Sphere Carving: Bounding Volumes for Signed Distance Fields

HUGO SCHOTT, Adobe, INSA Lyon, CNRS, LIRIS, UMR5205
THEO THONAT, Adobe
THIBAUD LAMBERT, Adobe
ERIC GUÉRIN, INSA Lyon, CNRS, LIRIS, UMR5205
ERIC GALIN, Université Claude Bernard Lyon 1, CNRS, LIRIS, UMR5205
AXEL PARIS, Adobe

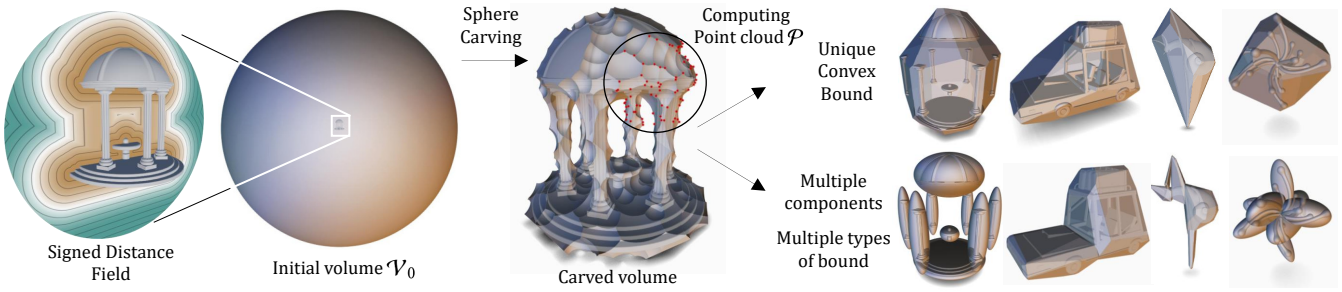

Fig. 1. Given an input black box conservative signed distance field, we automatically generate a convex bounding volume around the implicitly defined object, agnostic of its representation. Starting from a large initial volume around the object, we iteratively carve the space using spheres defined by the signed distance field. The bounding volume is then constructed as a single or multiple convex primitives (half-spaces, ellipsoids) and allows for faster field function queries.

We introduce *Sphere Carving*, a novel method for automatically computing bounding volumes that closely bound a procedurally defined implicit surface. Starting from an initial bounding volume located far from the object, we iteratively approach the surface by leveraging the signed distance function information. Field function queries define a set of empty spheres, from which we extract intersection points that are used to compute a bounding volume. Our method is agnostic of the function representation and only requires a conservative signed distance field as input. This encompasses a large set of procedurally defined implicit surface models such as exact or Lipschitz functions, BlobTrees, or even neural representations. *Sphere Carving* is conceptually simple, independent of the function representation, requires a small number of function queries to create bounding volumes, and accelerates queries in Sphere Tracing and polygonization.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

Additional Key Words and Phrases: Implicit surfaces, Signed Distance Fields, Bounding Volumes

ACM Reference format:
Hugo Schott, Theo Thonat, Thibaud Lambert, Eric Guérin, Eric Galin, and Axel Paris. 2025. Sphere Carving: Bounding Volumes for Signed Distance Fields. *ACM Trans. Graph.* 1, 1, Article 1 (May 2025), 12 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Implicit surfaces [Bloomenthal and Wyvill 1997] form a compact and efficient representation for modeling volumetric objects, and have demonstrated their effectiveness in constructive solid geometry, fluid simulation, rendering, or surface reconstruction. Unlike explicit representations such as point clouds, surface meshes, or voxels which rely on a discretization of space, an implicit representation indirectly defines an object as the zero level-set $\partial\mathcal{O} = \{\mathbf{p}, f(\mathbf{p}) = 0\}$ of a function f, often referred to as a potential field. In this article, we address implicit surfaces for which their function is also a lower bound of the distance to the surface, referred to in this paper as signed distance fields.

As objects become increasingly complex, the mathematical or procedural expression of the function can become computationally expensive, hindering the ability to create and edit complex scenes interactively. A well-established technique for optimizing queries involves using bounding volume hierarchies [Gourmel et al. 2010; Wyvill et al. 1999]. These are usually defined using simple geometric shapes and allow quickly discarding large portions of space to save computations. While the computation of the bounding volumes has been thoroughly investigated for implicit surfaces built from construction trees [Wyvill et al. 1999], the problem is challenging for general implicit surfaces, agnostic of their underlying representation. Moreover, we aim to automatically compute a lightweight bounding volume hierarchy whose representation may be incorporated in the function definition itself.

In this work, we present a method for computing such bounding volumes called *Sphere Carving*. We conceptualize space as an infinite block of rock, which we initially reduce to a large but finite region that encloses the object (see Figure 1). Similar to sculpting, we remove sections of rock to gradually reveal the underlying shape. We iteratively and progressively carve this rock by removing parts of spheres without damaging the object, using the empty sphere criterion defined by conservative signed distance fields. At the end of this process, we obtain a tight volume $\mathcal{V}$ that closely envelopes the object's surface (Figure 2). We convert $\mathcal{V}$ into a bounding proxy signed distance function that is less expensive to evaluate than the original function f. This lightweight approach is simpler than an

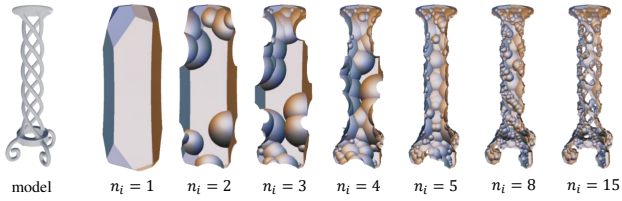

Fig. 2. Visualization of the spheres carving the initial volume through several iterations of *Sphere Carving*. The carved volume often approaches the model even for complex geometric shapes with a non-zero genus and several components.

external acceleration structure, such as a grid or an octree [Hart 1996].

The construction of the bounding volumes requires a small number of field function queries $\#f$, from a few thousand for complex objects to less than one hundred for an exact signed distance field. The algorithm operates for a large class of implicit surfaces: the only requirement is that the function f should be conservative. This allows us to perform faster queries in a large portion of space, which is typically useful for Sphere Tracing or polygonization algorithms. Moreover, our method produces a lightweight hierarchy of convex volumes, which can be directly used to generate a modified function $\widetilde{f}$ with accelerated field function queries.

Our contributions are as follows: 1) We introduce a novel and robust method to calculate a bounding volume given a conservative input signed distance field. 2) We propose a method for creating a lightweight hierarchy of convex volumes, whose representation can be incorporated into the function definition, resulting in small computational overhead. To the best of our knowledge, this method is the first to compute bounding volumes agnostic of the construction of the function f.

2 RELATED WORK

This related work focuses on techniques designed to accelerate queries on implicit surfaces. Regardless of the underlying representation, complex algorithms – such as surface normal evaluation, polygonization, or ray-surface intersection – ultimately rely on one fundamental query: evaluating the field function $f(\mathbf{p})$ at a specific position $\mathbf{p}$. Therefore, previously published methods either aim to reduce the *number* of field function calls $\#f$ or to decrease the *computational cost* of evaluating $f(\mathbf{p})$.

Implicit surfaces can be classified into three categories: discrete, procedural, and neural. Discrete models define the signed distance function through the interpolation of discrete samples, typically organized in a regular or adaptive grid [Frisken et al. 2000] or in a narrow band structure [Museth et al. 2002]. Neural implicit surfaces represent the field function as a neural network, allowing to encode arbitrary input shapes as implicit surfaces [Coiffier and Béthune 2024; Park et al. 2019; Sharp and Jacobson 2022]. Procedural implicit surfaces combine closed-form mathematical equations [Pasko et al. 1995] or utilize trees or graphs that combine various primitives and operators [Blinn 1982; Riso et al. 2024; Wyvill et al. 1999, 1986]. However, querying the field function can become computationally intensive as the complexity of the shapes increases.

Reducing the number of function calls $\#f$, i.e. the complexity, is a general strategy for improving performance that depends on the considered algorithm. Methods for processing implicit surfaces take advantage of the properties of the signed distance function f and the coherency of the queries $f(\mathbf{p})$ to reduce the number of function calls and accelerate algorithms. This is typically the case for optimizing ray-surface intersection algorithms through Lipschitz-based algorithms [Kalra and Barr 1989] such as *Sphere Tracing* [Bán and Valasek 2023; Bálint and Valasek 2018; Hart 1996], Segment Tracing [Galin et al. 2020], or second-order approximation of the signed distance function along the ray [Aydinlilar and Zanni 2023]. Those approaches need to evaluate a global or a local Lipschitz bound. This is also the case for grid-based polygonization algorithms that adopt a continuation [Wyvill et al. 1986] or a sweeping strategy [Lorensen and Cline 1987] to reuse signed distance function queries 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.[2015].

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. Caching distance data [Schmidt et al. 2005] can be effective but limited in its application because of the increasing memory usage. Local piecewise approximation of the field function in a grid or an octree [Pujol and Chica 2023] suffers from the same limitation. Simplifying the function expression is another method to improve computational efficiency. Screen-space pruning of BlobTrees [Zanni 2024] relies on the ability to bound the influence of primitives and operators in space but may require costly pre-processing steps. Interval arithmetic [Aydinlilar and Zanni 2023; Keeter 2020] 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 additional specific queries.

Finally, acceleration techniques usually rely on additional data structures *embedded* or *external* to the representation of the implicit object. Some models built from compactly supported primitives such as Blobs [Wyvill et al. 1986] or particularly the BlobTree [Wyvill et al. 1999], 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 [Gourmel et al. 2010] or octrees [Galin et al. 2020; Hart 1996] 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 [Pujol and Chica 2023]. 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 [Aydinlilar and Zanni 2023; Keeter 2020] or Lipschitz techniques [Galin et al. 2020] that need specific knowledge for every primitive and operator in the construction tree.

Recent results in implicit modeling [Sellán et al. 2023, 2024] utilize the field function value $f(\mathbf{p})$ to define empty spheres in space and provide efficient polygonization. Our method draws inspiration

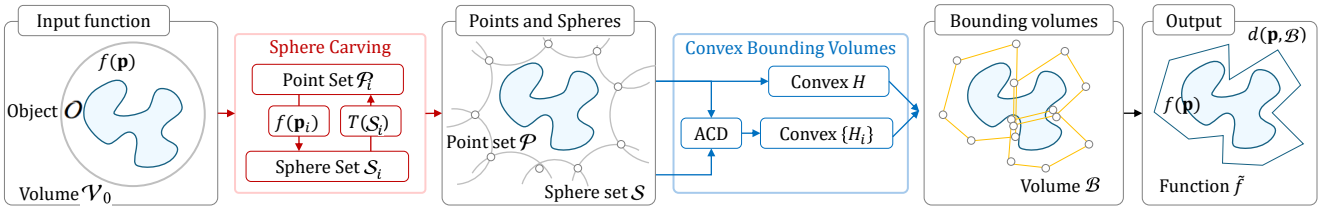

Fig. 3. Given a conservative signed distance function f and a volume $\mathcal{V}_0$, *Sphere Carving* progressively carves spheres $\mathcal{S}$ around the object while maintaining a point set $\mathcal{P}$ defined as the trilateration of $\mathcal{S}$. Next, we construct bounding volumes fitting the object using an approximate convex decomposition algorithm over $\mathcal{P}$. The resulting set of convex shapes is then used to define an accelerated function $\tilde{f}$.

from this approach by using empty spheres to quickly carve out space around the object. This process leads to a point set that is close to the surface and can be used to obtain a bounding volume with few queries.

3 OVERVIEW AND NOTATIONS

3.1 Fundamentals concepts

An implicit surface $\partial\mathcal{O}$ is defined as the 0-level set of a function $f : \mathbb{R}^3 \to \mathbb{R}$:

$$\partial\mathcal{O} = \{\mathbf{p} \in \mathbb{R}^3,\ f(\mathbf{p}) = 0\}$$

We consider the subset of implicit surfaces called signed distance fields, where the function f computes a geometric distance to the object's surface, with positive distance values outside and negative values inside. We distinguish between the following different families: *exact*, *Lipschitz*, and *conservative* (Figure 4). Functions that do not fit these categories belong to the general class of scalar functions on which we cannot make assumptions.

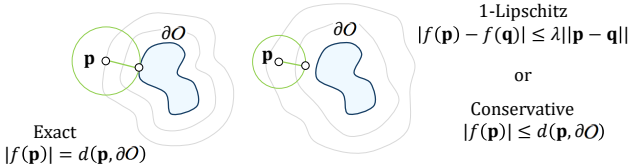

Fig. 4. Different classes of signed distance field, grey lines depict isopotential level sets. A key observation is that the empty sphere $S(\mathbf{p}, |f(\mathbf{p})|)$ is tangent to $\partial\mathcal{O}$ in the case of exact signed distance fields.

Exact signed distance functions compute the Euclidean distance d to the surface: $\forall \mathbf{p} \in \mathbb{R}^3, |f(\mathbf{p})| = d(\mathbf{p}, \partial\mathcal{O})$. Consequently, a sphere centered at any point $\mathbf{p}$ of radius $|f(\mathbf{p})|$ is always tangent to the surface of $\mathcal{O}$. *Lipschitz* functions admit a Lipschitz bound λ. Recall that a function f is Lipschitz if and only if there is a positive constant λ such that: $\forall (\mathbf{p}, \mathbf{q}) \in \mathbb{R}^3 \times \mathbb{R}^3, |f(\mathbf{p}) - f(\mathbf{q})| \leq \lambda\|\mathbf{p} - \mathbf{q}\|$. *Conservative* functions define a possibly not continuous lower bound for the distance to the surface. This means that $\forall \mathbf{p} \in \mathbb{R}^3, |f(\mathbf{p})| \leq d(\mathbf{p}, \partial\mathcal{O})$. They form a more general class than Lipschitz and their exact counterparts.

Although exact signed distance fields exist for some geometric primitives (sphere, box), this is no longer true when using some Boolean operators (union, intersection, difference) or any smooth operators (smooth union, *etc.*). At best, f may be *almost exact*, meaning that the distance to the object's surface may be slightly underestimated. *Sphere Carving* is general and only requires the function to be a conservative signed distance field. In that case, the fundamental empty-sphere criterion is verified. Let $S(\mathbf{c}, r)$ denote the sphere centered at $\mathbf{c}$ of radius r, we have:

$$\forall \mathbf{p} \in \mathbb{R}^3\ S(\mathbf{p}, |f(\mathbf{p})|) \cap \partial\mathcal{O} = \varnothing$$

When there is no ambiguity, we denote $S(\mathbf{p})$ the empty sphere $S(\mathbf{p}, |f(\mathbf{p})|)$.

3.2 Overview

Starting from an object $\mathcal{O}$ defined by a (black box) signed distance function f, we aim at computing a bounding volume using only the distance information (Figure 3). We start from an initial large bounding volume $\mathcal{V}_0$ (Section 4.1) enclosing the object and iteratively carve a volume $\mathcal{V}$ around $\mathcal{O}$, while guaranteeing never to intersect it. As an analogy to sphere tracing, which computes an intersection with the surface by iteratively discarding empty segments along a ray, *Sphere Carving* computes a bounding volume of the surface by iteratively carving empty spheres in space. At every iteration, we compute a set of empty spheres $\mathcal{S}_k$ that defines the carved volume as $\mathcal{V}_k = \mathcal{V}_0 \backslash \mathcal{S}_k$. From this sphere set $\mathcal{S}_k$, we manage to find a set of interesting points $\mathcal{P}_k$ on the surface of $\mathcal{V}_k$, by computing and selecting the relevant 3-spheres intersections of $\mathcal{S}_k$. The set of spheres is then augmented (Section 4.2) by querying f on those points $\mathcal{S}_{k+1} = \mathcal{S}_k \cup \{S(\mathbf{p}, f(\mathbf{p})) \mid \mathbf{p} \in \mathcal{P}_k\}$.

The process of *Sphere Carving* usually converges to a volume $\mathcal{V}$ in less than 20 iterations (depending on how conservative f is) and gives a final set of points $\mathcal{P}$ close to the surface $\partial\mathcal{O}$ that we exploit to build one or several bounding volumes $\mathcal{B}$ of different types (Section 5). Let $H(\mathcal{P})$ denote the geometric convex hull of a point set $\mathcal{P}$, we exploit the property $H(\mathcal{P}_k) \supseteq \mathcal{O}$, which is always satisfied for the point set $\mathcal{P}_k$, to generate convex bounding volumes around $\mathcal{O}$. The construction of those bounding volumes only exploits the information given by the *Sphere Carving* (*i.e.* $\mathcal{S}$ and $\mathcal{P}$) and never requires supplementary queries of f.

Ultimately, the bounding volume $\mathcal{B}$ is used to build a conservative variant $\tilde{f}$ whose field function queries are defined as the distance to its bounding volume $d(\mathbf{p}, \mathcal{B})$ if $\mathbf{p}$ lies outside of $\mathcal{B}$, and f otherwise, formally:

$$\tilde{f}(\mathbf{p}) = d(\mathbf{p}, \mathcal{B}) \text{ if } d(\mathbf{p}, \mathcal{B}) > \varepsilon, \quad \tilde{f}(\mathbf{p}) = f(\mathbf{p}) \text{ otherwise.}$$

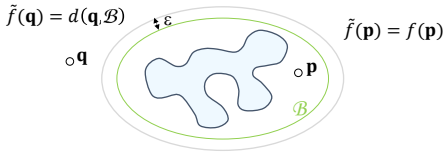

Fig. 5. Construction of the conservative signed distance function $\widetilde{f}$.

The $\varepsilon > 0$ constant is important and guarantees that the 0-level set of $\widetilde{f}$ is the same as f (see Figure 5). We fixed $\varepsilon = 0.01$ in all our experiments. This variant is constructed as a conservative signed distance function and provides fast queries of the field function whenever $\mathbf{p}$ is outside of $\mathcal{B}$.

Algorithm 1: *Sphere Carving.*

Input: Conservative signed distance field f, initial large volume $\mathcal{V}_0$, distance threshold $\tau > 0$.
Result: Set of empty spheres $\mathcal{S}$, point set $\mathcal{P}$ whose convex hull bounds the surface defined by f.

$\mathcal{P} \leftarrow$ sample surface($\mathcal{V}_0$); `// Section 4.1`
$\mathcal{S} \leftarrow \{$ sphere$(\mathbf{p}, f(\mathbf{p})) \mid \mathbf{p} \in \mathcal{P} \}$;
while $\max(f(\mathcal{P})) > \tau$ **do**
  $\mathcal{P} \leftarrow$ intersections(sphere triplets($\mathcal{S}$));
  $\mathcal{P} \leftarrow \mathcal{P} \setminus$ interior($\mathcal{S}$); `// Keep boundary points`
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{$ sphere$(\mathbf{p}, f(\mathbf{p})) \mid \mathbf{p} \in \mathcal{P}, f(\mathbf{p}) > \tau \}$;
end
return $S, \mathcal{P}$

4 SPHERE CARVING

At every step k of the algorithm (see Algorithm 1), we compute a set of carving spheres $\mathcal{S}_k$ and its corresponding point set $\mathcal{P}_k$. For the next iteration $k+1$ and to continue carving, we need to place new spheres located on the surface of $\mathcal{V}_k = \mathcal{V}_0 \backslash \mathcal{S}_k$. Thus, we need to build a set $\mathcal{P}_k$ of points on $\mathcal{V}_k$. While this may be done in various ways, we compute $\mathcal{P}_k$ as all the 3-spheres intersections T on the set of spheres $\mathcal{S}_k$ (*i.e.*, a *trilateration*, see Appendix A.2), and select the points on the surface of $\mathcal{V}_k$ (*i.e.*, the points outside of the spheres). Placing new points on 3-sphere intersections (see Figure 8) ensures the convergence of the process: placing a sphere on an intersection will always create new intersections for the next step, and therefore new empty spheres that decrease the carved volume. Although the process is not guaranteed to converge towards the exact shape O, it has always been the case in our experiments.

Let ∂S denote the surface of the sphere, S the ball whose boundary is ∂S, and interior($\mathcal{S}_k$) the interior of the volume defined by the sphere set, we have:

$$T(\mathcal{S}_k) = \{\partial S_a \cap \partial S_b \cap \partial S_c, (S_a, S_b, S_c) \in \mathcal{S}_k^3\}$$
$$\mathcal{P}_k = \widetilde{T}(\mathcal{S}_k) = T(\mathcal{S}_k) \backslash \text{interior}(\mathcal{S}_k)$$

In practice, every 3-sphere intersection $S_a \cap S_b \cap S_c$ is checked for inclusion in the spheres of the complementary set $\{S_i, i \notin \{a,b,c\}\}$.

The sets $\mathcal{S}_k$ and $\mathcal{P}_k$ satisfy two important properties (see Figure 6) that constitute the foundations of the method.

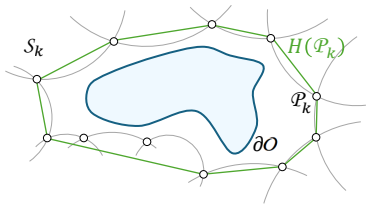

Fig. 6. Notations and properties: the set of points $\mathcal{P}_k$ is computed by *trilateration* from the sphere set $\mathcal{S}_k$ and the object remains inside the convex hull (even when all points do not lie on the convex hull): $O \subset H(\mathcal{P}_k)$.

Property 1. The sphere set $\mathcal{S}_k$ is outside the object O defined by the function f:

$$\mathcal{S}_k \cap O = \varnothing \Leftrightarrow \forall \mathbf{p} \in \mathcal{S}_k, f(\mathbf{p}) > 0$$

This comes from the construction of the sphere set: the initial points on $\mathcal{V}_0$ are outside O, and all carving spheres satisfy the empty-sphere criterion. The intersection points of carving spheres are necessarily outside O. Consequently, the new spheres built from the set $\mathcal{P}$ do not intersect O. This ensures that at every iteration, we cannot intersect the object of interest (Section 4.2).

Property 2. The convex hull $H(\mathcal{P}_k)$ of the point set $\mathcal{P}_k$ is bounding the object O:

$$O \subset H(\mathcal{P}_k) \Leftrightarrow \forall \mathbf{p} \notin H(\mathcal{P}_k), f(\mathbf{p}) > 0$$

This second property comes from the observation that the convex hull of the intersection points $H(\mathcal{P})$ always lies inside the sphere set (see Figure 6) , which is empty from the previous property. Therefore, no part of the object O will intersect the convex hull $H(\mathcal{P})$. Thus, the bounding volumes that we construct are guaranteed to strictly contain the object of interest, a property that we use in Section 5.

4.1 Initialization

For the carving process to effectively produce a valid bounding volume of O, its initialization needs to meet some simple yet important essential constraints.

As mentioned earlier, the carving starts from an already bounding region of space $\mathcal{V}_0$, that can be taken as large as needed, provided that f is well-defined in that volume. Points are sampled on its surface and used to produce the first sphere set $\mathcal{S}_0$ by evaluating f. As illustrated in Figure 7, to provide a valid initialization, this sphere set has to cover the entire surface of $\mathcal{V}_0$.

In practice, $\mathcal{V}_0$ is chosen as a large icosphere (a recursively subdivided icosahedron), for two main purposes: 1) the sampling is straightforward as we can use the vertexes of the icosphere, and 2) the coverage constraint of $\mathcal{V}_0$ can be checked per triangle $\triangle(\mathbf{a},\mathbf{b},\mathbf{c})$, using only the spheres of its three vertexes (see Appendix A.1 and Figure 25):

$$\forall \triangle(\mathbf{a},\mathbf{b},\mathbf{c}) \in \mathcal{V}_0,\ \triangle(\mathbf{a},\mathbf{b},\mathbf{c}) \subset S(\mathbf{a}) \cup S(\mathbf{b}) \cup S(\mathbf{c})$$

The subdivision level of the icosphere can then be adapted to the coverage criterion if needed. At the end of the process, we get an

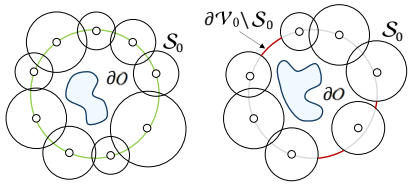

Fig. 7. Synthetic illustration of valid (left) and invalid (right) initial sphere set $\mathcal{S}_0$ for an initial sphere volume $\mathcal{V}_0$.

initial carved volume $\mathcal{V}_0\backslash\mathcal{S}_0$ and a point set $\mathcal{P}_0$. This icosphere sampling strategy is beneficial, as our coverage constraint is expressed per *triangle*. Other strategies may be used as long as this constraint is fulfilled.

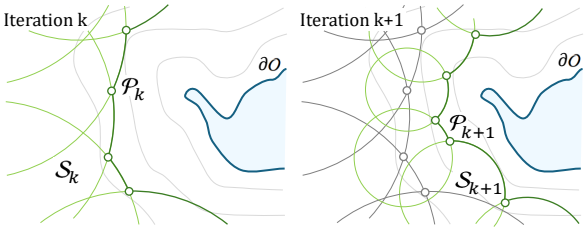

Fig. 8. Overview of the iterative process: the sphere set $\mathcal{S}$ progressively carves $\mathcal{V}_0$ towards the object at every iteration.

4.2 Iteration

The *Sphere Carving* algorithm consists in progressively carving new spheres from the volume $\mathcal{V}_0$ to approach the object O. We consider the system at iteration k, with the sphere set $\mathcal{S}_k$ and the associated point set $\mathcal{P}_k$. We iteratively define the new set of spheres $\mathcal{S}_{k+1}$ by adding new spheres located at the intersection points in $\mathcal{P}_k$ and with a radius defined by the signed distance function f evaluated at their centers:

$$\mathcal{S}_{k+1} = \mathcal{S}_k \cup \{S(\mathbf{p}, f(\mathbf{p})) \mid \mathbf{p} \in \mathcal{P}_k\}$$

The point set is updated to contain the valid intersections of the new sphere set by *trilateration*. Figure 8 illustrates how the series of point sets $\mathcal{P}_k$ progress toward the shape at each iteration.

The computation of the valid intersection points of the sphere set $\mathcal{S}_{k+1}$ is performed as follows. We compute the *trilateration* for the combination of all sphere triplets of $\mathcal{S}_k$, and only keep the intersections that do not belong to the set of previously computed carved spheres: $\mathcal{P}_k = \widetilde{T}(\mathcal{S}_k)$.

The carved volume $\mathcal{V}_0\backslash\mathcal{S}_{k+1}$ always contains the object O, while diminishing at every iteration. Figures 2 and 9 show its evolution at different iterations for one model, while Figure 10 shows three converged carved volumes, for various models.

The overall process requires a small number of field function evaluations. The most demanding part of the algorithm comes from evaluating every possible triplet of sphere intersections within the sphere set. Using an acceleration data structure may reduce this otherwise computationally intensive step, as discussed in Section 6.4.

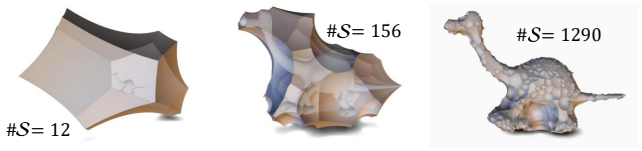

Fig. 9. Visualization of the carved volume $\mathcal{V}_0\backslash\mathcal{S}_k$ with k carving iterations for the dinosaur model, here $k \in \{1, 4, 7\}$. The first set of spheres is still far from ∂O, showing that the function is not exact but conservative.

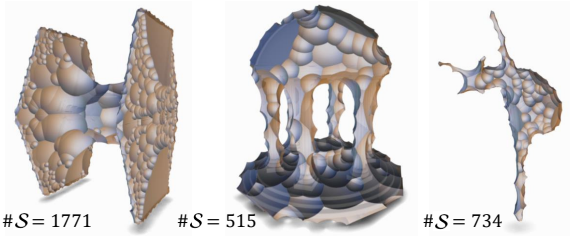

Fig. 10. Visualization of the final carved volume $\mathcal{V}_0\backslash\mathcal{S}$ for three models: a tie-fighter, a temple, and a snail.

4.3 Termination criterion

Sphere Carving involves only a few parameters: the size of the initial volume, denoted as $\mathcal{V}_0$, which is usually taken as large as the computation precision can allow (in practice, we take it two orders of magnitude larger than the object) and the distance threshold for the spheres, denoted as τ (set to 0.1 in our experiments). This last parameter serves as the primary stopping criterion for the algorithm. The algorithm will terminate if no new sphere is added during a given iteration, meaning that every point $\mathbf{p}$ in $\mathcal{P}$ satisfies $f(\mathbf{p}) < \tau$. The value of τ is influenced by the overall quality of the signed distance function, which is discussed in Section 6.1. In practice, the complexity of the sphere intersection routine is $O(n^3)$ and it is possible to impose a maximum number of spheres or limit the number of iterations.

5 BOUNDING SHAPES

The *Sphere Carving* algorithm produces a final sphere set $\mathcal{S}$ and its associated set of points $\mathcal{P}$. Recall that the convex hull of $\mathcal{P}$ is bounding the object (Property 2) and may thus be exploited in different ways to build efficient bounding volumes.

We compute the bounding volume $\mathcal{B}$ from the sphere set $\mathcal{S}$ as a set of convex polyhedra $\mathcal{H}$ or a set of ellipsoids $\mathcal{E}$. Recall that $\widetilde{f}$ is defined as the distance to its bounding volume $d(\mathbf{p}, \mathcal{B})$ if $\mathbf{p}$ lies outside of $\mathcal{B}$, and f otherwise:

$$\widetilde{f} = d(\mathbf{p}, \mathcal{B}) \text{ if } d(\mathbf{p}, \mathcal{B}) > \varepsilon, \quad \widetilde{f} = f(\mathbf{p}) \text{ otherwise}$$

Here we address the construction of $d(\mathbf{p}, \mathcal{B})$.

5.1 Implicit convex hulls

The convex hull $\mathcal{H} = H(\mathcal{P}_n)$ is defined as the intersection of a set of oriented half-spaces $\{\Pi_i\}$ characterized by a point $\mathbf{c}_i$ and their normal $\mathbf{n}_i$, corresponding to the faces of the polyhedron. A corresponding conservative signed distance function (different from

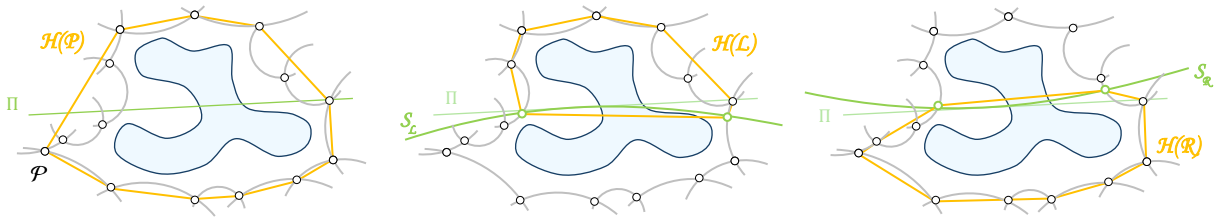

Fig. 11. The convex hull $H(\mathcal{P})$ of the set of points $\mathcal{P}$ is cut by a partitioning half-space Π. We approximate Π by two large-radius spheres, $S_{\mathcal{L}}$ and $S_{\mathcal{R}}$ to create two subsets of points: $\mathcal{L}$ and $\mathcal{R}$. We then add the intersections of the cutting sphere with the set of spheres $\mathcal{S}$ to these two subsets.

the Euclidean distance field) may be defined as:

$$d(\mathbf{p}, \mathcal{H}) = \max_i d(\Pi_i, \mathbf{p}) \qquad d(\Pi_i, \mathbf{p}) = (\mathbf{p} - \mathbf{c}_i) \cdot \mathbf{n}_i$$

The convex hulls often hold several hundreds of half-spaces, making the corresponding signed distance field computationally intensive. However, its cost should always remain negligible compared to the cost of the function f defining the object. Therefore, we simplify the set of plane $\mathcal{H}$ to a more manageable one $\widetilde{\mathcal{H}}$ while still preserving a convex bound.

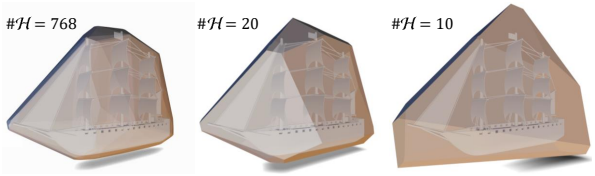

Fig. 12. Simplified convex bounding volumes: one with full resolution with 768 half-spaces, and simplified versions using only 20 and 10 planes. With ≈ 20 half-spaces, the shape of the convex hull is almost equivalent to the full resolution, while the evaluation of $d(\mathbf{p}, \widetilde{\mathcal{H}})$ is significantly more efficient.

Our approach comes from the observation that removing a half-space Π_i from the set $\mathcal{H}$ does not break the bounding property of the hull: removing a plane only adds volume to the convex shape. Therefore, selecting the most relevant half-spaces from the set $\mathcal{H}$ is an effective way to simplify the hull. We apply a k-means algorithm to the set of half-spaces $\mathcal{H}$, searching for m clusters and only considering their orientations $\mathbf{n}_i$. After convergence, we project the resulting clusters onto the existing half-spaces to get a reduced set $\widetilde{\mathcal{H}}$ which continues to bound the object and offers a more efficient evaluation of the distance $d(\mathbf{p}, \widetilde{\mathcal{H}})$. The parameter m serves as a trade-off between the accuracy of the bound (where higher values of m yield a better approximation of the convex hull) and the efficiency of the distance query.

Figure 12 illustrates various simplifications of an initial convex hull at full resolution (768 half-spaces) alongside simplified versions featuring 20 and 10 half-spaces. Notably, significantly reducing m does not adversely affect the quality of $\widetilde{\mathcal{H}}$. Figure 13 shows the volume of $\widetilde{\mathcal{H}}$ as a function of the number of half-spaces m, averaged for 17 different objects. In our implementation, we set $m \approx 20$.

5.2 Approximate Convex Decomposition

Extracting a single bounding volume does not fully utilize the information produced by the *Sphere Carving* algorithm. The point

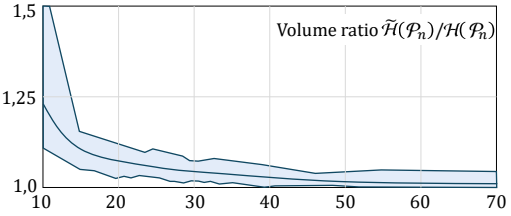

Fig. 13. Average and span of ratio of volumes $\widetilde{\mathcal{H}}/\mathcal{H}$ as a function of m.

set $\mathcal{P}$ contains more information than just its convex hull, as it conforms more closely to the shape (see final carved volumes in Figures 2, 9 and 10). Using a straightforward approximate convex decomposition algorithm, we show that this additional information can be leveraged to define a tighter volume as a union of simpler shapes.

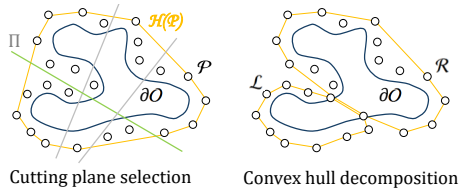

Fig. 14. From the point set $\mathcal{P}$ produced by *Sphere Carving*, we sample candidate cutting planes Π, compute their respective scores, and select the candidate minimizing the volume of the set of new hulls $\sigma(\mathcal{P}, \Pi)$ (left, green).

The approximate convex decomposition method [Thul et al. 2018] (see Figure 14) recursively cuts the point set $\mathcal{P}$ into a user-defined number of parts along carefully selected cutting planes. This process does not require further evaluations of the signed distance function f, as we only utilize the sphere set $\mathcal{S}$ and its corresponding point set $\mathcal{P}$. Below, we explain how to select the cutting planes and split each part.

Choosing an effective cutting plane is crucial for any convex decomposition process. To do so, we need a relevant metric that evaluates the relative quality of various cutting planes. When it comes to accelerating field function queries f using bounding volumes, the goal is to achieve the tightest possible bounding volume. Let $\mathcal{L}$ and $\mathcal{R}$ denote the two disjoint subsets of $\mathcal{P}$, separated by a plane Π:

$$\mathcal{L} = \{\mathbf{p} \in \mathcal{P}, d(\Pi, \mathbf{p}) < 0\} \qquad \mathcal{R} = \{\mathbf{p} \in \mathcal{P}, d(\Pi, \mathbf{p}) \geq 0\}$$

We apply the following straightforward score function σ to the subdivided point set (with V the volume function):

$$\sigma(\mathcal{P}, \Pi) = V(H(\mathcal{L})) + V(H(\mathcal{R}))$$

Exploring the space of cutting planes can rapidly become expensive and is not of paramount importance compared to the implied associated cost. Therefore, we only sample a few hundreds planes $\{\Pi_i\}$, and choose the cutting plane Π as follows:

$$\Pi = \arg\min_i \sigma(\mathcal{P}, \Pi_i)$$

Although simply taking the convex hull of the split point sets $\mathcal{L}$ and $\mathcal{R}$ is not sufficient to guarantee the bounding property of those hulls, formally: $\mathcal{O} \not\subset H(\mathcal{L}) \cup H(\mathcal{R})$. Instead, we must compute the intersections of the cutting plane Π with the sphere set $\mathcal{S}$. While this process is straightforward for the initial cutting plane, it becomes increasingly complex after several recursive decompositions as we need to manage intersections between an arbitrary number of spheres and planes in three dimensions.

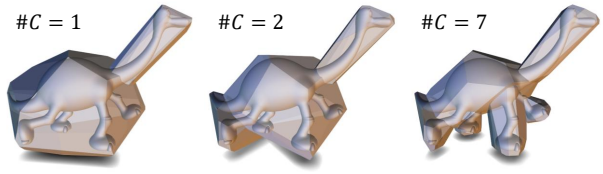

Fig. 15. Bounding volumes using an increasing number cutting plane #C.

We tackle this problem by considering each cutting plane as an infinite sphere added to the sphere set (see Figure 11). Indeed, a cutting plane carries the same type of information as any sphere of $\mathcal{S}$, as it marks a region of space as empty. By converting each cutting plane Π into an infinite sphere S_∞ and adding it to the sphere set $\mathcal{S}$, we maintain the same context as the *Sphere Carving* method and thus gain the same advantages: the convex hull created by the *trilateration* of this extended sphere set bounds a portion of the shape, formally:

$$H(\widetilde{T}(\mathcal{S} \cup S_\infty)) \supset \mathcal{O} \cap \Pi$$

We replace the cutting plane Π as a large sphere (we set its radius to the size of $\mathcal{V}_0$ in our implementation) that is tangent to Π, with its center located on a line perpendicular to the plane and passing through the centroid of the point set $\mathcal{L}$ or $\mathcal{R}$. This choice does not affect the bounding property of the resulting hull, as illustrated in Figure 15. Using large spheres instead of infinite spheres or planes will create slight overlaps between the different convex hulls, but will never break the bounding guarantee. Furthermore, it also allows us to utilize the intersection calculations from the *Sphere Carving* algorithm.

6 RESULTS

We implemented *Sphere Carving* (Section 4) in GLSL using Compute Shaders and Approximate Convex Decomposition (Section 5) in C++. A simple code is available at *link will be added upon acceptance* and a detailed breakdown of a parallel implementation is presented in Appendix A.3. All the models shown throughout this paper (Figure 1, 9, 10, 12, 15, 16, 17, 22) were rendered using *Sphere Tracing* [Hart 1996] in a standalone application (see accompanying video). Experiments were performed on a laptop computer equipped with an Intel Core i7-12800H clocked at 2.4 GHz with 32 GB of RAM, and NVIDIA GeForce RTX 3080 Ti Laptop GPU.

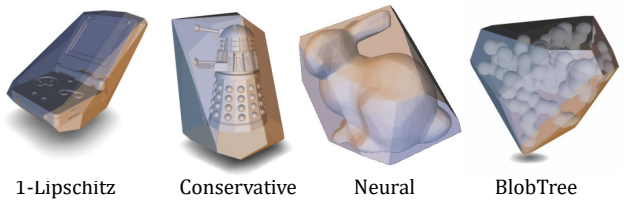

Fig. 16. Bounding volumes for various types of implicit surfaces (neural bunny from Coiffier et al. [2024], molecule inspired from Galin et al. [2020]).

Sphere Carving is compatible with a wide range of implicit surfaces, including neural 1-Lipschitz [Coiffier and Béthune 2024], Blob [Galin and Akkouche 1996] and BlobTree [Wyvill et al. 1999] models, conservative, Lipschitz, or exact signed distance field (Figure 16). We tested the algorithm on a set of 18 shapes from a public dataset [Takikawa et al. 2021] or made by professional artists, that we classified in three categories: almost exact, conservative, and overly conservative (also referred to as *bad*). The algorithm converges to a bounding volume provided that the function f is conservative. Figure 21 illustrates how an ill-defined signed distance function leads to a slower convergence of *Sphere Carving*. Overly conservative functions typically require a higher number of iterations to converge. Figure 17 shows the convex bounding volumes computed for various models, and Figures 15 and 19 show examples of approximate convex decomposition. Figure 17 illustrates the robustness of our method on a large set of shapes and topologies: high-genus (car, ship), flat areas (wings of the Tie Fighter, see also Figure 10), and multiple disconnected components (camera).

6.1 Performance

Speed. Sphere Carving performs best for exact or almost exact signed distance fields, where only one iteration is required to get a close bound to the surface. Table 1 reports various *Sphere Carving* statistics to obtain a single bound. The best performances are obtained with almost exact distance fields using a maximum iteration count of 1 and allowing up to 2000 spheres. In contrast, the conservative and overly conservative signed distance function require more iterations to get a precise volume, with a maximum iteration count set to 30 and the maximum sphere count to 30000. Other nearly exact fields share the same statistics and are not included in this table, such as Car, House, Raccoon, Mike, Tentacles, and House models, all referenced in Figure 17).

As the distance function becomes more conservative, more iterations n_i may be required to get a bound close to the surface of the object (see *conservative* and *bad* in Table 1). In all cases, constructing a single bounding volume is completed in a few seconds at most, and in less than 1ms with less than 100 function evaluations for exact or almost exact signed distance function, making it suitable for real-time application (see accompanying video). This efficiency is crucial when evaluating f becomes computationally intensive, such

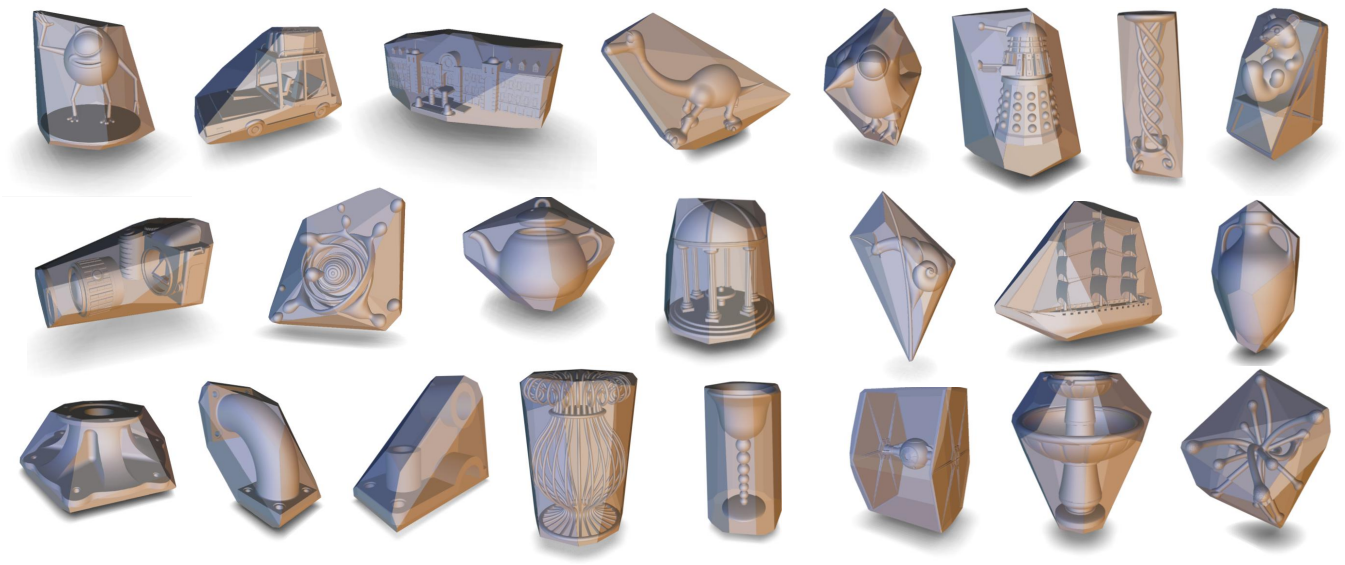

Fig. 17. Bounding volumes obtained by *Sphere Carving* on a variety of quasi-exact and conservative signed distance function.

Table 1. Statistics of *Sphere Carving* to get a single bound, with n_i the number of iterations, $\#\mathcal{S}$ the number of spheres, $\#\mathcal{P}$ the number of points, $\#f$ the number of field function calls, and t the time in ms.

Type	Scene (Figure)	n_i	Complexity		Performance	
			$\#\mathcal{S}$	$\#\mathcal{P}$	$\#f$	t
Bad	Dalek (17)	17	30778	19568	50843	2160
	Ship (12)	15	35450	21244	43213	4418
Conservative	Splash (17)	5	1790	3370	2369	6
	Teapot (17)	7	2276	2883	2713	18
	TieFighter (17)	10	2112	3686	4993	13
	Snail (10)	13	1285	1592	3013	15
	Dinosaur (9)	9	2219	3447	5304	24
	Camera (17)	13	256	562	906	6
	Glass (17)	6	2864	3176	2880	21
	Pump mount (17)	6	3318	5505	3539	8
≈ Exact	Temple (10)	1	42	80	42	< 1
	Car (17)					
	Console (16)					
	Pipe (17)					
	Joint (17)					
	Fountain (17)					
	Vase (17)					

as in complex construction trees that combine thousands of intricate primitives using blending, Boolean and deformation operators – a typical case in implicit modeling.

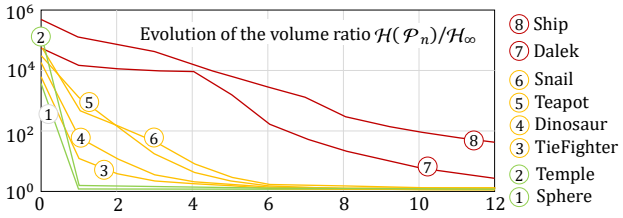

Fig. 18. Evolution of the volume ratio between the *Sphere Carving* result and the real convex hull of ∂O, for several objects (iterations in abscissa). The quality of f can be identified through the curves: the convergence is slower for very underestimated distances (in red), while a single iteration is enough for quasi-exact functions (in green).

The most computationally intensive part is the computation of all possible sphere intersections within the set of spheres $\mathcal{S}$, with a complexity of $O(n^3)$, where n represents the number of spheres. This may challenge the algorithm when dealing with overly conservative signed distance fields such as the *ship* model in Table 1 (see also Figure 21). In such cases, *Sphere Carving* requires more iterations to converge (see Figure 18), which often involves processing a large set of spheres $\mathcal{S}_k$ at each iteration. This case is similar to slow *Sphere Tracing*, which requires more steps for overly conservative functions. Still, our GPU implementation performs well for most models.

The approximate convex decomposition (see Section 5.2) requires a final point set $\mathcal{P}$ close to the shape surface, which in turn requires more iterations. Table 2 reports the statistics for an approximate convex decomposition. We purposely performed more iterations to get dense point clouds around the shapes enabling the decomposition into multiple bounds. The decomposition process usually takes a few seconds to complete. This is mainly due to the cutting plane selection step, which requires multiple convex hull computations to

Table 2. Statistics for approximate convex decomposition: n_i denotes the number of iterations, $\#c$, $\#\mathcal{S}$ and $\#\mathcal{P}$ the number of convex parts, spheres, and points, $\#f$ the number of field function calls, t the processing time (in ms), and t_d the approximate convex decomposition time (in s).

Scene (Figure)	n_i	$\#c$	Complexity		Performance		
			$\#\mathcal{S}$	$\#\mathcal{P}$	$\#f$	t	t_d
House (17)	10	4	548	963	1593	7	2.7
Raccoon (17)	20	4	1347	2875	4626	20	9.7
Mike (17)	13	5	601	1303	2026	7	8.1
Bird (17)	7	5	299	685	967	4	5.1
Tentacles (17)	14	8	806	1795	2830	10	12.0
Candlestick (17)	6	3	2212	4411	3646	9	25.7
Amphora (17)	7	3	2331	4750	3353	29	49.9

ensure that a good cutting plane is chosen. This step draws inspiration from approximate convex decomposition methods typically applied to meshes [Thul et al. 2018] and could be easily replaced with more efficient or faster methods if necessary. Nonetheless, results demonstrate that the point set $\mathcal{P}$ is accurate enough to bound the input object by a union of multiple convex polyhedra or ellipsoids (see Figures 15, 22 and 19).

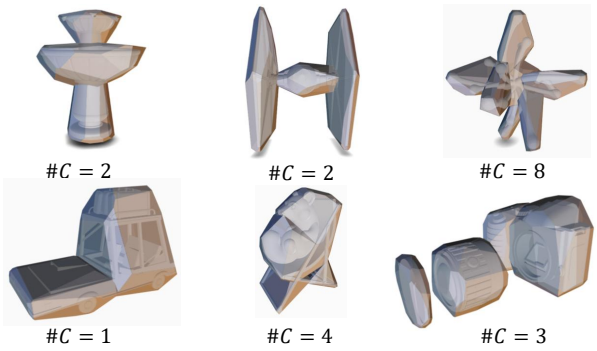

Fig. 19. Bounding volumes obtained using the approximate convex decomposition with a varying number of cutting planes $\#C$.

Memory. A key feature of the method is the compact representation of the bounding volume (either a set of convex polyhedra or a union of ellipsoids) as a signed distance field $d(\mathbf{p}, \mathcal{B})$ and embedded in the definition of $\widetilde{f}$. The simplification scheme (see Section 5.2) allows the bounding volume representation to remain compact in memory and be implemented on graphics hardware for sphere tracing applications. This approach is beneficial compared to external acceleration structures, such as grids or octrees, which tend to have a larger memory footprint and require additional code for querying the acceleration structure. Another advantage of embedding the bounding volume in the function definition is that we remain compatible with other specific acceleration techniques dedicated to specific algorithms, such as *Sphere Tracing* [Bálint and Valasek 2018; Keinert et al. 2014] or Segment Tracing [Galin et al. 2020], or even polygonization including early continuation methods [Wyvill et al. 1986] or grid-based approaches [Lorensen and Cline 1987], to recently published methods [Sellán et al. 2024].

Performance gain. The augmented field function $\widetilde{f}$ enables faster queries in various applications, including sphere tracing, polygonization, and simulation. Notably, queries located outside the bounding volume do not need to compute the computationally intensive function f of the object. In an experiment, we computed ~1 million queries within a box surrounding the object. We measured the query costs with and without using a bounding volume. The results indicate that the augmented $\widetilde{f}$ performs 3.3 to 4.9 times faster on average than the base field function in our set of test shapes, depending on the box size (from ×1.5 to ×2 larger than the object).

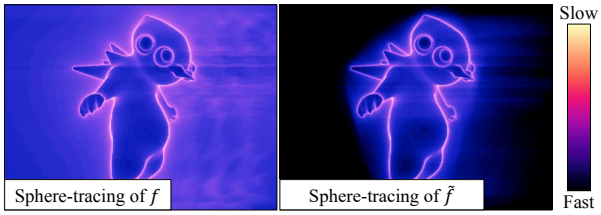

Fig. 20. Per-pixel cost comparison between sphere tracing of the original function f (left), and the augmented function $\widetilde{f}$ that uses the convex hull as a proxy (right). Artifacts on the right are due to GPU memory access pattern inconsistencies.

Figure 20 shows the cost of f against its augmented counterpart $\widetilde{f}$ in a Sphere Tracing scenario. Using a convex bound around the object can provide up to a 50% speedup.

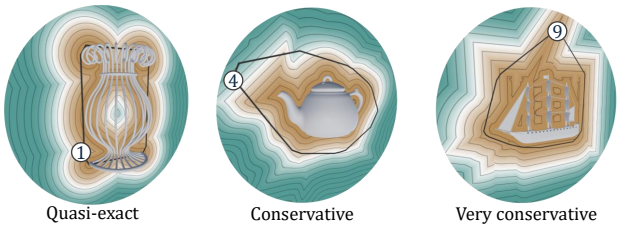

Fig. 21. Convex volumes for different qualities of f; as shown by the curves in Figure 18, more conservative functions require more iterations (reported in the circles).

6.2 Other bounding volumes

Our method is general and can be used to generate multiple types of bounding volumes. Oriented bounding boxes and k-dops can be directly derived from the set of point $\mathcal{P}$. We can adapt the algorithm to convex polyhedra defined as intersections of slabs, and to k-dops (Figure 22). Recall that slabs are formed by the intersection of two half-spaces that extend in opposite directions. k-dops are created by intersecting half-spaces, using a subset of directions [Klosowski et al. 1998]. Utilizing slabs and k-dops significantly speeds up the evaluation of the signed distance function $\widetilde{h}$, by dividing the number of scalar products by two.

Bounding ellipsoids can be generated by replacing the convex hull generation step (Section 5.1) from the set of points $\mathcal{P}$ with a minimum-volume enclosing ellipsoid generation. Figure 22 shows several decompositions produced with the same set of points $\mathcal{P}$, along with a single bounding volume as an inset.

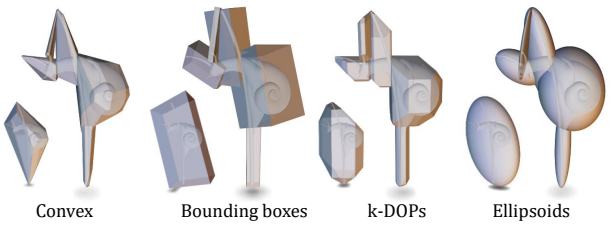

Fig. 22. Different types of bounding volumes: planes selected by k-means, oriented bounding boxes, k-dops (here $k = 18$) and ellipsoids.

Let $\mathcal{E}$ denote the ellipsoid of center $\mathbf{c}$, $\mathbf{R}$ its rotation matrix defining its axes, $\mathbf{D}$ a diagonal matrix, and $a < b < c$ the lengths of the axes. The following function defines a 1-Lipschitz (thus conservative) signed distance function:

$$d(\mathbf{p}, \mathcal{E}) = a\left(\|\mathbf{A}\,(\mathbf{p} - \mathbf{c})\| - 1\right) \qquad \mathbf{A} = \mathbf{D}(1/a, 1/b, 1/c)\,\mathbf{R}^t$$

Compared to convex polyhedra, ellipsoids may have a more conservative distance function depending on the relative axis lengths but are more compact in memory and computationally more efficient. The evaluation of the signed distance function requires only ≈ 4 scalar products and one square root, compared to n scalar products for a n-face convex polyhedron.

6.3 Comparison with other methods

Computing the bounding volume of a signed distance function is challenging because of the implicit characterization of the surface. An important feature of *Sphere Carving* is its ability to compute convex bounding volumes using a dozen to a few thousand field function queries at most.

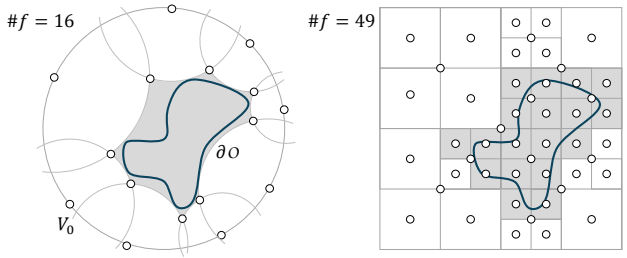

Fig. 23. Comparison of signed distance function evaluations for octree and *Sphere Carving*, in 2D, for a quasi-exact distance.

A first alternative strategy is to decompose space into a grid, potentially covering a very large domain around the shape, and compute the set of voxels straddling the surface and use them to extract the convex hull, which requires the costly evaluation of f at the vertices of the grid. Moreover, this approach does not provide any bounding guarantees. A variant is to rely on an *adaptive* decomposition of space using octrees [Hart 1996]. Starting from a large cube instead of the initial carving sphere $\mathcal{V}_0$, we recursively subdivide the cube if the value $f(\mathbf{c})$ at its center $\mathbf{c}$ is greater than the length of the diagonal of the cube, which guarantees that ∂O does not straddle it (see Figure 23). The convex hull H is computed from the eight vertices of the detected straddling cubes.

We conducted some experiments and compared the results with *Sphere Carving* (Figure 24). The octree typically requires significantly more evaluations of f to accurately obtain a good convex hull. For very conservative functions, however, this gain may be offset by the cost of intersecting many spheres. The final point set generated by the octree may have numerous coplanar points, particularly when the signed distance function f defines flat surface parts. This abundance of coplanar points can slow the computation of the convex hull, as these points may all contribute to the convex envelope. The snail shape is the only instance where the octree decomposition converges in fewer iterations compared to *Sphere Carving*. This may be attributed to significant anisotropy in the values of f, which results in many small spheres gradually moving towards the shape.

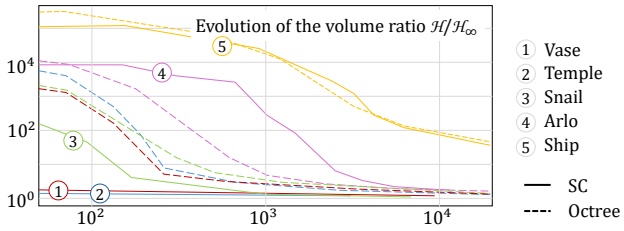

Fig. 24. Comparison of convex volume convergence between the lattice-free *Sphere Carving* and an octree-based approach. The graph reports the ratio $\mathcal{H}/\mathcal{H}_\infty$ as a function of the number of evaluation of f.

Interval and affine arithmetic are another way of computing bounds or excluding parts of the function expression in space [Duff 1992]. They have been successfully applied to the ray tracing of implicit surfaces [Keeter 2020; Knoll et al. 2009]. This family of methods relies on well-defined interval queries implemented for each primitive and operator of the implicit function. It is therefore less general than our method and cannot process black-box signed distance fields.

Recently, neural approaches have been employed to compute tight bounding volumes around complex objects [Liu et al. 2024]. While this method may be applied to signed distance field, it requires a long training time per shape (up to one hour) and does not provide strict guarantees regarding false negatives. In contrast, *Sphere Carving* can process a variety of implicit surface representations, terminates in a few seconds, and provides strict bounding guarantees.

6.4 Limitations

Sphere Carving involves maintaining a set of spheres from which we compute all possible intersection points. The size of this sphere set can increase rapidly with each iteration. Since we need to calculate the complete set of intersections among all spheres, this results in a computational complexity of $O(n^3)$, which can become computationally intensive. This issue typically arises in cases of conservative

signed distance fields that significantly underestimate the distance to the object's surface. Consequently,the *Sphere Carving* algorithm may require more iterations and more spheres to converge (see Figure 21).

In our experiments, we observed that the computation time rose to several seconds for the *ship* and *dalek* models (see Table 1). To alleviate this issue, the *trilateration* of all spheres could be optimized by using accelerating data structures adapted to the sphere set. We plan to explore these possibilities in future work.

Overly conservative functions may also require a larger initial volume $\mathcal{V}_0$, or a denser sampling of it to satisfy the initialization constraint (see Section 4.1). This is however not a problem for quasi-exact signed distance field. Finally, the bounding guarantee does not hold if the signed distance field overestimates the distance to the object. The algorithm does not break and still progress toward the shape, but the result is no longer guaranteed to be bounding. This behavior is similar to Sphere Tracing, where a ray may miss the shape due to overestimated distances.

7 CONCLUSION

Sphere Carving is an efficient method for computing bounding volumes agnostic of the representation of the signed distance function. This simple technique, straightforward to implement, generates point sets around the surface of an implicitly defined object. These point sets can be further processed to create a set of convex polyhedra or other types of bounding volumes, such as ellipsoids. Additionally, these bounding volumes can be used to define a variant of the signed distance function that provides speedup for function queries.

One advantage of our approach is that it is agnostic to the implicit surface representation; however, this characteristic also serves as a limitation since it does not take advantage of the construction model. Adapting to specific models and utilizing the power of commonly used construction trees or graphs may further reduce the cost of evaluating the signed distance function.

ACKNOWLEDGMENTS

We thank Luc Chamerlat, Inigo Quilez, and Nikie Monteleone for authoring some models used in this paper.

REFERENCES

Melike Aydinlilar and Cédric Zanni. 2023. Forward inclusion functions for ray-tracing implicit surfaces. *Computers & Graphics* 114 (2023), 190–200.

Róbert Bán and Gábor Valasek. 2023. Automatic Step Size Relaxation in Sphere Tracing. In *Eurographics 2023 - Short Papers*, Vahid Babaei and Melina Skouras (Eds.). The Eurographics Association.

James F. Blinn. 1982. A Generalization of Algebraic Surface Drawing. *ACM Trans. Graph.* 1, 3 (1982), 235–256.

Jules Bloomenthal and Brian Wyvill (Eds.). 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Csaba Bálint and Gábor Valasek. 2018. Accelerating Sphere Tracing. In *EG 2018 - Short Papers*, Olga Diamanti and Amir Vaxman (Eds.). The Eurographics Association.

Guillaume Coiffier and Louis Béthune. 2024. 1-Lipschitz Neural Distance Fields. *Computer Graphics Forum* 43, 5 (2024), e15128.

B. R. de Araújo, Daniel S. Lopes, Pauline Jepp, Joaquim A. Jorge, and Brian Wyvill. 2015. A Survey on Implicit Surface Polygonization. *ACM Computing Survey* 47, 4, Article 60 (2015), 39 pages.

Tom Duff. 1992. Interval Arithmetic Recursive Subdivision for Implicit Functions and Constructive Solid Geometry. *SIGGRAPH Computer Graphics* 26, 2 (1992), 131–138.

Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. 2000. Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'00)*. 249–254.

Eric Galin and Samir Akkouche. 1996. Blob Metamorphosis based on Minkowski Sums. *Computer Graphics Forum* 15, 3 (1996), 143–152.

Eric Galin, Eric Guérin, Axel Paris, and Adrien Peytavie. 2020. Segment Tracing Using Local Lipschitz Bounds. *Computer Graphics Forum* 39, 2 (2020), 545–554.

Olivier Gourmel, Anthony Pajot, Mathias Paulin, Loïc Barthe, and Pierre Poulin. 2010. Fitted BVH for Fast Raytracing of Metaballs. *Computer Graphics Forum* 29, 2 (2010), 281–288.

John C. Hart. 1996. Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces. *The Visual Computer* 12, 10 (1996), 527–545.

D. Kalra and A. H. Barr. 1989. Guaranteed Ray Intersections with Implicit Surfaces. *SIGGRAPH Computer Graphics* (1989).

Matthew J. Keeter. 2020. Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 39, 4 (2020), 141:1–141:10.

Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. 2014. Enhanced Sphere Tracing. In *Proceedings of Smart Tools & Apps for Graphics*. Eurographics Association, Cagliari, Italy.

James T. Klosowski, Martin Held, Joseph S. B. Mitchell, Henry Sowizral, and Karel Zikan. 1998. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics* 4, 1 (1998), 21—-36.

A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. Hansen, and H. Hagen. 2009. Fast Ray Tracing of Arbitrary Implicit Surfaces with Interval and Affine Arithmetic. *Computer Graphics Forum* 28, 1 (2009), 26–40.

Stephanie Wenxin Liu, Michael Fischer, Paul D. Yoo, and Tobias Ritschel. 2024. Neural Bounding. In *ACM SIGGRAPH 2024 Conference Papers*. Association for Computing Machinery, Article 98, 10 pages.

William E. Lorensen and Harvey E. Cline. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *SIGGRAPH Comput. Graph.* 21, 4 (1987), 163–169.

Ken Museth, David Breen, Ross Whitaker, and Alan Barr. 2002. Level Set Surface Editing Operators. *ACM Transactions on Graphics* 21, 3 (2002), 330–338.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Alexander Pasko, Valery Adzhiev, Alexei Sourin, and Vladimir Savchenko. 1995. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer* 11, 8 (1995), 429–446.

Eduard Pujol and Antonio Chica. 2023. Adaptive approximation of signed distance fields through piecewise continuous interpolation. *Computers and Graphics* 114 (2023), 337–346.

Marzia Riso, Élie Michel, Axel Paris, Valentin Deschaintre, Mathieu Gaillard, and Fabio Pellacini. 2024. Direct Manipulation of Procedural Implicit Surfaces. *ACM Transaction on Graphics* (2024).

Ryan Schmidt, Brian Wyvill, and Eric Galin. 2005. Interactive implicit modeling with hierarchical spatial caching. In *International Conference on Shape Modeling and Applications*. 104–113.

Silvia Sellán, Christopher Batty, and Oded Stein. 2023. Reach For the Spheres: Tangency-aware surface reconstruction of SDFs. In *SIGGRAPH Asia 2023 Conference Papers*. Article 73, 11 pages.

Silvia Sellán, Yingying Ren, Christopher Batty, and Oded Stein. 2024. Reach For the Arcs: Reconstructing Surfaces from SDFs via Tangent Points. In *SIGGRAPH 2024 Conference Papers*. Article 25, 11 pages.

Nicholas Sharp and Alec Jacobson. 2022. Spelunking the deep: guaranteed queries on general neural implicit surfaces via range analysis. *ACM Transactions on Graphics* 41, 4, Article 107 (2022), 16 pages.

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes. (2021).

Daniel Thul, L'ubor Ladický, Sohyeon Jeong, and Marc Pollefeys. 2018. Approximate convex decomposition and transfer for animated meshes. *ACM Transactions on Graphics* 37, 6, Article 226 (2018), 10 pages.

Brian Wyvill, Andrew Guy, and Eric Galin. 1999. Extending the CSG Tree - Warping, Blending, and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum* 18, 2 (1999), 149–158.

Geoff Wyvill, Craig McPheeters, and Brian Wyvill. 1986. Data Structure for Soft Objects. *The Visual Computer* 2 (1986), 227–234.

Cédric Zanni. 2024. Synchronized Tracing of Primitive-based Implicit Volumes. *ACM Transactions on Graphics* 44, 1, Article 6 (2024), 15 pages.

A APPENDIX

A.1 Triangle coverage test

The sphere carving algorithm relies on the property that the sampling of the initial volume $\mathcal{V}_0$ is dense enough to create sphere intersections (see Section 4.1).Therefore, we need to check whether a triangle T is completely outside of O:

$$\forall \mathbf{p} \in T,\ f(\mathbf{p}) > 0$$

The evaluation of f at the vertices of the triangle T with vertices $\mathbf{v}_i$, $i \in \{0, 1, 2\}$ yields three empty spheres $S_i(\mathbf{v}_i, f(\mathbf{v}_i))$. We need to verify whether those three spheres cover T completely. This is equivalent to checking that no point of T is lying outside of the circles centered at its vertices:

$$\forall \mathbf{p} \in T,\ \exists\, i \in \{0, 1, 2\},\ \|\mathbf{p} - \mathbf{v}_i\| < f(\mathbf{v}_i)$$

We designed a test based on the following observation: only two cases arise where the triangle is not covered: 1) one circle is disjoint from the other two, and 2) an intersection point of two circles is lying inside T and outside of the third circle. If any of those two conditions is true, the triangle is not entirely covered by spheres, and the test fails. Figure 25 illustrates the two cases. The code implementing this algorithm is available at *link will be added upon acceptance.*

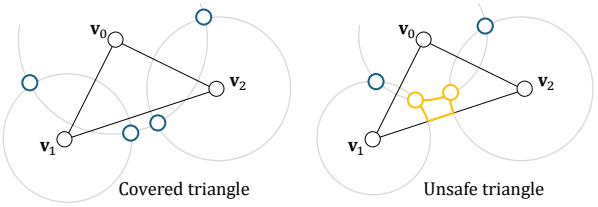

Fig. 25. The triangle on the left is completely covered by the three circles $S_i(\mathbf{v}_i, f(\mathbf{v}_i))$, whereas the triangle on the right doesn't satisfy the coverage test: 2 intersection points are lying inside $T$and outside the third circle.

A.2 Sphere trilateration

We detail here the computation of the intersections of three spheres $S_i(\mathbf{p}_i, r_i)$, $i \in \{0, 1, 2\}$. Finding intersections is equivalent to solving the following system for $\mathbf{p}$:

$$(\mathbf{p}_i - \mathbf{p})^2 = r_i^2 \quad \forall i \in \{0, 1, 2\} \tag{1}$$

We consider a local frame $(\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z)$, aligned with the plane passing through the centers $\mathbf{p}_i$:

$$\mathbf{e}_z = \frac{(\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)}{\|(\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)\|}$$
$$\mathbf{e}_x = \frac{\mathbf{p}_1 - \mathbf{p}_0}{\|\mathbf{p}_1 - \mathbf{p}_0\|}$$
$$\mathbf{e}_y = \mathbf{e}_z \times \mathbf{e}_x$$

Let $a = ||\mathbf{p}_1 - \mathbf{p}_0||$, $b = (\mathbf{p}_2 - \mathbf{p}_0) \cdot \mathbf{e}_x$, and $c = (\mathbf{p}_2 - \mathbf{p}_0) \cdot \mathbf{e}_y$, the equation 1 can be rewritten as:

$$x^2 + y^2 + z^2 = r_0^2$$
$$(x - a)^2 + y^2 + z^2 = r_1^2$$
$$(x - b)^2 + (y - c)^2 + z^2 = r_2^2$$

Solving the system yields:

$$x = \frac{r_0^2 - r_1^2 + a^2}{2a}$$
$$y = \frac{r_1^2 - r_2^2 - (x - a)^2 + (x - b)^2 + c^2}{2c}$$

Then, we can compute z, provided that the term under the radical is positive (meaning that the intersection exists):

$$z = \sqrt{r_0^2 - x^2 - y^2}$$

Finally, we return to the original frame:

$$\mathbf{p} = \mathbf{p}_0 + x.\mathbf{e}_x + y.\mathbf{e}_y \pm z.\mathbf{e}_z$$

A.3 Parallel Sphere Carving algorithm

We detail here a parallel-friendly implementation for a single *Sphere Carving* iteration. As shown in Algorithm 2, it requires four very simple kernels.

Algorithm 2: Parallel implementation for a single iteration.

Input : Conservative signed distance field f, current array of empty spheres $\mathcal{S}$.
Result: Updated array of empty spheres $\mathcal{S}$, array of points $\mathcal{P}$ whose convex hull bounds the surface defined by f.

```
pairs ← ∅;                                  // array of sphere index pairs
parallel for S_a ∈ S do
    for S_b ∈ S with b > a do
        if S_a ∩ S_b ≠ ∅ then
            pairs ← pairs ∪ (a, b);
P, P' ← ∅;                                  // arrays of 3D points
parallel for (a, b) ∈ pairs do
    for S_c ∈ S with c > b do
        P' ← P' ∪ intersections(S_a, S_b, S_c);
parallel for p ∈ P' do
    for S ∈ S do
        if p ∈ interior(S) then end thread;
    P ← P ∪ p;
parallel for p ∈ P do
    S ← S ∪ sphere(p, f(p));
return S, P;
```

To compute the intersection points for all ordered triplets of spheres, we first gather all ordered pairs of spheres that intersect using one kernel, and then test those pairs against all the sphere set with another kernel. To filter out an intersection point that lies on the interior of the sphere set volume, we simply iterate over all spheres and discard the point if for any sphere, its distance to the sphere center is less than the sphere radius, accounting for floating-point precision. The sphere set is augmented by evaluating in parallel the signed distance field at every point that was not discarded. All sets are implemented as arrays that are assumed to be pre-allocated with sufficient memory; pushing values in parallel onto an array requires one atomic counter per array.