through each brick along its path until the first intersection with a surface is reached.

The alternative ray–SDF intersection methods also support searching for relaxed boundary points. According to [reference], a point belongs to the relaxed boundary if the SDF value at that point is below a certain threshold called the relaxed epsilon, and the SDF derivative along the ray reaches its local minimum. The Newton and analytic methods are based on the fact that inside a voxel, the SDF along the ray can be represented as a cubic polynomial. Thus, within the voxel, a potential relaxed boundary point can be found as the root of a quadratic equation, after which it is sufficient to verify that the SDF value at this point is less than the relaxed epsilon. The case when the local minimum occurs on the boundary between voxels or bricks is handled separately.

SparseDR is implemented in C++ and Vulkan and does not use automatic differentiation. Gradients are accumulated in a float array whose size equals the number of updated distances. In the shader, when computing the color of a pixel for 16 samples, information about the voxels hit by the samples is stored; after every 16 samples (or after the last one), the gradients are added to the array. SparseDR optionally uses an additional shader that samples only pixels containing boundaries and evaluates only the boundary integral, without the need for color computation. This adaptive boundary sampling allows cheaper samples and yields more accurate gradients for smaller relaxed epsilon values, thereby reducing bias.

## 3.2 Redistancing

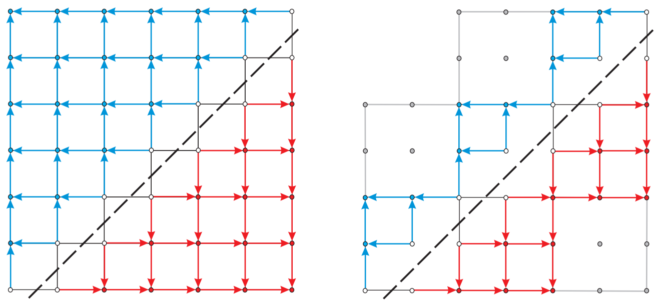

Figure 2: Comparison of the redistancing algorithm (on the left) with the accelerated version. Updating only the bricks with a surface is sufficient to get correct relaxed boundary points, as the brick size is larger than the relaxed epsilon.

For differentiable rendering of SBS in SparseDR, two versions of redistancing have been implemented. The first, full version, is an adaptation of the parallel fast sweeping method for SBS, which is used to acquire the correct SDF. In this version, it was necessary to address the problem of distance updates across empty space between bricks. Consider a brick without neighbors and without any surface inside it. Following the redistancing algorithm, the distances within it will be initialized with a constant value and will never be updated. In differentiable rendering, such extreme cases do not occur, but situations may arise where the distances within a brick have larger absolute values than they should, since adjacent bricks are not taken into account. To solve this issue, an extrapolation step between brick faces was introduced. For each face of a brick without a direct neighbor, the nearest neighbor bricks are found, and one of their faces is selected for extrapolating the values. These neighbors are determined once, and after each fast sweeping step, values are extrapolated considering the grid spacing. From the extrapolated and previous values, the one with the smallest absolute value is chosen. To propagate the updated distances, the fast sweeping method is always applied as the final step of this algorithm.

However, using SBS instead of a grid and Newton's method for intersection search inside the voxel instead of sphere tracing opened opportunities for optimization of the redistancing procedure. These resulted in an accelerated version, which is used after each iteration. The first idea is to exploit the properties of intersection based on the Newton method. Since each ray traverses all voxels along its path, it does not depend on exact distance values in voxels that do not contain boundaries. Therefore, to obtain accurate surface intersections and relaxed boundary points, it is sufficient to apply redistancing only for surface bricks. The second idea assumes that in surface bricks it is enough to compute distances considering only the surface stored within the same brick, without propagating distance values between bricks. This assumption was only partially confirmed, since during the initialization stage, redistancing recalculates distances defining the surface, and they must be as accurate as possible. However, after initialization, redistancing localized within individual bricks is sufficient. Figure Nn shows the MSE plot for optimization iterations of the Nefertiti bust model using both redistancing versions. The accelerated version neither degrades reconstruction quality nor affects convergence to the reference.

### 3.3 Sparsification and Upsampling

The sparsification step is performed before upsampling. During this stage, all bricks containing the surface are preserved, as well as all bricks within a radius of 1 voxel from them, including diagonal neighbors. If any of the adjacent bricks are missing, they are added at this stage. This is necessary in cases where the reconstructed object contains fine details, so that the bricks where these details' reconstruction should reside would otherwise be absent. Thus, this step not only contributes to reducing the model size, but also eliminates a potential limitation compared to the baseline method. After the sparsification step, the full version of redistancing is applied to recompute all distances before upsampling or saving the resulting SBS, ensuring that the model represents a valid SDF. The upsampling step doubles the resolution of the remaining bricks, generating eight new bricks from each original one. The new distance values are computed using trilinear interpolation.

## 4 Experiments

Experiments with the baseline method were conducted using [https://github.com/zichenwang01/relaxed-boundary/], which provides its official implementation. The baseline method employed the hyperparameters specified in the paper [reference], while the remaining parameters were taken from the configuration file "turbo.json". The only modifications concerned the number of viewpoints, the batch size, and the steps at which upsampling was performed. For consistency across experiments, an identical lighting setup was used: two directional light sources with directions $(1,1,1)$ and $(-1,-1,-1)$, along with ambient light. The experiments were carried out on six models. Reconstruction used 16 viewpoints uniformly distributed over a sphere around the scene, generated by the algorithm from [reference]. The batch size is 4 viewpoints, image resolution is 1024×1024. The full optimization process comprised 1000 steps, with upsampling applied at steps 200, 400, 600, 700, 800, and 900. Computations were performed on an AMD Ryzen 9 7950X CPU and an NVIDIA GeForce RTX 4090 GPU.

## 5 Conclusion

In this work, we present SparseDR, a system for 3D reconstruction based on the method of differentiable rendering for a sparse SDF representation. Using [reference] as a baseline, we leverage the advantages of SBS and modify all stages of the algorithm. As a result, SparseDR achieves performance and reconstruction quality comparable to the original approach while reducing memory consumption by an order of magnitude. Future work will focus on upgrading SparseDR to a full-fledged path tracer and addressing the challenges that complicate the practical use of differentiable rendering, in particular the need for precise camera parameter information.

### 5.1 Order of Sections

Sections should be arranged in the following order [**?**]:

1. Main content sections (numbered)
2. Appendices (optional, numbered using capital letters)
3. Ethical statement (optional, unnumbered)
4. Acknowledgements (optional, unnumbered)
5. Contribution statement (optional, unnumbered)
6. References (required, unnumbered)

## References

[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.

[Müller *et al.*, 2022] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4), 2022.

[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.

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