Lipschitz Pruning: Hierarchical Simplification of Primitive-Based SDFs

SUBMISSION ID: 1201

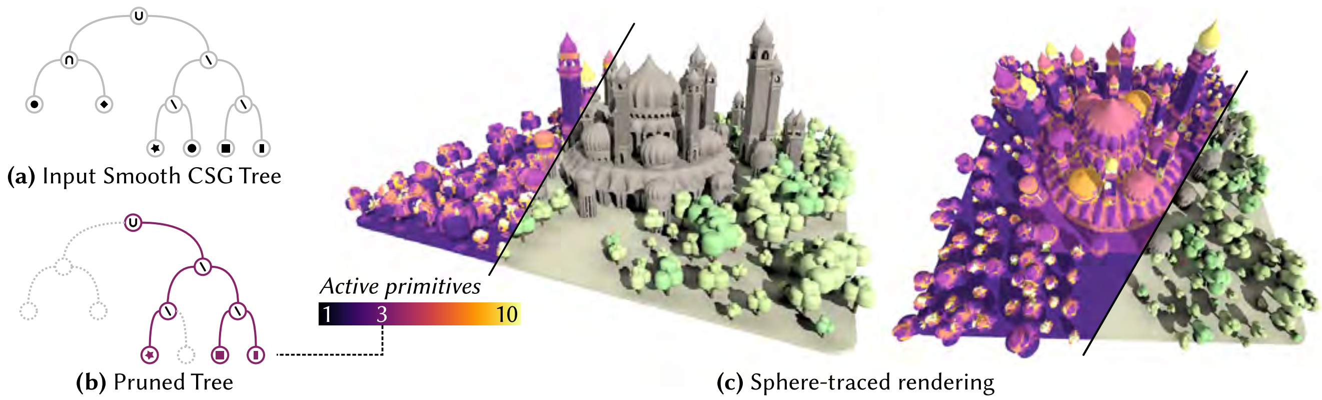


Figure 1: *We introduce an efficient primitive pruning algorithm for CSG trees that encode Signed Distance Fields (SDFs). For a region of space, we reduce binary operators to one of their operands, or completely replace sub trees with constant expression, which greatly reduces the complexity of the SDF. We achieve speedups up to two orders of magnitude when compared to classical sphere tracing, for instance we reach* $\times 629$ *on this scene made of* 6023 *nodes.*

Abstract

Rendering tree-based analytical Signed Distance Fields (SDFs) through sphere tracing often requires to evaluate many primitives per tracing step, for many steps per pixel of the end image. This cost quickly becomes prohibitive as the number of primitives that constitute the SDF grows. In this paper, we alleviate this cost by computing local pruned trees that are equivalent to the full tree within their region of space while being much faster to evaluate. We introduce an efficient hierarchical tree pruning method based on the Lipschitz property of SDFs, which is compatible with hard and smooth CSG operators. We propose a GPU implementation that enables real-time sphere tracing of complex SDFs composed of thousands of primitives with dynamic animation. Our pruning technique provides significant speedups for SDF evaluation in general, which we demonstrate on sphere tracing tasks but could also lead to significant improvement for SDF discretization or polygonization.

1. Introduction

Signed distance fields (SDFs) are a powerful surface representation for modeling and animating shapes of arbitrary topology. As opposed to meshes, they naturally provide non-trivial modeling operators such as Boolean operations, smooth blending and offset surfaces. They also come in an incredibly compact format, as one only needs to store the expression of the function representing the surface. The ability to build complex shapes using SDFs makes them the representation of choice for easy-to-use 3D modeling software, which have seen a recent rise of popularity [Wom22, Mag22, Neo23].

More precisely, SDFs are implicit surfaces representing shapes as the 0-level set of a scalar function $f : \mathbb{R}^3 \to \mathbb{R}$ with the following properties:

- $|f(\mathbf{p})|$ is the exact distance from $\mathbf{p}$ to the surface.
- $f(\mathbf{p})$ is positive if $\mathbf{p}$ is outside the shape, and negative if inside.

We consider the more general family of lower-bound (often quoted *conservative*) SDFs, for which $|f(\mathbf{p})|$ is a lower bound on the exact distance from $\mathbf{p}$ to the surface. A key property of lower-bound SDFs is that they are 1-Lipschitz: $\|\nabla f\| \leq 1$. In this paper, we refer to SDFs as the wider family of lower-bound and exact SDFs, which our method can process similarly.

The implicit nature of SDFs makes their efficient rendering challenging. One can either resort to indirect polygonization, or direct ray-tracing. Polygonization techniques first discretize the SDF over a finite domain – usually a regular grid or an octree – and extract a polygon mesh from this discretization. While these fit in traditional graphics pipeline, they are not without issues: on top of possible topological errors, sharp and detailed features are difficult to capture without a very fine grid, leading to a high memory footprint.

On the other hand, ray-tracing directly computes the intersection between a camera ray and the isosurface. Sphere-tracing [Har96] has been the go-to algorithm for the past three decades, thanks to its simplicity and its embarrassingly parallel nature. However, it remains computationally expensive, scaling linearly with the number of pixels as well as with the SDF complexity. Several approaches have been proposed to improve sphere tracing performance. These includes reducing the number of marching steps through heuristics or local Lipschitz bound computation [KSK*14, GGPP20, BV18, AZ23], but at the cost of increasing single step computations, mitigating the resulting gain. Alternatively, reducing the complexity of the function evaluation can be performed by pruning its expression, retaining only the instructions that contribute to the distance value for a given region of space. However, existing algorithms suffer from the significant overhead of an interval arithmetic interpreter [Kee20] or are limited to implicit functions with local support [FGW01] and thus not applicable to SDFs. We fill this gap in the literature by providing a pruning algorithm that is tailored to the SDF representation and exploits the 1-Lipschitz property to conservatively prune the input tree without the overhead of interval arithmetic, and achieves similar pruning efficiency. Our contributions are as follows:

1. A spatially-varying pruning algorithm that reduces the number of primitives and operators evaluated at a given point in space, compatible with CSG and smooth operators.
2. A hierarchical scheme for this pruning, compatible with GPU implementation, that allows our method to scale to large scenes.
3. A far-field culling method that reduces subtrees to constant distances, which in turn provides even faster evaluation.

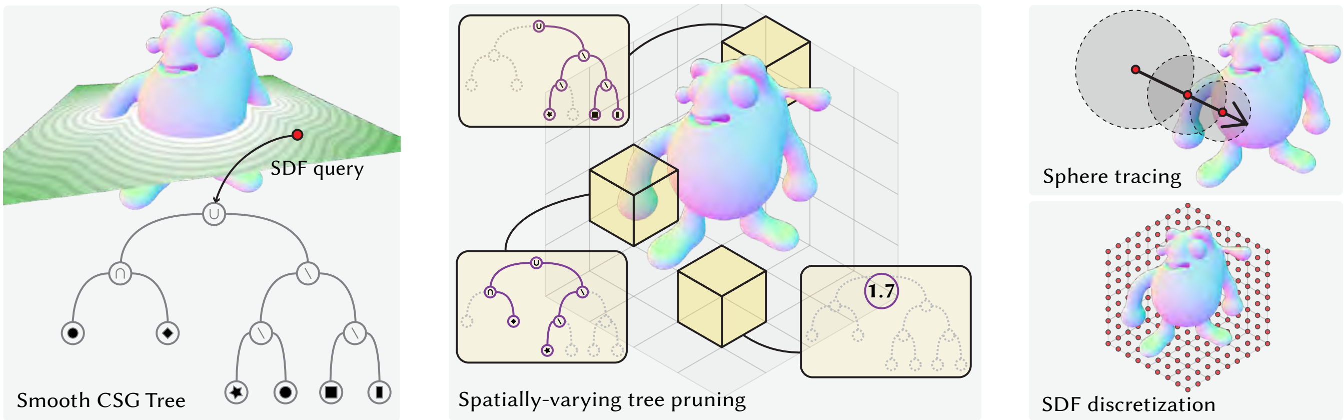
Our contributions are conceptually simple, easy to implement, and allow to speedup the function evaluation in scenarios such as sphere tracing and discretization, for SDFs scenes made of thousands of primitives and operators.

2. Related Work

In this section, we review existing techniques for ray tracing implicit surfaces, and particularly SDFs. They often come in two main flavors: they can be either *analytic*, i.e. the mathematical expression of the distance function is evaluated at runtime, or they can be *discrete*, i.e. the function is represented by a finite set of samples, and their corresponding values are interpolated at runtime to provide a distance. Even though our method is developed for analytic SDFs, we also briefly discuss discrete representations when relevant.

Sphere tracing and variants. Sphere tracing [Har96] is the standard algorithm to raytrace signed distance fields. The core idea is to advance along the ray by the distance evaluated at the current point, with the guarantee to not miss any intersection with the surface thanks to the unbounding sphere property. A common issue is that rays that get close to the shape require a significant number of steps to reach or discard the intersection (the *grazing rays* problem). Several variants were created to reduce the number of steps needed to compute the ray-surface intersection. Keinert et al's relaxed sphere tracing [KSK*14] extends the step size by a constant factor, using the fact that no intersection can be missed if the empty spheres between two consecutive steps overlap. Bálint and Valasek's enhanced sphere tracing [BV18] is built on the same idea, but rather than scaling the step size by a fixed factor, they compute a local linear approximation of the SDF and use this approximation to compute an optimal step size. Bàn and Valasek [BV23] extend this approach by using exponential averaging of the slope. Segment tracing [GGPP20] computes local Lipschitz bounds along ray segments, which allows to take longer steps without backtracking. While the first two approaches can process black-box SDFs, segment tracing require knowledge of the underlying Blob-Tree [WGG99], and is limited to primitives and operators where local Lipschitz bounds can be computed. Sphere tracing may also be viewed as the process of bounding the distance function by two affine functions with slopes equal to the (global or local) Lipschitz constant. In this spirit, forward inclusion functions [AZ23] generalize the standard Lipschitz bounds with asymmetric (lower and upper) and higher-order bounding functions. These bounds can be computed either by analyzing the CSG tree or using interval arithmetic [Duf92].

Primitive pruning. A common representation for an implicit surface is a construction tree or graph, where nodes are functions describing simple geometric primitives or composition operators such as CSG operators, smooth blending, or affine transformations. The goal of *pruning* is to determine which nodes in the construction tree contribute to the distance value for a given input region. Claybook [Aal18] evaluates the distance to every primitive at the center of the region and checks if it is lower than the radius of the region, with an additional safety margin for smooth blending operators. Unfortunately this approach leads to incorrect results when the blending radius grow beyond the safety margin. Pujol and Chica [PC23] accelerate the computation of the distance to a triangle mesh by precomputing an octree where every cell stores a list of active triangles. Their method is limited to triangle meshes, and is not usable in the context of hierarchical SDFs with arbitrary primitives and smooth blending operations. Closer to our method, Keeter [Kee20] describes a GPU-friendly hierarchical culling algorithm for general implicit surfaces. They translate the arithmetic expression of the implicit function into a tape of instructions later interpreted on the GPU. Then, for a given region they rely on interval arithmetic to determine which clauses can be safely removed from the tape without changing the result of the computation. This is done in a hierarchical manner: the large regions are split in smaller regions that use the reduced tape of their parent as a starting point for their own culling. Dreams [Eva15] uses a hierarchical culling scheme specifically tailored to SDFs, but their algorithm is limited to a linear array of primitives and is not fully described. Zanni [Zan23] describes how to design compact operators for SDFs, which in turn allows to prune them in screen-space and reduce computations. However, the Lipschitz bound is not preserved, which may lead to missing ray-surface intersections when rendering the SDF. In contrast,



(a) Input SDF **(b)** SDF update **(c)** Fast SDF queries

Figure 2: *Our method takes as input a smooth CSG tree that defines a SDF function and its associated surface **(a)**. Upon any modification of the SDF, which can happen in real-time, we compute a spatially-varying pruning of the tree **(b)**. At runtime, this pruning allows fast evaluations of the SDF, useful when sphere-tracing the surface or when discretizing the SDF into a dense grid **(c)**.*

our pruning algorithm preserves the Lipschitz property, operates in the 3D space (which allows for optimizing secondary rays), and guarantees that the SDF evaluates to the same distance value after pruning. Alternatively, hierarchical trees with compact support [WGG99], also called BlobTrees, have been widely studied over the years [SWSJ07, GLA00, FJW*05, FGW01, GDW*16], and are similar to the smooth CSG SDFs that our work focuses on. A key difference is that since the density field of primitives has a compact support, it is straightforward to compute bounding boxes for every node in a BlobTree. To our knowledge, all existing approaches to prune such structures are based on the computation of these bounding boxes [WGG99, FGW01], which do not translate to signed distance fields with global support that are widely used in practice [JQ14]. Still, some ideas developed in the literature do translate to our setting, such as efficient post-order traversal and left-heavy stack optimization [GDW*16] which can be applied to smooth CSG SDFs.

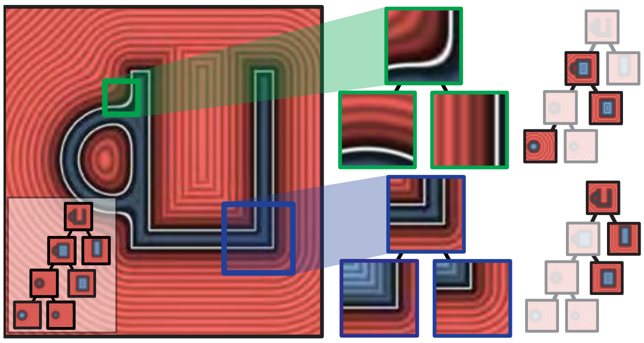
3. Method

3.1. Overview

We aim at reducing the cost of a single evaluation of the signed distance function by pruning or even replacing parts of its construction tree that are not relevant to a region of space. To that end, we developed two procedures: **(i)** a (hierarchical) pruning algorithm that reduces the number of active nodes in a region of space, and **(ii)** a far-field culling that replaces the whole tree with a constant expression when sufficiently far from the surface (Figure 2).

Our pruning algorithm (Sec. 3.4) relies on two important observations, namely **(i)** for a given point in space only a small subset of nodes needs to be evaluated to get the final value, and **(ii)** this subset tends to be coherent in space due to the inherent property of the distance function in Euclidean space. For a region of space e.g., a grid cell, we compute the subset of nodes needed to be evaluated for at least one point in the region – which we call *active* nodes – and output a pruned tree that is equivalent to the full tree when evaluated inside the input region (see Figure 3). This pruning is executed in a hierarchical manner (Sec. 3.5), which makes it scale gracefully to large scenes with thousands of nodes that would not be possible otherwise due to memory constraints.

We also observe that further away from the surface, the mathematical expression may be reduced even more. This is the focus of our far-field culling optimization (Sec. 3.6) that reduces trees to constant expressions. While not exact, the reduced constant remains a lower bound on the distance, and allows large sub trees to be evaluated in constant-time which proves to be more efficient.



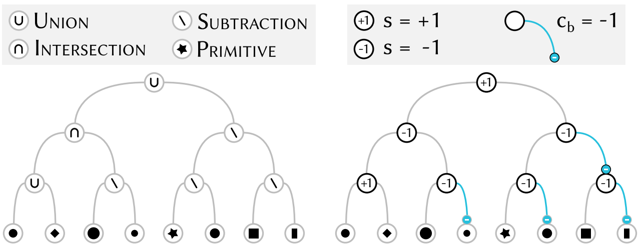
(a) Signed Distance Field **(b)** Per-cell pruned tree

Figure 3: *Our primitive pruning procedure (Section 3.4) consumes a smooth CSG tree that describes a Signed Distance Field made of different primitives (spheres, boxes) and operators (CSG operators and their smooth variants) **(a)**, as well as a region of space (e.g., blue or green squares). The output is a per-cell pruned tree **(b)** that evaluates to the same signed distance values within the region.*

Our pruning and far-field optimization contributions are complementary to each other. Pruning greatly reduces the number of primitives evaluated when *close to the surface*, and far-field culling reduces the construction tree to constant expressions *further away* *from the shape*. Together, they makes the SDF evaluation significantly faster in context such as sphere tracing, or when discretizing the function for a later mesh extraction (see Section 5).

3.2. Smooth CSG Tree

We encode SDFs as binary trees, in the spirit of the BlobTree [WGG99]. Leaf nodes represent primitives and compute the signed distance to a potentially transformed geometric shape (sphere, box, cone, etc.). Internal nodes are binary operators that combine the distances evaluated from their subtrees (CSG operators and their smooth variants). The actual distance function is defined as the expression for the root of this *smooth CSG tree*. We focus specifically on exact and conservative SDFs where the distance function f preserves the 1-Lipschitz property ($\|\nabla f\| \leq 1$ everywhere [Har96]).



(a) Input CSG tree **(b)** Annotated CSG tree

Figure 4: ***Tree annotation.*** *The operators used in the input smooth CSG Tree* ***(a)*** *need to provide our pruning algorithm with the sign s and the potential complementary flags c_a, c_b, stored in children* ***(b)****.*

3.3. Pruning Constraints

To enable effective pruning, we need an additional constraint on each binary operator: it must reduce to one of its operands or its complementary when they are sufficiently far apart. We show below that this is met by classical hard CSG operators (namely union, intersection, and difference) as well as their smooth counterparts.

Formally, for a binary operator OP we require a blending radius $k \in \mathbb{R}$, a sign $s = \pm 1$ and two complementary flags $c_a = \pm 1$ and $c_b = \pm 1$ to define this constraint. The flag c_a (resp. c_b) specifies whether the operator reduces to its operand A (resp. B) or to the complementary $-A$ (resp. $-B$). The sign s encodes whether the comparison is $\leq$ or $\geq$ in our formal constraint:

$$\text{if } |a' - b'| > k,\ \textsc{Op}(a,b) = \begin{cases} a' \text{ if } s \cdot a' \leq s \cdot b' \\ b' \text{ otherwise} \end{cases} \tag{1}$$

where $a' = c_a \cdot a$ is derived from the distance a returned by the operand A by flipping its sign when OP may reduce to the complementary of A. Similarly, $b' = c_b \cdot b$ is derived from the distance b returned by the operand B. The blending radius k represents the distance at which we can reduce the operator to one of its operands - its value varies depending the underlying operator formula, as shown below.

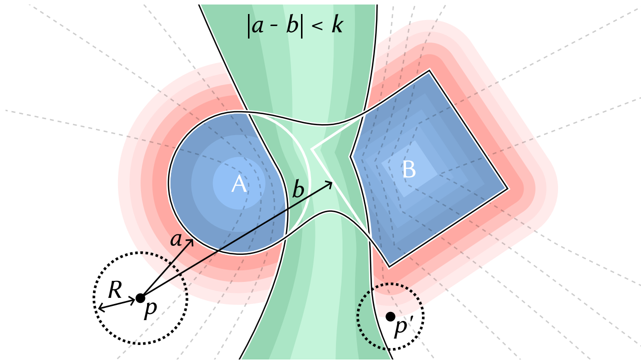


Figure 5: *An operator that combines operands A and B is marked as skipped by Procedure 1 if the bounding sphere of the region under consideration does not overlap with the points such that $|a-b| \leq k$ (green area). In this example, the* UNION *operator is skipped when pruning for region centered at* **p** *with radius R, but it cannot be skipped for the region centered at* $\mathbf{p}'$*. Shades of red are the positive isolines of the signed distance to* UNION(A,B)*, shades of blue are the negative isolines of this same distance and dashed lines are the positive isolines of* $|a-b|-k$*.*

Smooth Blending operators. Without loss of generality, the binary operators that we use are the *smooth* CSG operators with a user-specified blending radius k:

$$\textsc{Union}(a,b,k) = \min(a,b) - \phi(|a-b|,k)$$
$$\textsc{Inter}(a,b,k) = \max(a,b) + \phi(|a-b|,k)$$
$$\textsc{Sub}(a,b,k) = \max(a,-b) + \phi(|a+b|,k)$$

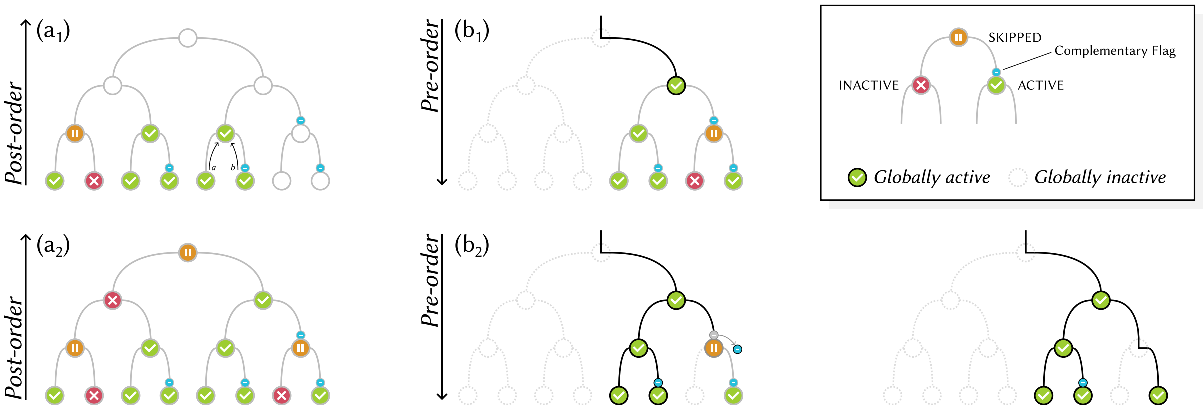
where $\phi(d,k)$ is a blending kernel that vanishes to 0 as d approaches k such that when $k=0$, they correspond to hard CSG operators. In practice, we use quadratic blending [DVOG04, Qui24b], namely $\phi(d,k) = \frac{1}{4k}\max(k-d,0)^2$. These smooth CSG operators fulfill constraint (1) since when $\phi(d,k) = 0$ they can all be reduced to one of their operands. For these operators, we use the following values for c_a, c_b, and s:

OP	c_a	c_b	s
UNION	+1	+1	+1
INTER	+1	+1	-1
SUB	+1	-1	-1

To efficiently handle all smooth CSG operators c_a (resp. c_b) are also stored as a sign `cFlag` in the child node A (resp. B) (Figure 4). For our pruning procedure, we also consider a bounding box that contains the scene to be provided as input.

3.4. Pruning algorithm

For complex scenes with hundreds of nodes or more, the smooth CSG tree can grow quite large, which increases the cost of a single SDF evaluation as it needs to traverse the whole tree. Our pruning algorithm takes as input a region of space (a grid cell in practice), computes the subset of *active* nodes needed to be evaluated for at least one point in the region and outputs a pruned tree that is equivalent to the full tree when evaluated inside the input region (see



(a) First traversal
Evaluating local state

(b) Second traversal
Pruning inactive subtrees and skipped nodes
Propagating complementary flags

(c) Pruned Smooth CSG tree
New parent assignment
and complementary flag

Figure 6: *Our pruning algorithm consists in two traversals of the input smooth CSG tree. The first one evaluates the local state of a node with respect to a given grid cell **(a)**, the second reassigns node parents to prune inactive sub-trees and remove skipped nodes **(b)**. The result is a pruned tree that is equivalent to the input one for points of the grid cell under consideration. **(c)**. Complementary flags are a compact way to keep track of sub-trees whose sign must be inverted when skipping some operators.*

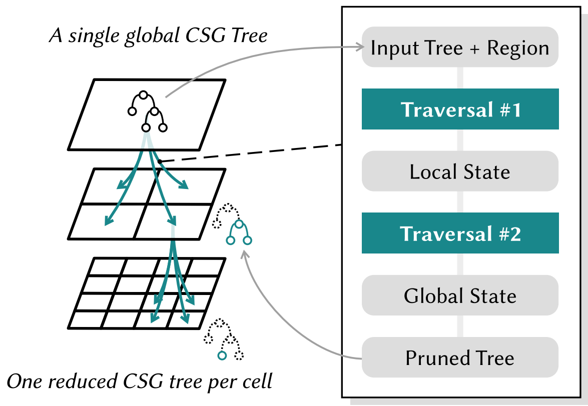
Figure 3). Since points that are close to each other in space are likely to share the same subset of contributing nodes, the set of active nodes tends to shrink rapidly as the size of the region of interest diminishes.

Overview. Our pruning procedure consists in two consecutive traversals of the original smooth CSG tree. The first one evaluates a *local state* of each node, relative to the input region. The second one deduces the *global* contribution of each node and rewires parenting so as to prune unused sub-trees and drop operators that can be skipped, effectively building a *pruned* tree valid for the whole region (Figure 7).

Traversal #1. We first do a post-order traversal during which we compute the *local state* of every node (Figure 6.a):

- A node is *inactive* if it does not contribute to the value of its parent node: the whole sub-tree rooted at this node can be pruned.
- A node is *skipped* if it contributes to the value of its parent, but can be replaced by one of its children or its complementary. For example, in the expression $min(min(a,b),c)$, if $a < b$ then the inner min can be replaced by its left operand a.
- A node is *active* if it contributes to the value of its parent and cannot be replaced by one of its children, or has no children.

We take advantage of the fact that all the expressions are 1-Lipschitz: we detect binary operators that can be safely *skipped* using a single evaluation of their operands. Considering an operator with child expressions f_1 and f_2, and a region of radius R centered on a point $\mathbf{p}$, if $|f_1(\mathbf{p}) - f_2(\mathbf{p})| \geq k + 2R$ it follows that $|f_1 - f_2| \geq k$ for all points in this region (see Appendix A). This implies that we can replace the operator with one of its children due to Equation 1 (see Figure 5). In this case we mark the operator node as *skipped*, and depending on the sign of $f_1(\mathbf{p}) - f_2(\mathbf{p})$ we mark one of the children as inactive (see Procedure 1).



(a) Spatial Hierarchy **(b)** Pruning

Figure 7: *Simplified overview of our hierarchical pruning algorithm in **2D**. Starting from the global smooth CSG tree, our pruning algorithm (right) computes pruned trees that are equivalent in distance evaluation but with lower complexity. The process is iterative and leads to simpler pruned trees that are stored per-cell in a hierarchical manner (left).*

Our Lipschitz constraint allows to estimate bounds of $f_1 - f_2$ on a region using a single evaluation. While range arithmetic is a more generic yet more costly approach to obtain SDF bounds on regions [SJ22], we show in Figure 8 that in practice, Lipschitz bounds, when available, have similar pruning capabilities than those obtained with affine or interval arithmetic.

Traversal #2. The role of the second traversal (Figure 6.b) is to determine the *global state* of the nodes. While the local state only tells whether a node contributes to the value of its direct parent, the global state tells whether the node contributes to the full tree evaluation. A node contributes to this final value if it is active and there are no inactive nodes among its ancestors.

We compute the global state for every node through a pre-order traversal of the tree (see Procedure 2). We set a node as *globally active* if there is no *locally inactive* node among its ancestors. During this traversal we also update the parent (line 11) – the new parent of a node is its closest ancestor that is not *skipped* – and we propagate the complementary flag of skipped nodes to their children (line 12). Finally, we also count the number of globally active nodes (i.e the size of the pruned tree) during this traversal (line 15).

Procedure 1: ComputeLocalState(nodes, p, R)

Input :
- nodes: array of nodes
- p: cell center
- R: cell radius

Output: Local state of the nodes relative to the cell centered at p with radius R.

```
stack ← {}    // Entry: (distance, node idx)
foreach i ∈ PostOrderTraversal(nodes) do
    n ← nodes[i]
    if IsLeaf(n) then
        n.state ← ACTIVE
        d ← EvalPrimitive(p, n.primData)
    else
        (b′, right) ← stack.pop()
        (a′, left) ← stack.pop()
        {Op, k, s, c_a, c_b} ← n.blendData    // see (1)
        if |a′ − b′| > k + 2R then
            if s · a′ ≤ s · b′ then
                nodes[right].state ← INACTIVE
            else
                nodes[left].state ← INACTIVE
            end
            n.state ← SKIPPED
        else
            n.state ← ACTIVE
        end
        d ← Op(c_a · a′, c_b · b′, k)
    end
    stack.push((n.cFlag · d, i))
end
```

3.5. Spatial hierarchy

Applying this procedure directly for all cells of a dense grid leads to two different issues in practice. First, the size of each output pruned tree is not known in advance. In the worst case where every cell outputs the full input tree, memory allocation would not scale up for large scenes and dense grids. Second, evaluating the full tree for each cell of a dense grid would be time-consuming. Indeed, the complexity of such an approach would scale linearly not only with the number of cells in the grid, but also with the tree size.

Procedure 2: ComputeGlobalState(nodes)

Input:
- nodes: array of nodes with their local state

Output:
- A boolean flag for every node (activeGlobal) set to TRUE if the node is part of the pruned tree.
- The number of globally active nodes.

```
1  numActiveGlobal ← 0;
2  foreach i ∈ PreOrderTraversal(nodes) do
3      n ← nodes[i];
4      if n.state = INACTIVE then
5          n.activeGlobal ← FALSE;
6          n.inactiveAncestors ← TRUE;
7      else
8          n.inactiveAncestors ← n.parent.inactiveAncestors;
9          n.activeGlobal ← n.state = ACTIVE and not
               n.inactiveAncestors;
10         if n.parent.state = SKIPPED then
11             n.parent ← n.parent.parent;
12             n.cFlag ← n.cFlag × n.parent.cFlag;
13         end
14         if n.activeGlobal then
15             numActiveGlobal ← numActiveGlobal + 1;
16         end
17     end
18 end
```

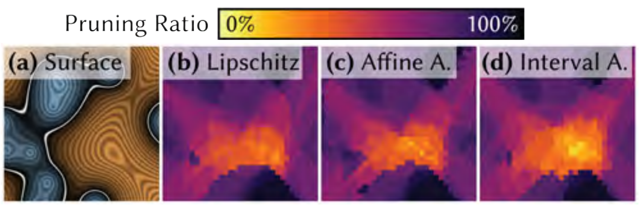


Figure 8: ***Pruning ratio.*** *Amount of pruned primitives on a 2D test scene **(a)**, computing the bounds of $f_1 - f_2$ for each cell of a regular grid and each binary operator using: our Lipschitz constraint **(b)**, affine arithmetic **(c)**, and interval arithmetic **(d)**. Our method leads to a slightly better pruning ratio and only requires a single evaluation of the SDF, while others are significantly more invasive.*

We address these two issues using a spatial hierarchy: starting from a coarse grid, we iteratively subdivide each cell and apply our pruning procedure to compute a finer grid until a target resolution is reached. More specifically, the input of the pruning procedure for a given cell is the pruned tree computed for the parent cell in the previous grid, as seen in Figure 7. In practice, this approach reduces the memory footprint and speedups computations. Indeed, at each step the pruned tree becomes smaller, allowing less memory to be allocated at the next iteration, and faster computing of the next iteration as opposed to using the full tree. The pruning effectiveness increases with the hierarchy levels resolution as illustrated in Figure 9.

Although we use a regular grid in our implementation, any hierarchical structure may be used instead (octrees, kd trees, etc.). In addition to its simplicity, the regular grid also provides efficient random access, which is suited for sphere tracing or any application that require parallel SDF queries on the GPU. All in all, our hierarchical pruning procedure is fast enough to be useful in contexts where the scene is modified (i.e. a parameter of a primitive is changed, new sub trees are added or deleted, etc.), as discussed in Section 5.

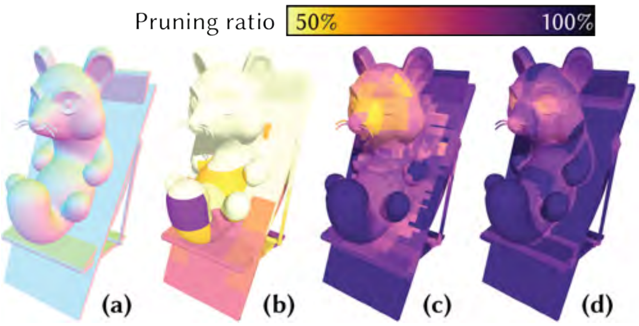


Figure 9: *Pruning ratio shown as a heat-map on the Raccoon scene surface* ***(a)****, for three different levels of our hierarchy with respective grid resolutions of* 16^3 ***(b)****,* 64^3 ***(c)****, and* 256^3 ***(d)****.*

3.6. Far-field culling

In order to further speed up tasks that can operate on SDFs while being conservative rather than exact, we optimize cells that are far away from the 0-isosurface, a.k.a. the *far-field*. We take inspiration from the *narrow-band* optimization commonly used in discrete SDF rendering [Eva15, Aal18, Söd21, SEAM22] to *replace* trees in far-field cells by an approximate tree made of a single constant node, whose value is given by the heuristic described below. Although this approximation is not differentiable nor continuous, it is guaranteed to give a lower-bound distance field that shares the same 0-isosurface as the input SDF, and can thus be safely used for sphere-tracing or contouring tasks (see Table 1).

Conservative heuristic. After the first traversal (Procedure 1), we check whether the SDF evaluated at the cell center $d = f(p)$ is greater than $C \cdot R$, where R is the cell radius and C is a constant factor that controls the size of the near-field. When $|d| > C \cdot R$, the cell is considered far and its tree is replaced with the constant $sign(d) \cdot (|d| - R)$. It is important that $C > 1$, to assert that the cell is surface free, which ensures that the approximate SDF remains conservative. When C gets larger, the narrow band in which we exactly preserve the input SDF gets thicker. As such, less cells are replaced by a single node, but the ones that get replaced contain a larger minimal constant as $||d| - R| > (C-1) \cdot R$.

When doing sphere tracing, a value of C too close to 1 leads to small step sizes which results in poor performance. On the contrary, when C is too large, the condition $|d| > C \cdot R$ is never met and thus none of the cell is deemed far. In practice, we use $C = 2$,

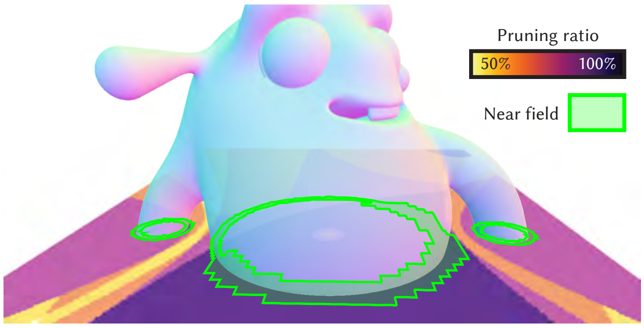


Figure 10: *Near field and pruning ratio for a* ***2D slice****. For all cell in the far field (i.e. outside the near field), we replace its tree with a single constant value. This drastically reduces the SDF complexity in regions of space with low pruning capabilities, typically far from the surface, while preserving a lower-bound on the distance.*

which guarantees that the tracing takes at most 2 steps to cross a far-field cell while keeping the size of the near-field small enough to significantly improve performance.

Benefits. This far-field culling has a good synergy with our pruning algorithm. Indeed, many regions that have a low pruning potential occur in the far-field, when multiple surfaces are equally far from the cell. Figure 10 shows that many of these low pruning ratio regions fall out of the near-field, allowing our far-field culling to simplify their SDF and thus avoid a lot of costly computation both during the second traversal of the pruning and during the final evaluation of the SDF.

4. GPU implementation

We implement our algorithm using compute shaders, with one dispatch per level of the hierarchy. Starting from the coarser level, we iteratively perform a single dispatch to compute the next level of our grid hierarchy. In each dispatch, we spawn one thread per target grid cell: every thread computes and writes the pruned tree of its cell, given the pruned tree of its parent cell from the previous level. Similarly to [Kee20] we use a $4 \times 4 \times 4$ subdivision for the hierarchical grid, which ensures that all threads in a warp have the same parent cell to eliminate divergence.

Data representation. At each iteration, each cell of the grid stores a tree of variable size. We then represent a grid as three arrays, namely the *cell array*, the *mutable node array* and the *immutable node array* (see Figure 11). The cell and mutable node arrays are double-buffered: one copy is read, from a coarser level, while the other is written, to a finer level. Arrays are switched after each dispatch.

The immutable node array is the input of our system, and contains all data related to the construction tree of the SDF: the primitives and binary operators data. This array is not affected by the pruning, hence stored once without needing duplication.

The mutable node array represents the data modified by the pruning, and thus needs to be stored separately for all cells. Each el-

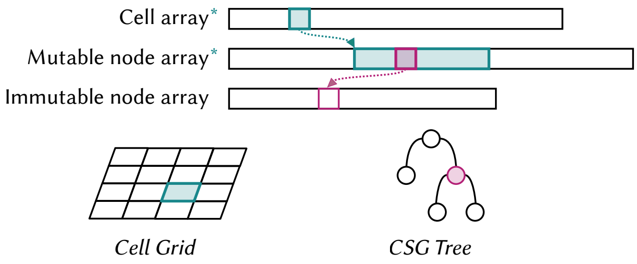


Figure 11: *For our hierarchical pruning, we store data in three different GPU arrays. Arrays marked with an asterisk are mutable and thus double buffered. The immutable node array is the input from which the two other arrays are constructed, and contains all the smooth CSG tree data.*

ement has an ancestor node index, a complementary flag, and a global index that references the immutable node array.

Each world-space cell owns a slice of the mutable node array where its pruned tree is stored using post-order indexing [GDW*16]. The cell array stores the start and length of this slice for every cell. We index this array using Morton order to map the 3D cell coordinates to 1D, which ensures that cells sharing the same parent in the hierarchy have consecutive indices.

Local & global state computation. The input nodes are stored using post-order indexing. Hence, post-order and pre-order traversal are implemented with forward and backward iteration through the array of input nodes. We use per-thread arrays for the stack used in Procedure 1, and a pre-allocated temporary buffer in global memory for all per-node temporary data: local state for the first traversal (2 bits per node); global state, inactive ancestors flag, updated ancestor indices and node signs for the second traversal (19 bits per node).

Writing the pruned tree. After computing the global state and updating ancestor and sign for every input node, we write out these results. Every thread increments a global atomic counter by the size of its pruned tree, computed during Procedure 2, to reserve a slice of the output mutable node array. Then the thread iterates through all the input nodes, checks whether they are globally active, and if so writes their global index, ancestor index and sign to the next unused location in the output slice. Since we want to write the output index of the ancestors and not their index in the input tree, we cache the output index of every active node in our temporary buffer during this traversal and translate the ancestor index before writing.

5. Results

We implemented our method in C++/Vulkan/GLSL in a standalone application (see the accompanying video). All models shown throughout this paper (Figures 1, 9, 10, 12) are represented by construction trees made of common SDF primitives and binary operators. In particular, our implementation supports all exact and conservative SDFs [Qui24a], as well as hard and smooth CSG operators. Scenes were rendered using sphere tracing, propelled with our hierarchical pruning procedure (Sections 3.4, 3.5) and far-field culling (Section 3.6). Our method can accelerate arbitrary SDF queries and is not limited to primary ray tracing as we demonstrate using shadow rays and grid discretization. Statistics for rendering the models shown in this paper are reported in Table 1, for a 1920×1080 image resolution on a laptop RTX 4060 with 8GB of GPU memory. Results are given with pruning on a 4-level grid hierarchy with resolutions 4^3, 16^3, 64^3, and 256^3 for all of our scenes.

5.1. Efficiency

Pruning and culling. We report in Table 1 the performance of pruning, and statistics about the number of active nodes in cells in Table 2. Our pruning algorithm significantly reduces the amount of active nodes, going as low as ≈ 1 active node per cell for many scenes when using far-field culling. Efficiency mostly depends on how primitives are positioned in space. For instance, the *fluid* scene is made of thousands of particles close in space, which makes pruning less efficient. The blending radius k (Equation 1) is also significant: since it impacts the range at which a primitive can be pruned, high blending radius have a negative effect on the runtime (Figure 14). Nonetheless, our method is more robust to this issue than state-of-the-art methods (Figure 13). Finally, while the pruning procedure only needs to be done once for static scenes, the algorithm is fast enough to be executed every frame for dynamic scenes.

Runtime performance. As exemplified in Table 1 and Figure 12, our method enables faster evaluation of the SDF on complex scenes. In ray tracing contexts with a primary ray and shadow ray per pixel, speedups go as high as two orders of magnitude (*City*, *Crowd*) when compared to naive sphere tracing. Our far-field culling improves the tracing performance by up to a factor of 2 (*City*, *Gameboy*, *Camera*), and our hierarchical pruning scheme significantly reduces the memory impact and runtime of the pruning procedure, allowing for larger scenes that would not be possible otherwise. As we optimize the SDF evaluation in 3D space, our method is also applicable in contexts where the SDF needs to be *discretized* over a finite domain, as it is the case for meshing purposes.

Handling of large scenes. Construction trees are a user-friendly paradigm to encode implicit surfaces, however their authoring experience may quickly suffer from poor rendering performance as the scene complexity grows. By efficiently pruning both hard and *smooth* boolean operators, which are notoriously difficult to handle, our method scales well and can render 3D scenes featuring thousands of nodes in real time (Figure 1, 12, 13), where other SDF rendering methods are limited to a few dozens to hundreds of nodes, going as high as few thousands for 2D SDFs [Kee20].

5.2. Comparison with other methods

Below we compare our approach against the method from Keeter [Kee20] (referred to as *parallel tape reduction*), and also discuss *object centric* techniques.

Object centric pruning. One classical approach for tree-based pruning is to compute a bounding volume (usually a box or sphere) around each node in the tree. These are then used to prune parts of

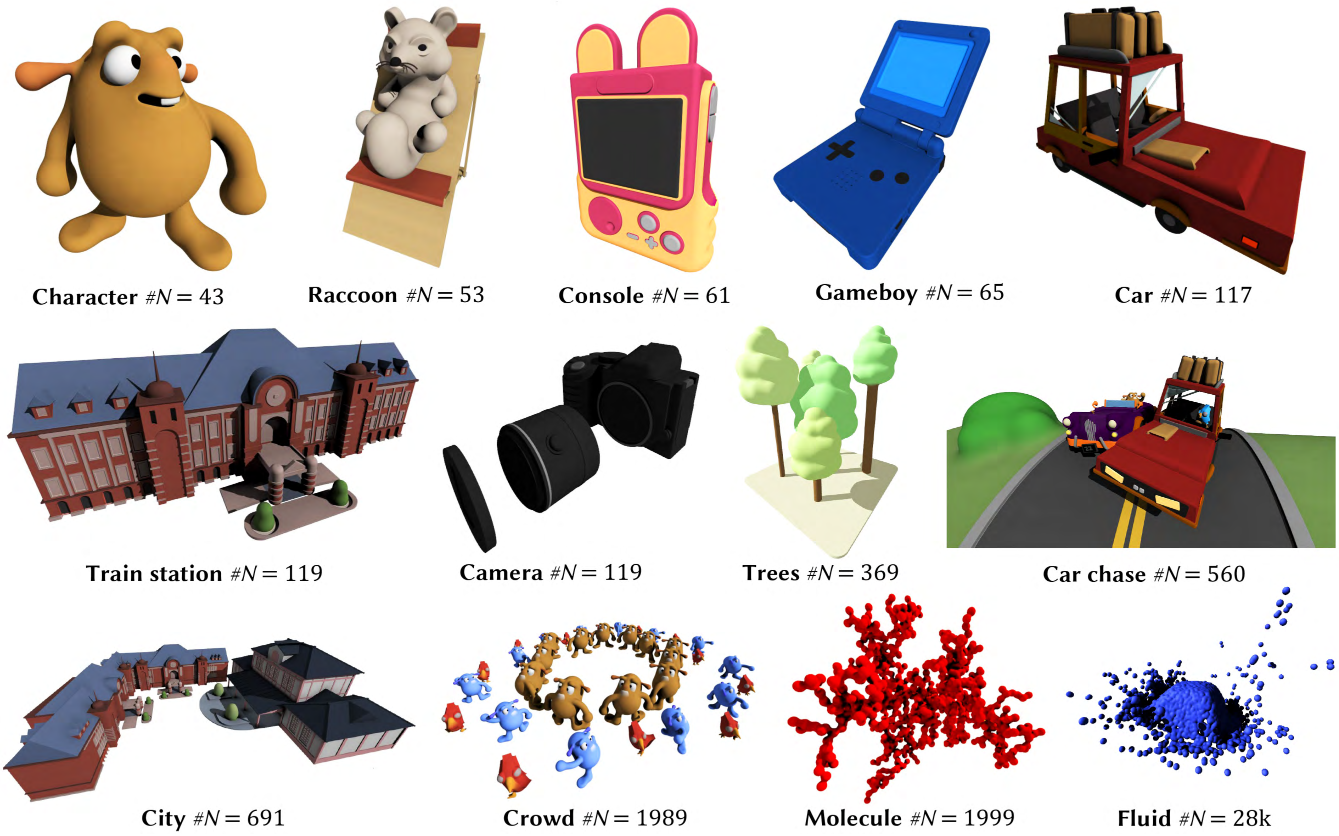


Figure 12: *Real time rendering of scenes with varying complexity with our method with ambient occlusion, shadow rays, and anti-aliasing.*

	Ours					**No far-field culling**					**No spatial hierarchy**					**Baseline**		
	$\mathcal{T}_p$	$\mathcal{T}_s$	$\mathcal{T}_d$	$\mathcal{M}_p$	$\mathcal{M}_s$	$\mathcal{T}_p$	$\mathcal{T}_s$	$\mathcal{T}_d$	$\mathcal{M}_p$	$\mathcal{M}_s$	$\mathcal{T}_p$	$\mathcal{T}_s$	$\mathcal{T}_d$	$\mathcal{M}_p$	$\mathcal{M}_s$	$\mathcal{T}_s$	$\mathcal{T}_d$	$\mathcal{M}_s$
Camera ($\#\mathcal{N} = 119$)	0.84	**1.47** (x34)	0.91	0.39	0.19	13.84	3.45	4.36	2.28	0.42	-	-	-	11.5*	-	50.31	70.66	0
Car ($\#\mathcal{N} = 117$)	1.17	**2.38** (x23)	1.0	0.46	0.2	4.09	3.32	2.57	1.01	0.32	-	-	-	11.3*	-	56.14	69.07	0
Car Chase ($\#\mathcal{N} = 560$)	2.53	**4.89** (x102)	1.05	0.5	0.2	10.31	8.65	4.4	1.75	0.44	-	-	-	52.8*	-	501.95	334.87	0
Character ($\#\mathcal{N} = 43$)	1.02	**2.79** (x8)	0.99	0.44	0.2	4.76	3.99	3.58	1.23	0.41	25.6	3.6	0.95	4.6	0.20	22.31	26.48	0
City ($\#\mathcal{N} = 691$)	2.52	**5.17** (x150)	1.08	0.52	0.2	12.72	9.7	5.01	2.0	0.45	-	-	-	64.1*	-	778.69	422.7	0
Console ($\#\mathcal{N} = 61$)	0.97	**1.17** (x13)	0.97	0.44	0.2	3.92	1.85	2.56	1.0	0.32	-	-	-	6.1*	-	15.83	36.77	0
Crowd ($\#\mathcal{N} = 1989$)	5.53	**8.35** (x227)	1.02	0.54	0.2	-	-	-	-	-	-	-	-	186*	-	1821	1223	0
Fluid ($\#\mathcal{N} = 27977$)	125.54	**78.52** (x-)	6.99	5.68	0.42	-	-	-	-	-	-	-	-	2625*	-	×	×	0
Gameboy ($\#\mathcal{N} = 65$)	0.72	**2.02** (x15)	0.91	0.38	0.19	10.14	4.39	4.0	1.92	0.43	-	-	-	6.5*	-	31.11	39.09	0
Molecule ($\#\mathcal{N} = 1999$)	8.9	**6.97** (x86)	1.48	1.47	0.21	-	-	-	-	-	-	-	-	188*	-	601.36	841.65	0
Monument ($\#\mathcal{N} = 6023$)	14.29	**15.15** (x629)	1.61	0.96	0.22	-	-	-	-	-	-	-	-	565*	-	9448	3604	0
Raccoon ($\#\mathcal{N} = 53$)	0.85	**2.18** (x9)	0.96	0.41	0.2	3.59	3.03	2.45	0.94	0.31	31.9	2.8	0.95	5.5	0.19	20.64	33.01	0
Train station ($\#\mathcal{N} = 119$)	1.53	**2.81** (x22)	1.12	0.51	0.21	6.94	4.65	3.73	1.38	0.38	-	-	-	11.5*	-	63.9	72.41	0
Trees ($\#\mathcal{N} = 369$)	2.73	**13.01** (x20)	1.24	0.64	0.21	-	-	-	-	-	-	-	-	34.9*	-	266.87	216.56	0

Table 1: *Statistics for the different scenes shown throughout our paper with $\#\mathcal{N}$ nodes, measured on a laptop RTX 4060 with 8GB of GPU memory. We report the pruning time $\mathcal{T}_p$, sphere tracing time $\mathcal{T}_s$, discretization time over a 256^3 grid $\mathcal{T}_d$, memory usage during pruning $\mathcal{M}_p$, and memory usage during tracing $\mathcal{M}_s$. All timings are in milliseconds (**ms**) and memory usage is in gigabytes (**GB**). We report numbers for our method with both spatial hierarchy and far-field culling, and also ablations without these additions, and compare against a naive SDF evaluation baseline. For sphere tracing, we use one primary and one shadow ray over 1920×1080 pixels. Empty cells marked with a dash ($-$) correspond to tests that overflowed the maximum temporary buffer size (5.7GB), and empty cells marked with a cross ($\times$) correspond to tests where rendering or discretization did not finish after a few minutes. Cells reporting memory marked with a star (*) give only a theoretical lower-bound memory usage that would be required for the pruning without spatial hierarchy.*

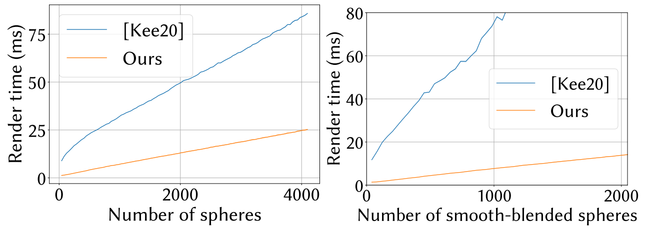


Figure 13: *Render time comparison between our method and parallel tape reduction [Kee20], using a RTX 3090, with a* 2048×2048 *screen resolution, on a synthetic scene with a varying number of spheres without smooth blending (left), and with smooth blending (right). Since parallel tape reduction couples pruning and rendering, our timings combine pruning and sphere tracing.*

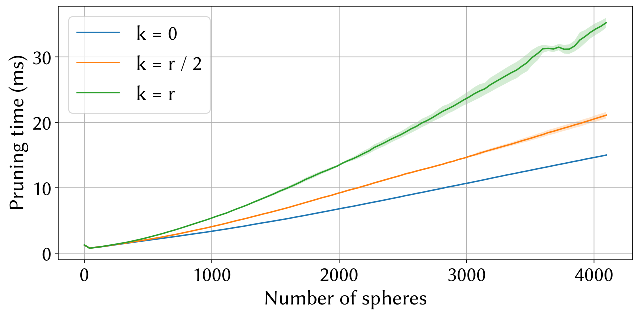


Figure 14: *Impact of the blending radius k on pruning time using a laptop RTX 4060, on a synthetic scene composed of a varying number of spheres with unit random positions and radius* $r = 0.05$. *Line thickness indicates standard deviations.*

the tree in certain regions of space. Some implicit model are well suited to this technique as they provides an integrated bounding volume hierarchy [WGG99]). While this is also theoretically possible with SDFs, it quickly becomes impractical as smooth CSG operators require the bounding box to be *extended* by the blending radius [Qui24b], which quickly leads to very large boxes that simply cannot be pruned. In contrast, our Lipschitz criteria is *space centric* rather than *object centric*, does not require nodes to compute a bounding box, and also guarantees distance exactness while being as efficient as possible with respect to the blending radius.

Parallel tape reduction. Our approach is similar in spirit to Keeter [Kee20]: we aim at simplifying the SDF expression in certain regions of space. The key difference is that we do not use interval arithmetic, but rather exploit the fact that all of our nodes are 1-Lipschitz to bound their range of influence. Our approach compares favourably in terms of speed (see Figure 13, left and right) and handles widely used smooth boolean operators more efficiently (see Figure 13, right). It is also simpler to implement as primitives are treated as black boxes, whereas tape reduction relies on well-defined interval arithmetic queries for each node in the tree.

6. Discussion

6.1. Limitations & Future work

Our method is restricted to primitives and operators that are 1-Lipschitz, meaning that $\|\nabla f\| \leq 1$ everywhere. In practice, this limitation is mitigated as any function f may be transformed into a lower-bound distance function by using f/K as the field function, with K a Lipschitz bound [Har96]. This induces that our method is compatible with all primitives and operators where a Lipschitz bound can be computed, which is the case for most primitives and operators used in implicit modeling.

Our approach provides foundations to explore several directions in the future. First, while its performance already allows real-time editing of the underlying CSG tree investigating the *partial update* of our pruning structure would improve the editing experience for very large scenes. Second, our dense space partition could be replaced by a sparse adaptive one, especially for scenes with significant variations in object scales. Third, while our hierarchical scheme scales efficiently memory-wise, we still need to preallocate a potentially large chunk of GPU memory at startup. A more efficient scheme could first compute the amount of memory required for the whole structure in a pre-pass.

6.2. Conclusion

We presented a real-time rendering method for signed distance fields encoded as construction trees of thousands of primitives and operators. Our hierarchical pruning algorithm, based on a Lipschitz criteria, greatly reduces the size of the tree near the surface while preserving exact distance values. Additionally, our far-field culling replaces entire sub trees with constants that remain valid lower distance bounds further away from the object. Our method is conceptually simple, non-invasive when compared to other approaches, and handles smooth CSG operators widely used in implicit modeling.

References

[Aal18] Aaltonen S.: Gpu-based clay simulation and ray-tracing tech in claybook. In *Game Developers Conference* (2018). 2, 7

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

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

[BV23] Bán R., Valasek G.: Automatic step size relaxation in sphere tracing. 2

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

[DVOG04] Dekkers D., Van Overveld K., Golsteijn R.: Combining csg modeling with soft blending using lipschitz-based implicit surfaces. *The Visual Computer 20* (2004), 380–391. 4

[Eva15] Evans A.: Learning from failure: A survey of promising, unconventional and mostly abandoned renderers for 'dreams ps4', a geometrically dense, painterly ugc game. In *SIGGRAPH Advances in Real-Time Rendering in Games* (2015). 2, 7

[FGW01] Fox M., Galbraith C., Wyvill B.: Efficient use of the blobtree for rendering purposes. In *2001 International Conference*

	Ours		**No far-field culling**		**Baseline**
	Avg $\pm$ std. dev.	Max	Avg $\pm$ std. dev.	Max	$\#\mathcal{N}$
Camera	$\mathbf{1.0 \pm 0.01}$	**7.0**	2.36 ± 2.93	15.0	119
Car	$\mathbf{1.03 \pm 0.05}$	**6.0**	1.51 ± 0.5	8.0	117
Car Chase	$\mathbf{1.06 \pm 0.15}$	**33.0**	2.52 ± 3.68	46.0	560
Character	$\mathbf{1.03 \pm 0.06}$	**8.0**	2.27 ± 1.39	8.0	43
City	$\mathbf{1.05 \pm 0.14}$	**14.0**	2.56 ± 2.82	24.0	691
Console	$\mathbf{1.02 \pm 0.02}$	**5.0**	1.56 ± 0.89	11.0	61
Crowd	$\mathbf{1.03 \pm 0.17}$	**17.0**	-	-	1989
Fluid	$\mathbf{2.83 \pm 776.07}$	**1202.0**	-	-	27 977
Gameboy	$\mathbf{1.0 \pm 0.0}$	**9.0**	2.44 ± 1.75	15.0	65
Molecule	$\mathbf{1.12 \pm 0.98}$	**62.0**	-	-	1999
Monument	$\mathbf{1.17 \pm 0.81}$	**45.0**	-	-	6023
Raccoon	$\mathbf{1.01 \pm 0.02}$	**8.0**	1.43 ± 0.87	10.0	53
Train station	$\mathbf{1.05 \pm 0.11}$	**10.0**	1.98 ± 1.99	16.0	119
Trees	$\mathbf{1.1 \pm 0.97}$	**36.0**	-	-	369

Table 2: *Statistics on active nodes per grid cell using our method with and without far-field culling, against the baseline where all nodes are considered active. When enabled, the average number of active nodes is close to* 1 *in many scenes, allowing efficient SDF evaluation.*

on Shape Modeling and Applications (2001), IEEE Computer Society, p. 306. 2, 3

[FJW*05] Foster K., Jepp P., Wyvill B., Sousa M. C., Galbraith C., Jorge J. A.: Pen-and-ink for blobtree implicit models. In *Computer Graphics Forum* (2005), vol. 24, Amsterdam: North Holland, 1982-, pp. 267–276. 3

[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. 3, 8

[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. In *Computer Graphics Forum* (2000), vol. 19, Wiley Online Library, pp. 257–270. 3

[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, 4, 10

[JQ14] Jeremias P., Quilez I.: Shadertoy. https://www.shadertoy.com/, 2014. 3

[Kee20] Keeter M. J.: Massively parallel rendering of complex closed-form implicit surfaces. *ACM Transactions on Graphics (TOG) 39*, 4 (2020), 141–1. 2, 7, 8, 10

[KSK*14] Keinert B., Schäfer H., Korndörfer J., Ganse U., Stamminger M.: Enhanced sphere tracing. 2

[Mag22] Magicacsg. http://ephtracy.github.io/index.html?page=magicacsg, 2022. 1

[Neo23] Project Neo. https://projectneo.adobe.com/, 2023. 1

[PC23] Pujol E., Chica A.: Triangle influence supersets for fast distance computation. In *Computer Graphics Forum* (2023), vol. 42, Wiley Online Library, p. e14861. 2

[Qui24a] Quilez I.: 3d signed distance functions, 2024. URL: https://iquilezles.org/articles/distfunctions/. 8

[Qui24b] Quilez I.: Bounding volumes for smooth sdf operators, 2024. Normalization, thickness and bounds. URL: https://iquilezles.org/articles/smin/. 4, 10

[SEAM22] Söderlund H. H., Evans A., Akenine-Möller T.: Ray tracing of signed distance function grids. *Journal of Computer Graphics Techniques Vol 11*, 3 (2022). 7

[SJ22] Sharp N., Jacobson A.: Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis. *ACM Transactions on Graphics (TOG) 41*, 4 (2022), 1–16. 5

[Söd21] Söderlund H. H.: Hardware-accelerated ray tracing of implicit surfaces: A study of real-time editing and rendering of implicit surfaces, 2021. 7

[SWSJ07] Schmidt R., Wyvill B., Sousa M. C., Jorge J. A.: Shapeshop: Sketch-based solid modeling with blobtrees. In *ACM SIGGRAPH 2007 courses*. 2007, pp. 43–es. 3

[WGG99] Wyvill B., Guy A., Galin E.: Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system. In *Computer Graphics Forum* (1999), vol. 18, Wiley Online Library, pp. 149–158. 2, 3, 4, 10

[Wom22] Womp 3d inc. https://womp.com/, 2022. 1

[Zan23] Zanni C.: Synchronized-tracing of implicit surfaces. working paper or preprint, 2023. 2

Appendix A: A condition for Lipschitz pruning

In order to prune binary operators, one needs find a lower bound on $|f_1 - f_2|$ for a region of space, where f_1 and f_2 are the operator's SDFs arguments. As depicted in Figure 5, given a closed ball $\mathcal{B}$ with center $\mathbf{p}$ and radius R, a binary operator with blending radius k can be pruned over $\mathcal{B}$ if $|f_1(\mathbf{q}) - f_2(\mathbf{q})| \geq k$ for all $\mathbf{q} \in \mathcal{B}$. Since the SDFs are 1-Lipschitz, this condition is met whenever $|f_1(\mathbf{p}) - f_2(\mathbf{p})| \geq k + 2R$, because:

$$\begin{aligned} |f_1(\mathbf{q}) - f_2(\mathbf{q})| &= |f_1(\mathbf{q}) - f_1(\mathbf{p}) + f_1(\mathbf{p}) - f_2(\mathbf{p}) + f_2(\mathbf{p}) - f_2(\mathbf{q})| \\ &\geq \underbrace{|f_1(\mathbf{p}) - f_2(\mathbf{p})|}_{\geq k+2R} - \underbrace{|f_1(\mathbf{q}) - f_1(\mathbf{p}) + f_2(\mathbf{p}) - f_2(\mathbf{q})|}_{\leq (\|\nabla f_1\| + \|\nabla f_2\|)\|\mathbf{p}-\mathbf{q}\| \leq 2R} \\ &\geq k \end{aligned} \tag{2}$$