# Simple and reliable boundary detection for meshfree particle methods using interval analysis

Marcos Sandim[a], Afonso Paiva[a,*], Luiz Henrique de Figueiredo[b]

[a]*Institute of Mathematical and Computer Sciences, USP, São Carlos, Brazil*
[b]*IMPA, Rio de Janeiro, Brazil*

**Abstract**

We present two novel algorithms for detecting boundary particles in 2D and 3D domains that are suitable for meshfree particle methods in Computational Fluid Dynamics. We combine a robust purely geometric sphere covering test based on interval analysis with an adaptive spatial subdivision of the sphere associated with a given particle. The methods are simple, fast, and easy to code. We report comparisons against state-of-the-art boundary detection methods in free-surface flow problems to demonstrate the effectiveness and accuracy of our approaches.

*Keywords:* boundary particles, interval analysis, geometric enclosure, free-surface flows, meshfree particle methods

*2020 MSC:* 76M28, 76B07

## 1. Introduction

Meshfree methods, such as Smoothed Particle Hydrodynamics (SPH) [1], Moving Least-Squares Particle Hydrodynamics (MLSPH) [2], and Moving Particle Semi-implicit (MPS) [3], provide attractive numerical discretizations for a wide range of Computational Fluid Dynamics applications. In these methods, particles typically carry a kernel function for a local approximation of attributes, such as density and pressure, instead of keeping track of the connectivity between

*Corresponding author
*Email addresses:* marcos.sandim@gmail.com (Marcos Sandim), apneto@icmc.usp.br (Afonso Paiva), lhf@impa.br (Luiz Henrique de Figueiredo)

particles during the simulation. Besides, meshfree interpolations can also be applied to local subdomains to perform numerical integration like Galerkin type methods [4, 5] or to achieve generalized finite difference stencils from scattered nodes [6].

An intrinsic advantage of meshfree methods over mesh-based methods is related to problems involving complex free-surface flows, efficiently capturing the topological changes (splitting and merging regions) that occur in the free surface, such as fragmentation, fracture, waves, splashing, and air bubbles creation. A delicate task for meshfree methods in handling boundary conditions at solid walls (e.g., no-slip and pressure Neumann condition) or across the free surface (e.g., constant pressure and kinematic condition), because such tasks demand accurate identification of *boundary particles*, that is, the particles that comprise the boundary of the fluid. This drawback is especially severe when solving problems where incompressibility must be satisfied [7] or surface tension has a meaningful effect on the flow behavior [8]. Beyond boundary conditions, detecting boundary particles is essential also in applications comprising level-set definition [9, 10] and particle shifting technology (PST) [11].

Despite its importance, the problem of accurately detecting boundary particles has not been extensively addressed in the literature. To detect the "exact" boundary, Dilts [2] provided a robust and reliable two-dimensional (2D) algorithm, where complex geometric predicates determine whether a particle is covered by its neighbor particles. Haque and Dilts [12] extended that algorithm to three-dimensional (3D) particles. However, their extension is neither straightforward to implement nor efficient, since computing mutual covering of spheres is expensive. Lo and Shao [13] proposed criteria where an SPH particle is classified as boundary particle if its density is less than a certain threshold with respect to a reference density. He et al. [14] and Liu et al. [15] presented a similar approach considering the SPH weighting of the particle distances. Although these methods involving SPH kernels are simple, they are inaccurate for free-surface flows with large deformations. Marrone et al. [9] performed boundary/non-boundary particle classification using a pre-processing step based on the spectrum of the SPH

correction matrix [16] combined with a geometrical test that verifies whether a neighbor particle lies in a conical region determined by a SPH approximation of normal vector. Barecasco et al. [17] simplified that method by replacing the normal vector by a cover vector defined by a weighted average of the particle positions. However, both approaches [9, 17] are sensitive to the non-uniformity of particle distribution due to the estimative of surface normals. Recently, Wang et al. [11] presented an optimization of the algorithm by Marrone et al. [9] just by changing the pre-processing step by the criteria regarding the SPH divergence of the particle positions as stated by Lee et al. [18]. Sandim et al. [10] introduced an accurate and efficient geometrical test reinterpreting the original problem as a point-cloud visibility problem. Lin et al. [8] proposed an algorithm to detect boundary particles in 2D. Their method first computes a Delaunay triangulation of the neighbor particles projected on a unit circle, and then classifies a particle as boundary if any circumcircle radius exceeds a certain threshold. Since Lin et al. [8] provided no hints on extending their method to 3D, we believe that extension is not straightforward.

In this paper, we present two novel algorithms for detecting boundary particles in 2D and 3D. Our approach is similar to the ones by Dilts and Haque [2, 12], in that we perform a purely geometric sphere covering test. The key difference is that we use interval analysis methods to ensure robustness. The accuracy of our methods relies on an adaptive spatial subdivision of the sphere associated with a given particle. The first method performs an interval evaluation of the implicit function defining the sphere for each particle. The second method uses geometric predicates on the enclosures of the boundary of the sphere. Both methods are robust: they do not produce false negatives; that is, no boundary particles are classified as interior. Moreover, the methods are simple, easy to code, and with competitive computational times. We perform a set of comparisons against state-of-the-art boundary detection methods in free-surface flow problems to demonstrate the effectiveness and accuracy of our approaches.

## 2. Boundary particles

Let $\mathcal{P}$ be a set of scattered particles sampling a compact region $\Omega \subset \mathbb{R}^d$ and let $\partial\Omega$ be the boundary surface of $\Omega$. We index the particles in $\mathcal{P}$ by $i \in \mathbb{N}$. Particle $i$ is located at the point $\mathbf{p}_i \in \mathbb{R}^d$. Our main goal is to identify the boundary particles in $\mathcal{P}$, that is, the set of particles in $\mathcal{P}$ that lie on $\partial\Omega$. Ideally, we would like to identify all boundary particles precisely, but we shall aim for a large subset of particles that are guaranteed to be on the boundary.

To define boundary particles precisely, we assume that $\mathcal{P}$ is an *r-sampling* of $\Omega$ [19]: for every point $\mathbf{x} \in \Omega$, there is a particle $i$ in $\mathcal{P}$ such that $\|\mathbf{x} - \mathbf{p}_i\| < r$. The parameter $r$ corresponds to the numerical resolution of the problem. For instance, in SPH solvers the radius $r$ coincides with the SPH smoothing length [12]. Thus, the radius $r$ dictates the accuracy of the method: the boundary detection method should be able to capture small-scale details of the fluid (like cavities, thin-sheets, ligaments, and drops) of diameter or thickness at least $2r$.

Let $B_i$ be the ball of radius $r$ centered at the point $\mathbf{p}_i$ and let $S_i = \partial B_i$ be the boundary sphere of $B_i$. A particle $i$ is called an *interior particle* when its sphere $S_i$ is completely covered by the neighboring balls; more precisely, when $S_i \subset \cup_{j \in \mathcal{N}_i} B_j$ where $\mathcal{N}_i = \{j \in \mathbb{N} : \|\mathbf{p}_j - \mathbf{p}_i\| \leq 2r\}$. Note that it is the boundary $S_i$ of $B_i$ that is completely covered other balls, not necessarily the whole ball $B_i$ (Figure 1). A particle $i$ is called a *boundary particle* when it is not an interior particle.

## 3. Boundary detection using interval analysis

To determine whether a particle is on the boundary, we perform a purely geometric covering test, following the definition above. This approach is similar to the ones by Dilts and Haque [2, 12]. The contribution of this paper is two robust and efficient solutions for this geometric problem. They rely on adaptive spatial subdivision and robust geometric tests. There is a single user parameter, the subdivision depth, which controls the tradeoff between speed of processing and accuracy of the results.

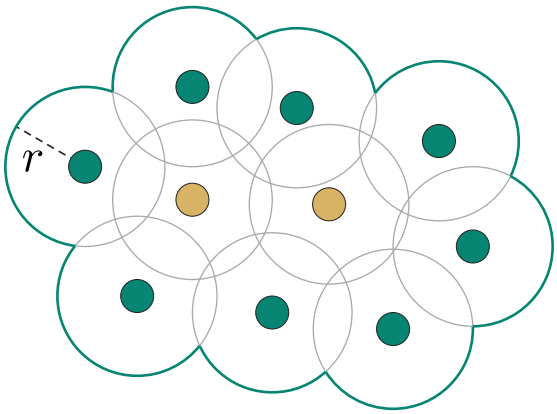



Figure 1: Interior particles (brown dots) and boundary particles (green dots).

Our methods use tools from interval analysis: interval arithmetic and geometric enclosures. The first method is easier to understand and to implement. It also serves as an introduction to the second method, which deals with slightly more complicated geometry. Recall that our main task is deciding whether particle $i$ is an interior particle. By definition, we have to check whether the boundary $S_i$ of $B_i$ is completely covered by neighboring balls $B_j$ for $j \in \mathcal{N}_j$.

### *3.1. Using interval arithmetic*

In our first method, we perform an adaptive spatial subdivision of the bounding box of $B_i$ into *query boxes*. We test the query boxes that intersect $S_i$ against the neighboring balls $B_j$. The geometric tests in this method rely on the implicit formulation of balls:

$$B_j = \{\mathbf{x} \in \mathbb{R}^d : f_j(\mathbf{x}) \leq 0\}, \quad \text{where} \quad f_j(\mathbf{x}) = \|\mathbf{x} - \mathbf{p}_j\|^2 - r^2$$

We test a query box $Q$ against a ball $B_j$ by computing the interval $I = f_j(Q)$. We compute this interval exactly using *interval arithmetic* [20][1]. There are three

[1]Interval arithmetic is a numerical technique that provides estimates for the whole range of values taken by a function in a box in $\mathbb{R}^d$. For the functions $f_j$ defining spheres, it happens that the estimates computed by interval arithmetic are exact.

possible outcomes of this computation, according to the position of the bounds of $I$, given by $\min I$ and $\max I$, with respect to 0:

- $\max I \leq 0$: Then $I \subseteq [-\infty, 0]$ and $Q$ is *completely inside* $B_j$.
- $\min I > 0$: Then $I \subseteq (0, \infty)$ and $Q$ is *completely outside* $B_j$.
- $\min I \leq 0 \leq \max I$: Then $Q$ *straddles the boundary* $S_j$ of $B_j$.

The query boxes that straddle the boundary $S_i$ of $B_i$ are called *boundary boxes*. They are found by testing the interval $f_i(Q)$. Boundary boxes are the interesting query boxes; the other ones are discarded. We test whether a boundary box $Q$ is completely covered by neighboring balls $B_j$ using the same test on the interval $f_j(Q)$. There are three possible outcomes:

- $Q$ is completely inside $B_j$ for *some* $j$: Then the part of $S_i$ inside $Q$ is completely **covered** by $B_j$.
- $Q$ is completely outside $B_j$ for *all* $j$: Then the part of $S_i$ inside $Q$ is completely **uncovered** by the neighboring balls.
- Otherwise, $Q$ is **partially covered**. Then, $Q$ is subdivided by bisection at its center into $2^d$ children boxes, the test is applied recursively to the children, and the results are combined.

If all boundary boxes are completely covered, then particle $i$ is an interior particle. Otherwise, particle $i$ is a boundary particle.

The function `is_interior` provided by Algorithm 1 implements these ideas: it performs an adaptive spatial subdivision of the sphere associated to particle $i$, starting the recursion with the bounding box of $B_i$, that is, the cube centered at $\mathbf{p}_i$ with side $2r$. The algorithm follows a subdivision tree (a quadtree in 2D and an octree in 3D), trying to classify query boxes according to the criteria above, up to a maximum depth chosen by the user to control the tradeoff between speed and accuracy.

**Algorithm 1:** Recursive classification of a particle $i$

```
function is_interior(i):
    Q ← bounding box of B_i
    return query(Q, i, 0)
end

function query(Q, i, depth):
    if Q is completely inside B_i or Q is completely outside B_i then
        return true // uninteresting boxes
    end
    uncovered ← true
    foreach j ∈ N_i do
        if Q is completely inside B_j then
            return true
        end
        if Q straddles the boundary S_j of B_j then
            uncovered ← false
        end
    end
    if uncovered then
        return false // definitely boundary
    end
    if depth = maximum depth then
        return false // probably boundary
    end
    return subdivide(Q, i, depth)
end

function subdivide(Q, i, depth):
    Q = Q_1 ∪ ··· ∪ Q_{2^d} // subdivision
    return query(Q_1, i, depth + 1) and ... and
     query(Q_{2^d}, i, depth + 1)
end
```

Figure 2 illustrates this algorithm in 2D using a quadtree with maximum depth 3. In this case, the algorithm classifies particle $i$ as a boundary particle without having to process the whole quadtree. For this, we assume that the conjuction in `subdivide` is short-circuited, that is, stops at the first term that is false. (The algorithm is correct even if this assumption does not hold; it just processes more query boxes.)

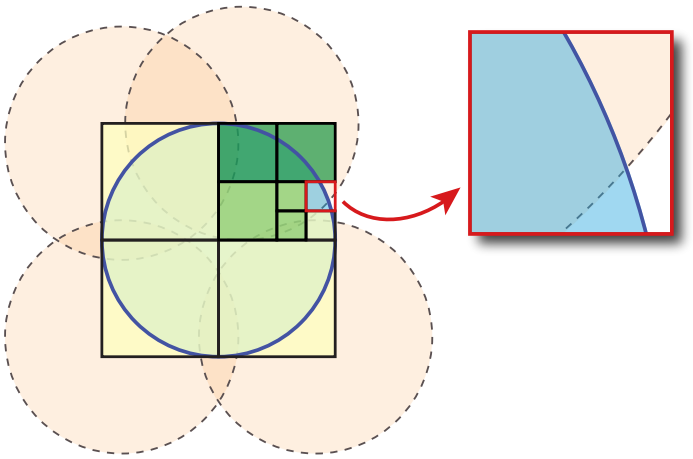


Figure 2: The 2D version of our boundary detection method using interval arithmetic. The ball $B_i$ around particle $i$ is shown in light blue. We check whether the circle $S_i$ (dark blue) is covered by its neighboring balls (orange). Some boundary boxes are not processed (yellow), and some are covered by a neighbor ball (dark green), while the non-boundary boxes (light green) are discarded by our method. A query box (red) at depth 3 reveals an uncovered region of $S_i$. Thus, particle $i$ is a boundary particle.

We shall now discuss some important implementation details.

*Finding neighboring particles.* Crucial to the performance of the algorithm is a fast way to identify the set $\mathcal{N}_i$ of neighboring particles. For this task, we use the *linked-list* algorithm [1], where the cells of the search grid have a size of $2r$.

*Faster tests.* To test whether a query box $Q$ is a boundary box, we skip interval arithmetic and test the signs of $f_i$ on the vertices of $Q$. Then $Q$ is a boundary box iff the signs are not the same. This change does not affect correctness because query boxes are subdivisions of the bounding box of $B_i$ and so are in special positions with respect to $B_i$. In particular, a boundary box never has all of its vertices outside $B_i$. The one exception is the bounding box of $B_i$, which is the initial query box. We avoid this case by starting the recursion at depth 1, by changing `is_interior` slightly to call `subdivide` instead of `query`, as shown in Algorithm 2:

**Algorithm 2:** Modified recursive classification of a particle $i$

```
function is_interior(i):
    Q ← bounding box of B_i
    return subdivide(Q, i, 0)
end
```

This change improves overall performance by about 16%. On the other hand, we cannot use the signs of $f_j$ on the vertices of $Q$ to test $Q$ against neighboring balls $B_j$ because their relative positions are arbitrary. We must rely on interval arithmetic for those tests.

*Interval arithmetic libraries.* For computations with intervals, we used PyInterval[2] in 2D and Boost C++[3] in 3D, both easy-to-use libraries. For the simple case of a sphere equation, we could avoid a full interval arithmetic library and use the simple ad-hoc code given in the Appendix. We could also probably avoid using outward rounding, because the geometric resolution is much lower than the numerical resolution of the floating point system.

*Avoiding recursion.* While Algorithm 1 is easy to understand and to check correctness, our actual implementation in Algorithm 3 simulates recursion by keeping query boxes that need to be checked in a stack. This makes it easier to terminate the process earlier without changing the final result, by following the simple rules below:

**R1:** If $Q$ is uncovered, we can safely say that $S_i$ has an uncovered region and thus the particle $i$ is a boundary particle;

**R2:** If $Q$ is covered, we stop the subdivision at $Q$;

**R3:** If $Q$ is partially covered and the maximum depth has not been reached, we subdivide $Q$ and continue recursively;

[2] `https://pypi.org/project/pyinterval/`
[3] `https://www.boost.org/doc/libs/1_66_0/libs/numeric/interval/doc/interval.htm`

**R4:** If the maximum depth has been reached, we declare that the particle $i$ is a boundary particle. These are the potential false positives, i.e., interior particles misclassified as boundary particles.

Rules **R1** and **R2** have precedence over **R3** and **R4**. If **R2** is satisfied, we must keep evaluating the sibling nodes until we find an uncovered node or determine that all nodes are covered, in which case that the particle $i$ is interior. In **R4**, although $Q$ is partially covered, we classify the particle $i$ as boundary particle for two reasons: (i) we ensure that the algorithm will never produce a false negative and (ii) it allows us to a short circuit the evaluation process and thus speeds up the detection.

Whenever we need a box to evaluate, we get the one on the top of the stack; when we subdivide a box, we push its children ton top of the stack. The simulated depth-first traversal is beneficial since we can reach rules **R1** and **R4** quicker and terminate the process earlier. If we do not arrive at **R1** or **R4**, and we reach the bottom of the stack, then all boxes fell on rule **R2**, and thus they are all covered; consequently, the particle $i$ in an interior particle.

### *3.2. Using geometric enclosures*

Our first method is simple to understand and to implement, but it spends effort classifying query boxes against a ball $B_i$: it finds and discards boxes that are completely inside or completely outside $B_i$. However, we care only about the boundary $S_i$ of $B_i$ and the only interesting boxes are the boundary boxes.

Our second method performs an adaptive spatial subdivision of an enclosure of $S_i$, a process that takes place in a lower dimension. We decompose $S_i$ into a coarse mesh of linear elements and enclose $S_i$ around that mesh with a union of thin convex polytopes which we shall call *slabs*. This cover is refined as needed by adaptively refining the underlying mesh.

In 2D, the mesh is initially given by the sides of an equilateral triangle inscribed in $S_i$. The mesh is then adaptively refined into an inscribed polygon. The slab around a mesh segment $L$ is the rectangle having one side on $L$ and the opposite side tangent to the circle $S_i$ at the projection of the midpoint $\mathbf{b}$

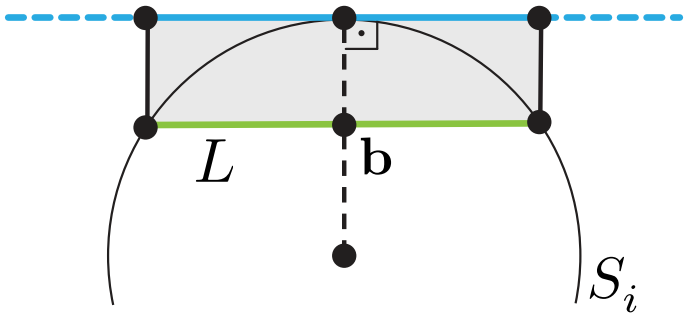



Figure 3: Slab (gray) enclosing a segment $L$ (green).

of $L$ onto $S_i$ (see Figure 3). To test a segment $L$ against a neighboring ball $B_j$, we test the corresponding slab $Q$ against $B_j$ by testing the signs of $f_j$ at the vertices of $Q$, as follows:

- If $f_j(\mathbf{v}) < 0$ for all vertices $\mathbf{v}$ of $Q$: Then $Q$ is *completely inside* $B_j$.
- If $f_j(\mathbf{v}) > 0$ for all vertices $\mathbf{v}$ of $Q$: Then $Q$ is *completely outside* $B_k$.
- Otherwise, $Q$ *straddles the boundary* $S_j$ of $B_j$.

As in the first method, there are three possible outcomes:

- $Q$ is completely inside $B_j$ for *some* $j$: Then $L$ is completely **covered** by $B_j$.
- $Q$ is completely outside $B_j$ for *all* $j$: Then $L$ is completely **uncovered** by the neighboring balls.
- Otherwise, $L$ is **partially covered**. Then, $L$ is subdivided at its midpoint into two children segments, the test is applied recursively to the children, and the results are combined.

If all segments are completely covered, then particle $i$ is an interior particle. Otherwise, particle $i$ is a boundary particle. Figure 4 illustrates our second method in 2D.

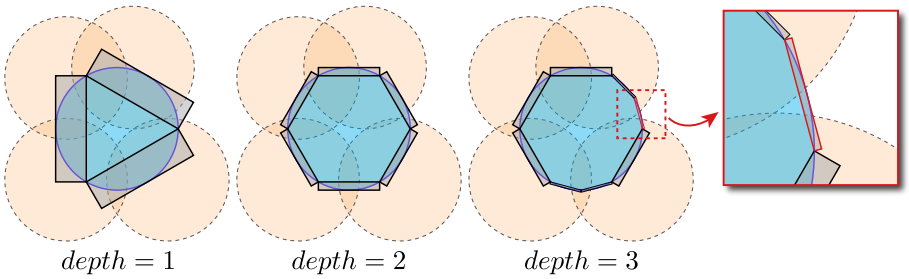



Figure 4: The 2D version of our method based on geometric enclosures. At each level, we check whether the slabs (gray) are covered by its neighboring balls (orange). At maximum depth 3, a slab (red) shows an uncovered region of $S_i$. Thus, the particle $i$ is a boundary particle.

*Computing slabs in 3D.* Firstly, we apply the affine transformation $T_i(\mathbf{x}) = r^{-1}(\mathbf{x} - \mathbf{p}_i)$ in the sphere $S_i$ and its neighboring balls. Note that the center of $S_i$ is translated for the origin $\mathbf{0}$ and all balls become unit balls.

In 3D, the linear elements are triangles; the mesh is initially given by the faces of a regular tetraheadron inscribed in $S_i$. Let $L$ be a mesh triangle and $\mathbf{b}$ its barycenter. The slab around $L$ is the frustum obtained by projecting $L$ onto the plane $P$ that is tangent to $S_i$ at the projection $\overline{\mathbf{b}} = \mathbf{b}/\|\mathbf{b}\|$ of $\mathbf{b}$ onto $S_i$ (see Figure 5).

Each vertex $\mathbf{a} \in L$ is projected onto $P$ to mimic a perspective projection from the viewpoint $\mathbf{0}$. Considering $\overline{\mathbf{a}}$ as the projection of $\mathbf{a}$ onto $P$, by construction as shown in Figure 5, we have that $\mathbf{a} \cdot \overline{\mathbf{b}} = \cos\theta = \|\overline{\mathbf{a}}\|^{-1}$. Thus, the projected vertex is given by $\overline{\mathbf{a}} = \mathbf{a}/(\mathbf{a} \cdot \overline{\mathbf{b}})$.

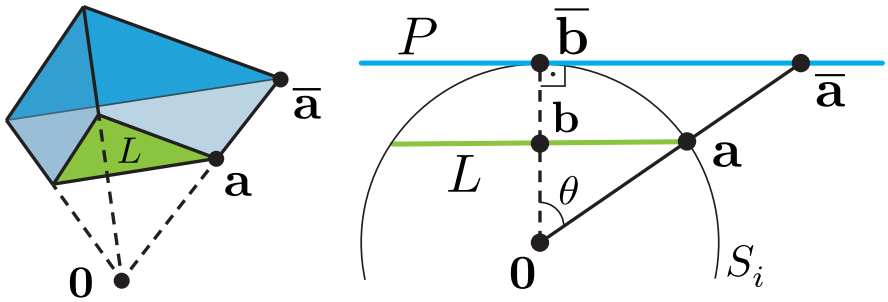



Figure 5: Perspective projection of a triangle $L$ (green) onto the tangent plane $P$ (blue) of the unit sphere $S_i$: the frustum formed by $L$ and its projection (left) and the projection of a single vertex $\mathbf{a} \in L$ on the plane $P$ (right).

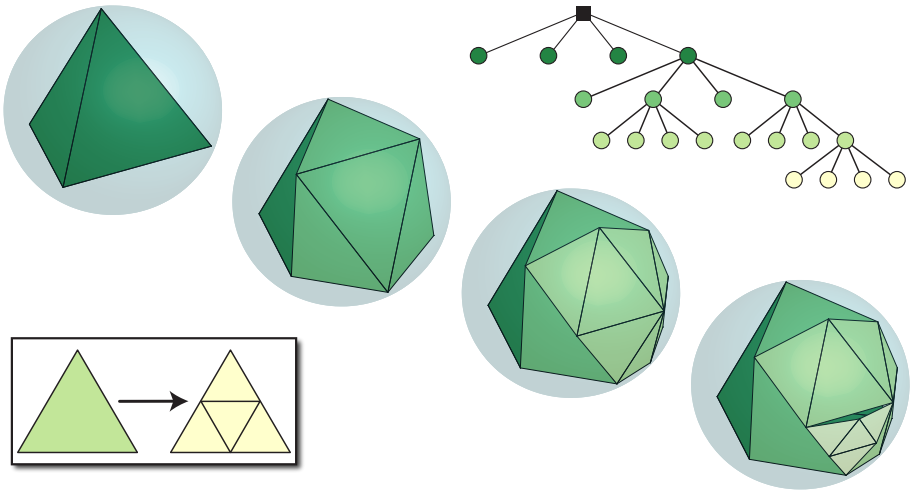


Figure 6: Mesh refinement in 3D (center) and its underlying quadtree (top-right): a regular tetrahedron inscribed in $S_i$ is refined using midpoint subdivision scheme (bottom-left).

The mesh triangle $L$ is refined when necessary by using the standard midpoint subdivision scheme (see Figure 6). Then, we update the location of these new vertices by projecting them on $S_i$. Although the refined mesh provides a better approximation of $S_i$, this process can produce hanging nodes and so a non-conforming mesh. However, this is not a problem because we are not interested in the resulting mesh itself, only in the slabs that enclosure the boundary.

The mesh refinement process rapidly reduces the total volume of the enclosure because the combined volumes of the slabs associated with the four child nodes will be a fraction of the volume associated with the original parent node. So each subdivision step improves the accuracy of the covering test, consequently its convergence.

We apply the rules **R1**–**R4** of the previous method in each slab. Assuming that a slab is a boundary box, the labeling of a generic element (box or slab) regarding its covering is summarized in Algorithm 3.

**Algorithm 3:** Classification of a particle $i$

```
function is_interior(i):
    initialize the stack 𝒯 with the nodes from the first tree-level
    while 𝒯 ≠ ∅ do
        Q ← the topmost element from 𝒯
        remove Q from 𝒯
        q ← label (Q, i)
        if q is uncovered then
            return false
        else if q is covered then
            continue // proceed to the next element of 𝒯
        else // Q is partially covered
            ℓ ← the depth of Q
            if ℓ < maximum depth then
                Q = Q_1 ∪ ··· ∪ Q_m // subdivision
                foreach child node Q_k do
                    if Q_k is boundary box then
                        add Q_k to the top of 𝒯 with depth ℓ + 1
                    end
                end
            else
                return false
            end
        end
    end
    return true
end

function label(Q, i):
    q ← uncovered
    foreach j ∈ 𝒩_i do
        if Q is completely outside B_j then
            continue // keep the current label
        else //either covered or partially covered
            if Q is completely inside B_j then
                return covered
            end
            q ← partially covered
        end
    end
    return q
end
```

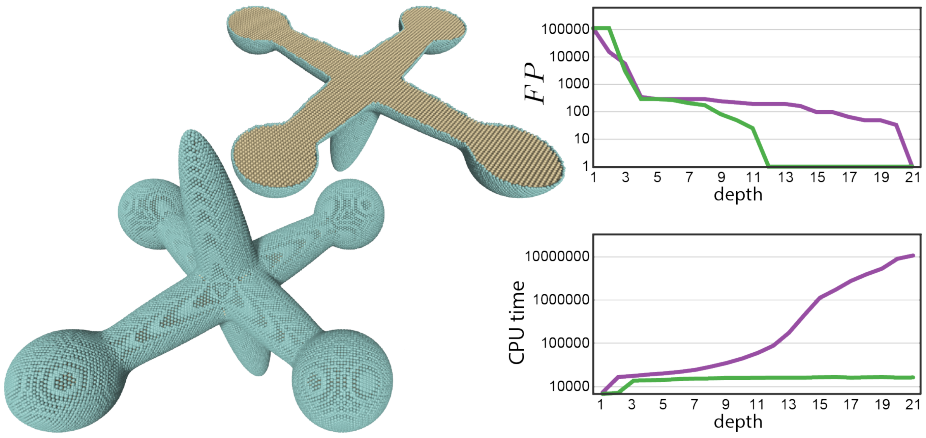


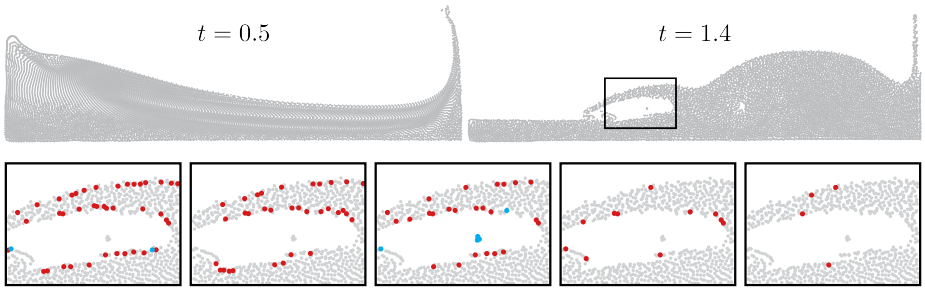

Figure 7: Comparison between IA (■) and GE (■) methods. On the left, the classification of the boundary particles (green) and interior particles (brown) in a model with 170k particles, and a cutaway view thereof. On the right, the symmetric log (symlog) plots of the $FP$ and the computational times (in milliseconds) w.r.t. the subdivision depth.

## 4. Results

We first analyze the performance of our novel techniques based on interval arithmetic (IA) and geometric enclosure (GE). Assuming the classification provided by Dilts and Haque [2, 12] as ground truth, we count the number of particles according to their assignment:

- *True Positive* ($TP$): a boundary particle correctly classified;
- *True Negative* ($TN$): an interior particle correctly classified;
- *False Positive* ($FP$): an interior particle classified as boundary;
- *False Negative* ($FN$): a boundary particle classified as interior.

Since our interval approaches do not produce false negatives (i.e., $FN = 0$), the comparison between IA and GE approaches is performed by analyzing the number of false positives. Figure 7 shows an error analysis based on $FP$ in a static model and a comparison of the computational performance between both approaches as well. We observe that GE approach converges to the exact classification faster than IA with less computational effort.




(a) Marrone [9] (b) Sandim [10] (c) Lin [8] (d) Our IA (e) Our GE

Figure 8: Comparison between different boundary detection methods in a 2D dam-break simulation using SPH (top) at $t = 1.4$. The misclassified particles are highlighted (bottom): $FP$ (red) and $FN$ (blue).

In search of a balance between accuracy and efficiency in IA and GE methods, we choose the maximum depth of 6 in all experiments carried out below.

Figure 8 provides a qualitative comparison by showing the misclassified particles ($FP$ and $FN$) resulting from each technique in an SPH simulation of a 2D dam-break problem, as described in [9]. The zoomed rectangle in this figure shows that our geometric approaches can capture cavities and thin sheets of fluid better than any other detection method due to the reduced number of misclassified particles.

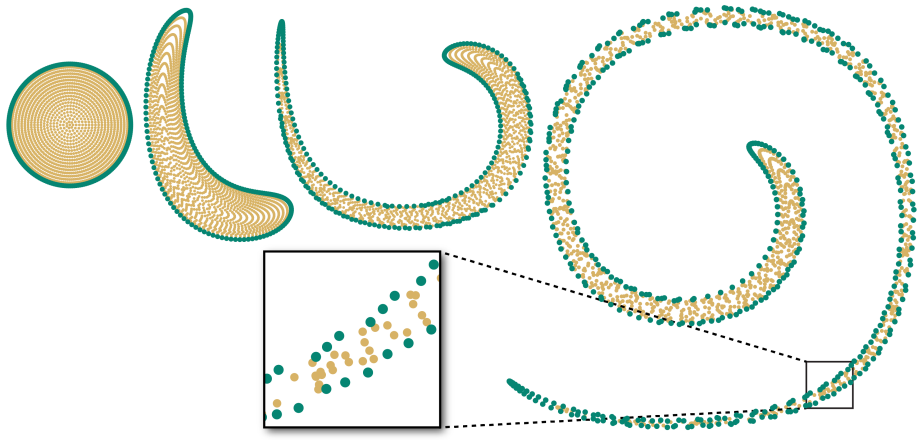
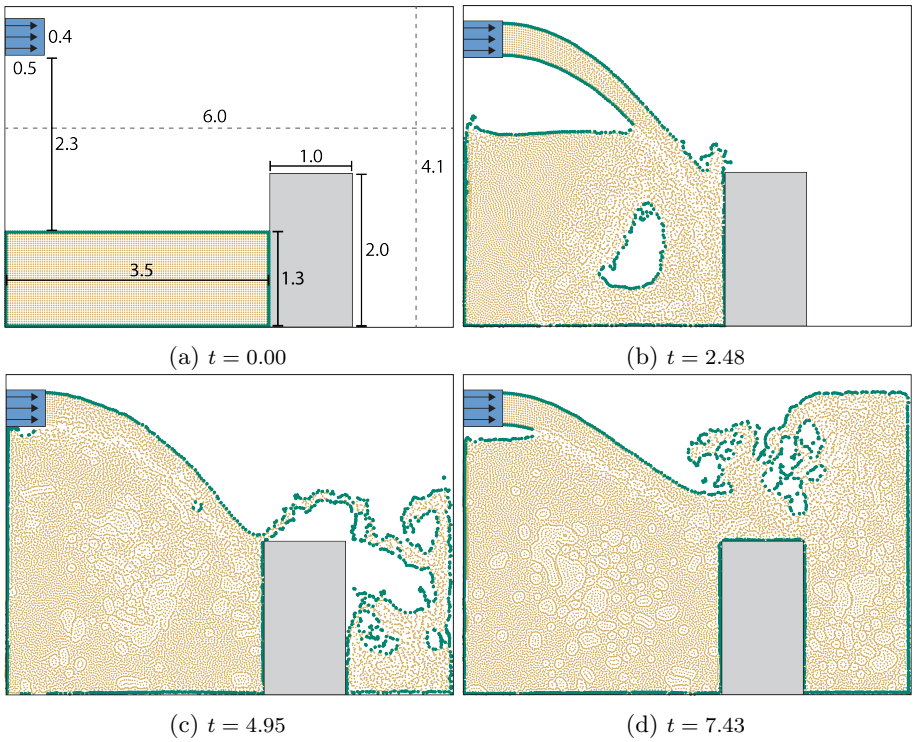


Figure 9: Boundary detection using IA in a single vortex at different times $t = 0.0, 0.25, 0.5$ and 1.0 (from left to right): boundary particles (green) and interior particles (brown).




(a) $t = 0.00$ (b) $t = 2.48$

(c) $t = 4.95$ (d) $t = 7.43$

Figure 10: Boundary detection using GE in an SPH flow simulation of an injector (blue) with inlet velocity of $4\,m/s$ in a domain with an obstacle (gray) at different times ($t$): boundary particles (green) and interior particles (brown).

Figure 9 shows the behavior of the particles in the classical *single vortex* experiment, as detailed by Enright et al. [21]. Our IA method detects the boundary particles gracefully, even when the particles are compressed due to a considerable stretching caused by the vortex flow field.

Figure 10 shows an SPH flow simulation of a liquid been injected in a domain with an obstacle. Our GE method captures the bubbles formed by the impact of the liquid against a rigid obstacle.

Figures 11 and 12 depict the boundary detection using IA and GE methods in complex free-surface flows in 3D resulting from the impact of a double dam-break and against a rigid tall obstacle after a single dam-break, respectively. Our geometric methods are resilient to the fragmentation of the interface and the thin layers of particles created by the impact.

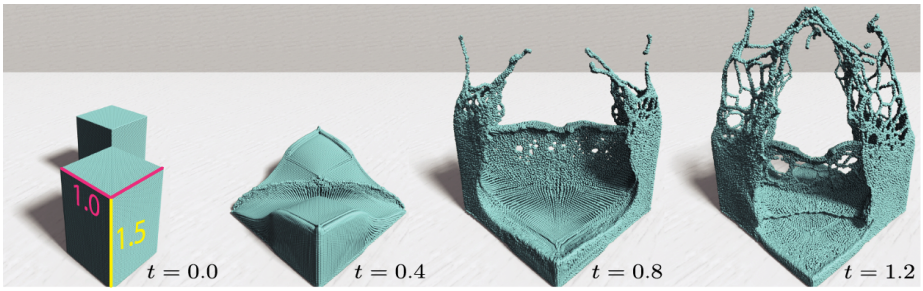
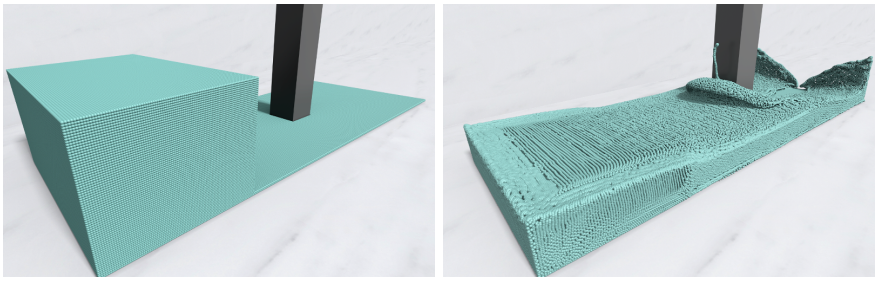



Figure 11: Boundary particles detected by our IA method in an SPH flow simulation of a dam-break problem of two liquid columns at opposite corners of a container with a square base of size 2.5 and height 3.5 at different times ($t$).

All SPH flow simulations presented in this section were performed using the computational platform SPHysics [22].

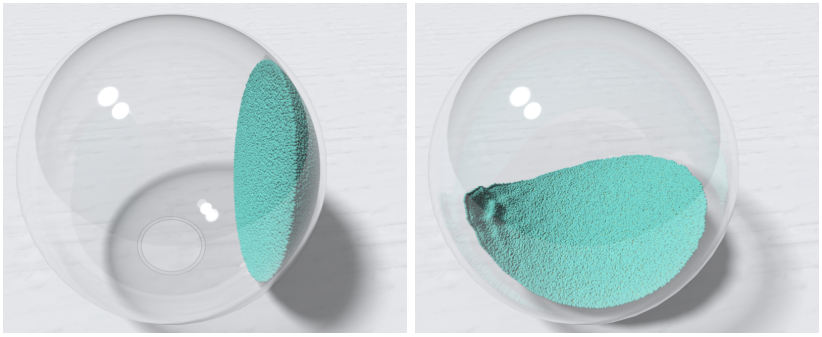
Beyond SPH, Figure 13 shows the versatility of our GE method when applied to detect the boundary particles in an *Affine Particle-in-Cell* (APIC) [23] flow simulation of a liquid sloshing in a spherical tank. In this experiment, we use the APIC implementation provided by Kim [24].

Similar to the level-set definition from the boundary particles introduced by Marrone et al. [9], Figure 14 demonstrates the effectiveness of our geometric approach when the free-surface is reconstructed from the level-set of the *Enright test* [21] generated by Sandim et al.'s algorithm [25].



(a) $t = 0.0$ (b) $t = 0.2$

Figure 12: Boundary particles detected by our GE method in an SPH flow simulation of the impact against a rigid obstacle after a dam-break, as reported in [9].



(a) $t = 0.0$ (b) $t = 0.3$

(c) $t = 0.6$ (d) $t = 0.9$

Figure 13: APIC flow simulation of liquid sloshing starting in a section of a sphere of diameter 0.8: boundary particles detected by our GE method at different time instants.

### *4.1. Quantitative analysis*

To perform quantitative comparisons between different boundary detection methods, we measure the accuracy of each method using the metric proposed by Sandim et al. [10]:

$$M = \ Rec \cdot (1 - FPR)\,,$$

where $Rec = TP/(TP+FN)$ and $FPR\ = FP/(P+TN)$ are well-known metrics in data analysis [26] called *Recall* and *False Positive Rate* (FPR), respectively. Recall measures the accuracy of a method in detecting boundary particles precisely among the actual set of boundary particles. The best result occurs when $Rec = 1$, meaning that all boundary particles were classified correctly, although of the possible presence of false positives. While FPR quantifies how many interior particles were misclassified as boundary. The best case occurs

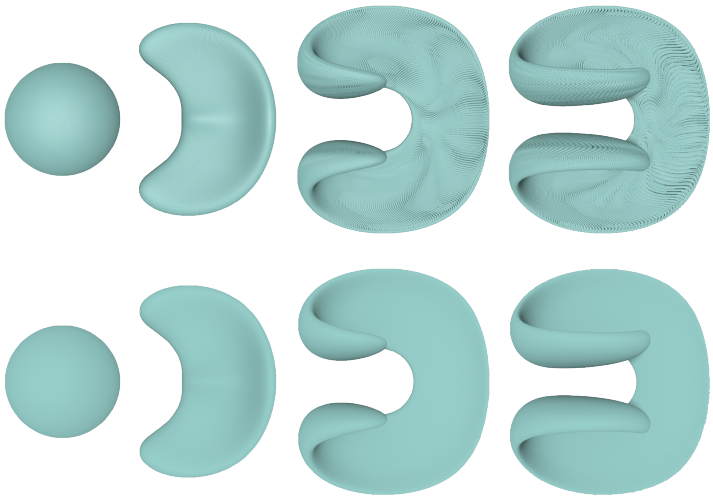


Figure 14: The Enright test at different times $t = 0.0, 0.5, 1.0$ and $1.5$ (from left to right): the boundary particles detected by our GE method (at top) and its level-set (at bottom).

when $FPR = 0$, when no interior particles have been classified as boundary. Thus, the best classification occurs when the combined metric $M$ reaches its maximum value of 1, i.e. when the Recall is maximum, and FPR is minimum simultaneously.

Table 1 and Table 2 show the number of particles ($|\mathcal{P}|$), the parameter $r$, and the average scores resulting from each technique in 2D and 3D, respectively. We apply the same parameters as suggested in the corresponding paper of each technique. Remembering, for the assessments of IA and GE, we choose the maximum depth of 6. Notice that our interval approaches reliably detect all boundary particles, providing certified results with $Rec = 1$. Moreover, the GE approach outperforms the other methods in all experiments.

We implemented the 2D version of our techniques in Python and the 3D version in C++. All boundary detection methods in C++ were parallelized using OpenMP, except the ground truth [12] due to its complexity. All experiments have been performed on a computer equipped with processor Intel i7-8750H with six 2.2GHz cores and 16GB RAM. Table 3 shows the computational times regarding the number of processor cores for the experiments in 3D. The column *speedup* is the relation 1-core/6-core, and the last column is how much faster

Table 1: Quantitative analysis between different boundary detection methods in 2D (best results are shown in bold).

| **Experiment** | $\lvert\mathcal{P}\rvert$ | $r$ | **Methods** | $Rec$ | $FPR$ | $M$ |
|---|---|---|---|---|---|---|
| Figure 8 | 11.2k | 0.009 | Marrone et al. [9] | 0.9767 | 0.0245 | 0.9523 |
| | | | Sandim et al. [10] | 0.9979 | 0.0184 | 0.9796 |
| | | | Lin et al. [8] | 0.9762 | 0.0189 | 0.9577 |
| | | | Our IA | **1.0000** | 0.0036 | 0.9964 |
| | | | Our GE | **1.0000** | **0.0016** | **0.9984** |
| Figure 9 | 837 | 0.022 | Marrone et al. [9] | 0.9779 | 0.0701 | 0.9093 |
| | | | Sandim et al. [10] | **1.0000** | 0.0776 | 0.9224 |
| | | | Lin et al. [8] | 0.9982 | 0.0642 | 0.9341 |
| | | | Our IA | **1.0000** | 0.0569 | 0.9431 |
| | | | Our GE | **1.0000** | **0.0472** | **0.9528** |
| Figure 10 | [5k, 23.4k] | 0.051 | Marrone et al. [9] | 0.9735 | 0.0136 | 0.9602 |
| | | | Sandim et al. [10] | 0.9540 | 0.0040 | 0.9502 |
| | | | Lin et al. [8] | 0.9307 | 0.0106 | 0.9209 |
| | | | Our IA | **1.0000** | 0.0052 | 0.9948 |
| | | | Our GE | **1.0000** | **0.0014** | **0.9986** |

Table 2: Quantitative analysis between different boundary detection methods in 3D (best results are shown in bold).

| **Experiment** | $\lvert\mathcal{P}\rvert$ | $r$ | **Methods** | $Rec$ | $FPR$ | $M$ |
|---|---|---|---|---|---|---|
| Figure 11 | 275.4k | 0.033 | Marrone et al. [9] | 0.9127 | 0.0304 | 0.8850 |
| | | | Sandim et al. [10] | 0.9992 | 0.0269 | 0.9723 |
| | | | Our IA | **1.0000** | 0.0094 | 0.9906 |
| | | | Our GE | **1.0000** | **0.0065** | **0.9935** |
| Figure 12 | 1.1M | 0.007 | Marrone et al. [9] | 0.9708 | 0.0376 | 0.9343 |
| | | | Sandim et al. [10] | 0.9998 | 0.0298 | 0.9700 |
| | | | Our IA | **1.0000** | 0.0122 | 0.9878 |
| | | | Our GE | **1.0000** | **0.0070** | **0.9930** |
| Figure 13 | 550k | 0.007 | Marrone et al. [9] | 0.6928 | 0.0269 | 0.6742 |
| | | | Sandim et al. [10] | 0.9938 | 0.0338 | 0.9602 |
| | | | Our IA | **1.0000** | 0.0169 | 0.9831 |
| | | | Our GE | **1.0000** | **0.0089** | **0.9911** |
| Figure 14 | 1.9M | 0.005 | Marrone et al. [9] | 0.9708 | 0.0376 | 0.9343 |
| | | | Sandim et al. [10] | 0.9998 | 0.0298 | 0.9700 |
| | | | Our IA | **1.0000** | 0.0122 | 0.9878 |
| | | | Our GE | **1.0000** | **0.0070** | **0.9930** |
| Figure 15 | [22k, 3.2M] | 0.010 | Marrone et al. [9] | 0.6727 | 0.1729 | 0.5564 |
| | | | Sandim et al. [10] | 0.99908 | 0.2027 | 0.7965 |
| | | | Our IA | **1.0000** | 0.2219 | 0.7781 |
| | | | Our GE | **1.0000** | **0.0787** | **0.9213** |

is a detection method than the exact method, i.e., the *rate* of CPU time of a detection method over the exact method using a single core. As can be seen, the GE method is almost twice as fast as the exact method in the worst case.

Table 3: Average computational times (in seconds) per time-step.

| **Experiment** | $\lvert\mathcal{P}\rvert$ | **Methods** | 1-core | 6-core | speedup | rate |
|---|---|---|---|---|---|---|
| Figure 11 | 275.4k | Haque and Dilts [12] | 19.70 | ✗ | ✗ | 1.00 |
| | | Marrone et al. [9] | 4.77 | 0.97 | 4.91 | 4.13 |
| | | Sandim et al. [10] | 4.68 | 1.02 | 4.59 | 4.21 |
| | | Our IA | 15.26 | 2.85 | 5.35 | 1.29 |
| | | Our GE | 10.60 | 2.38 | 4.46 | 1.86 |
| Figure 12 | 1.1M | Haque and Dilts [12] | 113.00 | ✗ | ✗ | 1.00 |
| | | Marrone et al. [9] | 31.09 | 6.52 | 4.77 | 3.63 |
| | | Sandim et al. [10] | 23.37 | 5.37 | 4.35 | 4.84 |
| | | Our IA | 81.32 | 15.07 | 5.39 | 1.39 |
| | | Our GE | 61.45 | 13.68 | 4.49 | 1.84 |
| Figure 13 | 550k | Haque and Dilts [12] | 112.89 | ✗ | ✗ | 1.00 |
| | | Marrone et al. [9] | 15.51 | 3.60 | 4.30 | 7.28 |
| | | Sandim et al. [10] | 17.03 | 5.73 | 2.97 | 6.63 |
| | | Our IA | 48.36 | 9.95 | 4.86 | 2.33 |
| | | Our GE | 37.10 | 8.48 | 4.38 | 3.04 |
| Figure 14 | 1.9M | Haque and Dilts [12] | 649.05 | ✗ | ✗ | 1.00 |
| | | Marrone et al. [9] | 149.16 | 35.53 | 4.20 | 4.35 |
| | | Sandim et al. [10] | 74.85 | 21.21 | 3.53 | 8.67 |
| | | Our IA | 362.61 | 70.32 | 5.16 | 1.79 |
| | | Our GE | 300.71 | 71.73 | 4.19 | 2.19 |

The scalability of the detection methods is measured on a mushroom jet simulation using SPH, as illustrated by Figure 15. In this simulation, more than 3 million particles are inserted in the system along the time. The computational time of GE method is located between the exact and the fastest methods.

## 5. Conclusion

We have presented two novel methods for detecting boundary particles in both 2D and 3D domains. These methods are tailored to particle-based methods in free-surface flow simulations. We combine a robust purely geometric sphere covering tests based on interval analysis with an adaptive spatial subdivision of the sphere associated with a given particle. Our approaches outperform the state-of-the-art boundary detection methods as attested by the set of experiments and comparisons carried out in the paper. As an application, we show that the proposed methods can be applied to define a level-set function from the boundary particles by using the strategy provided by Sandim et al. [25].

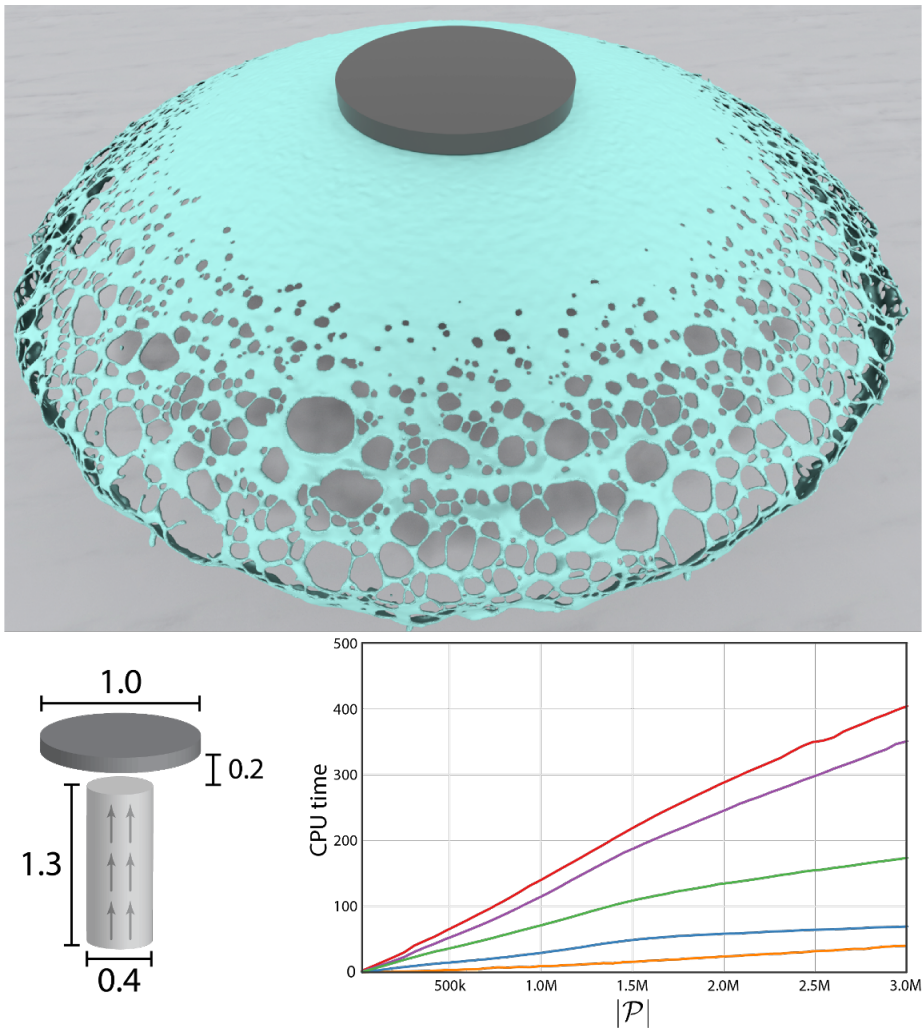



Figure 15: SPH flow simulation of a mushroom jet with inlet velocity of $5\,m/s$. On the top, the level-set from the boundary particles detected by our GE method at $t = 0.5$. On the bottom-left, the sketch of the problem geometry. On the bottom-right, the computational times (in seconds) of each detection method: Haque and Dilts [12] (■), Marrone et al. [9] (■), Sandim et al. [10] (■), our IA (■), and our GE (■).

As future work, we intend to port our geometric methods to GPU architecture, since the particle classification is performed locally and independently for each particle. Another direction is to apply our boundary detection to adapt tree-based grids dynamically around the liquid interface in fluid simulations [27] and in particle remeshing applications [28] as well.

## Acknowledgements

We want to thank the anonymous reviewers for their suggestions. We also thank Cristin Barghiel from SideFX for their kind donation of the Houdini software. This study was financed in part by the Coordenação de Aperfeiçcoamento de Pessoal de Nível Superior – Brazil (CAPES), by the National Council for Scientific and Technological Development – Brazil (CNPq) under grants 301642/2017-6 and 301244/2017-0, and the São Paulo Research Foundation (FAPESP) under grant 2019/23215-9. The computational resources provided by the Center for Mathematical Sciences Applied to Industry (CeMEAI), also funded by FAPESP (grant 2013/07375).

## References

[1] G. Liu, M. Liu, Smoothed Particle Hydrodynamics: A Meshfree Particle Method, World Scientific, 2003.

[2] G. A. Dilts, Moving least-squares particle hydrodynamics II: conservation and boundaries, Int. J. Numer. Meth. Eng. 48 (10) (2000) 1503–1524. doi:10.1002/1097-0207(20000810)48:10<1503::AID-NME832>3.0.CO;2-D.

[3] S. Koshizuka, K. Shibata, M. Kondo, T. Matsunaga, Moving Particle Semi-Implicit Method, Academic Press, 2018.

[4] G. F. Fasshauer, Meshfree Approximation Methods with MATLAB, World Scientific, 2007.

[5] V. P. Nguyen, T. Rabczuk, S. Bordas, M. Duflot, Meshless methods: A review and computer implementation aspects, Math. Comput. Simulat. 79 (3) (2008) 763 – 813. doi:https://doi.org/10.1016/j.matcom.2008.01.003.

[6] N. Flyer, G. A. Barnett, L. J. Wicker, Enhancing finite differences with radial basis functions: experiments on the Navier-Stokes equations, J. Comput.

Phys. 316 (2016) 39–62. doi:https://doi.org/10.1016/j.jcp.2016.02.078.

[7] S. M. Hosseini, J. J. Feng, Pressure boundary conditions for computing incompressible flows with SPH, J. Comput. Phys. 230 (19) (2011) 7473–7487. doi:https://doi.org/10.1016/j.jcp.2011.06.013.

[8] Y. Lin, G. Liu, G. Wang, A particle-based free surface detection method and its application to the surface tension effects simulation in smoothed particle hydrodynamics (SPH), J. Comput. Phys. 383 (2019) 196 – 206. doi:https://doi.org/10.1016/j.jcp.2018.12.036.

[9] S. Marrone, A. Colagrossi, D. Le Touzé, G. Graziani, Fast free-surface detection and level-set function definition in SPH solvers, J. Comput. Phys. 229 (10) (2010) 3652–3663. doi:https://doi.org/10.1016/j.jcp.2010.01.019.

[10] M. Sandim, D. Cedrim, L. G. Nonato, P. Pagliosa, A. Paiva, Boundary detection in particle-based fluids, Comput. Graph. Forum 35 (2) (2016) 215–224. doi:10.1111/cgf.12824.

[11] P.-P. Wang, Z.-F. Meng, A.-M. Zhang, F.-R. Ming, P.-N. Sun, Improved particle shifting technology and optimized free-surface detection method for free-surface flows in smoothed particle hydrodynamics, Comput. Meth. Appl. Mech. Eng. 357 (2019) 112580. doi:https://doi.org/10.1016/j.cma.2019.112580.

[12] A. Haque, G. A. Dilts, Three-dimensional boundary detection for particle methods, J. Comput. Phys. 226 (2) (2007) 1710 – 1730. doi:https://doi.org/10.1016/j.jcp.2007.06.012.

[13] E. Y. Lo, S. Shao, Simulation of near-shore solitary wave mechanics by an incompressible SPH method, Appl. Ocean Res. 24 (5) (2002) 275 – 286. doi:https://doi.org/10.1016/S0141-1187(03)00002-6.

[14] X. He, N. Liu, G. Wang, F. Zhang, S. Li, S. Shao, H. Wang, Staggered meshless solid-fluid coupling, ACM Trans. Graph. 31 (6) (2012) 149:1–149:12. doi:10.1145/2366145.2366168.

[15] X. Liu, P. Lin, S. Shao, An ISPH simulation of coupled structure interaction with free surface flows, J. Fluids Struct. 48 (2014) 46–61. doi:https://doi.org/10.1016/j.jfluidstructs.2014.02.002.

[16] P. Randles, L. Libersky, Smoothed particle hydrodynamics: Some recent improvements and applications, Comput. Meth. Appl. Mech. Eng. 139 (1) (1996) 375– 408. doi:https://doi.org/10.1016/S0045-7825(96)01090-0.

[17] A. Barecasco, H. Terissa, C. Naa, Simple free-surface detection in two and three-dimensional SPH solver, arXiv (2013) 1–10arXiv:1309.4290.

[18] E.-S. Lee, C. Moulinec, R. Xu, D. Violeau, D. Laurence, P. Stansby, Comparisons of weakly compressible and truly incompressible algorithms for the SPH mesh free particle method, J. Comput. Phys. 227 (18) (2008) 8417–8436. doi:https://doi.org/10.1016/j.jcp.2008.06.005.

[19] S. Katz, A. Tal, R. Basri, Direct visibility of point sets, ACM Trans. Graph. 26 (3) (2007) . doi:10.1145/1276377.1276407.

[20] M. J. C. Ramon E. Moore, R. Baker Kearfott, Introduction To Interval Analysis, SIAM, 2009.

[21] D. Enright, R. Fedkiw, J. Ferziger, I. Mitchell, A hybrid particle level set method for improved interface capturing, J. Comput. Phys. 183 (1) (2002) 83–116. doi:https://doi.org/10.1006/jcph.2002.7166.

[22] M. Gomez-Gesteira, B. Rogers, A. Crespo, R. Dalrymple, M. Narayanaswamy, J. Dominguez, SPHysics – development of a free-surface fluid solver – Part 1: Theory and formulations, Comput. Geosci. 48 (2012) 289 – 299. doi:https://doi.org/10.1016/j.cageo.2012.02.029.

[23] C. Jiang, C. Schroeder, J. Teran, An angular momentum conserving affine-particle-in-cell method, J. Comput. Phys. 338 (2017) 137–164. doi:10.1016/j.jcp.2017.02.050.

[24] D. Kim, Fluid Engine Development, CRC Press, 2016.

[25] M. Sandim, N. Oe, D. Cedrim, P. Pagliosa, A. Paiva, Boundary particle resampling for surface reconstruction in liquid animation, Computers & Graphics 84 (2019) 55 – 65. doi:https://doi.org/10.1016/j.cag.2019.08.011.

[26] T. Fawcett, An introduction to ROC analysis, Pattern Recogn. Lett. 27 (8) (2006) 861 – 874. doi:https://doi.org/10.1016/j.patrec.2005.10.010.

[27] M. A. Olshanskii, K. M. Terekhov, Y. V. Vassilevski, An octree-based solver for the incompressible Navier–Stokes equations with enhanced stability and low dissipation, Computers & Fluids 84 (2013) 231 – 246. doi:https://doi.org/10.1016/j.compfluid.2013.04.027.

[28] A. Obeidat, S. P. Bordas, An implicit boundary approach for viscous compressible high reynolds flows using a hybrid remeshed particle hydrodynamics method, J. Comput. Phys. 391 (2019) 347 – 364. doi:https://doi.org/10.1016/j.jcp.2019.01.041.

## Appendix

Algorithm 4 shows pseudo code for computing the image of a 3D query box $Q = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$ by the function $f_k$ defining the sphere $S_k$. As mentioned in the text, we avoid using outward rounding, because the geometric resolution is much lower than the numerical resolution of the floating point system.

**Algorithm 4:** Ad-hoc interval evaluation of $f_k(Q)$

**function** `FK`$(x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max})$**:**
- **global** $x_k, y_k, z_k, r$
- $[a_{min}, a_{max}] \leftarrow$ `square` $(x_{min}, x_{max}, x_k)$
- $[b_{min}, b_{max}] \leftarrow$ `square` $(y_{min}, y_{max}, y_k)$
- $[c_{min}, c_{max}] \leftarrow$ `square` $(z_{min}, z_{max}, z_k)$
- $f_{min} \leftarrow a_{min} + b_{min} + c_{min} - r^2$
- $f_{max} \leftarrow a_{max} + b_{max} + c_{max} - r^2$

**return** $[f_{min}, f_{max}]$

**function** `square`$(t_{min}, t_{max}, t)$**:**
- $t_{min} \leftarrow t_{min} - t$
- $t_{max} \leftarrow t_{max} - t$
- **if** $t_{min} \geq 0$ **then**
  - $f_{min} \leftarrow t_{min}^2$
  - $f_{max} \leftarrow t_{max}^2$
- **else if** $t_{max} \leq 0$ **then**
  - $f_{min} \leftarrow t_{max}^2$
  - $f_{max} \leftarrow t_{min}^2$
- **else**
  - $f_{min} \leftarrow 0$
  - $f_{max} \leftarrow \max\{t_{min}^2, t_{max}^2\}$
- **end**

**return** $[f_{min}, f_{max}]$