Accelerating Signed Distance Functions

Pierre Hubert-Brierre[1], Eric Guérin[2], Adrien Peytavie[1], Eric Galin[1]

[1] Université Claude Bernard Lyon 1, CNRS, LIRIS, UMR 5205, France
[2] INSA Lyon, CNRS, LIRIS, UMR 5205, France

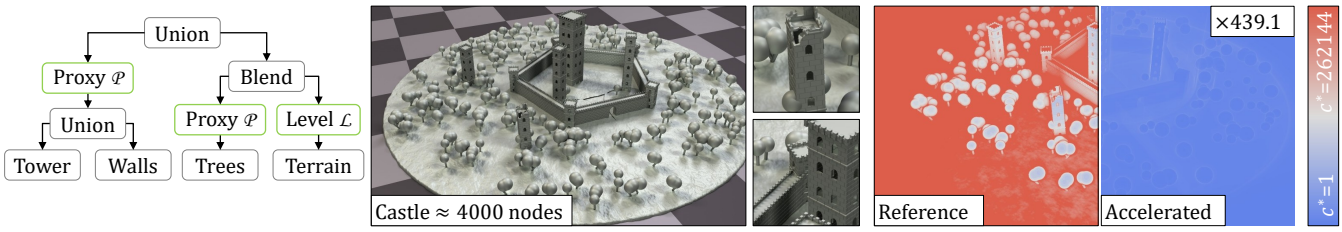

Figure 1: *Given a signed distance field f defined by a construction tree, we insert proxy and levels of detail nodes to accelerate the evaluation of f. In this scene made of 3889 nodes, we improve the performance of signed distance field queries up to $\times 400$.*

Abstract

Processing and particularly visualizing implicit surfaces remains computationally intensive when dealing with complex objects built from construction trees. We introduce optimization nodes to reduce the computational cost of the field function evaluation for hierarchical construction trees, while preserving the Lipschitz or conservative properties of the function. Our goal is to propose acceleration nodes directly embedded in the construction tree, and avoid external, accompanying data-structures such as octrees. We present proxy and continuous level of detail nodes to reduce the overall evaluation cost, along with a normal warping technique that enhances surface details with negligible computational overhead. Our approach is compatible with existing algorithms that aim at reducing the number of function calls. We validate our methods by computing timings as well as the average cost for traversing the tree and evaluating the signed distance field at a given point in space. Our method speeds-up signed distance field evaluation by up to three orders or magnitude, and applies both to ray-surface intersection computation in Sphere Tracing applications, and to polygonization algorithms.

CCS Concepts

• ***Computing methodologies*** → ***Shape modeling;***

1. Introduction

Implicit surfaces form a particularly compact and expressive class of objects for modeling various shapes. They have demonstrated their effectiveness in constructive solid geometry, fluid simulation, rendering, or surface reconstruction. Unlike explicit surface representations such as meshes, an implicit surface is indirectly defined as the zero level-set $\partial\mathcal{O} = \{\mathbf{p} \in \mathbb{R}^3, f(\mathbf{p}) = 0\}$ of a continuous function f, often referred to as a potential field [BW97].

A perennial challenge is to provide designers with tools for modeling complex shapes while maintaining an interactive processing, in particular interactive visualization. Regardless of the underlying representation, algorithms for processing implicit surfaces – such as polygonization, or ray-surface intersection – ultimately rely on one fundamental query: evaluating the field function $f(\mathbf{p})$ at a specific position $\mathbf{p}$.

Reducing the number of function queries $\#f$ is a general strategy for improving performance, *agnostic* of the underlying representation of the implicit surface. Algorithms for processing implicit surfaces take advantage of the properties of the function f and the coherency of the queries $f(\mathbf{p})$ to reduce $\#f$ and improve performance. Still, large objects featuring complex primitives based on curves or noise involve computationally demanding distance computation.

Our work is based on the observation that reducing the computational cost, denoted as $c(\mathbf{p})$, for evaluating the signed distance function $f(\mathbf{p})$ is essential for handling complex objects. This reduction requires explicit knowledge of the object's representation, such as its construction tree. Furthermore, the function queries $f(\mathbf{p})$ can be simplified when points are located far from the surface $\partial\mathcal{O}$. Therefore, we propose embedding acceleration nodes directly within the construction tree to eliminate the need for external data structures, such as octrees.

In this work, we propose a framework for accelerating the eval-

uation of $f(\mathbf{p})$ for implicit surfaces built from construction trees (Figure 1). We introduce two kinds of nodes: proxy nodes that modify f into an accelerated function $\widetilde{f}$ for points sufficiently far away from the surface $\partial\mathcal{O}$, and levels of detail nodes simplifying the geometry of the surface $\partial\widetilde{\mathcal{O}}$ and using the gradient ∇f to warp the normal and obtain classical bump mapping effects.

More precisely, our contributions are as follows: 1) We propose *proxy nodes* and continuous levels of detail nodes preserving the Lipschitz property of the field function in space, so that it can be consistently used in a modeling context; 2) We introduce a simple and efficient normal warping equivalent to traditional bump mapping for meshes, without the need for inverse coordinates. 3) We show that our methods significantly improve performances (up to three orders of magnitude speedup) in a variety of contexts. 4) A code release `https:https://github.com/orgs/Arches-Team/repositories`

In the context of rendering and shading, levels of detail operators can be completed by normal warping in the construction tree. Although we present our algorithms for signed distance fields [RMD11, RMP*24], our method can be generalized to other hierarchical models such as the BlobTree [WGG99] or function representations [PASS95]. Moreover, our approach directly applies to existing implicit surface algorithms, including ray-surface intersection or polygonization algorithms.

2. Related work

Here we review acceleration techniques for processing implicit surfaces, organized into two categories: algorithms that aim at reducing the number of function queries $\#f$, and methods trying to reduce the computational cost $c(\mathbf{p})$ of field function evaluation $f(\mathbf{p})$. Methods can be classified based on whether they are independent or agnostic of the underlying field function representation and computation, or whether they leverage knowledge of how they are built and computed.

2.1. Number of field function queries

Accelerated algorithms for implicit surfaces take advantage of the properties of the function f and the spatial coherency of the queries $f(\mathbf{p})$ to reduce the number of function queries. This is typically the case for optimizing ray-surface intersection algorithms through Lipschitz-based algorithms [KB89] such as Sphere Tracing [Har96] and variants [KSK*14, BV18, BV23], Segment Tracing [GGPP20], second-order approximation of the field function $f \circ \delta$ along the ray [AZ23], or affine arithmetic and interval arithmetic that require the closed form expression of f.

This is also the case for grid-based polygonization algorithms that adopt a continuation [WMW86] or a sweeping strategy [LC87] to reuse values computed at the vertexes of a virtual grid. A complete analysis of polygonization techniques is beyond the scope of this paper and may be found in Araujo et al. [DALJ*15]. Recent results in implicit modeling [SBS23, SRBS24] utilize the field function value $f(\mathbf{p})$ to define regions of empty spheres in space, which in turn facilitates efficient polygonization.

Existing accelerated ray implicit surface intersection techniques aim at reducing the number of field function queries along the ray. They are usually associated with a set of constraints on the field function such as global continuity, Lipschitz property or continuous differentiability.

Optimized evaluation and data-structures Acceleration techniques usually rely on additional data structures, such as octrees or bounding box hierarchies, *embedded* or *external* to the representation of the implicit object. Some models built from compactly supported primitives such as Blobs [WMW86] or particularly the BlobTree [WGG99] naturally contain a bounding volume hierarchy in their construction trees. In contrast, other models including procedurally defined signed distance fields do not benefit from the compact support property of their primitives and operators. External accompanying acceleration structures, usually bounding box hierarchies [FGW01, GPP*10] or octrees [Har96, GGPP20], may be used to eliminate empty regions on space, store additional data such as a local Lipschitz bound, or a computationally efficient local approximation of the function [PC23]. Such acceleration structures usually require a possibly intensive preprocessing step, significantly increase memory requirements, and are often model-specific. This is particularly the case for Interval Arithmetic based approaches that require the closed-form expression of nodes [Kee20, AZ23] or a neural model [SJ22], or Lipschitz techniques [GGPP20] that need specific knowledge for every primitive and operator in the construction tree.

Large implicit models pose a challenge in terms of performance, as it is not feasible to evaluate arbitrarily complex functions even on modern graphics hardware. A common approach is to prune the construction tree [FGW01] to constrain computations to a subset of the tree that affects a given region. While octrees [Har96, FGW01] necessitate the construction of an accompanying structure, bounding box hierarchies [GPP*10] can be embedded in the construction tree of the field function itself [GGPP20]. Bounding volumes can be embedded in the field function representation to reduce the evaluation cost at every step in the BlobTree model [WGG99], but this approach can only be used for compactly supported primitives.

2.2. Field function evaluation cost

Reducing the computational cost of $f(\mathbf{p})$, which may be regarded as a low-level optimization, plays a crucial role in accelerating processes such as ray-object intersection, polygonization, and collision detection. It can be used in addition to reducing the number of field evaluations, so that the benefits are combined.

Field function approximation and simplification Hierarchical spatial caching [SWG05] introduced the concept of a caching node inserted into the implicit construction tree. Caching nodes store potential field values at the vertices of a voxel grid and rely on trilinear and tri-quadratic reconstruction filters to locally approximate the potential field of a sub-tree. A lazy evaluation scheme is used to avoid expensive pre-computation. Caching distance data [SWG05] can be effective but limited in its application because of the increasing memory usage. In this context, tree reordering can be used to improve memory coherence [GDW*16]. Local piecewise approximation of the field function in a grid or an octree [PC23] is impacted by the same limitation. Barbier *et al.* [BSP*25] introduced a

method for pruning the construction tree of an input signed distance function into a region of space to simplify computations and accelerate Sphere Tracing [Har96]. The pruning algorithm reduces the number of active nodes and far-field culling replaces the whole tree with a constant expression when sufficiently far from the surface. Hansson-Söderlund *et al.* [HSEAM22] compare different strategies to optimize the field evaluation function in a grid context using a bounding volumes hierarchy and different types os solvers. Moinet *et al.* [MN25] improved the performance of Fractional Brownian Motion noise rendering by improving the stepping efficiency of Sphere Tracing considering the first and second derivatives, and treating cascaded sums as nested bounding volumes.

For ray-surface intersection algorithms, the field function along the ray $f \circ \delta(t)$ may be computed for some categories of implicit surfaces and roots found using analytic techniques [WT90] or numerical approaches such as Bézier clipping [KSN08, NN94]. For arbitrary scalar fields, approximating polynomials can be constructed along a ray, to which various root finding methods are applied [She99], or using Bernstein polynomials [KSN08]. However, increasing the degree of a polynomial arbitrarily does not necessarily yield a better approximation which is a fundamental limitation when using interpolation-based approximations in existing rendering approaches. The method proposed in [WMK24] addresses these limitations by utilizing adaptive Chebyshev polynomials proxies along each ray to perform an accurate intersection test via a robust and multi-stage searching method.

Simplifying the function expression is another method to improve computational efficiency. Screen-space pruning of construction trees [Zan24] relies on the ability to bound the influence of primitives and operators in space. Interval arithmetic [Duf92, Kee20, AZ23] may also be used to eliminate parts of the expression that do not contribute significantly in regions of space, but requires all primitives and operators to define specific additional queries, and may be constrained by the types of primitives and operations supported.

Levels of detail Cani *et al.* [CH01] pioneered the use of skeletal primitives with levels of detail by adapting the subdivision level of some curves and surfaces according to the visualization requirements. The model used relies on convolution surfaces which are computationally expensive. Barbier *et al.* [BGA04] improved and proposed a hierarchical system with an automatic levels of detail management for interactive modeling with an accelerated rendering using the BlobTree formalism [WGG99, GLA00]. They handle levels of detail with smooth transitions and tree optimizations speeding up visualization by an order of magnitude, which allows an interactive editing of the shapes.

The same procedure has been explored using a hierarchy of neural networks to learn an implicit surface at different levels of detail [TLY*21]. This method reconstructs an input black box surface, allowing it to work on various underlying representations, for instance defined by a tree as well as a neural network or even a grid.

3. Overview and notations

Recall that an implicit surface is mathematically defined as the set of points in space $\mathbf{p}$ that satisfy the equation $f(\mathbf{p}) = 0$, formally: $\partial\mathcal{O} = \{\mathbf{p} \in \mathbb{R}^3 | f(\mathbf{p}) = 0\}$.

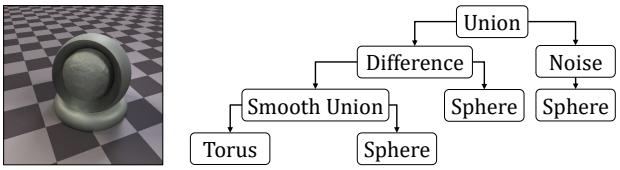

Figure 2: *Example of a hierarchical implicit surface model and its corresponding construction tree $\mathcal{T}$.*

Construction tree In this paper, we consider models defined from a hierarchical construction tree, in the spirit of the BlobTree [WGG99] or the smooth CSG tree [BSP*25]. The leaves of the tree are simple shapes implemented as signed distance primitives. Primitives are continuous 1-Lipschitz field functions, *i.e.*, $\forall \mathbf{p} \in \mathbb{R}^3, \|\nabla f(\mathbf{p})\| \leq 1$. For instance a sphere with center $\mathbf{c}$ and radius r is defined as $f(\mathbf{p}) = \|\mathbf{p} - \mathbf{c}\| - r$. More primitives can be defined as the Euclidean distance d to a skeleton $\mathcal{S}$, such as capsules or rounded curves : $f(\mathbf{p}) = d(\mathbf{p}, \mathcal{S}) - r$. Internal nodes are operators such as Boolean and blending functions that combine the field functions of their sub-trees, as well as deformations nodes including affine transformations or more complex deformations such as Barr's transformations [Bar84], which are also 1-Lipschitz. The field function value $f(\mathbf{p})$ is computed by recursively traversing the tree: primitives yield signed distance values, and the operators located at the internal nodes of the tree combine the values produced by their sub-trees. Figure 2 illustrates the construction tree of a simple object.

Implicit surfaces taxonomy Here, we recall the categorization of signed distance field models introduced in [MSLJ23] and complete it with *Lipschitz scalar fields* (Figure 3).

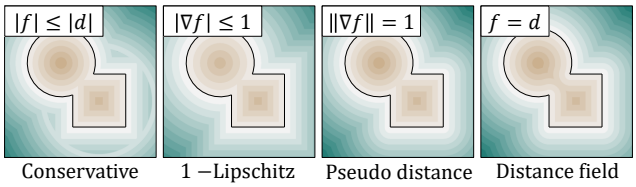

Figure 3: *Categorization of signed distance field.*

When a function $f = d$, *i.e.*, obeys the distance property to the object, it is truly a *signed distance field*. A *pseudo signed distance field* is piecewise differentiable function that only satisfies the eikonal property $\|\nabla f\| = 1$ wherever the gradient is well defined, they often arise when performing geometric editing on exact signed distance field, such as Boolean operations. *Conservative distance fields* constrain f to be smaller than the true Euclidean distance: $|f| \leq |d|$; they do not need to be continuous and may have unexpected high gradients in some regions. We introduce *Lipschitz scalar fields* to the categorization [KB89, Har96], satisfying

the Lipschitz property and $\|\nabla f\| \leq 1$. They have stronger properties than conservative functions, particularly they are continuous and have a bounded gradient.

Even though the original Sphere Tracing algorithm was proposed for Lipschitz scalar fields, it is still applicable to conservative distance fields [BSP*25]. However, some operators need the function to satisfy at least the Lipschitz property. Contrary to Boolean nodes that operate on conservative fields, smooth Boolean operators introduced in [DVOG04] require the function to be Lipschitz, or at least continuous. When applied to discontinuous conservative distance fields, smooth Boolean operators may produce fields that are not continuous in some regions, and that do not consistently define an implicit surface (see Figure 8 for rendering, and Figure 3 for a depiction of the field f).

However, the importance of this classification lies in the modeling process itself. Since most implicit operators require a true signed distance to function correctly, feeding them with a pseudo signed distance or a conservative one may introduce artifacts. Even very simple operators are affected. For instance, the inflate operator, defined as $I(\epsilon) = f(\epsilon) - k$, is impacted by the degradation of the surface quality (see Figure 4).

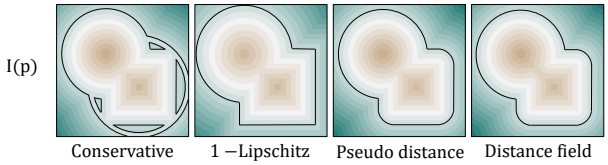

Figure 4: *Artifact created by applying the same operator on a set of signed distance fields from different classes.*

Costs Throughout this paper, we denote $c_{\mathcal{N}}$ the average cost for evaluating a node $\mathcal{N}$ in the tree, and $c(\mathbf{p})$ the average cost of a query $f(\mathbf{p})$. The cost $c(\mathbf{p})$ for evaluating the signed distance function $f(\mathbf{p})$ is calculated by summing the costs $c_{\mathcal{N}}(\mathbf{p})$ of each node encountered during the tree traversal. Without any optimization, this cost equals the total of all node costs since every node is visited. Our goal is to insert acceleration nodes into the tree in order to reduce $c(\mathbf{p})$ whenever possible.

Overview To reduce the evaluation cost $c(\mathbf{p})$, we transform the tree and introduce new nodes that branch to simpler computations of field functions whenever possible. To this end, we introduce two acceleration strategies: inserting *proxy nodes* in the tree that replace the complex evaluation of a sub-tree by simpler evaluation, and *levels of detail nodes* replacing a sub-tree by a simpler approximation according to some input accuracy requirement (Figure 5). The goal of *proxy nodes* is to modify the original tree $\mathcal{T}$ and obtain a new construction tree $\widetilde{\mathcal{T}}$ with the same surface $\mathcal{O} = \widetilde{\mathcal{O}}$, but with a more efficient signed distance field $\widetilde{f}$. Note that *proxy nodes* modify f in regions Ω entirely inside or outside of the object $\mathcal{O}$, *i.e.*. $\Omega \cap \partial\mathcal{O} = \varnothing$.

Inserting *proxy nodes* relies on the following fundamental observations: Boolean and warping nodes correctly handle conservative distance functions sub-trees, whereas smooth Boolean operators require that the sub-tree should be 1-Lipschitz. Therefore, we

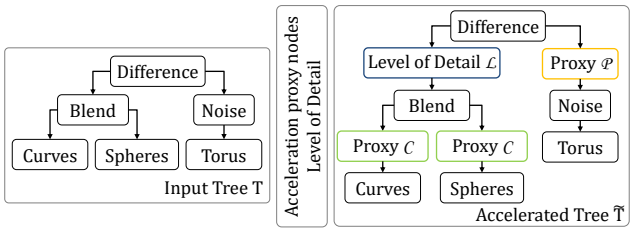

Figure 5: *Overview of the tree optimization.*

introduce two different types of nodes: conservative proxy nodes $\mathcal{P}$ (Section 4.1) yield a conservative distance field $\widetilde{f}$ that can be discontinuous, and more computationally demanding Lipschitz proxy nodes $\mathcal{C}$ (Section 4.2) that build 1-Lipschitz function $\widetilde{f}$.

The second strategy involves inserting *levels of detail nodes* in the tree to replace the computationally intensive sub-trees by simpler models. Contrary to proxy nodes that modify the definition of f in regions $\mathcal{V}$ that do not straddle the surface, and therefore do not change $\mathcal{O}$, levels of detail nodes change the surface $\partial\mathcal{O}$. We introduce *continuous levels of detail nodes* (Section 5.1) that operate on regions that partially or entirely intersect the surface and define a progressive simplification of the implicit surface. Then, we introduce normal warping (Section 5.2) as an affordable method to enhance surface details on a simpler model, while maintaining the overall consistency of the tree-based structure.

4. Field function simplification

Field function simplification nodes reduce the cost $c_{\mathcal{N}}$ of a node $\mathcal{N}$ in a given region of space $\mathcal{V}$. We introduce proxy nodes that generate a new field function $\widetilde{f}$. Simple proxy nodes produce a conservative only function, whereas improved proxy nodes yield a 1-Lipschitz function, compatible with smooth Boolean operators.

4.1. Proxy nodes

A proxy node, denoted as $\mathcal{P}$, is a unary node that replaces the definition of the function f outside of a volume $\mathcal{V}$ by a simpler, inexpensive 1-Lipschitz function, denoted as b, defined over the subdomain $\mathbb{R}^3 - \mathcal{V}$:

$$\widetilde{f}(\mathbf{p}) = \begin{cases} f(\mathbf{p}) & \text{if } \mathbf{p} \in \mathcal{V} \\ b(\mathbf{p}) & \text{otherwise} \end{cases}$$

In general, $\widetilde{f}$ is not continuous at the boundary $\partial\mathcal{V}$, although piecewise Lipschitz over $\mathcal{V}$ and $\mathbb{R}^3 - \mathcal{V}$. Therefore, it is only a conservative signed distance field and can only be used if the nodes above $\mathcal{N}$ are Boolean or warping operators. The advantage of this operator is its efficient evaluation: assuming that $c_b(\mathbf{p})$ is negligible compared to $c_f(\mathbf{p})$, this node offers accelerated queries in $\mathbb{R}^3 - \mathcal{V}$. Moreover, *proxy nodes* are also exactly 1-Lipschitz over $\mathbb{R}^3 - \mathcal{V}$, which further increases the performance of algorithms such as Sphere Tracing.

In practice, the volume $\mathcal{V}$ is defined as a simple shape, such as a sphere, an ellipsoid, or convex polyhedron [SLT*25]. The corresponding function b is defined as the Euclidean distance to the

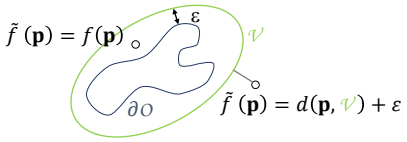

Figure 6: *Notation and field function computation for a discontinuous proxy node $\mathcal{P}$.*

volume, with an offset ε (Figure 6) that is a lower bound of the distance to the surface of the object $\partial\mathcal{O}$:

$$b = d(\mathbf{p}, \partial\mathcal{V}) + \varepsilon$$

Formally, ε can be defined as:

$$\varepsilon = \min_{\mathbf{p}\in\mathcal{V}} f(\mathbf{p})$$

In practice, an approximation of the lower bound derived from a priori knowledge of the signed distance field and when defining the region $\mathcal{V}$.

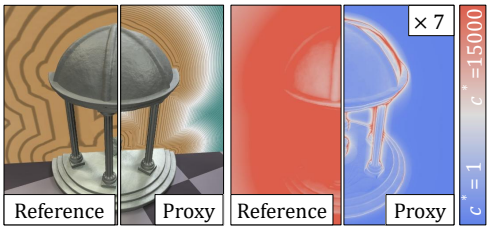

Figure 7: *Comparison of ray-object intersection performance, using Sphere Tracing, with the reference object, and using proxy nodes with an average $\times 7$ speedup. Brown lines are the isovalues of the distance field away from the surface.*

Figure 7 illustrates the field function query acceleration obtained for the *Temple* model. A capsule, a half-sphere and a cylinder proxy volume were respectively located around the pillars, roof and stairs to reduce the cost c_f of their corresponding sub-trees; a valid strategy as those elements were combined using a union node. The speed-up factor was $\approx \times 6.86$.

In some cases, proxy nodes can be also used to accelerate the evaluation of f inside $\mathcal{O}$, which is particularly effective for rendering complex transparent objects, or for computing the difference between two complex objects (see the *Cake* and *Saturn* models in Figure 22). Since $b(\mathbf{p}) < 0$ if $\mathbf{p} \in \mathcal{V}$, the accelerated function b for the interior is simply defined using the Euclidean distance to the boundary as: $b = -d(\mathbf{p}, \partial\mathcal{V}) - \varepsilon$; the negative sign handles the fact that $\mathbf{p} \in \mathcal{O}$.

An important limitation of proxy nodes is that they only generate conservative functions $\widetilde{f}$ and therefore cannot be inserted in the construction tree below nodes requiring Lipschitz or continuous fields, such as the smooth union operator. Figure 8 shows a typical rendering artifact when using smooth union operator with proxy nodes: the field function is no longer continuous in the neighborhood of $\partial\mathcal{O} = \{\mathbf{p}, f(\mathbf{p}) = 0\}$.

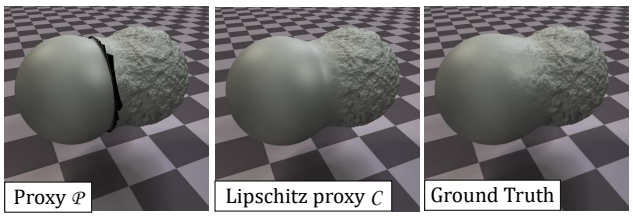

Figure 8: *Artifacts due to f discontinuity when combining a sphere and a noisy sphere with a smooth union using a discontinuous proxy. With a continuous proxy (presented in 4.2), the surface differs slightly as a side effect of the smooth union.*

4.2. Continuous simplification

Generating an accelerating 1-Lipschitz signed distance field proxy, denoted as $\mathcal{C}$, is a challenging problem. Let $v(\mathbf{p}) = d(\mathbf{p}, \mathcal{V})$ define the distance to region $\mathcal{V}$. The challenge stems not only from the variety of shapes $\mathcal{V}$ that can be used for prescribing the accelerated computation, but also from the interpolation between two functions f and v that must be handled with care.

First, we define the interpolation region as $\Omega = \mathcal{V} \oplus \mathcal{B}(\delta r) - \mathcal{V}$ where $\oplus$ is the Minkowski sum and $\mathcal{B}(\delta r)$ the sphere of radius δr (Figure 9). This allows us to easily define an interpolating, compactly supported function α of the distance in the region $\mathcal{V}$, with values in $[0,1]$, and $\alpha = 0$ over $\partial\mathcal{V}$ and $\alpha = 1$ over $\partial(\mathcal{V} \oplus \mathcal{B}(\delta r))$. Let $s : [0,1] \to [0,1]$ denote a monotonically non-decreasing function, such as identity or the smoothstep function, we define:

$$\alpha(\mathbf{p}) = s(v(\mathbf{p})/\delta r)$$

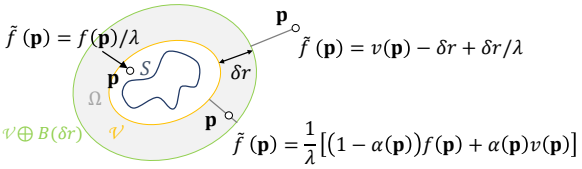

Figure 9: *Notations and computation for 1-Lipschitz proxy node.*

Interpolating between the Lipschitz field functions f and v over the domain Ω generates a smoothly continuous interpolating function.

The function $\alpha(\mathbf{p}) : \mathbb{R}^3 \to [0,1]$ weights the two argument fields and we can define the 1-Lipschitz function:

$$\widetilde{f}(\mathbf{p}) = \begin{cases} f(\mathbf{p})/\lambda & \text{if } \mathbf{p} \in \mathcal{V} \\ 1/\lambda\left[(1-\alpha(\mathbf{p}))\, f(\mathbf{p}) + \alpha(\mathbf{p})\, v(\mathbf{p})\right] & \text{if } \mathbf{p} \in \Omega \\ (v(\mathbf{p}) - \delta r) + \delta r/\lambda & \text{otherwise} \end{cases}$$

The constant λ is needed to guarantee that $\widetilde{f}$ is 1-Lipschitz. Consider the gradient of the numerator of $\widetilde{f}$ when $\mathbf{p} \in \Omega$:

$$\nabla\alpha\,(v - f) + (1-\alpha)\,\nabla f + \alpha\,\nabla v$$

The second term represents the interpolation between two gradient fields and may be bounded by the maximum of the norm of their

gradients, *i.e.*, 1 since f and v are 1-Lipschitz. Moreover, since $\alpha(\mathbf{p})$ is computed as a function of the Euclidean distance to the region $s(v(\mathbf{p})/\delta r)$, the norm of the gradient $\|\nabla\alpha\|$ is simply equal to the Lipschitz constant of the interpolating function s. Therefore, we have:

$$\lambda = 1 + \lambda_s \sup_{\Omega} |f - v|$$

One difficulty comes from the fact that the Lipschitz bound λ is in general not equal to 1, and can be large because of the term $\sup_\Omega |f-v|$, and difficult to evaluate. Fortunately, it is possible to obtain a bound under simple hypotheses.

If nothing is known about f, then in the worst case $\forall \mathbf{p} \in \mathcal{V} \oplus \mathcal{B}(\delta r)$, we have $0 \leq f \leq 2r + \delta r$ and $0 \leq b \leq \delta r$. Thus, by interval analysis, $-\delta r \leq f - v \leq 2r + \delta r$, thus:

$$\sup_{\Omega} |f - v| \leq 2r + \delta r$$

Finally, we have:

$$\lambda \leq 1 + \frac{2r + \delta r}{\delta r} \lambda_s$$

Better approximations may be obtained in particular cases with a better knowledge of f. For instance, if $f(\mathbf{c}) < 0$, *i.e.*, then the point $\mathbf{c}$ is inside the object and in the worst case $0 \leq f \leq r + \delta r$, which in turn lowers the constant λ to:

$$\lambda \leq 1 + \frac{r + \delta r}{\delta r} \lambda_s$$

This demonstrates the impact of δ_r on the node. The smaller δ_r, the bigger the Lipschitz bound. The choice of δ_r is a tradeoff between a large transition area (strong resiliense to side effect Figure 10) and a small Lipschitz bound (big steps in sphere tracing).

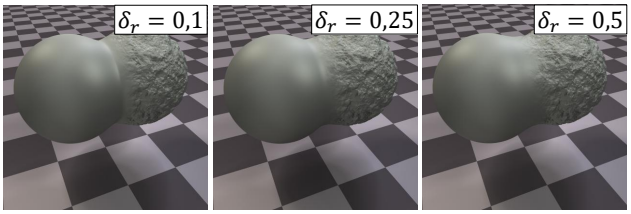

Figure 10: *The parameter δr has no direct impact on the surface itself. However, it has potential side effects caused by a degradation of the distance field. The smaller δr, the larger the resulting deformation.*

4.3. Proxy nodes placement constraint

Let $\mathcal{T}$ denote the construction tree of an object $\mathcal{O}$, we propose a method for automatically inserting proxy nodes to obtain a new tree $\widetilde{\mathcal{T}}$. The algorithm recursively traverses $\mathcal{T}$ from its root node and at every node $\mathcal{N}$, it inserts *proxy nodes* or 1-Lipschitz *proxy nodes* (denoted as $\mathcal{P}$ and $\mathcal{C}$ respectively) above $\mathcal{N}$ depending on the presence of smooth operators above it (see Figure 11). For each node $\mathcal{N}$:

1. If $\mathcal{N}$ is a smooth operator node or if it has a smooth operator above it in the tree hierarchy, then insert a 1-Lipschitz *proxy node* between $\mathcal{N}$ and its parent;
2. Otherwise insert a *proxy node* between $\mathcal{N}$ and its parent.

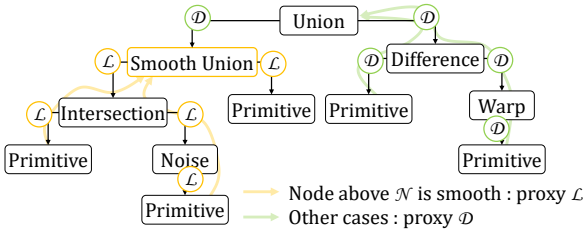

Figure 11: *Constraints when inserting a proxy node in the tree $\mathcal{T}$: yellow arc illustrate that there exists a node above candidate insertion positions, which necessitates that the node should be 1-Lipschitz; other cases are highlighted in green.*

Recall that the volumes $\mathcal{V}$ can represent any geometric object for which we can compute the Euclidean distance $d(\mathbf{p}, \mathcal{V})$. In our implementation, we meticulously construct simple trees that are congruent with the overall morphology of our surface. The automated generation of proxy nodes presents a fruitful avenue for further exploration. To enhance the process of proxy generation, we propose several strategies that may contribute to advancing the efficacy and accuracy of proxy generation. One potential approach involves generating convex shapes, which can be readily combined through Boolean and affine transformation operators. Alternatively, the simplification of complex warping nodes or the smoothing of Boolean operators may facilitate the automated attainment of coarse approximations. Lastly, the application of the Sphere Carving algorithm [SLT*25] holds significant promise in this context.

The Sphere Carving algorithm [SLT*25] generates a point cloud around the surface. This point cloud can be further processed to extract a convex object that encloses the surface. Using this reconstructed surface as a proxy enables the automatic generation of proxy nodes. With this method, we can recursively compute a set of convex shapes, each enclosing a sub-tree of the signed distance tree, and then select the proxy shape that minimizes the overall cost. The corresponding proxy node is inserted into the tree only if it decreases the original cost (see Figure 12).

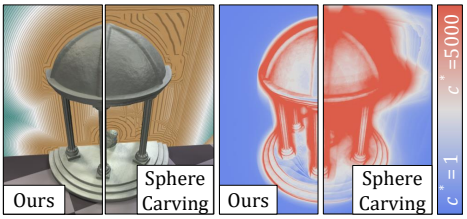

Figure 12: *Comparison of proxies obtained either manually or by using the Sphere Carving algorithm. Brown lines are the isovalues of the distance field away from the surface.*

5. Levels of detail nodes

We introduce levels of detail nodes that replace the function f with a more efficient expression l. Unlike proxy nodes, whose region of influence $\mathcal{V}$ does not intersect the surface of the sub-tree, levels of detail nodes simplify the definition of the signed distance field and yield a different surface $\partial\widetilde{\mathcal{O}}$. We first define *continuous levels of detail nodes*, denoted as $\mathcal{L}$, which define a continuous model for progressively interpolating between a high and low-resolution model. We then present an efficient normal warping algorithm, embedded in the construction tree, that allows to augment the low-resolution model with details, at a reduced computational cost.

5.1. Continuous levels of detail nodes

The goal of these nodes is to provide a continuous and seamless interpolation between two sub-trees depending on the argument query point $\mathbf{p}$. In general, levels of detail adapt to the distance between the viewer $\mathbf{e}$ and the object, denoted as $d(\mathbf{e},\mathbf{p})$. The objective is that the implicit surface of the object should continuously adapt to $d(\mathbf{e},\mathbf{p})$, rather than selecting an interpolating model according to the distance to the center $\mathbf{c}$ of the object $d(\mathbf{e},\mathbf{c})$.

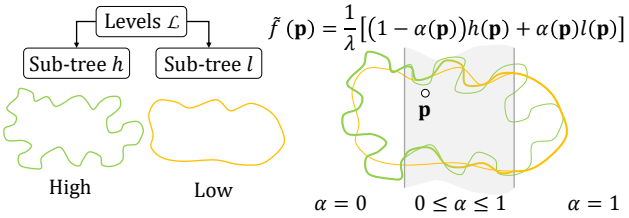

Figure 13: *Continuous levels of detail obtained by interpolating between two signed distance field; the transition region $\mathcal{V}$ depends on the view point.*

Let h and l denote the signed distance field of the high and low-resolution models respectively (Figure 13). We compute the levels of detail function $\widetilde{f}$ by considering the interpolation:

$$\widetilde{f}(\mathbf{p}) = 1/\lambda\Big[(1-\alpha(\mathbf{p}))h(\mathbf{p}) + \alpha(\mathbf{p})l(\mathbf{p})\Big]$$

Although the equation is the same as for proxy node, the difference stems from the fact that the region $\mathcal{V}$ depends on the viewer position and can straddle and even embed the surface $\partial\mathcal{O}$.

Figure 14 displays three different levels of detail for the *Temple* model. We defined the medium-resolution model by removing noise nodes, while the low-resolution model was created by replacing the carved pillars with simple cylinders and removing the decorations from the top of the roof.

A typical case for interpolating function consists in defining α as a smoothly decreasing function of the distance to the viewer:

$$\alpha(\mathbf{p}) = \begin{cases} 0 & \text{if } \|\mathbf{p}-\mathbf{e}\| \leq r \\ 1 & \text{if } \|\mathbf{p}-\mathbf{e}\| \geq r+\delta r \\ (\|\mathbf{p}-\mathbf{e}\|-r)/\delta r & \text{otherwise} \end{cases}$$

Figure 15 illustrates the continuous levels of detail on a train. Three different functions are interpolated in this example, from a highly detailed model to a simplified geometry. In the transition zone $r \leq d(\mathbf{p},\mathbf{e}) \leq r+\delta r$, the cost is higher.

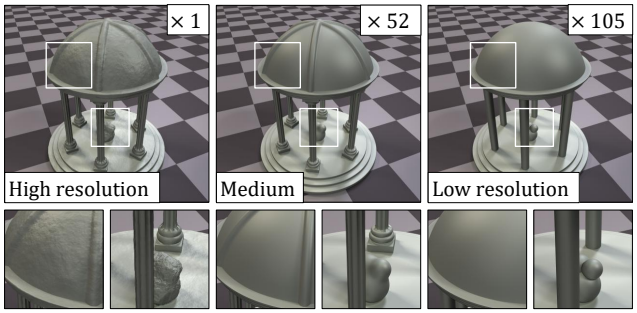

Figure 14: *The Temple model at three different levels of detail. The rendering of the low-resolution model is up to $\times 100$ faster.*

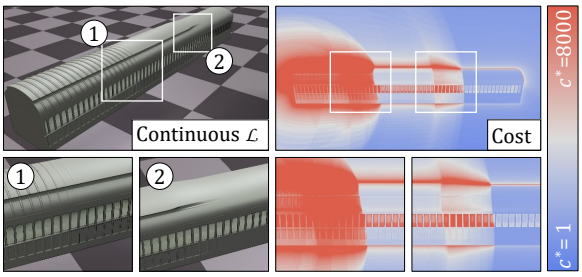

Figure 15: *Continuous levels of detail is used to progressively transition between the different functions is a continuous way. The transition zone is more expensive because two functions have to be evaluated. In this illustration, the interpolation is decoupled from the viewpoint in order to clearly highlight the LOD transition.*

5.2. Normal warping

Without computationally intensive geometric details, simplified geometry often looks smooth. A classic technique for meshes consists in using a normal map to add surface details. This approach requires a global or semi-global [GGLW25] parametrization of the implicit surface.

Taking inspiration from [TW99], we introduce an inexpensive *normal warping* approximation strategy adapted to the tree representation of the object with levels of detail. The key aspect of our approach is speed: we avoid any parametrization of the surface, and avoid performing several steps following the gradient ∇f.

Recall that the normal of a low level of detail implicit surface $\partial\widetilde{\mathcal{O}}$ with its field function $\widetilde{f}$ is defined as $\widetilde{\mathbf{n}}(\mathbf{p})$. The objective of normal warping is to map the normal details $\mathbf{n}$ of a detailed signed distance field f to the smoother signed distance field $\widetilde{f}$.

The strategy consists in projecting $\mathbf{p}$ on the surface $\partial\mathcal{O}$ to a new point $\pi(\mathbf{p})$ by following the gradient $\nabla f(\mathbf{p})$ and use the warped normal $\widetilde{\omega}(\mathbf{p}) \propto \nabla f \circ \pi(\mathbf{p})$. This can be interpreted as doing a gradient descent to find the closest point to the surface $\partial\mathcal{O}$ (see Figure 16). This gradient descent can be performed using n steps, 1

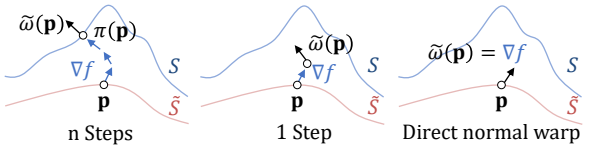

Figure 16: *Computation of the warped normal $\widetilde{\omega}(\mathbf{p})$: following the gradient ∇f to project $\mathbf{p}$ onto $\partial\mathcal{O}$ is computationally intensive and prone to errors. Instead, we perform only one step and compute $\widetilde{\omega}(\mathbf{p}) \propto \nabla f \circ (\mathbf{p} + f(\mathbf{p})\widetilde{\mathbf{n}}$, or even directly $\widetilde{\omega}(\mathbf{p}) \propto \nabla f(\mathbf{p})$.*

step or 0 step. In some cases, following the gradient of the reference signed distance field f may lead to visible discontinuities in the normal map. Moreover, it requires several evaluations of the gradient ∇f, which is computationally intensive. Our experiments demonstrate that using only one step or even evaluating the gradient directly $\nabla f(\mathbf{p})$ yields correct approximations provided $\partial\widetilde{\mathcal{O}}$ is not too far away from $\partial\mathcal{O}$ (see Figure 17).

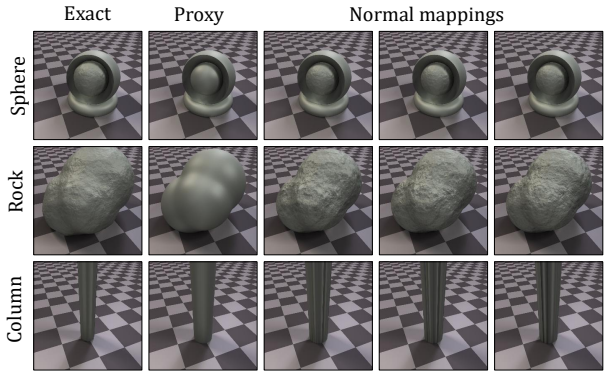

Figure 17: *Normal warping on several objects, from left to right: the reference object, the simplified object, and its augmented version with the normal warping applied with 3, 1, and 0 steps. Notice that the last column, despite being the simplest method, yield faster and sometimes better results.*

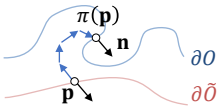

Figure 18: *Penetrating normals in some extreme concave settings.*

In some rare cases, particularly when the implicit surface $\partial\mathcal{O}$ shows small concave parts near the surface, the normal warping algorithm could yield a normal $\widetilde{\mathbf{n}}$ penetrating inside the surface $\partial\widetilde{\mathcal{O}}$ (see Figure 18) which in turn may yield shadowing and refraction inconsistencies. This case can be easily detected and solved as follows: if $\mathbf{n} \cdot \widetilde{\mathbf{n}} < 0$, then we use the geometric normal $\mathbf{n}$ (see Figure 19).

Figure 20 shows a typical example of a continuous level of detail model with normal warping. The function f of the high-resolution model generates the geometric cracks produced, whereas the low-resolution model is defined using simple box function, decorated with normal warping produced by ∇f. The continuous levels of detail node increases speed by a $\times 4.5$ factor. However, this method

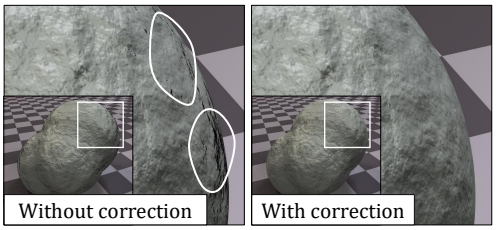

Figure 19: *Comparison of the normal map with and without correction of the normal.*

may produce artifacts if the normal map is not well aligned with the model's geometry (Figure 21).

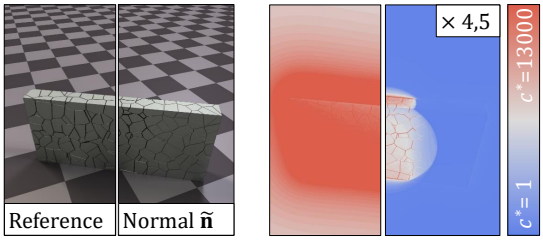

Figure 20: *The wall model with two levels of detail: the red region at the center corresponds to the region of space where the interpolation requires the computation of the functions of both high and low resolution models.*

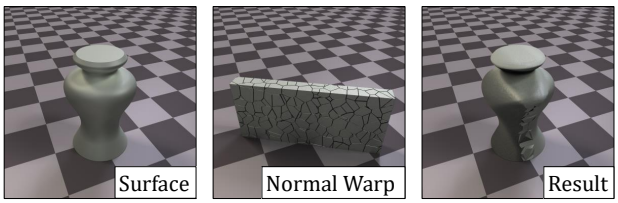

Figure 21: *The vase model is shown on the left, the wall model used for the normal warp in the middle, and on the right the warped vase, where artifacts appear due to the different topology of the two objects.*

Finally, note that defining the function to be $\alpha(\mathbf{p})$ constant, *i.e.*, not depend on $\mathbf{p}$, is equivalent to traditional levels of detail.

6. Results

We implemented the hierarchical construction tree model and our algorithm in C++ and Qt. All the objects were coded in C++, and then automatically converted into a GLSL function, directly streamed to real-time shading and off-line global illumination rendering Widgets. CPU timings were obtained on a desktop computer equipped with Intel® Core *i*9, clocked at 3 GHz with 32 GB of RAM, GPU timings were clocked on an nVidia® GeForce 4070.

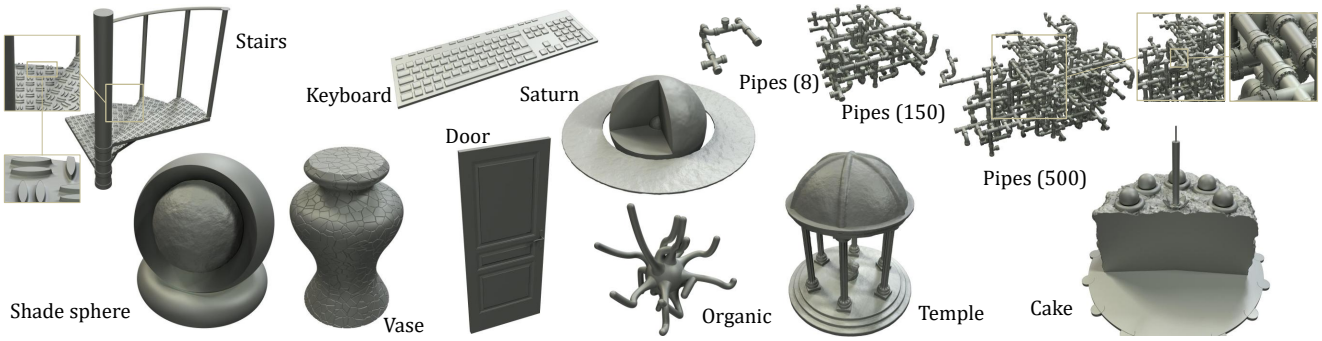

Figure 22: *Some test models from our database.*

Model	Figure	#$\mathcal{N}$	Speedup CPU			Speedup GPU		
			Proxy	LOD	Proxy+LOD	Proxy	LOD	Proxy+LOD
Shade sphere	Figure 22, Figure 2	8	×6.56	×14.8	×19.7	×2.04	×22.9	×20.1
Cake	Figure 22	101	×12.5	×1.21	×11.8	×2.75	×2.85	×4.18
Door	Figure 22	175	×20.4	×4.86	×51	×1.82	×3.2	×6.09
Vase	Figure 22	9	×4.27	×1.02	×4.38	×2.06	×8.71	×9.12
Stair	Figure 22	244	×34	×11.8	×35.5	×12.1	×48.8	×75.3
Saturn	Figure 22	14	×2.46	×10.7	×9.08	×2.94	×72.9	×58.9
Wall	Figure 1, Figure 20	10	×27.1	×74.3	×73.9	×2.6	×493	×382
Temple	Figure 22, Figure 14	63	×5.37	×3.23	×6.42	×6.86	×24.6	×26
Keyboard	Figure 22	340	×9.78	×1.9	×11	×2.1	×1.8	×3.52
Castle	Figure 1, Figure 26	3889	×19.7	×1.39	×22.4	×117.4	×125	×439.1
Pipes (8)	Figure 22	3137	×133	×27.2	×148	×3.57	×6.2	×5.66
Pipes (150)	Figure 22	63746	×1330	×27.1	×1940	×56.2	×43.1	×206
Pipes (500)	Figure 22	174790	×4310	×25.2	×5960	×161	×30.9	×439
Organic	Figure 22	92	×8.35	×1.04	×3.51	×1.97	×1.5	×2.52

Table 1: *Comparison of speedups with and without optimization nodes, in CPU and GPU implementations. CPU speedup is obtained by measuring the time needed to query samples in a box surrounding the object. GPU speedup is obtained by measuring the time needed to render the objects on GPU using sphere tracing.*

6.1. Performance

Signed distance field evaluation cost Traditionally, implicit surface algorithms are designed to minimize the number of field function queries, represented as $\#f$. However, these queries tend to be more computationally intensive due to the increased complexity of the construction tree $\mathcal{T}$ and its corresponding signed distance function f. Additionally, different primitives and operators exhibit varying performance characteristics.

Figure 23 and 24 show some primitives and operators along with their relative costs. Notably, primitives with axes of revolution or symmetries (planar or axial) are efficient compared to more complex geometric primitives such as tubes created by offsetting quadric or cubic curves. Table 2 (see Appendix) reports the computational time for evaluating different primitives and operators with 10^7 random points taken in the bounding cube, and their relative cost. Note that all the images shown throughout the paper indicate a color ramp for each pixel that uses this reference cost evaluation. The sum of all costs are cumulated along the ray.

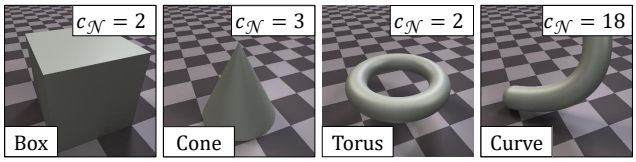

Figure 23: *Example of a primitives and their relative cost.*

Timings Table 1 reports the average speedup we obtain using either the proxy nodes, levels of detail nodes, and both of them. The corresponding cost images are shown in Figure 25. In most cases,

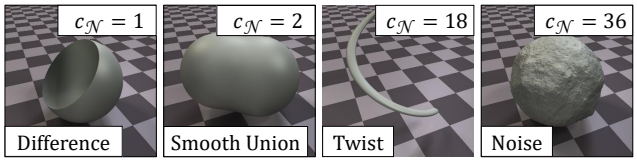

Figure 24: *Example of a operators and their relative cost.*

combining both nodes yields the best results, in both CPU and GPU setups. When used in a realistic rendering setup with three bouncing rays, the speedup is still comparable but is limitted by an increase of queries near the surface.

In some cases, as shown in Table 1, it is preferable to use LOD without proxy optimization, because the assumption made in Section 4.1 is no longer valid. We assumed that the cost of evaluating the proxy was negligible compared to the cost of evaluating the object. However, in certain cases, the LOD is simple enough to be used directly as a proxy. In such situations, the assumption breaks down because both objects are effectively the same. When combining LOD with proxy optimization, the same object is evaluated twice, which decreases performance.

6.2. Comparison with other methods

We focus on techniques that directly process the construction tree and that do not require an external accompanying data-structure such as a bounding box hierarchy or an octree. Thus, we compare our method to model-oriented approaches [MN25] and techniques based on space-decomposition [Kee20, BSP*25]. Moreover, even though we are trying to simplify the initial signed distance field, our method is not suited for signed distance reconstruction (like [FC24]). Indeed we actually show the raw distance field and the LOD one, both of them need to be cleaned out.

Recently, Moinet *et al.* [MN25] proposed a specific acceleration technique improving the stepping efficiency of Sphere Tracing for objects built from Fractional Brownian Motion (FBM)-based noise. The method requires the analysis of the the first and second derivatives, and interprets the cascaded sums of signals as nested bounding volumes. While probably less efficient for the very reason that we do not make as many hypotheses, our approach is more general and can also handle objects created with noise operators with a significant acceleration (see Figure 1 and Figure 22).

Another category of methods aims at simplifying the signed distance field expression in prescribed regions of space. Keeters [Kee20] relies on interval arithmetic to simplify the closed-form expression of the function f in a restricted region of space. Barbier *et al.* [BSP*25] follow a similar strategy but take advantage of 1-Lipschitz property of the primitives to bound their range of values in space. Our approach is more straightforward to implement than the interval arithmetic strategy: primitives and operators operate as black boxes, and the *levels of detail* and *proxy* operators can be directly implemented in any hierarchical construction tree model. Compared to [BSP*25], we do not need a two-pass evaluation of the tree to prune the model: our approach directly operates with standard Sphere Tracing [Har96] and variants [KSK*14].

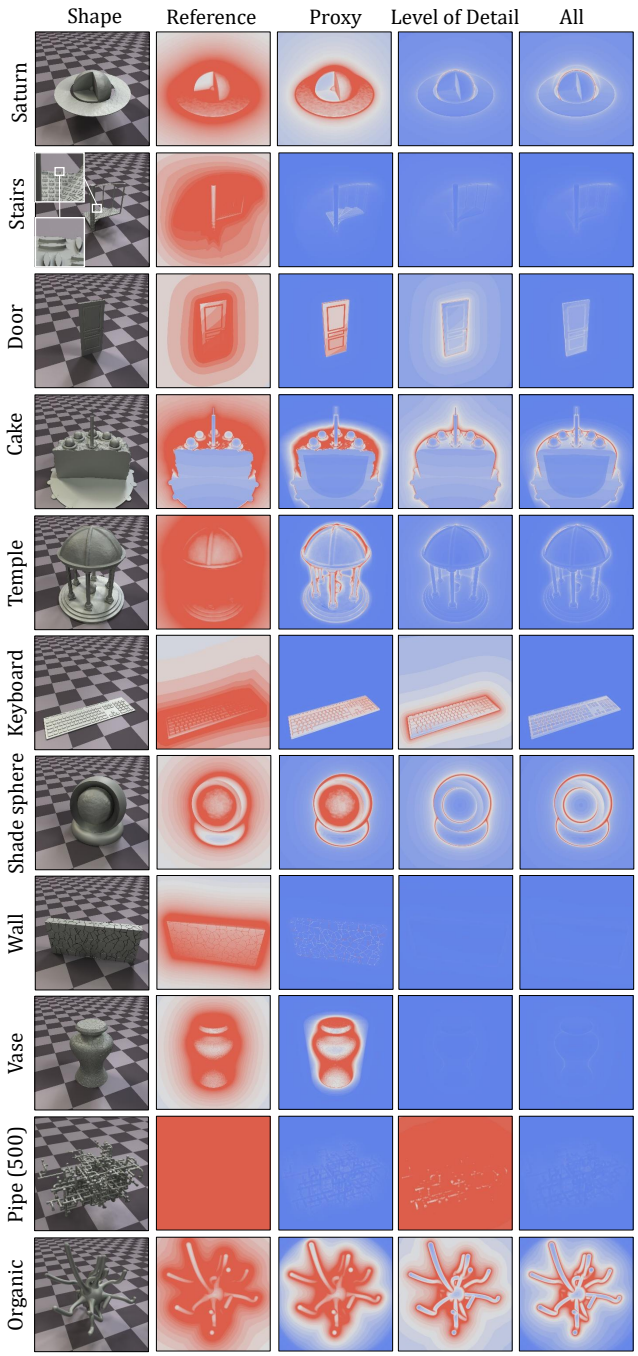

Figure 25: *Some models in our database, along with the color depiction of the reduction of the cost using proxy and levels of detail nodes.*

Moreover, our method correctly handles affine transformations and warping nodes (see Figure 1). Note that our approach is complementary as the proxy and level of detail nodes provide sufficient information to be, in theory, combined with tree-pruning.

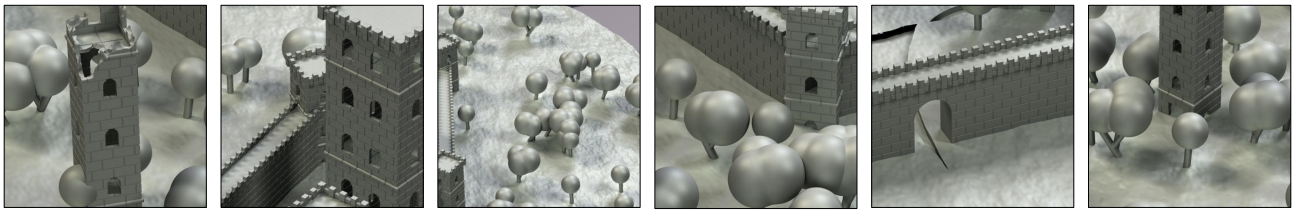

Figure 26: *Closeups of some models in the large Landscape model featuring* ≈ 3889 *primitives. The terrain is optimized using a continuous levels of detail operator, towers and walls use proxy nodes combined with levels of detail operators and normal warping, trees with smooth union operators use 1-Lipschitz proxy nodes.*

6.3. Limitations

Our method also has limitations. While it improves the evaluation of the function f, adding nodes to the construction tree makes the model more memory-intensive. For example, the Castle model originally contains $3,889$ nodes, which increases to $4,880$ after inserting proxy nodes and LOD nodes. Moreover, when using automatic proxy generation, this number may rise significantly with little benefit, as the algorithm attempts to add a proxy at every node.

For LOD and normal warping optimizations, additional memory is required to store both the detailed tree and the LOD tree. Furthermore, in these cases, there is no automatic way to generate the LOD nodes, which must instead be created manually. If done carelessly, this may lead to severe artifacts (see Figure 21).

7. Conclusion

In this paper, we propose new nodes for accelerating field function queries in complex signed distance field models constructed from construction trees. Our method can handle complex models featuring any kind of signed distance field primitives combined with Boolean, blending, warping operators. These proxy nodes can be inserted into the construction tree to lower the overall evaluation cost while effectively preserving the surface details. In addition, our levels of detail operators can simplify the surface to speed up ray-surface intersection or polygonization and resort to normal warping for adding details during shading, a typical tradeoff between accuracy and efficiency. Within this framework, procedural normal mapping nodes are utilized to simulate fine details, although this requires the storage of two sub-trees representing both high and low-resolution models. Moreover, our approach is compatible with other algorithms aimed at reducing the number of function calls.

Finally, there is a need for an algorithm that optimizes the overall tree structure. A better automatic generation of proxy and levels of detail nodes could be a potential avenue for future work, along with the automatic balancing of the tree. Another promising direction to explore is determining the minimal cost for a signed distance tree by strategically adding a well-suited proxy nodes and a levels-of-detail nodes at appropriate locations within the tree. Furthermore, it could be beneficial to generate the proxy object and the simplified object in an optimal manner. Lastly one could try out different interpolating function for the LOD transition.

Acknowledgments

This work was supported by the project EOLE ANR-23-CE56-0008 of Agence Nationale de la Recherche, France.

References

[AZ23] AYDINLILAR M., ZANNI C.: Forward inclusion functions for ray-tracing implicit surfaces. *Computers & Graphics 114* (2023), 190–200. 2, 3

[Bar84] BARR A. H.: Global and local deformations of solid primitives. *SIGGRAPH Computer Graphics 18*, 3 (1984), 21–30. 3

[BGA04] BARBIER A., GALIN E., AKKOUCHE S.: Complex skeletal implicit surfaces with levels of detail. In *Proceedings of WSCG* (2004), pp. 35–42. 3

[BSP*25] BARBIER W., SANCHEZ M., PARIS A., MICHEL E., LAMBERT T., BOUBEKEUR T., PAULIN M., THONAT T.: Lipschitz pruning: Hierarchical simplification of primitive-based SDFs. *Computer Graphics Forum 44*, 2 (2025), e70057. 2, 3, 4, 10

[BV18] BÁLINT C., VALASEK G.: Accelerating Sphere Tracing. In *Eurographics - Short Papers* (2018), Diamanti O., Vaxman A., (Eds.), The Eurographics Association. 2

[BV23] BÁN R., VALASEK G.: Automatic step size relaxation in sphere tracing. In *Eurographics - Short Papers* (2023), Babaei V., Skouras M., (Eds.), The Eurographics Association. 2

[BW97] BLOOMENTHAL J., WYVILL B. (Eds.): *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. 1

[CH01] CANI M.-P., HORNUS S.: Subdivision-curve primitives: a new solution for interactive implicit modeling. In *Proceedings of Solid Modeling* (2001), pp. 82–88. 3

[DALJ*15] DE ARAÚJO B. R., LOPES D. S., JEPP P., JORGE J. A., WYVILL B.: A survey on implicit surface polygonization. *ACM Computing Surveys 47*, 4 (2015), 60:1–60:39. 2

[Duf92] DUFF T.: Interval arithmetic recursive subdivision for implicit functions and constructive solid geometry. *SIGGRAPH Computer Graphics 26*, 2 (1992), 131–138. 3

[DVOG04] DEKKERS D., VAN OVERVELD K., GOLSTEIJN R.: Combining csg modeling with soft blending using lipschitz-based implicit surfaces. *The Visual Computer 20*, 6 (2004), 380—-391. 4

[FC24] FENG N., CRANE K.: A heat method for generalized signed distance. *ACM Transactions on Graphics (TOG) 43*, 4 (2024), 1–19. 10

[FGW01] FOX M., GALBRAITH C., WYVILL B.: Efficient use of the BlobTree for rendering purposes. In *2001 International Conference on Shape Modeling and Applications* (2001), IEEE Computer Society, p. 306. 2

[GDW*16] GRASBERGER H., DUPRAT J.-L., WYVILL B., LALONDE P., ROSSIGNAC J.: Efficient data-parallel tree-traversal for BlobTrees. *Computer-Aided Design 70* (2016), 171–181. 2

[GGLW25] GENEST B., GUETH P., LEVALLOIS J., WANG S.: Implicit uvs: Real-time semi-global parameterization of implicit surfaces. *Computer Graphics Forum 44*, 2 (2025). 7

[GGPP20] GALIN E., GUÉRIN E., PARIS A., PEYTAVIE A.: Segment tracing using local lipschitz bounds. *Computer Graphics Forum 39*, 2 (2020), 545–554. 2

[GLA00] GALIN E., LECLERCQ A., AKKOUCHE S.: Morphing the BlobTree. *Computer Graphics Forum 19*, 2 (2000), 257–270. 3

[GPP*10] GOURMEL O., PAJOT A., PAULIN M., BARTHE L., POULIN P.: Fitted BVH for Fast Raytracing of Metaballs. *Computer Graphics Forum 29*, 2 (2010), 7–288. 2

[Har96] HART J. C.: Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer 12*, 10 (1996), 527–545. 2, 3, 10

[HSEAM22] HANSSON-SÖDERLUND H., EVANS A., AKENINE-MÖLLER T.: Ray tracing of signed distance function grids. *Journal of Computer Graphics Techniques 11*, 3 (2022), 94–113. 3

[KB89] KALRA D., BARR A. H.: Guaranteed Ray Intersections with Implicit Surfaces. *SIGGRAPH Computer Graphics 23*, 3 (1989), 297–306. 2, 3

[Kee20] KEETER M. J.: Massively parallel rendering of complex closed-form implicit surfaces. *ACM Transactions on Graphics 39*, 4 (2020). 2, 3, 10

[KSK*14] KEINERT B., SCHÄFER H., KORNDÖRFER J., GANSE U., STAMMINGER M.: Enhanced sphere tracing. In *Proceedings of Smart Tools & Apps for Graphics* (Cagliari, Italy, 2014), Eurographics Association. 2, 10

[KSN08] KANAMORI Y., SZEGO Z., NISHITA T.: GPU-based fast ray casting for a large number of metaballs. *Computer Graphics Forum 27*, 2 (2008), 351–360. 3

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (1987), 163–169. 2

[MN25] MOINET M., NEYRET F.: Fast sphere tracing of procedural volumetric noise for very large and detailed scenes. *Computer Graphics Forum 44*, 2 (2025). 3, 10

[MSLJ23] MARSCHNER Z., SELLÁN S., LIU H.-T. D., JACOBSON A.: Constructive solid geometry on neural signed distance fields. In *SIGGRAPH Asia 2023 Conference Papers* (New York, NY, USA, 2023), Association for Computing Machinery. 3

[NN94] NISHITA T., NAKAMAE E.: A method for displaying metaballs by using Bézier clipping. *Computer Graphics Forum 13*, 3 (1994), 271–280. 3

[PASS95] PASKO A., ADZHIEV V., SOURIN A., SAVCHENKO V.: Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer 11*, 8 (1995), 429–446. 2

[PC23] PUJOL E., CHICA A.: Adaptive approximation of signed distance fields through piecewise continuous interpolation. *Computers and Graphics 114* (2023), 337–346. 2

[RMD11] REINER T., MÜCKL G., DACHSBACHER C.: Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions. *Computer & Graphics, Proceedings of Shape Modeling International 35*, 3 (2011), 596–603. 2

[RMP*24] RISO M., MICHEL E., PARIS A., DESCHAINTRE V., GAILLARD M., PELLACINI F.: Direct manipulation of procedural implicit surfaces. *ACM Transaction on Graphics* (2024). 2

[SBS23] SELLÁN S., BATTY C., STEIN O.: Reach for the spheres: Tangency-aware surface reconstruction of SDFs. In *SIGGRAPH Asia 2023 Conference Papers* (2023). 2

[She99] SHERSTYUK A.: Fast ray tracing of implicit surfaces. *Computer Graphics Forum 18*, 2 (1999), 139–147. 3

[SJ22] SHARP N., JACOBSON A.: Spelunking the deep: guaranteed queries on general neural implicit surfaces via range analysis. *ACM Transactions on Graphics 41*, 4 (2022). 2

[SLT*25] SCHOTT H., LAMBERT T., THONAT T., GUÉRIN E., GALIN E., PARIS A.: Sphere carving: Bounding volumes for signed distance fields. *ACM Transactions on Graphics 44*, 4 (2025). 4, 6

[SRBS24] SELLÁN S., REN Y., BATTY C., STEIN O.: Reach for the arcs: Reconstructing surfaces from SDFs via tangent points. In *SIGGRAPH 2024 Conference Papers* (2024). 2

[SWG05] SCHMIDT R., WYVILL B., GALIN E.: Interactive implicit modeling with hierarchical spatial caching. In *International Conference on Shape Modeling and Applications* (2005), pp. 104–113. 2

[TLY*21] TAKIKAWA T., LITALIEN J., YIN K., KREIS K., LOOP C., NOWROUZEZAHRAI D., JACOBSON A., MCGUIRE M., FIDLER S.: Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 11353–11362. 3

[TW99] TIGGES M., WYVILL B.: A field interpolated texture mapping algorithm for skeletal implicit surfaces. *Proceedings of Computer Graphics International* (1999), 25–33. 7

[WGG99] WYVILL B., GUY A., GALIN E.: Extending the CSG tree – warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (1999), 149–158. 2, 3

[WMK24] WINCHENBACH R., MÖLLER M., KOLB A.: Lipschitz-agnostic, efficient and accurate rendering of implicit surfaces. *The Visual Computer 10*, 1 (2024), 7925–7944. 3

[WMW86] WYVILL G., MCPHEETERS C., WYVILL B.: Data structure for soft objects. *The Visual Computer 2*, 4 (1986), 227–234. 2

[WT90] WYVILL G., TROTMAN A.: Ray-tracing soft objects. In *Computer Graphics International* (1990), Springer Japan, pp. 469–476. 3

[Zan24] ZANNI C.: Synchronized tracing of primitive-based implicit volumes. *ACM Transactions on Graphics 44*, 1 (2024). 3

Appendix

Primitive cost

Table 2 report the cost (computational time) for evaluating different primitives and operators. Timings were obtain by evaluating the signed distance field of the node with 10^7 random points a cube twice as big as the primitive. The relative cost c^* was computed using the half-space (plane) primitive as the reference.

Primitive	c	c^*	Operator	c	c^*
Half space	198	1	Boolean	105	0.5
Sphere	378	2	Smooth Boolean	326	1
Box	382	2	Scaling	308	1
Disc	484	2	Rotation	507	3
Capsule	496	3	Translation	166	0.8
Torus	521	3	Affine	852	4
Cylinder	683	3	Symmetry	645	3
Curve[1]	3775	19	Twisting	1714	9
Curve[2]	79845	403	Cloning	710	4

Table 2: *Cost and relative cost for some primitives and operators.* [1] *Quadric,* [2] *Cubic*