# Robust event-driven particle tracking in complex geometries

Strobl, Severin; Campbell Bannerman, Marcus Nigel; Poeschel, Thorsten

*Published in:*
Computer Physics Communications

*Publication date:*
2020

*Document Version*
Early version, also known as pre-print

# Robust event-driven particle tracking in complex geometries

Severin Strobl[a], Marcus N. Bannerman[b], Thorsten Pöschel[a,*]

[a] *Institute for Multiscale Simulation, Friedrich-Alexander-Universität Erlangen-Nürnberg, Cauerstraße 3, 91058 Erlangen, Germany*
[b] *School of Engineering, University of Aberdeen, Fraser Noble Building, AB24 3UE, UK*

**Abstract**

Particle tracking, that is, the repeated localization of particles within a grid by means of tracking the particles' trajectories, is routinely applied in particle-based schemes where the domain is described by an unstructured polyhedral grid. A range of tracking algorithms are available in the literature, which are inherently similar to algorithmic approaches common both in event-driven particle dynamics (EDPD) and ray-tracing methods. We propose a reformulation of existing particle tracking algorithms in the context of EDPD. On the one hand, this resolves inconsistencies in the mapping between particle positions and grid cells triggered, e.g., by imperfect grids. More importantly, it allows the specification of solid objects via constructive solid geometry (CSG), a standard technique for the modeling of solids in computer-aided design. While usually considered contrary approaches, our description of the computational domain as the combination of a bounding volume defined by an unstructured grid and solids modeled via CSG embedded into this volume can be highly advantageous. The two different approaches of modeling the computational domain complement each other perfectly, as the CSG representation is not only efficient in terms of memory and computing time, but also avoids the challenges of generating finely resolved unstructured grids in the presence of complicated boundaries. These benefits, as well as the positive impact of several algorithmic optimizations of the extended tracking algorithm, are exemplified via a particle-based simulation of a gas flow through a highly porous medium.

*Keywords:* particle tracking, particle sorting, event-driven dynamics, constructive solid geometry, complex geometries

## 1. Introduction

Particle models are one of the earliest applications of computer simulation, first appearing in 1957 [1]. Initially, the technique could only be applied to relatively simple problems due to limitations in the available computational resources; however, modern applications of particle-based methods include both complex applied engineering problems [2–4] and fundamental research into coupled fluid–particle problems [5–7]. These simulations typically require accurate descriptions of fluid–particle and fluid–boundary interactions at both the macroscopic scale, such as in the optimization of fluidized bed reactors [8–11], and on the microscopic scale, such as in colloidal suspensions [12–15].

A common motif of these simulations is the requirement to handle boundary conditions of complicated shape. The usage of unstructured polyhedral grids/meshes, as they are most prominent in the context of finite element analysis, allows for a convenient definition of the computational domain also in the case of particle simulations. In general, the mesh here serves merely as a geometric description of the domain, the dynamics of the system are tied to the particles. The specific role of the mesh in the context of the simulation scheme depends on the exact method. Particles typically interact in some way or another with the boundaries of the system, linked to the surface elements of the grid. Additionally, neighbor-lists [16] are commonly employed, which logically group particles based on their spatial proximity. These neighbor-list are used to optimize the calculation of physical interactions between particles by accelerating the search for nearby particles. The concept of neighbor-lists is independent of the shape of the grid cells, as long as a mapping between the particles' position and the corresponding grid cells can be obtained at any time.

*Corresponding author
*Email address:* thorsten.poeschel@fau.de (Thorsten Pöschel)

This mapping is also essential if a particle-based method such as the discrete element method (DEM) is coupled with an Eulerian method, e.g., for the simulation of a fluidized bed reactor [17].

Independent of the exact use of the mesh, it is necessary to derive a mapping from the current particle position to the enclosing mesh element and vice versa. There are two general approaches to this: (1) particles are repeatedly sorted into the elements of the mesh at regular intervals or (2) particles are tracked as they move through the mesh (after an initial sorting step). For structured grids, the sorting operation can usually be implemented efficiently by evaluating a closed-form expression; for unstructured grids, however, the naive attempt of checking the position against all elements in the mesh is too computationally expensive to perform. Assuming, however, the mapping of the initial position is available for each particle, e.g., by resorting to optimized search data structures such as quad-/octrees [18, 19] or k–d trees [20], the tracking of particles provides an efficient method to maintain the mapping, independent of the structure of the grid. Tracking the trajectory of particles avoids the potentially high computational cost of regular sorting but requires the overhead of determining the transition times when particles cross the boundaries of their enclosing mesh element (or cell). Even for complex meshes, this overhead is usually relatively small and thus a wide variety of algorithms implementing this approach have become available [21–26]. While the basic algorithms are well studied and variations supporting specialized configurations are available, we here want to address two remaining issues with existing approaches.

First, particle tracking algorithms are susceptible to failures triggered by imperfect meshes as well as floating-point imprecision itself. This can result in inconsistencies between a particle's position and the derived mapping to a cell, up to the point where particles are "lost" in-between cells. Several other publications [21–27] address this issue, yet the resulting algorithms are constructed without an underlying general formalism. The equivalent challenges for event detection algorithms in event-driven particle dynamics (EDPD) [28, 29] as well as algorithmic solutions are presented by the authors in previous contributions [30, 31]. Based on the equivalence of particle tracking and EDPD which is established in Section 2, the framework for event-driven tracking algorithms is introduced and an inherently robust yet computationally efficient tracking algorithm for unstructured meshes is derived in Section 3.

Second, while particle-based simulations using particle tracking in unstructured meshes allow for simulation domains of complex shape, this does come at a cost. The process of generating the volumetric mesh can be tedious, time consuming, and often require manual steps. This becomes especially problematic in the presence of larger numbers of (not necessarily stationary) obstacles embedded into the domain. Additionally, the computational cost of the particle tracking is a function of the ratio between the distance particles travel during one tracking step and the spatial extent of the mesh elements. For some simulation models where the computational cost of the interaction model capturing the physics of the system is limited, the particle tracking can then dominate the execution time of the simulation. In Section 4 we thus propose an extension to the tracking algorithm that relies on the description of parts of the domain using constructive solid geometry (CSG). To limit the impact of this extension towards CSG on the computational performance, in Section 5 several optimization strategies are presented. While still an underlying grid has to be generated using the typical procedures, this grid does not have to capture the surface features of the obstacles embedded into the domain. Especially for simulations where the obstacles are placed dynamically during the initialization or even follow their own dynamics this is a significant advantage. As an example, a single mesh can be reused to study a large number of different packings, simply by adjusting the parameters of the obstacles. The capabilities of this hybrid approach of using a mesh and additional obstacles defined using CSG are illustrated in Section 6. An open-cell foam is modeled using analytical geometric shapes for the simulation of a gas flow using a particle-based method. By changing the radii of the pores packings of different porosity can be automatically generated without any re-meshing.

## 2. Event-driven particle tracking

Consider a bounded and connected domain, $\Omega \subset \mathbb{R}^d$, of dimensionality $d$ which is then decomposed into a mesh, $\mathcal{M} = \{E_i\}_{i=1}^{N_E}$, of $N_E$ closed non-empty convex polyhedral elements/cells, $\{E_i\}$, satisfying,

$$\Omega = \bigcup_{i=1}^{N_E} E_i \,. \tag{1}$$

It is also required that the interiors of the elements are exclusive of each other, i.e.,

$$\forall\, i \neq j, \mathrm{int}(E_i) \cap \mathrm{int}(E_j) = \varnothing \qquad \text{where } i, j \in \{1, \ldots, N_E\}. \tag{2}$$

Although the interiors of elements are distinct, neighboring elements must share (depending on the dimensionality) vertices, edges, and facets as they are closed sets. This guarantees that there is a complete and unique mapping of positions to elements, except at the boundaries of each element which are within both neighboring elements to allow the smooth transition of particles.

Let us assume that at an initial time, $t_0$, a particle $i$ has already been located within an element of index $c_i(t_0) \in \{1, \ldots, N_E\}$, such that $\boldsymbol{r}_i(t_0) \in E_{c_i(t_0)}$, where $\boldsymbol{r}_i(t) : \mathbb{R} \to \Omega$ is the center-of-mass position of the particle. The objective is then to obtain at a later point in time, $t_1 > t_0$, the mapping between the position, $\boldsymbol{r}_i(t_1)$, and the element $E \in \mathcal{M}$, such that $\boldsymbol{r}_i(t_1) \in E$. Let us further assume, that the trajectory of the particle connecting the points $\boldsymbol{r}_i(t_0)$ and $\boldsymbol{r}_i(t_1)$ is known. By tracing this trajectory starting at $\boldsymbol{r}_i(t_0)$ to the boundary of the element $E_{c_i(t_0)}$, the intersection point between the trajectory and the boundary of $E_{c_i(t_0)}$ can be obtained. By identifying the corresponding facet and resorting to the connectivity information provided as part of the mesh description, the neighboring mesh element the trajectory crosses over into can then be obtained. Repeating this procedure iteratively until the final position $\boldsymbol{r}_i(t_1)$ is reached, the mesh element corresponding to $\boldsymbol{r}_i(t_1)$ can be found (see Fig. 1). In certain cases, complex trajectories obtained via numerical integration, such as $\boldsymbol{r}_2(t)$ in Fig. 1, can be approximated by a linear trajectory, here $\boldsymbol{r}'_2(t)$, simplifying the search for the intersection points significantly.

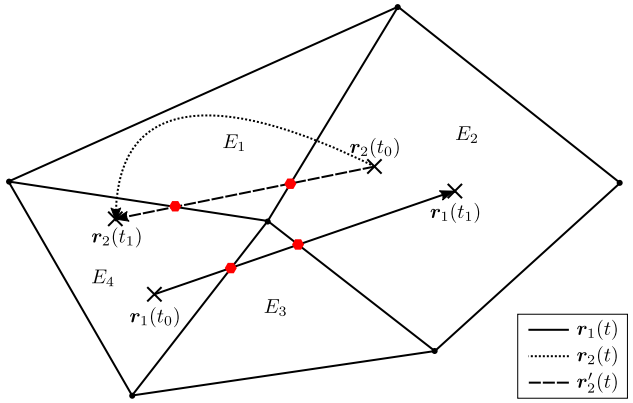



Figure 1: The principle of particle tracking illustrated for two particles $i \in \{1, 2\}$ moving in an unstructured two-dimensional grid, with $\Omega = \bigcup_{j=1}^{4} E_j$. Starting from the initial position $\boldsymbol{r}_i(t_0)$, with knowledge of the initial element, $c_0(t_0) = 4$, and $c_1(t_0) = 2$, respectively, the aim is to determine the element index $c_i(t_1)$ corresponding to the final position $\boldsymbol{r}_i(t_1)$. The transition points from one element to the next are highlighted by the red hexagons, where the actual curved trajectory of the second particle is approximated by a linear trajectory $\boldsymbol{r}'_2(t)$ for the purpose of tracking.

The particle tracking algorithm as outlined above forms the basis for a range of algorithms proposed in the literature [21–26], yet the different algorithms vary in their complexity with respect to the supported mesh elements as well as their numerical robustness. An often found extension is the assignment of physical boundary conditions to the surface facets of the mesh. In this case, an interaction based on some physical model between the particle and the boundary at this intersection point is performed whenever a particle trajectory intersects with a boundary facet $F \subset \partial\Omega$. Such discrete interactions are routinely employed for direct simulation Monte Carlo [32] (DSMC) or multi-particle collision dynamics [33] (MPCD). Especially for the case of DSMC, the integration of boundary interactions into a particle tracking algorithm is a commonly employed technique [21, 23, 24, 34]. Likewise, in case the simulation framework relies on a decomposition of the computational domain as part of a parallelization scheme, the transition of a individual particle from one subdomain to another can be triggered by transitions between mesh elements assigned to different subdomains [26].

Albeit the long history of these algorithms and possibly due the alleged simplicity of the problem, there seems to be no general framework how to derive numerically robust tracking algorithms. At least parts of a number of publications [21–27] can be attributed to various efforts to increase the efficiency and robustness of the scheme. In this work, "robustness" refers to the resilience of the algorithms with respect to common numerical issues such as the limited precision of floating-point calculations, imperfect meshes with non-planar faces, leading to small "gaps" between adjacent mesh elements [24, 25]. These issues can result in errors in detecting the correct transition points between neighboring elements, which in turn may prevent the tracking algorithm from obtaining a valid mapping

between particle positions and mesh elements. While this may happen only in very few specific constellations during the run of a simulation for each individual particle, with simulations routinely employing $10^6$ or even $10^9$ particles, these issues are encountered in practice and can lead to issues which are hard to debug, especially in simulations on large parallel systems. Also, when combined with the handling of physical boundary conditions as introduced above, erroneous repeated interactions caused by numerical errors can occur. While this is not as critical for some boundary conditions such as walls resulting in a diffuse reflection, for other boundary conditions the correct number of boundary interactions is crucial. An example are surfaces tied to some chemical reaction model where repeated interactions can have a significant impact on the simulation result.

Conceptionally, the problem of formulating a robust tracking algorithm is analogous to the construction of a robust event detection for event-driven particle dynamics (EDPD): The challenge lies in the reliable and computationally efficient calculation of the next *event*. In the absence of dissipative forces, EDPD particles follow a ballistic trajectory in between discrete interactions or *events*. As the motion of each individual particle is obtainable analytically between these events, the scheme becomes *event-driven*, where each algorithmic step is a search for the next event. Reformulating the general tracking algorithm as a search for the next event has two benefits: First, recent improvements in the robustness of the event detection [30, 31] can now be integrated directly into the particle tracking routine, which is discussed in the next section. Second, it allows the consistent extension of the algorithm to handle complex composite objects, which we explore further in Section 4.

## 3. Robust event detection for particle tracking

In order to formulate a robust EDPD methodology, the authors systemized the approach for obtaining stable event detection algorithms in a previous work [30]. There, several fundamental concepts are introduced: (1) the definition of *valid states*, (2) suitable *overlap functions*, and (3) *stabilizing interactions*. For EDPD, a valid state can be defined as a physically meaningful configuration of particles, e.g., for a hard-sphere system a configuration where no particles are overlapping. Whether a given pair of particles is in valid state or not at the time $t$ is indicated by the overlap function, $f(t)$. This function is specific to the implemented interaction model, but generally the evaluation of the time-dependent overlap function returns a positive value for a valid and a negative value for an invalid state. It is then the task of the event detection algorithm to generate stabilizing interaction events between particles entering invalid states in order to prevent the further deterioration of the configuration.

Before applying the EDPD methodology, the significance of maintaining a valid state also for particle tracking should be emphasized. As the tracking algorithm is typically not concerned with the interactions between particles due to some (pair-wise) interaction model, the individual particles can be considered independently. A valid state for a single particle $i$ is then a valid mapping between the physical position of the particle, $\boldsymbol{r}_i(t)$, and the logical position in the mesh, described by the index $c_i(t)$. Consequently, in this context an invalid state does not correspond to a violation of some physical law, but rather a state where the mapping between the logical position in the mesh and a particle's (center-of-mass) position is violated.

For particle tracking, the choice of the overlap function is dependent both on the type of mesh elements and the complexity of the particle trajectories. There is some freedom in the choice of $f(t)$, but ideally the overlap function should be simple enough that its roots can be found efficiently using an analytical scheme. In this work we focus on linear trajectories, but depending on the application, higher order trajectories can be considered analogously. For convex mesh elements bounded by line segments (2D) or planar polygons (3D), only one type of overlap function is required, as each boundary segment divides space into two half-spaces (indicated by the dashed red lines in Fig. 2 for the three-dimensional case). Let $\mathcal{F}_j$ denote the set of facets forming the boundary of an element $E_j \in \mathcal{M}$ and $k$ the index of a facet, such that $F_k \in \mathcal{F}_j$. Then, assuming a unit normal $\hat{\boldsymbol{n}}_k$ pointing inwards into $E_j$ and a point on the planar facet, e.g., the centroid of the facet, $\boldsymbol{c}_k \in F$, an overlap function for a particle $i$ can be defined as

$$f_{\mathrm{P}}(t + \Delta t) = \hat{\boldsymbol{n}}_k \cdot [\boldsymbol{r}_i\,(t + \Delta t) - \boldsymbol{c}_k]\ , \tag{3}$$

where $\Delta t$ is the relative time of the event. This somewhat unconventional choice of the normal direction is required by the definition of the overlap function to evaluate to positive values for valid states. It should be noted that in order to determine the first intersection point between the trajectory and the facets, it is not necessary to test explicitly whether this point on the surface of a half-space is indeed contained within a certain facet (compare $\mathcal{F}_{k_2}$ in the example of

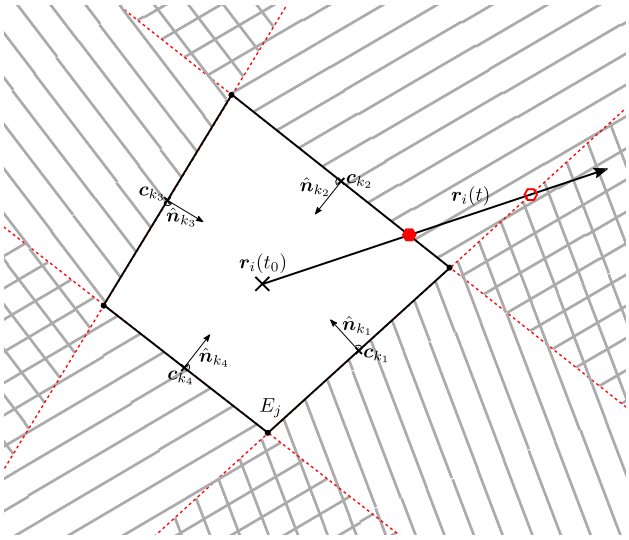



Figure 2: The exterior of a convex element $E_j \in \mathcal{M}$ can be described using planar half-spaces determined by the set of facets $\mathcal{F}_j$ bounding the element. Each half-space is defined by a point $\boldsymbol{c}_k$ contained in the facet $F_k \in \mathcal{F}_j$ and the inward pointing unit normal vector $\hat{\boldsymbol{n}}_k$. The points marked by the hexagons correspond to the intersection points of the trajectory of particle $i$ and the facets $k_2$ and $k_1$, respectively.

Fig. 2). This is ensured by the geometric properties of the element and greatly reduces the computational cost, but requires convex mesh elements. While the overlap function of Eq. (3) is restricted to convex mesh elements, this does not limit the general applicability as concave meshes can be composed of convex elements. Additionally, if concave elements have to be employed, these can be decomposed into convex sub-elements. The restriction to planar boundary facets significantly reduces the complexity of the overlap function. It is possible, in principle, to employ an overlap function of higher complexity, but the additional cost in finding the roots of this overlap function will limit the practical relevance to simple cases, for example, bilinear patches [35].

In the stable formulation of the EDPD algorithm [30], an event is scheduled for the smallest non-negative time interval, $\Delta t$, that satisfies the condition

$$\Big(f\,(t+\Delta t) \leq 0\Big) \text{ and } \Big(\dot{f}\,(t+\Delta t) < 0\Big)\,. \tag{4}$$

The sign of the derivative of $f(t)$ with respect to time is used as an indicator whether an invalid state ($f(t) < 0$) deteriorates with time ($\dot{f}(t) < 0$) or not. So in addition to Eq. (3), it is also necessary to consider the time derivative of $f_\mathrm{P}(t)$, which for the linear trajectory of particle $i$ is given by

$$\dot{f}_\mathrm{P}(t+\Delta t) = \hat{\boldsymbol{n}}_k \cdot \dot{\boldsymbol{r}}_i\,. \tag{5}$$

The condition $\dot{f}_\mathrm{P}(t) < 0$ limits the viable set of facets for an event to the ones the particle approaches along its trajectory. It is noteworthy, that this condition is commonly found in other particle tracking algorithms as an optimization technique [21, 24], but here it additionally stabilizes the algorithm and prevents, e.g., spurious recollisions with faces the particle has already passed through. Solving for the root of the overlap function, that is, $f_\mathrm{P} = 0$, under consideration of the sign of $\dot{f}_\mathrm{P}$ yields

$$\Delta t = \begin{cases} \max\left(0, \frac{\hat{\boldsymbol{n}}_k \cdot [\boldsymbol{c}_k - \boldsymbol{r}_i]}{\hat{\boldsymbol{n}}_k \cdot \dot{\boldsymbol{r}}_i}\right) & \text{if } \hat{\boldsymbol{n}}_k \cdot \dot{\boldsymbol{r}}_i < 0 \\ \infty & \text{if } \hat{\boldsymbol{n}}_k \cdot \dot{\boldsymbol{r}}_i \geq 0 \end{cases}, \tag{6}$$

where the case of no intersection between the particle trajectory and the facet with index $k$ is treated explicitly as an infinite event time. For a precise simulation without any numerical errors, clamping the result to zero in the first case of Eq. (6) is not required, as no negative values can appear. In real implementations, however, the limited machine

precision of floating-point calculations will allow particles to enter invalid states. Such invalid states can be entered, for example, when the final tracking position of particles numerically coincides with a facet (see Fig. 3 of [30] for a similar example). Another typical source of invalid states are faces which are not precisely planar, resulting in small gaps between mesh elements [24, 25]. Independent of the source of the invalid state, a robust event-driven particle tracking algorithm must be able to recover and stabilize the system, and, if possible, return it to a valid state. This becomes feasible by the generation of *stabilizing events*, that is, events with $\Delta t = 0$, as according to Eq. (6), instead of ignoring these cases and/or allowing negative values for $\Delta t$.

## 4. Constructive solid geometry

While particle tracking in combination with unstructured meshes allows for particle-based simulations in complex geometries, there is a serious downside: The computational cost is a function of the distance particles travel in between tracking steps and the resolution of the mesh. In order to capture boundary features that are small compared to the overall size of the computational domain, a finely resolved mesh is required, at least locally. This is feasible for many technical devices such as fluidized bed reactors, where only some parts of the domain might necessitate a refined mesh. For studies of flows through porous media or studies of microfluidic devices, however, the boundaries dominate the behavior of the system. Here, the required high spatial resolution of the mesh would severely limit the applicability of the tracking scheme when the mobility of the computational particles is high as it is the case for particle-based simulations of flows.

### *4.1. Combining CSG with particle tracking*

There are other means of representing domain boundaries in particle methods compatible with the idea of particle tracking, such as the embedding of obstacles represented via triangulated surfaces [36, 37]. Nevertheless, when finely resolved boundaries are required, for particles near these boundaries large numbers of triangles would have to be taken into account during the tracking step. Considering the similarity between tracking algorithms on the one hand and EDPD and ray-tracing algorithms on the other hand, we propose an alternative method of modeling complex geometries. Instead of relying solely on a unstructured grid, a second logical layer can be added, representing additional domain boundaries. This layer consists of objects modeled via constructive solid geometry (CSG), a standard technique both in computer graphics and computer-aided design [38], as well as simulations of radiation transport problems [39].

In CSG, simple solid objects termed *primitives*, regularized Boolean operations, and affine transformations are combined to represent a more complicated solid object. For a $d$-dimensional Euclidean space the primitives are formed by algebraic half-spaces

$$H = \{\boldsymbol{x} \in \mathbb{R}^d \mid g(\boldsymbol{x}) \leq 0\}\,, \tag{7}$$

with $g(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}$ being an irreducible polynomial. The regularized Boolean operations *union*, *intersection*, and *difference* are defined such that the result is the closure of the interior of applying the Boolean operation to the two operands. This ensures that the object generated by a regularized Boolean operator is again a solid object without any "dangling" lower-dimensional features, e.g., a plane in three dimensions [38]. A CSG solid is usually represented with the help of a (binary) tree, where the leaf nodes are primitives and the internal nodes convey regularized Boolean operations. Affine geometric transformations can then be applied at arbitrary nodes in order to transform primitives and, via sub-trees, parts of the model. The overlap function $f(t)$ as it was introduced in Section 3 shares a core characteristic with the function $g(\boldsymbol{x})$ defining a CSG half-space: Both functions serve as indicators to distinguish two sets of points or sub-spaces. For constructive solid geometry, $g(\boldsymbol{x})$ divides space into two sub-spaces, namely the closure of the interior of the solid ($g(\boldsymbol{x}) \leq 0$) and the exterior ($g(\boldsymbol{x}) > 0$). Similarly, the overlap function partitions the configurational space into invalid ($f(t) \leq 0$) and valid ($f(t) > 0$) configurations or states.

In the interest of clarity, in the following we will only consider the case of point particles which are to be tracked in complex geometries defined by CSG. Further, we will limit the discussion to the three-dimensional case and the two primitives given by an infinite planar half-space and a spherical half-space or closed ball, respectively. The

corresponding algebraic half-spaces $H_\mathrm{P}$ and $H_\mathrm{B}$ are defined using a point, $\boldsymbol{c} \in \mathbb{R}^3$, and a unit normal vector, $\hat{\boldsymbol{n}} \in \mathbb{R}^3$, for the direction, or the scalar radius, $r \in \mathbb{R}_{>0}$, respectively, according to:

$$H_\mathrm{P}(\boldsymbol{c}, \hat{\boldsymbol{n}}) = \{\boldsymbol{x} \in \mathbb{R}^3 \mid (\boldsymbol{x} - \boldsymbol{c}) \cdot \hat{\boldsymbol{n}} \leq 0\}\,, \text{ and} \tag{8}$$

$$H_\mathrm{B}(\boldsymbol{c}, r) = \{\boldsymbol{x} \in \mathbb{R}^3 \mid (\boldsymbol{x} - \boldsymbol{c})^2 \leq r^2\}\,. \tag{9}$$

The result of applying the regularized Boolean operator *union* denoted by $\cup^*$ to two half-spaces $A$ and $B$, with $A, B \subset \mathbb{R}^3$ is defined as

$$A \cup^* B = \mathrm{cl}\,(\mathrm{int}\,(\{\boldsymbol{x} \mid \boldsymbol{x} \in A \vee \boldsymbol{x} \in B\}))\,, \tag{10}$$

while the regularized *intersection* of the two half-spaces denoted by $\cap^*$ is given as

$$A \cap^* B = \mathrm{cl}\,(\mathrm{int}\,(\{\boldsymbol{x} \mid \boldsymbol{x} \in A \wedge \boldsymbol{x} \in B\}))\,, \tag{11}$$

with $\mathrm{cl}\,(\cdot)$ denoting the closure and $\mathrm{int}\,(\cdot)$ denoting the interior of the argument.

Again, the union of planar half-spaces is already used implicitly in Section 3, when selecting the next event for a particle located in a convex mesh element. For the event-driven tracking algorithm, a CSG tree containing only unions can be used directly to detect the next event a particle encounters. To this end, the tree is traversed recursively starting from the root and each primitive is tested using the corresponding overlap function. The resulting events are then sorted in ascending order according to their respective event times and the earliest event is selected, corresponding to finding the *minimum* event time for all leaf nodes.

### *4.2. Logical states for particle tracking*

The event detection for intersecting half-spaces is significantly more complex. As an example for the intersection of half-spaces, the case of three intersecting closed balls with equal radii shall be considered. A two-dimensional projection of the configuration into the plane containing the center points of the balls is shown in Fig. 3. The trajectories of the two particles illustrate different cases, where in the first case the composite CSG object, i.e., the intersection of the three balls is missed, while in the second case the trajectory actually intersects the solid object. The intersection points of the trajectories and the surface of the individual primitives are marked by red hexagons and crosses, indicating the points where the trajectories enter and leave the half-spaces defined by the primitives.

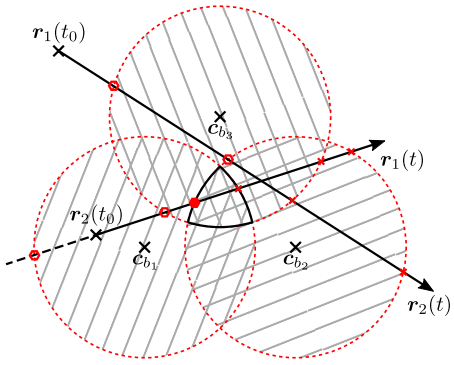



Figure 3: The projection of three intersecting balls $H_{b_1}$, $H_{b_2}$, and $H_{b_3}$ onto the plane defined by their center points. The trajectory of two tracked particles with starting points $\boldsymbol{r}_1(t_0)$ and $\boldsymbol{r}_2(t_0)$, respectively, intersects the individual primitives at the highlighted points, where red hexagons and crosses mark the points where the trajectory enters/leaves a primitive half-space and the filled hexagon indicates the point of interaction with the CSG solid for the second particle.

In classic ray-tracing algorithms, all points in time when the trajectory intersects the surfaces of the primitives in an intersection are taken into account and the *maximum* time the trajectory enters an object (red circles in Fig. 3) bounded by the *minimum* time the trajectory exits one of the objects intersecting (red crosses) is picked. If one of the objects in an intersection is missed, as it is the case for the second particle in Fig. 3, the particle does not hit the

composite object and the ray tracer does not schedule an interaction in any way. While this approach has the benefit of being straightforward to implement, it is inadequate for formulating a numerically robust tracking algorithm. Unlike for ray tracing, when applying tracking algorithms, particles can be located close to/on surfaces or inside tiny gaps in-between objects at the end of a tracking step. When resuming the tracking in the next step, this can result in cases where is not always possible to correctly predict the next event for certain configurations like (locally) very thin CSG objects resulting in numerically coinciding roots of the respective overlap functions. Here, in some cases, it is possible to miss an interaction due to the numerical imprecision of floating-point calculations.

Due to these difficulties, an alternative approach for handling intersections is taken. The concept of the *logical state* is introduced by the authors in a previous work [31] to distinguish captured and uncaptured square-well particles in EDPD. It is here applied to CSG modeling in order to determine whether a tracked particle is located within a given CSG primitive or not without relying on the relative position on the particle. Accordingly, the state of a particle is augmented by $\mathcal{P}_i$, the set of primitives within the CSG tree that the particle is currently contained in logically. Ignoring the limitations of floating-point precision and assuming a CSG tree consisting of $N_H$ primitive half-spaces forming the set of leaf nodes $\mathcal{P} = \{H_j\}_{j=1}^{N_H}$, for a particle $i$ this results in

$$\mathcal{P}_i = \{H \in \mathcal{P} \mid g_H(\boldsymbol{r}_i(t)) \leq 0\}\,, \tag{12}$$

where $g_H$ denotes the indicator function for the half-space $H$. As it is not possible to robustly recover $\mathcal{P}_i$ from a particle's current position $\boldsymbol{r}_i(t)$ due to the peculiarities of floating-point calculations, the set must be manipulated exclusively via the execution of events. This is done via *virtual events* analogous to the events for the tracking in meshes. Reconsidering the configuration sketched in Fig. 3, virtual events are scheduled at the points highlighted with red rings as well as any point where the trajectory exits a primitive half-space marked by red crosses; these events only add or remove primitives to/from $\mathcal{P}_i$. By examining this set and the number of primitive sub-nodes of an intersection in a CSG tree it is then possible to accurately decide that an interaction event has to be scheduled at the point of the trajectory in Fig. 3 marked with the full circle. This scheme can directly be combined with the handling of unions to allow for compound CSG objects.

As an additional benefit, this explicit tracking of the particle state allows the modeling of permeable objects. Such an CSG object does not have to correspond to a solid obstacle in combination with an appropriate boundary condition any more. Instead, it is possible to define complex-shaped regions within the computational domain where for example certain interactions are activated locally. One example could be the surrounding volume of an actual solid where additional forces act on the particles.

### *4.3. Transformation of Boolean operators*

The regularized *difference* or *relative complement* of the half-spaces $A$ and $B$, given by

$$A \setminus^* B = \mathrm{cl}\left(\mathrm{int}\left(\{\boldsymbol{x} \in A \mid \boldsymbol{x} \notin B\}\right)\right), \tag{13}$$

is not easily translatable into a robust event-driven particle tracking scheme. Instead, the absolute complement $A^C = U \setminus^* A$ of a half-space $A$ and the universe $U \equiv \mathbb{R}^3$ is used. With $\left(A^C\right)^C \equiv A$ and recursively applying De Morgan's laws,

$$(A \cup B)^C \equiv A^C \cap B^C \qquad \text{and} \qquad (A \cap B)^C \equiv A^C \cup B^C\,, \tag{14}$$

any occurrence of $A \setminus^* B$ in a CSG tree can be replaced by $A \cap^* B^C$. For the internal nodes of the tree only these laws have to be applied, whereas for a primitive half-space $H$ with the corresponding indicator function $g_H$ at a leaf node of the tree a different transformation is required according to

$$\begin{aligned} H^C &= \mathbb{R}^3 \setminus^* H \\ &= \mathbb{R}^3 \setminus^* \{\boldsymbol{x} \in \mathbb{R}^3 \mid g_H(\boldsymbol{x}) \leq 0\} \\ &= \mathrm{cl}\left(\mathrm{int}\left(\{\boldsymbol{x} \in \mathbb{R}^3 \mid g_H(\boldsymbol{x}) > 0\}\right)\right) \\ &= \{\boldsymbol{x} \in \mathbb{R}^3 \mid -g_H(\boldsymbol{x}) \leq 0\}\,, \end{aligned} \tag{15}$$

leading to an inversion of the indicator function. This inversion can be performed analogous for the overlap function $f(t)$ used in the event detection for the particle tracking. In the case of planar half-spaces, as an optimization, the normal vector $\hat{\boldsymbol{n}}$ can be inverted directly, obviating the need to treat the complement explicitly. For the complement of a spherical half-space, the overlap function provided in Appendix 2 of [31] can be employed.

### *4.4. Event detection for CSG objects*

A numerically robust method for the detection of tracking events between a particle $i$ and a CSG tree including the manipulation of the logical state $\mathcal{P}_i$ can be implemented according to Algorithm 1. In addition to the particle trajectory and state, the routine requires a non-empty set of nodes $\mathcal{N}$ as input, which initially is set to the root node of the CSG tree. The tree is then processed recursively, examining the type of each node and either descending further down the tree, or, if a leaf node is encountered, detecting the next event for the corresponding primitive using the subroutine NEXTPRIMITIVEEVENT. The handling of CSG objects requires three distinct event types, indicating a particle

**Algorithm 1** Implementation of the event detection for CSG objects employing the logical state $\mathcal{P}_i$ of a tracked particle $i$ and a set of nodes $\mathcal{N}$, initialized with the root node of the CSG tree $T$.

```
1:  procedure NEXTEVENT(r_i, ṙ_i, P_i, N)
Require: CSG tree T with the set of primitives P = {H_j}_{j=1}^{N_H} as leaf nodes
Require: P_i ⊆ P ∧ N ⊆ NODES(T)
2:      Δt ← ∞, e ← nil, ς ← 0
3:      for n ∈ N do
4:          if n ∉ P then                                   ▷ n is an internal node, i.e., a Boolean operation
5:              Δt_n, e_n, ς_n ← NEXTEVENT(r_i, ṙ_i, P_i, CHILDREN(n))
6:              if TYPE(n) = intersection then
7:                  if TYPE(e_n) = csgBoundary ∧ (ς_n + 1) < |CHILDREN(n)| then
8:                      SETTYPE(e_n, csgEnter)
9:                  else if ς_n = |CHILDREN(n)| then
10:                     ς ← ς + BOOLEAN(ς_n)
11:                 end if
12:             else if TYPE(n) = union then
13:                 ς ← ς + BOOLEAN(ς_n)
14:             end if
15:         else                                            ▷ n is a leaf node, i.e., a CSG primitive
16:             if n ∉ P_i then
17:                 Δt_n, e_n ← NEXTPRIMITIVEEVENT(r_i, ṙ_i, n, P_i)
18:                 SETTYPE(e_n, csgBoundary)
19:             else
20:                 Δt_n, e_n ← NEXTPRIMITIVEEVENT(r_i, ṙ_i, COMPLEMENT(n), P_i)
21:                 ς ← ς + 1
22:                 SETTYPE(e_n, csgLeave)
23:             end if
24:         end if
25:         if Δt_n < Δt then
26:             Δt ← Δt_n, e ← e_n
27:         end if
28:     end for
29:     return Δt, e, ς
30: end procedure
```

*entering* (line 8, event type `csgEnter`) or *leaving* (line 22, event type `csgLeave`) a half-space, as well as actually hitting the surface or boundary of a CSG object (line 18, event type `csgBoundary`). The first two are virtual events, leading only to an adjustment of $\mathcal{P}_i$, while the latter indicates an interaction with a boundary, as detailed in Section 2.

The recursive calculation of the number of intersected primitives $\varsigma$ and the modification of the event type on line 8 are only required to correctly handle intersections. Hence, if the CSG tree is known to consist solely of unions, the algorithm can be simplified accordingly.

## 5. Optimization of particle tracking using CSG

For simulations using more elaborate CSG models than the examples depicted in Fig. 3 consisting only of a few primitives, the event prediction using the full CSG tree can become a bottleneck. It is therefore vital to integrate standard optimization techniques from the field of CSG modeling into the tracking scheme.

### *5.1. Bounding volumes for particle tracking*

One such technique to reduce the number of required intersection tests is the usage of bounding volumes, which can be integrated directly into the CSG tree at the internal nodes [40]. The idea is to determine bounding volumes for the primitives at the leaf nodes and iteratively combine these bounding volumes at the internal nodes based on the respective Boolean operation. By applying this procedure not only upward but also downward, it can also be used to prune effectively empty sub-trees from a CSG model, based, e.g., on an intersection higher up in the tree. A commonly used bounding volume is the *axis-aligned bounding box* (AABB), that is, the minimum $d$-dimensional cube bounding an object under the constraint that its edges are parallel to the coordinate axes. Such bounding boxes are trivial to determine for typical primitives and the Boolean operations required for CSG modeling are efficiently computable for any two given AABBs. Combining the bounding volumes with additional data structures such as octrees [19] or k–d trees [20] leads to further improvements in the computational efficiency.

In principle, the integration of such AABBs into the particle tracking scheme is straightforward, yet here we shall explore an alternative. Instead of using the AABBs directly during the event generation phase, they are combined with the underlying mesh. This results in a two-level approach, where for each CSG object first the corresponding AABB is calculated and then the AABBs of the grid cells intersecting the first AABB are determined. An example for applying this procedure to the CSG object of Fig. 3 is sketched in Fig. 4, where the cells intersecting the AABB of the solid object are shaded in red. Each CSG obstacle is then registered only with these selected cells, as no `csgBoundary`

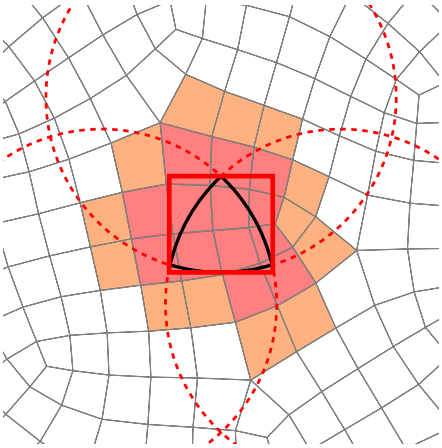


Figure 4: Two-level approach to determine an bounding volume via an underlying grid using intersection tests for the axis-aligned bounding box (AABB) of the CSG object introduced in Fig. 3 and the AABBs of the individual grid cells. The cells whose AABBs intersect the AABB of the CSG object directly are shaded in red. The cells sharing facets with the marked cells are shaded in orange, leading to an expanded, numerically robust bounding volume.

events (see Algorithm 1) can be generated involving the obstacle and particles contained in other cells.

The motivation for using the mesh elements as bounding volumes is twofold: First, the mapping between particle positions and grid cells is already maintained by the tracking algorithm, eliminating the need for additionally testing against the bounding volumes of the CSG objects. This significantly reduces the computational cost of the event

generation. Second, instead of storing a reference to the full CSG tree, a locally pruned version of the tree containing only the relevant parts of the tree for each cell can be used. While seemingly a technical detail, reusing the mesh elements instead of additional bounding volumes represents a major difference in the extended particle tracking method as compared to traditional ray-tracing algorithms.

There is, however, an issue with respect to the handling of the logical state, as introduced in Section 4.2: As the event detection for CSG objects is only performed for the cells marked according to the bounding volume, particles can enter CSG primitives "unnoticed" (compare Fig. 4). When then applying Algorithm 1, based on the test on line 16, the algorithm would fail to generate the correct event. In order to overcome this shortcoming, it is necessary to evaluate Eq. (12) for each primitive in the (local) CSG tree whenever a particle enters a cell referencing a CSG object and adjust the state $\mathcal{P}_i$ accordingly. Due to the inherent imprecision of floating-point calculations, however, this would violate the robustness of the event-driven tracking algorithm, as close to the surface of CSG primitives the indicator function become unreliable. The bounding volume implemented via the underlying mesh has hence be extended to the elements sharing facets with the originally marked elements, as indicated by the cells shaded in orange for the example in Fig. 4. As the indicator function for the individual primitives is only evaluated at the interface between two cells, this modification guarantees, that the indicator functions are only evaluated in a safe distance to the surface of the CSG object. From this, $\mathcal{P}_i$ can be safely adjusted based on the CSG objects registered with the two cells on either side of a facet. For CSG obstacles present in both cells, it is crucial that the logical state is transferred without modifications.

### *5.2. Bounding volumes using surface extraction*

Even though the locally pruned CSG trees greatly improve the efficiency of the tracking algorithm in many cases, the selection of the nodes in the trees to keep based on the intersection of the AABBs is not always sufficient. While the approach works well for smaller convex objects, for infinite objects, such as, non-intersecting planar half-spaces or the complement of spherical half-spaces, AABBs cannot provide an efficient bounding volume as they also become infinite. A solution to this issue can be found by exploiting the property of Algorithm 1 that CSG events are only generated at the surface of primitives. Instead of pruning the CSG tree based on the AABBs, a more aggressive reduction can be performed by pruning all primitives whose surface does not intersect a certain bounding volume. Here, we again use the polyhedral cells of the grid analogous to the case of AABBs. For a numerically robust intersection detection, the surface of the primitives is transformed into a solid object by taking the Minkowski sum of the respective surface and a spherical half-space with a certain radius. This radius can, for example, be determined as a fixed fraction of the diagonal of the AABB of the polyhedral cell being tested.

Special care has again to be taken to maintain the validity of the logical state $\mathcal{P}_i$ by adjusting it when a particle enters/leaves a mesh element. Additionally, as primitives can be pruned for cells contained entirely within a CSG primitive, a record has to be kept in this case in order to ensure the correct working of the event detection routine. To this end, line 5 of Algorithm 1 has to be adapted to account for this number of implicit intersections.

### *5.3. Memory-optimized storage of particle state*

The introduction of $\mathcal{P}_i$ as a per-particle property required by Algorithm 1 comes at the cost of storing and maintaining this dynamic set. One possibility to store $\mathcal{P}_i$ is the usage of associative containers provided, e.g., by the C++ standard library. While the asymptotic complexity per particle can be reduced to $\mathcal{O}(1)$ by employing hash tables instead of binary search trees, the storage overhead for the container data structure itself as well as the cost of dynamic memory allocations can become prohibitive. As an alternative, we use a fixed-size bit array or bitset $\mathcal{B}_i$ for each particle. The individual bits are assigned to the primitives in the CSG tree, so the manipulation of the state $\mathcal{P}_i$ is performed via bit operations on $\mathcal{B}_i$. This approach has the benefit of low memory requirements as well as profiting from the fact that bit operations are highly optimized for modern CPUs (`POPCNT` or an emulation using other SSE instructions). A potential drawback of this method is that the number of primitives that can be handled is limited by the number of bits in $\mathcal{B}_i$. While even for moderate storage requirements of 4 or 8 bytes per $\mathcal{B}_i$ the number of allowed primitives is already 32 or 64, respectively, for large systems this may still become an unacceptable constraint. But when combined with the previously introduced concept of locally pruned CSG models and hence reduced number of primitives, the limited number of possible primitives *within one cell* is sufficient, even for complex geometric models.

### 5.4. Performance evaluation

In order to evaluate the efficiency of the optimization strategies described above, we examine the performance of the event detection algorithm for a CSG object consisting of the union of 790 spherical half-spaces. Here, we refer only to the computational aspect of our work, the discussion of the physical problem is subject of ongoing work and shall be published elsewhere, including a detailed description of the system. The CSG object forming a cluster of spherical obstacles is embedded into a cuboidal domain described by a mesh consisting of $64^3$ uniform hexahedra (see Fig. 5). The cluster is positioned in the center of the domain, covering approximately 1.77 % of the simulation volume. The ratio between the average diameter of the slightly polydisperse balls forming the cluster and the width

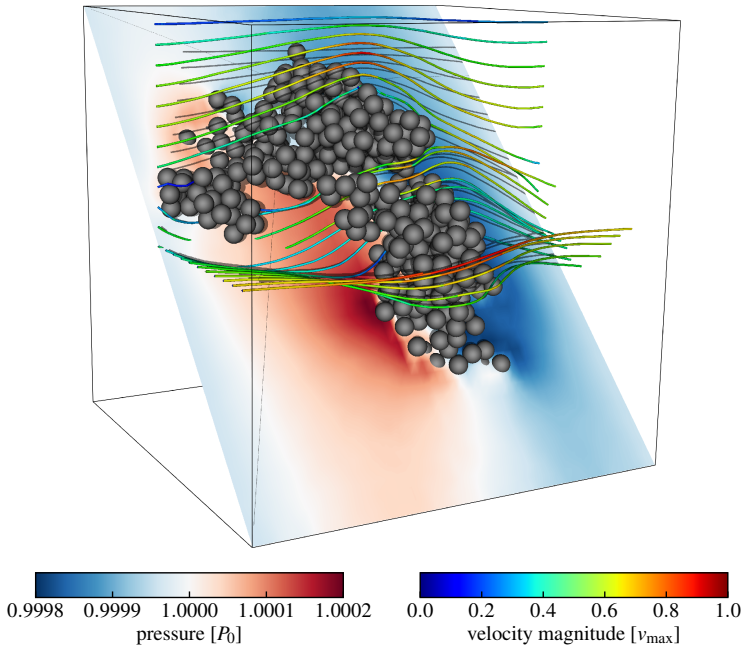



Figure 5: Particle-based simulation of a plane Poiseuille flow around a cluster of 790 spherical obstacles embedded into a domain defined by $64^3$ hexahedra. The fluid phase is modeled using a variant of the direct simulation Monte Carlo method optimized for flows with low Mach numbers [41]. The pressure profile and streamlines are obtained by averaging over the particles contained in the individual grid cells.

of the channel is 3.5 %. The number of randomly placed tracked particles inside of the domain is approximately $25.2 \times 10^6$, so that on average 100 particles are located within each cell of the grid. Each particle is assigned random velocity components and the time scale is set such that, again on average, each particle travels about $\frac{3}{4}$ of a cell width within one tracking step. When the particle tracking is performed taking into account both the unstructured grid and the CSG obstacle, the number of events generated for the CSG object is recorded. Additionally, the number of actually executed events is obtained which is a significantly smaller number, as not every detected event is executed (compare Figs. 1 to 3). Both numbers are averaged over 1000 time steps, in between which the tracked particles are assigned updated velocities.

To determine the efficiency of the bounding volumes as described in Section 5.1, as a base line, the performance of the event detection is examined without any bounding volumes (first line in Table 1). Next, a series of runs is performed where for each instance a progressively finer mesh covering the entire domain is used to provide the bounding volumes for the CSG object. Starting from a uniform mesh with $2^3$ hexahedral elements, the resolution is increased stepwise up to $64^3$ elements, identical to the main mesh used for the particle tracking. As the finer meshes are derived from the coarser ones by uniform refinement, there is a trivial mapping between the cells of the tracking mesh and the cells of the additional mesh serving as the bounding volumes for the CSG object. During the initialization of each run, the full tree consisting of 790 primitives is pruned based on the bounding volumes and an optimized CSG tree with local validity is created. This unsurprisingly leads to a decreasing number of primitives

| | Primitives per cell | | | Number of events | | Relative efficiency | |
|---|---|---|---|---|---|---|---|
| Grid cells | Min. | Max. | Avg. | Generated | Executed | Event generation | Computational |
| – | – | – | – | $3.79 \times 10^{10}$ | $7.93 \times 10^{4}$ | 1.00 | 1.00 |
| $2^3$ | 173 | 294 | 228 | $1.09 \times 10^{10}$ | $7.93 \times 10^{4}$ | 3.46 | 3.21 |
| $4^3$ | 0 | 236 | 29.1 | $1.20 \times 10^{9}$ | $7.93 \times 10^{4}$ | 31.5 | 17.4 |
| $8^3$ | 0 | 86 | 4.80 | $1.74 \times 10^{8}$ | $7.93 \times 10^{4}$ | 217 | 35.8 |
| $16^3$ | 0 | 27 | 1.02 | $3.14 \times 10^{7}$ | $7.93 \times 10^{4}$ | $1.2 \times 10^{3}$ | 42.2 |
| $32^3$ | 0 | 12 | 0.291 | $7.54 \times 10^{6}$ | $7.93 \times 10^{4}$ | $5.02 \times 10^{3}$ | 43.6 |
| $64^3$ | 0 | 8 | 0.122 | $2.70 \times 10^{6}$ | $7.93 \times 10^{4}$ | $1.4 \times 10^{4}$ | 44.5 |

Table 1: Impact of the resolution of the grid used to determine the bounding volumes of the CSG primitives forming the cluster depicted in Fig. 5 when embedded as a solid obstacle into a fluid flow simulated using a particle-based method. The primitives are filtered via the intersection of their individual AABBs and the AABBs of the grid cells.

per cell with increasing resolution of the grid. As shown in Table 1, this in turn results in a dramatic decrease in the number of generated events and hence a corresponding increase in the efficiency of the event detection. The number of actually executed events remains constant, which serves as a sanity check for the correct working of the event detection in combination with the bounding volumes. The computational efficiency is determined based on the average runtime of applying the tracking algorithm to all particles in the system. This thus includes the generation and processing of the events related to tracking the movement of the particles in the mesh.

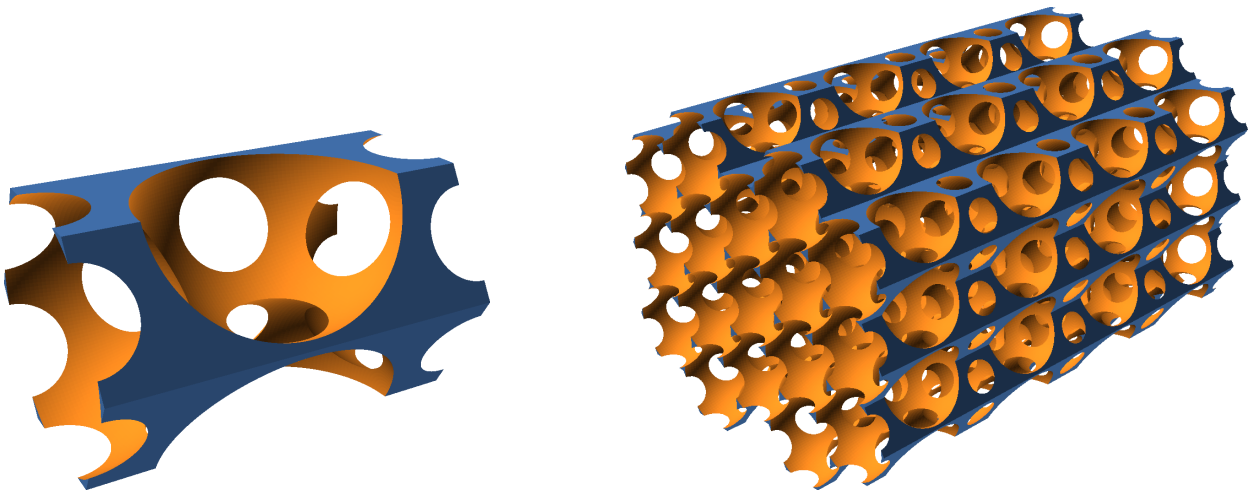
These results clearly show the effect of the optimization via the bounding volumes, leading to an increase in the computational efficiency of up to a factor of 44.5. The increase of the computational efficiency levels out for finer grids, as the event-driven tracking scheme also has to generate and process the events related to the tracked particles moving in the grid defining the cells. The number of events due to particles crossing facets in-between mesh elements is independent of the resolution of the grid defining the bounding volumes and with about $2.26 \times 10^{7}$ events per tracking step progressively dominates the computational performance in this test case.

Depending on the CSG models employed in a specific simulation configuration, several of the strategies introduced in this section can be combined. While the last two concepts, that is, the extraction of the surface of CSG models and the usage of a fixed-size bitset to store the particle state, are of lesser relevance for the simulation of the flow around the cluster, in the next section a more advanced CSG model is constructed, where especially these two optimizations are crucial.

## 6. Application example: Open-cell foams

The advantages of using the hybrid representation of a computational domain via a mesh and CSG objects embedded into it for a particle-based simulation can be illustrated using the example of fluid flows through open-cell foams. These foams feature a high porosity which makes them attractive for reactive gas flows due to the combination of a large surface area with a low pressure drop. It is this high porosity, however, which poses a challenge for numerical simulations as the struts between the pores of the foam require a finely resolved mesh [42]. Instead of generating a conforming mesh which can be challenging both in terms of mesh quality and computing time, we borrow an approach used in analytical studies of open-cell foams [43]. Here, an unit cell for an open-cell foam is constructed as the inversion of five overlapping spheres arranged in a hexagonal packing. The model is able to reproduce the characteristics of metal foams with a porosity of up to 96 %. An example unit cell is depicted in Fig. 6(a), revealing that this inversion of a sphere packing can be modeled using the CSG approach as the difference of a polyhedron and several spherical half-spaces. When several of these unit cells are combined periodically, a highly porous material resembling an open-cell foam is formed as shown in Fig. 6(b).

The simulation domain is constructed as a mesh consisting of about $1.4 \times 10^{6}$ cuboidal hexahedra and a single CSG object embedded into it. The CSG model consists of the difference of a planar half-space and the union of 266 spherical half-spaces. The planar half-space is positioned such, that the full mesh is contained in it, while the spherical half-spaces form the open cells of the foam structure. With the extensions to the particle tracking algorithm introduced



(a) (b)

Figure 6: The unit cell of an open-cell foam constructed from an inverted sphere packing [43] is shown in sub-figure (a). This unit cell can be modeled using CSG as the relative complement of a polyhedron and five spherical half-spaces. In sub-figure (b), multiple unit cells are combined, resulting in a fully analytical description of a highly porous material consisting of 267 CSG primitives.

in Section 4, this model can be used directly in a particle-based fluid simulation, without any discretization of the surface. This eliminates the need to generate a volume mesh covering the free volume, i.e., the part of the domain not obstructed by the solid foam. More importantly, the surface description used in the simulation is independent of the underlying mesh. This mesh can thus be generated solely based on the requirements of the particle method.

The fluid phase is modeled using a variant of the direct simulation Monte Carlo method optimized for flows featuring low Mach numbers [41], where the cells of the underlying mesh form the interaction volume for the fluid particles. The boundary conditions applied are no-slip boundaries on the surface of the foam and periodic boundaries in the two directions perpendicular to the pressure gradient.

During the initialization of the simulation, the approach discussed in Section 5 to extract for each bounding volume only the primitives featuring a surface intersection with this bounding volume is employed. The logical state $\mathcal{P}_i$ for each particle implemented as a fixed-size bitset with 32 entries, resulting in a storage requirement of 4 bytes per particle. Employing these optimizations is crucial, as the example simulation uses approximately $115 \times 10^6$ particles to resemble the fluid phase, so a minimal memory footprint for each particle is desirable. A total of 320 processor cores across 16 physical nodes each containing two Intel Xeon E5-2660 v2 CPUs are used in the example run. During each tracking step, approximately $1.1 \times 10^8$ events are executed, with the fraction of CSG events encountered by particles impinging the surface of the foam structure being about 8 %.

In Fig. 7, the pressure profile and streamlines obtained for the steady state solution of a pressure-driven laminar flow through the foam structure is shown. The surface-triangulation of the foam used in this snapshot is generated purely for the purpose of visualization, for the actual simulation the whole geometry is described analytically and no discretization of the foam surface is required.

## 7. Conclusions

The tracking of particles in unstructured grids can be considered a special case of event-driven particle dynamics and as such can benefit from recent improvements in the numerical robustness of the method. With this, it is possible to construct an inherently stable event-driven particle tracking scheme. This event-driven tracking scheme can be extended to allow the integration of complex-shaped boundaries modeled using the technique of constructive solid geometry (CSG). Using this mathematically exact representation of boundaries, efficient particle-based simulations in domains of challenging shape are achievable. The benefit of this approach lies in the independence of the boundary

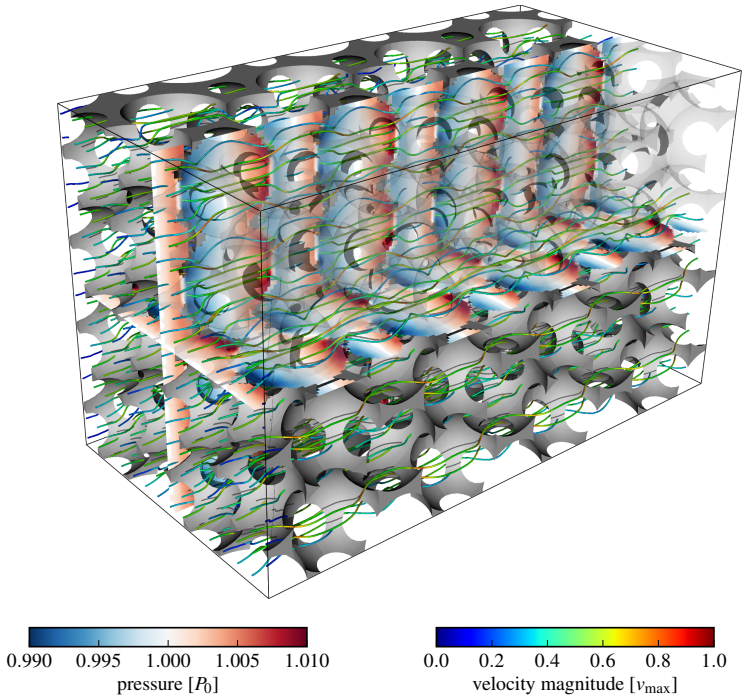



Figure 7: Pressure profile and streamlines obtained from the steady-state solution of a gas flow through a foam-like structure driven by a pressure gradient along the longitudinal axes. The simulation employs a variant of the direct simulation Monte Carlo method optimized for flows with low Mach numbers [41]. The streamlines a colored according to the velocity magnitude at the respective position.

description from any surface discretization. This is a clear advantage over previously published particle-tracking algorithms, where the simulation domain is discretized and represented solely via unstructured grids and the resolution of the boundaries is hence tied directly to the resolution of the mesh. With the method proposed in this work, depending on the specifics of the problem, a combination of unstructured grids and analytically described boundaries can be used. Several optimization strategies to improve the efficiency of the event-driven tracking scheme in the presence of complex CSG objects are suggested. Along with the integration of discrete boundary interactions, the abilities of the event-driven tracking algorithm are illustrated by the particle-based simulation of a gas flowing through an open-cell foam structure.

## Acknowledgements

The authors gratefully acknowledge the support of the German Research Foundation (DFG) through Grants PO 472/20 and SFB-814. FPS and ZISC are thanked for support. We would like to thank Prapanch Nair for helpful discussions.

## References

[1] B. J. Alder, T. E. Wainwright, Phase transition for a hard sphere system, J. Chem. Phys. 27 (5) (1957) 1208–1209. doi:10.1063/1.1743957.

[2] A. A. Alexeenko, D. Fedosov, S. F. Gimelshein, D. Levin, R. Collins, Transient heat transfer and gas flow in a MEMS-based thruster, J. Microelectromech. Syst. 15 (1) (2006) 181–194. doi:10.1109/JMEMS.2005.859203.

[3] M.-C. Lo, C.-C. Su, J.-S. Wu, K.-C. Tseng, Modelling rarefied hypersonic reactive flows using the direct simulation Monte Carlo method, Comm. Comput. Phys. 18 (4) (2015) 1095–1121. doi:10.4208/cicp.080115.010515s.

[4] E. J. Parteli, T. Pöschel, Particle-based simulation of powder application in additive manufacturing, Powder Technol. 288 (Supplement C) (2016) 96 – 102. doi:10.1016/j.powtec.2015.10.035.

[5] E. W. C. Lim, C.-H. Wang, A.-B. Yu, Discrete element simulation for pneumatic conveying of granular material, AIChE Journal 52 (2) (2006) 496–509. doi:10.1002/aic.10645.
[6] M. Robinson, M. Ramaioli, S. Luding, Fluid–particle flow simulations using two-way-coupled mesoscale SPH–DEM and validation, Int. J. Multiphase Flow 59 (Supplement C) (2014) 121 – 134. doi:10.1016/j.ijmultiphaseflow.2013.11.003.
[7] D. A. Fedosov, H. Noguchi, G. Gompper, Multiscale modeling of blood flow: from single cells to blood rheology, Biomech. Model. Mechanobiol. 13 (2) (2014) 239–258. doi:10.1007/s10237-013-0497-9.
[8] J. A. M. Kuipers, K. van Duin, F. van Beckum, W. van Swaaij, A numerical model of gas-fluidized beds, Chemical Engineering Science 47 (8) (1992) 1913–1924. doi:10.1016/0009-2509(92)80309-Z.
[9] B. Xu, A. Yu, Numerical simulation of the gas-solid flow in a fluidized bed by combining discrete particle method with computational fluid dynamics, Chem. Eng. Sci. 52 (16) (1997) 2785 – 2809. doi:10.1016/S0009-2509(97)00081-X.
[10] C. Wu, Y. Cheng, Y. Ding, Y. Jin, CFD–DEM simulation of gas–solid reacting flows in fluid catalytic cracking (FCC) process, Chem. Eng. Sci. 65 (1) (2010) 542 – 549, 20th International Symposium in Chemical Reaction Engineering—Green Chemical Reaction Engineering for a Sustainable Future. doi:10.1016/j.ces.2009.06.026.
[11] L. Fries, S. Antonyuk, S. Heinrich, S. Palzer, DEM–CFD modeling of a fluidized bed spray granulator, Chem. Eng. Sci. 66 (11) (2011) 2340–2355. doi:10.1016/j.ces.2011.02.038.
[12] M. Hecht, J. Harting, M. Bier, J. Reinshagen, H. J. Herrmann, Shear viscosity of claylike colloids in computer simulations and experiments, Phys. Rev. E 74 (2006) 021403. doi:10.1103/PhysRevE.74.021403.
[13] C.-C. Huang, R. G. Winkler, G. Sutmann, G. Gompper, Semidilute polymer solutions at equilibrium and under shear flow, Macromolecules 43 (23) (2010) 10107–10116. doi:10.1021/ma101836x.
[14] A. Zöttl, H. Stark, Hydrodynamics determines collective motion and phase behavior of active colloids in quasi-two-dimensional confinement, Phys. Rev. Lett. 112 (2014) 118101. doi:10.1103/PhysRevLett.112.118101.
[15] D. S. Bolintineanu, G. S. Grest, J. B. Lechman, F. Pierce, S. J. Plimpton, P. R. Schunk, Particle dynamics modeling methods for colloid suspensions, Comput. Part. Mech. 1 (3) (2014) 321–356. doi:10.1007/s40571-014-0007-6.
[16] M. P. Allen, D. J. Tildesley, Computer Simulation of Liquids, Oxford Science Publications, Oxford University Press, USA, 1989.
[17] N. G. Deen, M. Van Sint Annaland, M. A. Van der Hoef, J. A. M. Kuipers, Review of discrete particle modeling of fluidized beds, Chem. Eng. Sci. 62 (1–2) (2007) 28–44. doi:10.1016/j.ces.2006.08.014.
[18] R. A. Finkel, J. L. Bentley, Quad trees a data structure for retrieval on composite keys, Acta Inform. 4 (1) (1974) 1–9. doi:10.1007/BF00288933.
[19] D. Meagher, Geometric modeling using octree encoding, Comput. Graph. Image Process. 19 (2) (1982) 129–147. doi:10.1016/0146-664X(82)90104-6.
[20] J. L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (9) (1975) 509–517. doi:10.1145/361002.361007.
[21] K. C. Kannenberg, Computational method for the direct simulation Monte Carlo technique with application to plume impingement, Ph.D. thesis, Cornell University (1998).
[22] S. Subramaniam, D. C. Haworth, A probability density function method for turbulent mixing and combustion on three-dimensional unstructured deforming meshes, Int. J. Engine Res. 1 (2) (2000) 171–190. doi:10.1243/1468087001545128.
[23] J.-S. Wu, Y.-Y. Lian, Parallel three-dimensional direct simulation Monte Carlo method and its applications, Comput. Fluids 32 (8) (2003) 1133–1160. doi:10.1016/S0045-7930(02)00083-X.
[24] A. Haselbacher, F. Najjar, J. Ferry, An efficient and robust particle-localization algorithm for unstructured grids, J. Comput. Phys. 225 (2) (2007) 2198–2213. doi:10.1016/j.jcp.2007.03.018.
[25] S. B. Kuang, A. B. Yu, Z. S. Zou, A new point-locating algorithm under three-dimensional hybrid meshes, Int. J. Multiphase Flow 34 (11) (2008) 1023–1030. doi:10.1016/j.ijmultiphaseflow.2008.06.007.
[26] G. B. Macpherson, N. Nordin, H. G. Weller, Particle tracking in unstructured, arbitrary polyhedral meshes for use in CFD and molecular dynamics, Commun. Numer. Meth. Engng. 25 (3) (2009) 263–273. doi:10.1002/cnm.1128.
[27] G. Martin, E. Loth, D. Lankford, Particle host cell determination in unstructured grids, Comput. Fluids 38 (1) (2009) 101–110. doi:10.1016/j.compfluid.2008.01.005.
[28] B. J. Alder, T. E. Wainwright, Studies in molecular dynamics. I. General method, J. Chem. Phys. 31 (2) (1959) 459–466. doi:10.1063/1.1730376.
[29] A. Donev, S. Torquato, F. H. Stillinger, Neighbor list collision-driven molecular dynamics simulation for nonspherical hard particles. I. Algorithmic details, J. Comput. Phys. 202 (2) (2005) 737–764. doi:10.1016/j.jcp.2004.08.014.
[30] M. N. Bannerman, S. Strobl, A. Formella, T. Pöschel, Stable algorithm for event detection in event-driven particle dynamics, Comput. Part. Mech. 1 (2) (2014) 191–198. doi:10.1007/s40571-014-0021-8.
[31] S. Strobl, M. N. Bannerman, T. Pöschel, Stable algorithm for event detection in event-driven particle dynamics: logical states, Comput. Part. Mech. 3 (3) (2016) 383–388. doi:10.1007/s40571-016-0106-7.
[32] G. A. Bird, Molecular gas dynamics and the direct simulation of gas flows, Clarendon, 1994.
[33] G. Gompper, T. Ihle, D. Kroll, R. Winkler, Multi-particle collision dynamics: A particle-based mesoscale simulation approach to the hydrodynamics of complex fluids, in: C. Holm, K. Kremer (Eds.), Advanced Computer Simulation Approaches for Soft Matter Sciences III, Vol. 221 of Advances in Polymer Science, Springer Berlin Heidelberg, 2009, pp. 1–87. doi:10.1007/978-3-540-87706-6_1.
[34] T. Scanlon, E. Roohi, C. White, M. Darbandi, J. Reese, An open source, parallel DSMC code for rarefied gas flows in arbitrary geometries, Comput. Fluids 39 (10) (2010) 2078–2089. doi:10.1016/j.compfluid.2010.07.014.
[35] S. D. Ramsey, K. Potter, C. Hansen, Ray bilinear patch intersections, J. Graph. Tool. 9 (3) (2004) 41–47. doi:10.1080/10867651.2004.10504896.
[36] R. Wilmoth, A. Carlson, G. LeBeau, Fluid Dynamics and Co-located Conferences, American Institute of Aeronautics and Astronautics, 1996, Ch. DSMC grid methodologies for computing low-density, hypersonic flows about reusable launch vehicles. doi:10.2514/6.1996-1812.
[37] D. Gao, C. Zhang, T. E. Schwartzentruber, Particle simulations of planetary probe flows employing automated mesh refinement, Journal of

Spacecraft and Rockets 48 (3) (2011) 397–405. doi:10.2514/1.52129.
[38] C. M. Hoffmann, Geometric and Solid Modeling: An Introduction, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.
[39] C. Theis, K. H. Buchegger, M. Brugger, D. Forkel-Wirth, S. Roesler, H. Vincke, Interactive three-dimensional visualization and creation of geometries for Monte Carlo calculations, Nucl. Instrum. Methods Phys. Res., Sect. A 562 (2) (2006) 827–829. doi:10.1016/j.nima.2006.02.125.
[40] S. Cameron, Efficient bounds in constructive solid geometry, IEEE Comput. Graphics Appl. 11 (3) (1991) 68–74. doi:10.1109/38.79455.
[41] S. Ramanathan, D. L. Koch, An efficient direct simulation Monte Carlo method for low mach number noncontinuum gas flows based on the bhatnagar–gross–krook model, Phys. Fluids 21 (3) (2009) 033103. doi:10.1063/1.3081562.
[42] K. Boomsma, D. Poulikakos, Y. Ventikos, Simulations of flow through open cell metal foams using an idealized periodic cell structure, Int. J. Heat Fluid Flow 24 (6) (2003) 825–834. doi:10.1016/j.ijheatfluidflow.2003.08.002.
[43] O. Smorygo, V. Mikutski, A. Marukovich, A. Ilyushchanka, V. Sadykov, A. Smirnova, An inverted spherical model of an open-cell foam structure, Acta Mater. 59 (7) (2011) 2669–2678. doi:10.1016/j.actamat.2011.01.005.