Large-Scale Multiple-GPU-based DEM Simulation of Polyhedral Particle Systems

Jiayu Xu [a, b], Shuai Zhang [a], Yong Zhang [a], Haolei Zhang [a], Ji Xu [a, b, *], Wei Ge [a, b, *]

[a] State Key Laboratory of Mesoscience and Engineering, Institute of Process Engineering (IPE), Chinese Academy of Sciences (CAS), Beijing 100190, China

[b] School of Chemical Engineering, University of Chinese Academy of Sciences, Beijing 101408, China

Abstract

Polyhedral particles are ubiquitous in natural and industrial processes. Recent advances in GPU computing have greatly enhanced the feasibility of discrete element method (DEM) simulations for polyhedral particles, yet accurately simulating their collective behavior remains computationally intensive for large-scale simulations. In this study, a distributed parallel DEM simulation framework with multiple-GPU computing for polyhedral particles is developed to achieve high-performance large-scale simulations. The framework integrates the Message Passing Interface (MPI) with NVIDIA's Compute Unified Device Architecture (CUDA), in which the main compute pipeline, including domain decomposition, neighbor-list construction, contact search, and evaluation of contact is executed on GPUs. The proposed method is validated through both numerical and experimental studies. Numerical stability is verified through simulations of particle–wall impact and wall force evaluation at different mesh resolutions. The simulated static packing structures and velocity fields in a quasi-two-dimensional rotating drum show good agreement with experimental measurements. A scalability test involving 2×10^7 polyhedral particles on 16 GPUs demonstrates excellent parallel performance, achieving a 14.8 times speedup compared with the single-GPU case. Moreover, the large-scale applications, e.g., the silo deposition and fixed bed containing cylindrical catalyst particles, further demonstrate the capability of the proposed framework for industrial-scale applications.

Keywords: polyhedral particle, parallel computing, GPU, discrete element method

1 Introduction

Granular materials are ubiquitous in natural and industrial processes, including mining [1], chemical engineering [2], pharmaceuticals [3], and grain processing [4]. In contrast to continuum materials, granular materials can exhibit solid-like, liquid-like, and gas-like macroscopic states under external loading and can transition between these states [5]. Such transitions are accompanied by complex microscale phenomena, arching [6], shear thickening [7], and clogging [8], which make the prediction and control of granular flows a challenging research problem. To quantify the dynamics of granular matter, Cundall and Strack [9] introduced the discrete element method (DEM), which models particle-scale motion and interactions directly. Since then, DEM has been progressively applied to the prediction and optimization of granular material systems [10–12].

However, conventional DEM usually models granular materials as ideal spherical particles. This treatment is algorithmically simple and efficient but fails to capture the non-spherical features commonly observed in real grains [13]. Angularity and shape-induced force chains can significantly change stress distributions and flow patterns, while particle corners and interfacial friction often promote interlocking that suppresses rolling and changes the macroscopic response [14]. To better represent these effects, various shape methods have been developed within the DEM framework, such as multi-sphere approximations [15], level-set methods [16], superquadrics [17], and non-uniform rational B-splines (NURBS) [18] etc. These approaches improve the geometric fidelity to some extent, yet still struggle to reproduce sharp edges and corners, which limits the accuracy of predicted granular dynamics. In contrast, polyhedral representations mitigate this deficiency and offer greater fidelity in reproducing the behavior of real granular materials [19].

Industrial processes often involve billions or even more particles, and thus updating particle states within practical time remains a primary bottleneck for DEM applications. For large-scale simulations, the parallel-computing-based frameworks have emerged, e.g., with the Message Passing Interface (MPI), software packages such as LIGGGHTS [20], Mercury [21], ExaDEM [22], and MFIX [23] can simulate systems with millions of particles on CPU clusters. Although these approaches expand the computational scale, contact search and overlap evaluation remain the dominant costs for non-spherical particles of complex geometry. In superquadric methods, for example, resolving particle–particle and particle–wall contacts typically requires multiple iterations to reach a stable solution [24], which limits overall efficiency. Therefore, more effective parallel strategies are therefore required to overcome the performance limits of CPU-based DEM.

With the rapid development of the graphics processing unit (GPU), GPU-based parallel computing has become a high-performance alternative to conventional CPU architectures [25]. The combination of massive thread-level parallelism and high memory bandwidth provides a marked advantage for repeated contact search and force evaluation in particulate DEM. Over the past decade, GPU-based parallelism has been progressively incorporated into DEM, greatly increasing the simulation scale and efficiency [26]. Xu et al. [27] developed the DEMms software in the Mole-8.5 system with the CPU and GPU heterogeneous computing, enabling near real-time simulation of industrial-scale drum and supporting subsequent applications in the optimization and design of commercial-scale equipment [28,29]. Subsequently, several GPU-accelerated open-source frameworks, including PhasicFlow [30] and HOOMD-blue [31], have further demonstrated the feasibility of large-scale DEM simulations on GPUs. For non-spherical particles, Ji et al. [32] proposed SDEM, which implements GPU-based simulations of superquadric particles. Govender et al. [33,34] implemented contact detection for polyhedral particles within the BlazeDEM framework, establishing a foundation for industrial-scale non-spherical DEM simulations. Liu et al. [35] introduced the CoSim-DEM software, which incorporates Feng's energy-conserving contact theory [36] to achieve accurate polyhedral geometry evaluation and high-performance simulations of screw discharging on a single GPU. Commercial packages such as Rocky DEM and EDEM have also added GPU-accelerated modules for industrial applications.

However, most GPU-based DEM approaches for polyhedral particles rely on a single GPU or a single node, which limits scalability at the engineering scale and prevents the full use of multiple-GPU clusters. Beyond this limitation, the geometric complexity of polyhedral particles further complicates contact resolution, which involves convex-hull intersection, overlap-volume evaluation, and normal extraction. As a result, efficient solution workflows and unified parallel implementations are rarely described in the literature.

To address these limitations, in this work, a distributed polyhedral DEM framework for CPU–GPU heterogeneous clusters is established, which integrates the MPI with NVIDIA's Compute Unified Device Architecture (CUDA) [37]. Key computational strategies including domain decomposition, neighbor-list construction, contact search, and contact evaluation are executed on GPUs, while CPUs handle the inter-process communication, lightweight scheduling, and input–output operations. The geometric characteristics of polyhedral contacts are described in detail, including particle–particle intersections and particle–triangulated facet interactions. The proposed framework is subsequently verified, and its efficiency is evaluated across cases of different scales.

2 Methods

The overall large-scale multiple-GPU-based DEM framework for polyhedral particles is shown in Fig. 1. On the CPU side, parameter reading, polyhedron topology parsing, initial domain decomposition, and I/O are handled, while all remaining compute-intensive tasks are executed on GPUs. At each time step, the polyhedral particle data are exchanged between GPUs to synchronize boundary regions of each domain to enable parallel computing. Each GPU subsequently performs spatial binning and neighbor-list construction [38,39]. To achieve finer-grained parallelism, a one-dimensional array of candidate contact pairs is employed. Contact detection proceeds in two stages. In the broad phase, candidate contacting particle pairs are first filtered using bounding spheres. In the narrow phase, following the Gilbert–Johnson–Keerthi (GJK) evaluation, polyhedron–polyhedron contacts are resolved in the dual space using an incremental Quickhull algorithm that constructs the overlap region and evaluates its volume. Polyhedron–wall contacts on triangulated surfaces are computed using the Sutherland–Hodgman clipping method to obtain overlap areas. Based on these geometric quantities, contact forces and torques are accumulated on GPUs, and particle states are subsequently updated. The dominant computational costs arise from the inter-process communication, neighbor-list construction, and geometric contact computations, which are described in the following subsections.

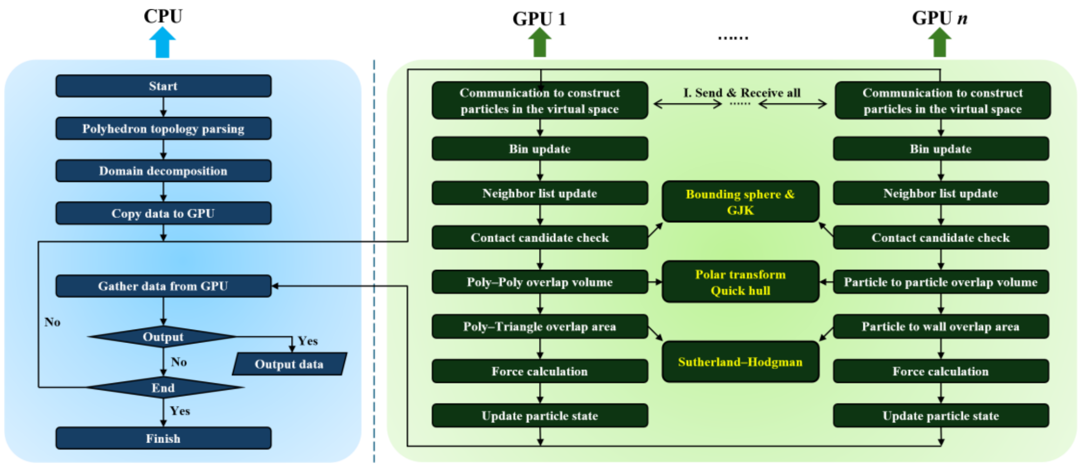
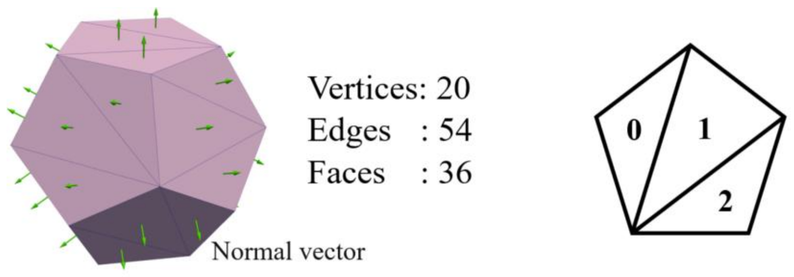


Fig. 1 Flowchart of the multiple-GPU-based polyhedral DEM.

2.1 Representation of convex polyhedral particles on GPUs

Precisely representing a convex polyhedral particle requires storing its vertex coordinates together with the incidence relations that define faces. However, such a complete representation is memory intensive. In parallel computation, storing the full geometric representation of each polyhedral particle and exchanging it across processes significantly increases communication overhead. To address this issue, the strategy proposed by Govender et al. [33,40,41] is adopted. NVIDIA GPU’s constant memory is used to speed up loading of the particle’s geometric representation, and only the triangular facets forming the convex polyhedron are stored, together with their vertex indices and outward normals, which serve as the template of polyhedral particles. During computation, each thread batch-loads these template datasets to reconstruct the particle geometry through rotation and scaling. For example, in Fig. 2 (a), the dodecahedron particle template consists of 20 vertices, 54 edges, and 36 faces, all of which are stored in constant memory.



(a) Dodecahedron particle representation (b) One face of the dodecahedron particle

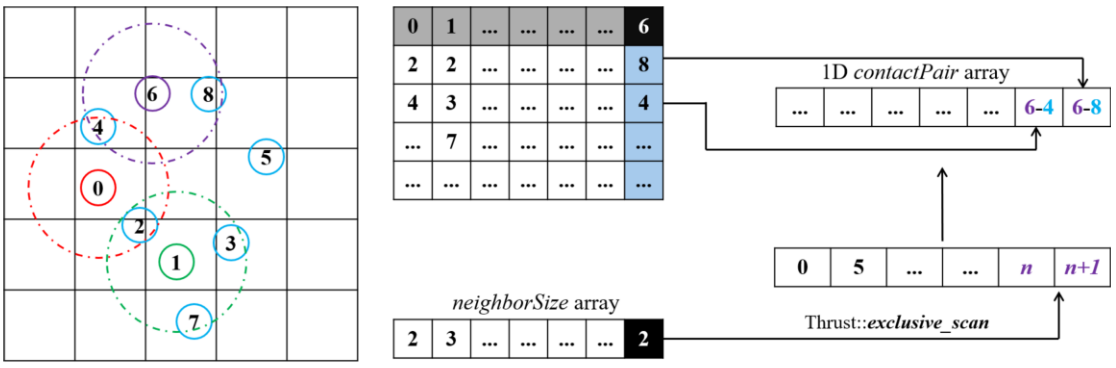
Fig. 2 Representation of a convex polyhedral particle, e.g., the dodecahedron particle.

During the computation, the complete description of all faces is unnecessary. On the GPU, the contact detection and overlap evaluation require only the local geometric data, rather than the full topological details of each facet. Consequently, in preprocessing, adjacent triangular facets sharing a common outward normal are merged to reduce storage redundancy. For instance, as shown in Fig. 2 (b), facets 0, 1, and 2 belong to the same pentagonal face and share a common outward normal, which means that they can be stored as a single entity. The complete face information is reconstructed only for post-processing and visualization.

2.2 Neighbor search and contact-pair extraction of polyhedral particles

In DEM simulations, neighbor search serves as the preliminary step in contact resolution. To facilitate it, the computational domain is divided into uniform spatial bins, as shown in Fig. 3 (a), so that each particle searches the nearby bins within the cutoff distance to reduce the computational complexity [38,39]. The cutoff radius is defined as k times (typically > 1.0) the diameter of the largest circumscribed sphere to ensure complete coverage of potential neighbors.

In previous implementation, as illustrated in Fig. 3 (b), the neighbor-list is stored in a 2D array to achieve coalesced memory access and improve cache locality on GPUs [42]. During neighbor-list construction, the number of neighbors for each particle is recorded in the *neighborSize* array. However, the 2D neighbor list must conform to the maximum neighbor count, which leads to sparse storage and lower memory efficiency. Furthermore, uneven neighbor counts lead to large differences in execution time across threads, when one thread processes all the neighbors of one particle in the subsequent particle contact computation.



(a) Binning the particles (b) 2D neighbor-list (c) Contact-pair parallelism

Fig. 3 Neighbor-list construction.

To address these issues, a contact-pair parallel strategy is introduced as illustrated in Fig. 3 (c). The 2D neighbor-list is replaced with a 1D *contactPair* array. The total number of candidate contact pairs is obtained through a parallel reduction on the GPU using the Thrust library [43]. A contiguous buffer is needed and each element is stored as type *int2*, where the two components are the indices of the two particles in the contact pair. An exclusive prefix sum is subsequently performed on the *neighborSize* array to determine the starting storage position of each particle in the *contactPair* array. Taking particle 6 as an example, its storage offset is *n*, and the two contact pairs are stored at *n* and *n*+1. During following computation of the interaction of particles, the parallelism strategy is assigning an independent GPU thread to handle each contact pair, which will avoid load imbalance and thread waiting caused by unequal neighbor counts.

To reduce the computational cost, the contact detection for convex polyhedra follows a two-stage scheme. First, in the broad phase, a lightweight bounding-sphere test efficiently removes distant pairs before detailed geometry checks. As illustrated in Fig. 4, each polyhedral particle is enclosed within a circumscribed sphere whose radius equals the maximum distance from the centroid to any vertex. Potential contact pairs are generated when the centroid distance between two particles is smaller than the sum of their bounding-sphere radii. The resulting candidates are passed to the narrow phase for precise intersection evaluation.

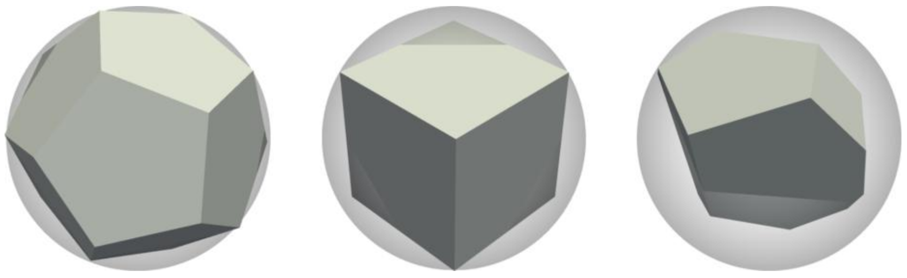


Fig. 4 Collision detection using bounding spheres in the broad phase.

Second, in the narrow phase, the GJK algorithm [44–46] is adopted for convex polyhedra. For two convex polyhedra A and B, the Minkowski difference is defined as:

$$\mathrm{A} \ominus \mathrm{B} = \{ a - b \mid a \in \mathrm{A}, b \in \mathrm{B} \}. \tag{1}$$

Here, *a* and *b* denote points inside polyhedra A and B, respectively. Intersection occurs only if the origin belongs to $\mathrm{A} \ominus \mathrm{B}$. The GJK algorithm does not explicitly construct the Minkowski difference, but approximates its boundary iteratively through the support function. The corresponding support points are defined as:

$$support(\mathrm{A}, \boldsymbol{dir}) = max(a \cdot \boldsymbol{dir}), \tag{2}$$

$$s = support(\mathrm{A}, \boldsymbol{dir}) - support(\mathrm{B}, -\boldsymbol{dir}), \tag{3}$$

where $\boldsymbol{dir}$ denotes the search direction. The overall GJK approach is shown in **Algorithm I.**

Algorithm I: GJK detection

```
1:   $\boldsymbol{dir} \leftarrow (x_\mathrm{B} - x_\mathrm{A})$                      # Initial search direction
2:   $simplex \leftarrow \varnothing$                              # Initialize empty simplex
3:   $s \leftarrow$ support (A, dir) – support (B, -dir)          # support points, Eq. 3
4:   $\boldsymbol{dir} \leftarrow -s$                                  # Search direction toward origin
5:   while True do
6:       $s \leftarrow$ support (A, dir) – support (B, -dir)      # New support point, Eq. 3
7:       if dot (s, dir) $\leq 0$ then                           # Check separation
8:               return False
9:           end if
10:          simplex.add (s)
11:      inside, dir ← UpdateSimplex (simplex)                 # Update simplex & direction
12:      if inside is True then
13:              return True                                   # Intersection detected
14:          end if
1:   end while
```

At the initial stage, the centroid-to-centroid vector defines the initial search direction ***dir***, where x_B and x_A are the centroids of particles B and A. The *simplex* is then grown incrementally from successive support points. The function ***add***(*s*) inserts the new support point into the current simplex, which represents the portion of the Minkowski difference explored up to the current iteration. The function ***UpdateSimplex***(*simplex*) selects the *simplex* feature (vertex, edge, face) nearest to the origin and updates the *simplex* and search direction. To minimize memory transfers, vertex data of each polyhedron are preloaded from constant memory on the GPU during execution. The algorithm typically converges within a few iterations, and the small variation in iteration counts among threads has negligible impact on the load balance.

2.3 Overlap volume between two convex polyhedral particles

For two convex polyhedral particles, the overlap volume is obtained from the convex hull of the intersection-defining points, as illustrated in Fig. 5. Since convexity is preserved under intersection, the intersection of two convex polyhedra is still convex [47]. However, directly constructing the intersection polyhedron in the primal space can be computationally demanding. Therefore, to obtain an exact solution for the intersection volume in an efficient and numerically robust manner, the problem is transformed into the dual space [34,35]. Within the parallel computing framework, only the contact pairs identified during the GJK stage are passed to compute the overlap volume, thereby minimizing thread divergence in GPU computing. Throughout this process, one thread is assigned to deal with one contact pair, achieving parallel computing on GPUs.

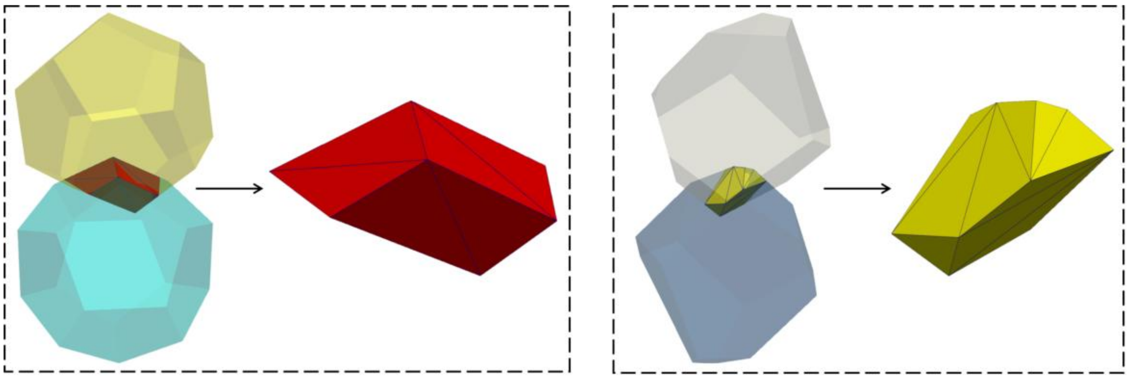

(a) Overlap between dodecahedron particles (b) Overlap between convex polyhedral particles

Fig. 5 Overlap volume of two convex polyhedral particles.

Each triangular face of the two polyhedra is polar-mapped with respect to a reference point x_{o}. For a face with unit outward normal vector $\boldsymbol{n}_{\mathrm{f}}$ and a representative point p, the signed offset d is defined as the oriented distance from x_{o} to the supporting plane. With this convention:

$$d = -\boldsymbol{n}_{\mathrm{f}} \cdot (p - x_{\mathrm{o}}), \tag{4}$$

the corresponding dual point is:

$$y = \frac{\boldsymbol{n}_{\mathrm{f}}}{-d}. \tag{5}$$

The resulting set of dual points y is stored in array *dualPoints* with its size tracked by the counter *dualCount*. The overall polar transformation procedure is shown in **Algorithm II**. These dual points are subsequently used to construct the convex hull and obtain the dual representation of the intersection polyhedron.

Algorithm II: Polar transformation

```
1:  dualCount ← 0
2:  for face in (polyA ∪ polyB) do                         # Iterate all faces
3:      $n_f$ ← face.normal ; p ← face.centroid              # Face normal & centroid
4:      d ← − dot ($n_f$, (p − $x_o$))                        # Signed offset, Eq. 4
5:      dualPoints[dualCount] ← $n_f$ / (−d)                  # Polar point, Eq. 5
6:      dualCount ← dualCount + 1
7:  end for
```

At the convex hull stage, an outward-facing initial tetrahedron is generated as the seed structure. In each iteration, active dual points are assigned to their farthest visible faces, and each face storing the index of its farthest external point in *farthestPoint*. This process identifies the candidate expansion vertices. One of these vertices is then selected as the *apex*, after which the corresponding visible faces are deleted, the separating horizon is extracted, and new faces including the *apex* are created while maintaining a consistent outward orientation. This process repeats until no face is associated with an external point, at which point the convex hull in the dual space is complete. The overall procedure is summarized in **Algorithm III**.

Algorithm III: ConvexHull construction

```
1:  initSimplex()                                  # Construct initial tetrahedron hull
2:  while ∃ face with farthestPoint ≥ 0 do         # Iterative expansion loop
3:      call Algorithm IV                          # Assign active points to faces
4:      call Algorithm V                           # Expand hull from selected apex
6:  end while                                      # Terminate when no active point lies outside
```

For each face, the maximum distance is initialized to negative infinity. Then, all active dual points are traversed, and their signed distances to the face are computed. If a point lies outside the face and its distance exceeds the current record, both the farthest distance and the associated point index are updated. As a result, each face is linked to a candidate *apex* point, which provides the input for the subsequent convex hull expansion.

Algorithm IV: **A**ssign points to faces

```
1:  for f ∈ [0, faceCount) do
2:      farthestDist[f] ← -∞                                # Initialize max distance
3:      farthestPoint[f] ← −1                               # Initialize with no farthest point
4:  end for
5:  for p ∈ [0, numPoints) do
6:      if isActive[p] = false then continue                # Skip inactive points
7:      for f ∈ [0, faceCount) do
8:          plane ← facePlanes[f]                           # Fetch plane coefficients
9:          dist ← dot (plane.xyz, dualPoints[p]) + plane.w # Signed distance
10:         if dist > farthestDist[f] then
11:             farthestDist[f] ← dist                      # Update farthest distance
12:             farthestPoint[f] ← p                        # Record point index
13:         end if
14:     end for
15: end for
```

Here, *facePlanes*[*f*] denotes the plane equation of face *f*, expressed as (n_x, n_y, n_z, w). *isActive*[*p*] indicates whether a dual point is still available for the hull expansion. For each face, *farthestDist*[*f*] records the maximum distance among all currently active points. Since individual GPU threads cannot efficiently allocate dynamic memory, and such operations often lead to severe memory fragmentation, an incremental Quickhull [48] is adopted, in which the convex hull is iteratively constructed by adding one point at a time to the existing hull and updating the visible facets and conflict lists. This strategy enables parallel and memory-efficient execution, and the overall processes is summarized in **Algorithm V**.

Algorithm V: expandOnce

```
1:  $baseFace$ ← first $f$ with $farthestPoint[f] \geq 0$
2:  $apexIdx$ ← $farthestPoint[baseFace]$                          # Select apex
3:  $apex$ ← $dualPoints[apexIdx]$
4:  for $f \in [0, faceCount)$ do
5:      $visible[f]$ ← (sgnVolume ($apex$, $tri[f]$) > 0)          # Mark visible faces
6:  end for
7:  $horizon$ ← {$e \in$ visible ∧ reversed ($e$) $\in$ non-visible}   # Collect horizon edges
8:  removeAllVisibleFaces ()                                        # Retain non-visible structure
9:  for $e$ in $horizon$ do                                         # Iterate horizon
10:     $plane$ ← makePlane ($apex$, $v_1$, $v_0$)                  # Construct candidate plane
11: end for
12: if dot ($plane.xyz$, $simplexCenter$) + $plane.w$ > 0 then
13:     swap ($v_0$, $v_1$)                                         # Enforce outward normal
14: end if
15: appendNewFace ($apexIdx$, $v_1$, $v_0$)                         # Commit new dualTriangles
16: compactIndices ()                                               # Tighten index sets
17: $triCount$ ← $faceCount$
```

Here, the *visible* array indicates whether face *f* is visible from the current *apex*, while the *horizon* array collects the separating edges (with endpoints v_0 and v_1) between visible and non-visible regions. For each such edge, a candidate plane is constructed with the *apex*, and its orientation is corrected using the reference point *simplexCenter* to enforce consistent outward normals. The counters *faceCount* and *triCount* maintain the current numbers of faces and triangles. The incremental Quickhull expansion process is illustrated in Fig. 6, where the polytope is expanded by introducing a new *apex* and updating its combinatorial structure.

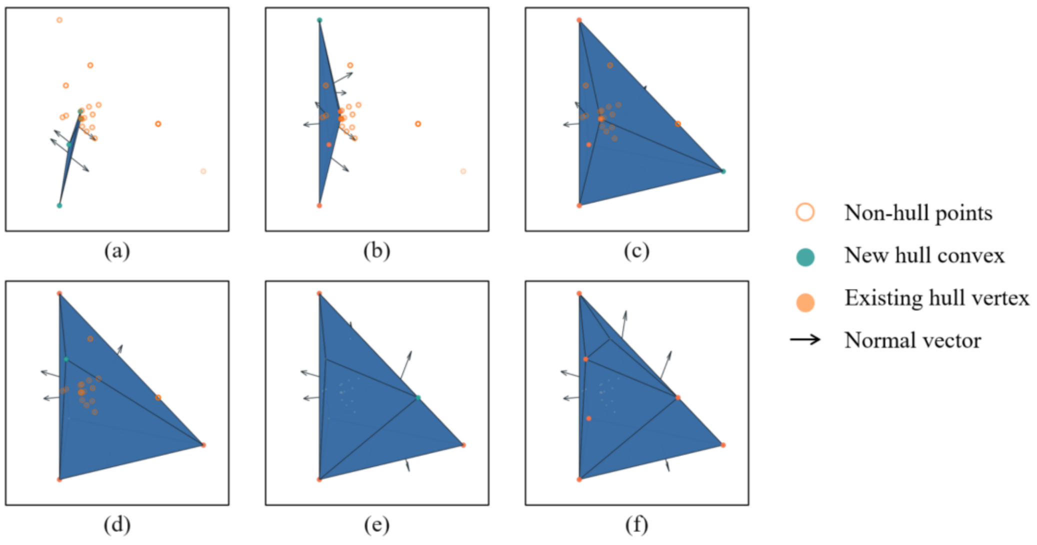


Fig. 6 Dual-space incremental Quickhull approach for convex polyhedra intersection.

A candidate base face is chosen, and the *farthestPoint* provides the index of *apex* (*apexIdx*). The corresponding *apex* is then marked as inactive to prevent reuse in subsequent iterations. Visible faces are detected via the signed volume test function ***sgnVolume***, and the separating horizon is determined. All visible faces are then removed using function ***removeAllVisibleFaces***, leaving only the stable non-visible subset. For each ridge edge on the *horizon*, a candidate plane is constructed. New dual-triangle faces are then created by ***appendNewFace***, each storing three vertex indices of the dual space. Finally, the function ***compactIndices*** reindexes the adjacency arrays to restores a consistent, compact layout, while the assignment *triCount*←*faceCount* enforcing alignment between face and triangle counts.

After the dual convex hull has been constructed, the intersection polyhedron is obtained by unpolarizing each dual face into a vertex. For a dual triangle $t = (i, j, k)$ with vertices $a = dualPoints[t.x]$, $b = dualPoints[t.y]$, and $c = dualPoints[t.z]$ in the dual space, the outward normal is computed as:

$$\hat{\boldsymbol{n}}_{\mathrm{f}} = \boldsymbol{unit}\big((b-a)\times(c-a)\big), \tag{6}$$

the signed offset:

$$d = \boldsymbol{min}(-\hat{\boldsymbol{n}}_{\mathrm{f}}\cdot a, -\hat{\boldsymbol{n}}_{\mathrm{f}}\cdot b, -\hat{\boldsymbol{n}}_{\mathrm{f}}\cdot c)\,, \tag{7}$$

and the corresponding primal-space vertex is then reconstructed by:

$$p = x_0 + \frac{\hat{\boldsymbol{n}}_{\mathrm{f}}}{-d}. \tag{8}$$

The overall procedure is summarized in **Algorithm VI**.

Algorithm VI: Unpolarization

```
1:  for $t = (i, j, k)$ in dualTris do                                   # Iterate over dual faces
2:      $a \leftarrow dualPoints[t.x]$, $b \leftarrow dualPoints[t.y]$, $c \leftarrow dualPoints[t.z]$
3:      $\hat{\boldsymbol{n}}_{\mathrm{f}} \leftarrow$ normalize $((b - a) \times (c - a))$          # Outward unit normal, Eq. 6
4:      $d \leftarrow$ min $(-\hat{\boldsymbol{n}}_{\mathrm{f}} \cdot a, -\hat{\boldsymbol{n}}_{\mathrm{f}} \cdot b, -\hat{\boldsymbol{n}}_{\mathrm{f}} \cdot c)$            # Signed offset
5:      $p \leftarrow \boldsymbol{x}_0 + \hat{\boldsymbol{n}}_{\mathrm{f}}/(-d)$                          # Recovered primal-space vertex
6:  end for
```

The unpolarized vertices are connected to form a triangular mesh with consistent normal vectors, defining the topology of the intersection polyhedron. Using the arithmetic mean of all vertices as the interior reference point o, each oriented face (v_0, v_1, v_2) forms an oriented tetrahedron and the signed overlap volume V is given by:

$$V = \frac{1}{6} \times \sum_{f} (v_0 - o) \cdot \big((v_1 - o) \times (v_2 - o)\big). \tag{9}$$

Convex-hull computation frequently requires temporary buffers for faces, edges, and vertices, whose numbers vary during the expansion process. In GPU implementations, dynamic allocations incur high overhead and lead to memory fragmentation. To address these issues, each hull is assigned a preallocated workspace sized to the estimated maximum number of faces, edges, and vertices. Storage positions in the memory are managed by counters like *triCount*, *faceCount*, and *dualCount*. All temporary geometric elements are stored sequentially to avoid dynamic resizing.

Computation of a typical convex hull between two polyhedra with approximately 48 faces 64 edges, and 32 vertices requires about twenty kilobytes of global memory in the most conservative case. Due to limited GPU global memory, it is not feasible to process all contact pairs simultaneously. The computation of the overlap volume is partitioned into batches according to the GPU's computing capability, namely the *contactPair* array is divided into equal-sized small arrays that are executed by the GPU kernel one after another. This batch execution strategy provides predictable memory usage and maintains high GPU utilization.

2.4 Overlap area between convex particle and triangle wall

For complex geometries, walls are typically modeled as triangular-facet meshes in DEM simulations, using standard formats such as STL and OBJ. In practice, particle–facet interactions can suffer from duplicate contact evaluations, which may result in nonphysical behaviors such as excessive adhesion or unrealistic rebounds during particle–wall interactions. To alleviate this issue, several studies have proposed filtering strategies that selectively exclude redundant contact facets [32,49], thereby yielding more reasonable force distributions. Building upon this, the area-weighted contact force approach [35,50] has been introduced, in which the overlap area between a triangular facet and a sphere or polyhedron is explicitly computed and used to weight the contact force.

Contact detection between a convex polyhedron and a triangular facet begins with identifying their overlap region. Since the intersection is restricted to the facet plane, it can be equivalently formulated as a two-dimensional polygon clipping operation. The Sutherland–Hodgman clipping algorithm is adopted to compute the overlap area between the polyhedron and the triangular facet. Specifically, the triangular facet is treated as the clipping polygon, while the intersection of the polyhedron with the plane is projected to obtain the subject polygon. Both polygons are represented in the same plane. Edges of the clipping polygon are processed sequentially, and in/out tests update the output vertex list. After all oriented edges have been processed, the resulting polygon corresponds to the overlap region between the polyhedron and the triangular facet. The complete procedure is summarized in **Algorithm VII**.

Algorithm VII: Sutherland–Hodgman

```
1:  $P \leftarrow$ projected polyhedron                                  # Subject polygon in projection plane
2:  $clip \leftarrow$ triangleWall                                      # Clip polygon
3:  for $(v_0, v_1) \in edges(clip)$ do                                  # Iterate oriented edges of clip polygon
4:      $edgeDir \leftarrow v_1 - v_0$
5:      $v_2 \leftarrow$ last $(P)$, $Q \leftarrow \varnothing$
6:      for $v_3 \in P$ do
7:          $inA \leftarrow$ cross2D $(edgeDir, v_3 - v_0) \le 0$            # Inside/outisde tests, right-hand side
8:          $inB \leftarrow$ cross2D $(edgeDir, v_2 - v_0) \le 0$            treated as inside
9:          if $inA \wedge inB$ then
                                                                             # inside → inside
10:             append $(Q, v_3)$
```

```
11:             end if
12:             if $inA \wedge inB$ then
13:                 append (Q, $I(v_2, v_3)$)          # outside → inside
14:                 append (Q, $v_3$)
15:             end if
16:             if $inA \wedge inB$ then
17:                 append (Q, $I(v_2, v_3)$)          # inside → outside
18:             end if
19:             $v_2 \leftarrow v_3$                    # Update previous vertex
20:         end for                                  # Replace input polygon
21:             P ← Q
22:     end for
```

In **Algorithm VII**, both the subject (*P*) and clipping polygons (*clip*) are defined in the same projection plane and are represented as ordered vertex lists. $\boldsymbol{I}(v_2, v_3)$ denotes the intersection point of segment v_2v_3 with the line through (v_0, v_1), given by:

$$I(v_2, v_3) = v_2 + (v_3 - v_2)t, \tag{10}$$

and

$$t = \frac{cross2D(v_0 - v_2)}{cross2D(v_3 - v_2)}. \tag{11}$$

The edge of *clip* is traversed in order, for each oriented edge (v_0, v_1) with direction $edgeDir = v_1 - v_0$, the half-plane on the right of the edge is treated as inside. Scan once over the vertices of the current subject polygon *P*, using v_2 for the previous vertex and v_3 for the current vertex. Process each clipping edge in order and update Q according to the standard Sutherland–Hodgman in/out rule. After the scan set $P \leftarrow Q$ and continue with the next clipping edge. After all edges have been processed, P equals the polygon that represents the overlap of the triangular facet and the polyhedron in the projection plane. To illustrate the clipping workflow, Fig. 7 visualizes a convex polyhedron intersected by a triangle facet.

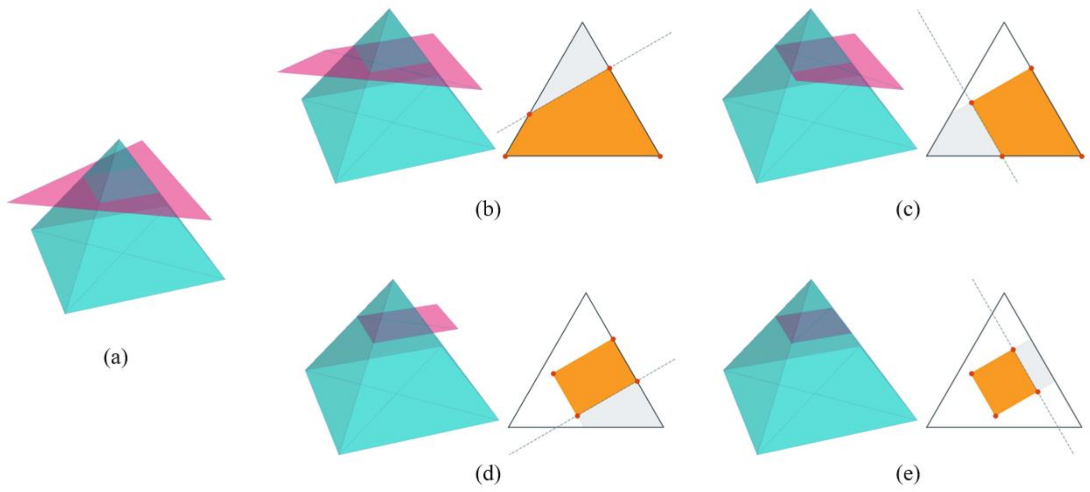


Fig. 7 Polyhedron–plane clipping for wall contact.

In the right column of panels (b) to (e) in Fig. 7, the facet is shown in its projection plane. At each step, the dashed line indicates the current clipping edge, the grey area marks what is discarded by clipping, while the orange polygon shows the retained overlap. After all edges are processed, the final orange polygon represents the overlap region, whose area can be obtained by standard triangulation.

2.5 Contact force calculation for polyhedral particles

Based on the energy conservation principle, Feng et al. [19,51] proposed a unified theoretical framework for contact forces. The core idea is that the normal contact force should originate from a potential energy function, ensuring that the energy remains consistent within the physical system. For polyhedral particles, the contact potential is determined by the overlap volume ΔV and material parameters, leading to the following expression for the elastic normal force:

$$\boldsymbol{F}_{\mathrm{n}}^{\mathrm{e}} = k_{\mathrm{n}} \Delta V^{\frac{1}{3}} \boldsymbol{n}, \tag{12}$$

where $\boldsymbol{n}$ is the unit normal vector, ΔV is the overlap volume, and k_{n} is the normal stiffness [35], given by:

$$k_{\mathrm{n}} = \left(\frac{E_{P_1}}{R_{P_1}} + \frac{E_{P_2}}{R_{P_2}} \right) A_{\mathrm{n}}, \tag{13}$$

with E_{P_1} and E_{P_2} being the Young's moduli of the two contacting particles, R_{P_1} and R_{P_2}

being the equivalent radii defined as the radii of spheres that have the same volume as the polyhedra, and A_n denoting the projected contact area. Since the purely elastic formulation cannot capture the energy dissipation, a viscous damping term is introduced to suppress oscillations and account for energy loss [34]:

$$\boldsymbol{F}_\mathrm{n}^\mathrm{d} = -c_\mathrm{n}(\boldsymbol{v}_\mathrm{r} \cdot \boldsymbol{n})\boldsymbol{n}, \tag{14}$$

where $\boldsymbol{v}_\mathrm{r}$ is the relative velocity at the contact point. The damping coefficient c_n is expressed as:

$$c_\mathrm{n} = \frac{2\ln(\varepsilon)\sqrt{k_\mathrm{n} m_\mathrm{eff}}}{\sqrt{\ln(\varepsilon)^2 + \pi^2}}, \tag{15}$$

where ε is the restitution coefficient and $m_\mathrm{eff} = (m_1^{-1} + m_2^{-1})^{-1}$ denotes the effective mass. The total normal contact force is then expressed as the sum of the elastic and damping contributions:

$$\boldsymbol{F}_\mathrm{n} = \boldsymbol{F}_\mathrm{n}^\mathrm{e} + \boldsymbol{F}_\mathrm{n}^\mathrm{d}. \tag{16}$$

For tangential interactions, the model proposed by Cundall and Strack [34] is employed. The tangential force consists of elastic and viscous parts, which is constrained by the Coulomb friction:

$$\boldsymbol{F}_\mathrm{t} = \boldsymbol{F}_\mathrm{t}^\mathrm{e} + \boldsymbol{F}_\mathrm{t}^\mathrm{d}, |\boldsymbol{F}_\mathrm{t}| \le \mu|\boldsymbol{F}_\mathrm{n}|. \tag{17}$$

The elastic tangential force is defined as:

$$\boldsymbol{F}_\mathrm{t}^\mathrm{e} = -k_\mathrm{t}\Delta\boldsymbol{s}, \tag{18}$$

where $\Delta\boldsymbol{s}$ is the accumulated tangential displacement during contacting and k_t is the tangential stiffness, which is formulated as:

$$k_\mathrm{t} = \left(\frac{E_{P_1}}{2(1+\nu_{P_1})R_{P_1}} + \frac{E_{P_2}}{2(1+\nu_{P_2})R_{P_2}}\right)A_\mathrm{n}, \tag{19}$$

with ν_{P_1} and ν_{P_2} denoting the Poisson's ratios.

The viscous component of the tangential force is expressed as:

$$\boldsymbol{F}_\mathrm{t}^\mathrm{d} = -c_\mathrm{t}\boldsymbol{v}_\mathrm{t}, c_\mathrm{t} = \frac{2\ln(\varepsilon)\sqrt{k_\mathrm{t} m_\mathrm{eff}}}{\sqrt{\ln(\varepsilon)^2 + \pi^2}}, \tag{20}$$

where $\boldsymbol{v}_\mathrm{t}$ is the tangential relative velocity.

In addition to the translational motion, the rotational dynamics of polyhedral particles also play a crucial role. The torque contribution from a contact pair c is calculated as:

$$\boldsymbol{T}_\mathrm{c} = \boldsymbol{r}_\mathrm{c} \times \boldsymbol{F}_\mathrm{c}, \tag{21}$$

where $\boldsymbol{r}_\mathrm{c}$ is the vector from the particle centroid to the contact point and $\boldsymbol{F}_\mathrm{c}$ is the total contact force. The torque acting on a particle is obtained by summing over all contact pairs:

$$\boldsymbol{T} = \sum\nolimits_\mathrm{c} \boldsymbol{T}_\mathrm{c}. \tag{22}$$

The above formulations define the complete contact force model for polyhedral particles. In practice, these computations are executed on GPUs following the contact-pair parallel scheme. Each contact-pair is processed by an independent thread, and the resulting force and torque are accumulated to the two interacting particles through atomic operations provided by CUDA to ensure thread-safe memory updates.

The rotational dynamics are governed by the conservation of the angular momentum:

$$\dot{\boldsymbol{L}} = \boldsymbol{T}, \boldsymbol{L} = \boldsymbol{I} \cdot \boldsymbol{\Omega}, \tag{23}$$

where $\boldsymbol{L}$ is the angular momentum, $\boldsymbol{\Omega}$ is the angular velocity in the global coordinate system, and $\boldsymbol{I}$ is the inertia tensor. For polyhedral particles, the inertia depends on the particle orientation. The principal inertia tensor in the body-fixed coordinate system is $\hat{\boldsymbol{I}} = diag(I_\mathrm{xx}, I_\mathrm{yy}, I_\mathrm{zz})$, and the global inertia tensor is obtained by the rotation matrix $\boldsymbol{\mathcal{H}}$:

$$\boldsymbol{I} = \boldsymbol{\mathcal{H}}\, \hat{\boldsymbol{I}}\, \boldsymbol{\mathcal{H}}^\mathrm{T}. \tag{24}$$

In the body-fixed frame, the rotational equation of motion is:

$$\hat{\boldsymbol{I}}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \left(\hat{\boldsymbol{I}}\boldsymbol{\omega}\right) = \hat{\boldsymbol{T}},$$

where $\boldsymbol{\omega}$ is the angular velocity in the body-fixed system and $\hat{\boldsymbol{T}} = \boldsymbol{\mathcal{H}}^{-1}\boldsymbol{T}$ is the torque in the same frame. Particle orientation is represented by quaternions $\boldsymbol{Q} = (q_0, q_1, q_2, q_3)$, which provide a numerically stable rotation description without the singularities inherent to Euler angles, and the corresponding rotation matrix is:

$$\boldsymbol{\mathcal{H}} = \begin{pmatrix} 1 - 2(q_2^2 + q_3^2) & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & 1 - 2(q_1^2 + q_2^2) \end{pmatrix}. \tag{25}$$

The quaternion components are updated using:

$$\begin{cases} \dot{q}_0 = \left(q_1\omega_\mathrm{x} + q_2\omega_\mathrm{y} + q_3\omega_\mathrm{z}\right) \times \left(-\frac{1}{2}\right) \\ \dot{q}_1 = \left(q_0\omega_\mathrm{x} + q_2\omega_\mathrm{z} - q_3\omega_\mathrm{y}\right) \times \left(+\frac{1}{2}\right) \\ \dot{q}_2 = \left(q_0\omega_\mathrm{y} - q_3\omega_\mathrm{x} - q_1\omega_\mathrm{z}\right) \times \left(+\frac{1}{2}\right) \\ \dot{q}_3 = \left(q_0\omega_\mathrm{z} - q_1\omega_\mathrm{y} - q_2\omega_\mathrm{x}\right) \times \left(+\frac{1}{2}\right) \end{cases}, \tag{26}$$

where $(\omega_x, \omega_y, \omega_z)$ are the angular velocity components in the body-fixed frame. To ensure consistent transient particle state in the virtual space, particle orientations are kept read only within each time step and are updated before the inter-process communication to construct the virtual space particles.

2.6 Inter-process communication

In this study, a domain decomposition strategy based on MPI is implemented to achieve large-scale parallelization across multiple GPUs. The communication strategy is illustrated in Fig. 8. Each computational domain assigned to a GPU is divided into the *Inner Region*, *Outer Region*, and *Virtual Region*. The *Inner Region* contains the particles that are completely managed and updated by the current process. The *Outer Region* serves as a buffer zone separating neighboring *Inner Regions*, ensuring that particle interactions do not directly span processes within a single time step. During each time step, particles located in the *Outer Region* of one process are transmitted to the adjacent process, where they are mapped to the corresponding *Virtual Region*. These virtual particles provide the necessary boundary information for the local contact detection and force evaluation. This exchange ensures that all neighboring particles within the effective interaction range are properly included in the computations near domain boundaries.

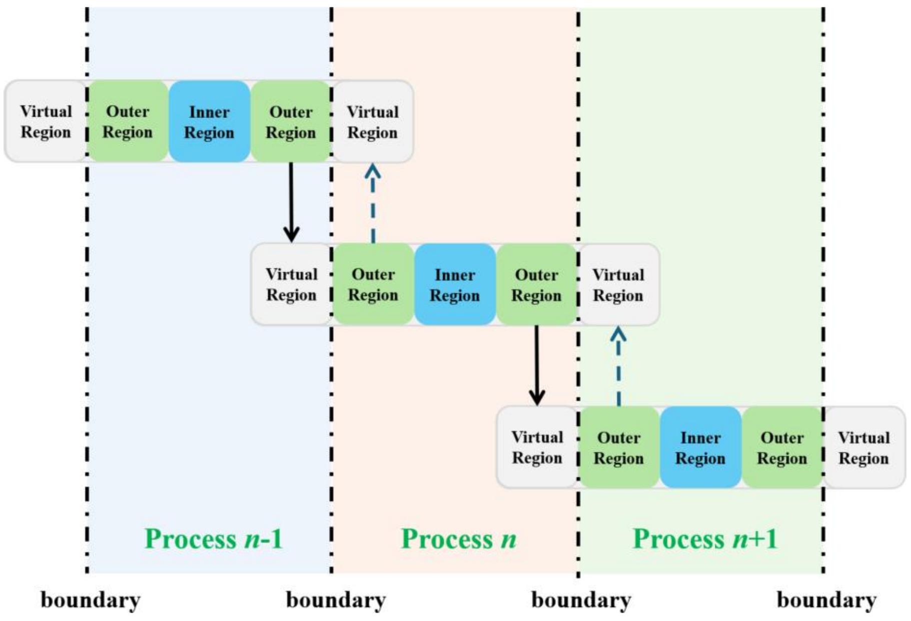


Fig. 8 Domain decomposition and inter-process communication for parallel computing of multiple-GPU-based DEM framework for polyhedral particle system.

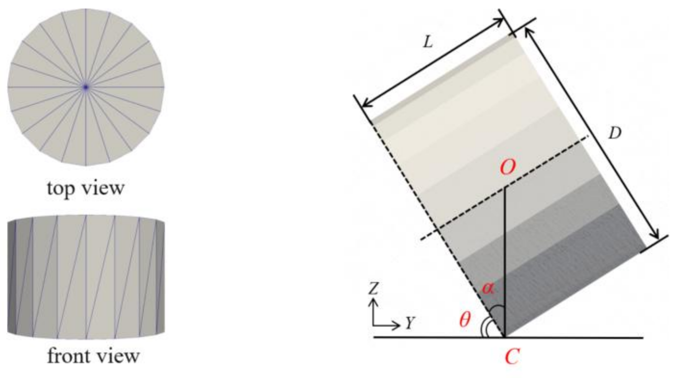
Particle transfer across processes is triggered when a particle centroid crosses a domain boundary. The particle state is then sent to the neighboring process. Communication between GPUs is handled through MPI and only minimal information is exchanged, including the centroid position, velocity, template type, and quaternion. After receiving this information, the neighboring process retrieves the corresponding particle template from constant memory and uses the quaternion to reorient the particle. In this way, the complete particle state is restored without transferring large geometric data.

3 Verification and performance evaluation

All simulations were conducted on the Mole-H supercomputer at the Huairou Center, Institute of Process Engineering, Chinese Academy of Sciences. The compute node is equipped with two Intel Xeon Silver 4410Y processors and eight NVIDIA RTX 4090 GPUs. The operating system is the Linux CentOS 8.

3.1 Cylindrical particle and wall impact

The proposed method is verified by simulating the impact process between a cylindrical particle and a wall and comparing the numerical results with the analytical solution. As shown in Fig. 9 (a), the cylindrical particle is modeled with 42 vertices, 120 edges, and 80 triangular faces. It should be noted that the polyhedral discretization does not perfectly reproduce a smooth cylindrical surface. Thus, the resulting moment of inertia may differ slightly from the values reported in Refs. [17,24]. Following the approach of Kodam et al. [52], the cylindrical particle impacts a wall at a specified orientation angle as shown in Fig. 9 (b).



(a) Cylindrical particle model (b) Cylindrical particle and wall impact

Fig. 9 Scheme of the cylindrical particle and wall impact.

The initial angular velocity is set to zero, and the contact process is assumed to be frictionless and without gravity. According to the analytical model, the post-impact angular and translational velocities can be expressed as:

$$\omega_y^+ = \frac{mV_z^-(1+\varepsilon)rcos(\alpha+\theta)}{I_{yy}+mr^2cos^2(\alpha+\theta)} \tag{27}$$

and

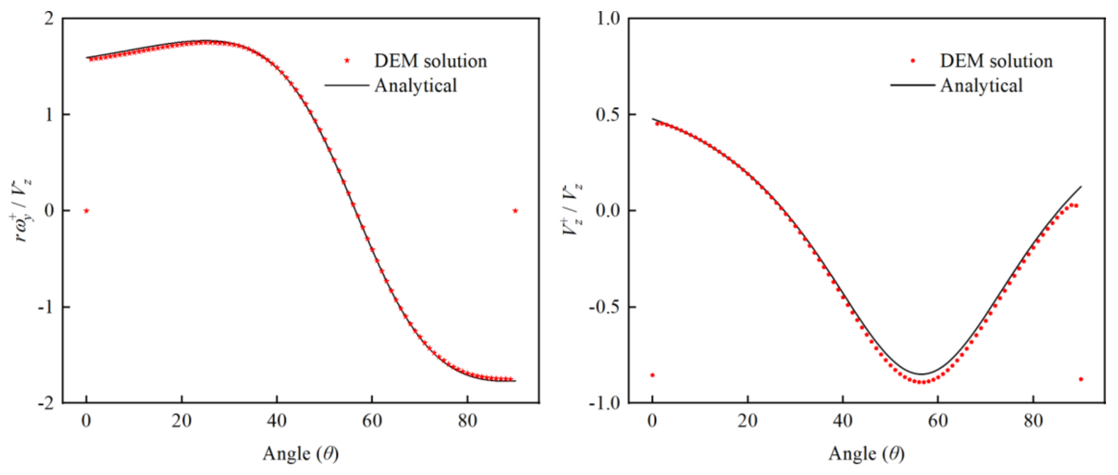
$$V_z^+ = \omega_y^+ rcos(\alpha+\theta) - \varepsilon V_z^-, \tag{28}$$

where *m* is the mass of the cylindrical particle, ε is the coefficient of restitution at the contact point, and V_z^- is the pre-impact translational velocity normal to the wall. The angle α denotes the angle between the cylinder's face and the line connecting the particle center to the contact point, while θ is the angle between the cylinder's face and the wall. I_{yy} represents the moment of inertia about the *y*-axis, and *r* is the distance from the particle center to the corner point *C*, which is assumed to be fixed. The detailed parameters are listed in Table. 1.

Table. 1 Parameters in the cylindrical particle and wall impact simulation

Parameters	**Value**
Diameter, *D* (m)	8×10^{-3}
Length, *L* (m)	5.3×10^{-3}
Volume, *V* (m^3)	2.62×10^{-7}
Density, ρ (kg/m^3)	1245
Moment of inertia, I_{xx} ($kg{\cdot}m^2$)	2.047×10^{-9}
Moment of inertia, I_{yy} ($kg{\cdot}m^2$)	2.047×10^{-9}
Moment of inertia, I_{zz} ($kg{\cdot}m^2$)	2.567×10^{-9}
Shear modulus, *E* (Pa)	1.15×10^{9}
Poisson's ratio, *v* (-)	0.35
Coefficient of friction, μ_s (-)	0.0
Coefficient of restitution, *e* (-)	0.85
Time step, Δt (s)	5×10^{-7}

As shown in Fig. 10, the simulation results are in excellent agreement with the analytical predictions and accurately capture the variations in post-impact angular and translational velocities. Deviations occur at very low and very high impact angles, mainly attributed to geometric approximations and uncertainties in contact point determination [17,24]. Overall, the results demonstrate that the polyhedral DEM can effectively capture the dynamics of collisions between a cylindrical particle and a wall with good accuracy and applicability.



(a) Post-impact angular velocity (b) Post-impact translational velocity

Fig. 10 Comparison of analytical solution and DEM results.

3.2 Effect of the mesh resolution on the particle and wall contact

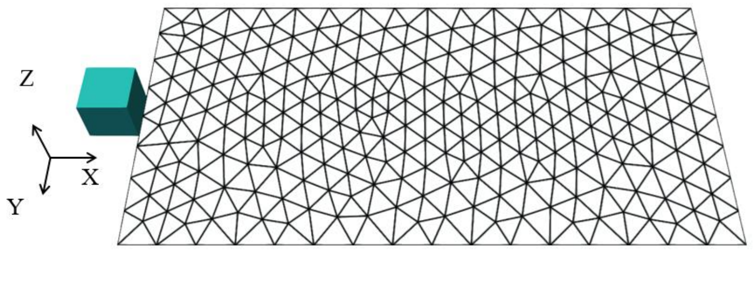
To represent boundaries of real apparatus of complex geometry, walls are often modeled as triangulated surfaces in DEM simulations. To quantify the effect of mesh resolution on the wall contact force calculation, a test of cuboid particle moving on the plane surface is conducted, as shown in Fig. 11 (a). Six STL walls across a range of mesh resolutions were generated, each consisting of a different number of triangular facets, as illustrated in Fig. 11 (b). To characterize the relation between the particle size and mesh resolution, a dimensionless parameter is introduced:

$$\lambda = \frac{L_c}{L_m}, \tag{29}$$

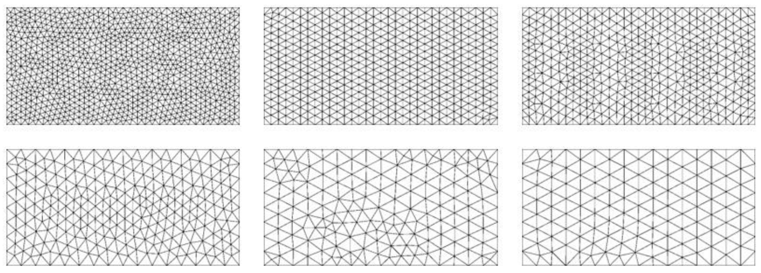
where L_c is the edge length of the cuboid particle and L_m is the average size of the STL mesh elements. A smaller λ indicates a coarser wall mesh relative to the cuboid. During the test, the cuboid particle was placed at the left side of the wall and moved along the *X*-axis at a constant speed until it completely left. The normalized displacement is defined as:

$$\kappa = \frac{\Delta x}{L_c}, \tag{30}$$

where Δx is the cuboid displacement along the *X*-axis. The parameters used in this test are summarized in Table. 2.



(a) Cuboid particle moves on the plane surface



(b) Walls with different mesh resolutions

Fig. 11 Walls with different resolutions and the schematic diagram of the cuboid movement.

Table. 2 Parameters in the particle and wall contact

Parameters	**Value**
Cuboid edge length, *L* (m)	2.5×10^{-2}
Density, *ρ* (kg/m³)	1500
Shear modulus, *E* (Pa)	1×10^{6}
Poisson's ratio, *v* (-)	0.1
Coefficient of friction, μ_s (-)	0.8
Coefficient of restitution, *e* (-)	0.1
Time step, Δt (s)	1×10^{-5}

Fig. 12 presents the contact force acting on the cuboid as a function of κ. The sensitivity to the size ratio λ was evaluated. The force–displacement curves almost perfectly overlap under different mesh resolutions, indicating that the current polyhedral DEM is sufficiently accurate to calculate the interaction force between the convex polyhedral particle and the STL wall.

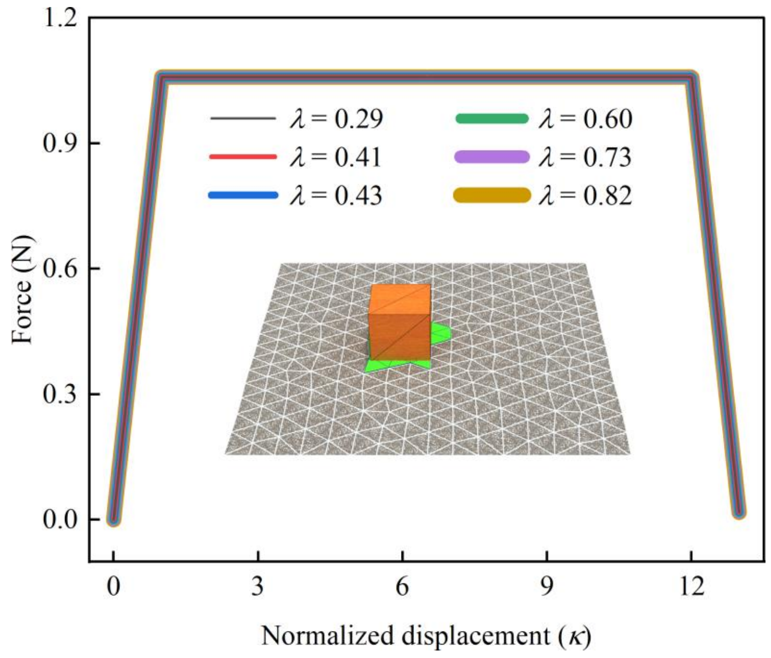


Fig. 12 Cuboid and wall interaction test under different STL mesh resolutions.

3.3 Dodecahedron particle packing

To assess the reliability of the proposed polyhedral DEM in static confined packing, a comparative test with regular dodecahedron particles in a cylindrical container was conducted. As shown in Fig. 13 (a), sixty-three dodecahedron particles were fabricated by 3D printing (BambuLab X1 Carbon) with a layer height of 0.2 mm. The container, as shown in Fig. 13 (b), was a transparent cylinder with an inner diameter of 10 cm and a height of 20 cm. Particles were loaded individually, and the final fill height was recorded after the system reached a stable state. In the simulation, the same number of particles and container geometry were used. Particles were initialized with random positions and orientations above the container and settled under gravity. The simulation parameters were identical to those specified in Section 3.2.

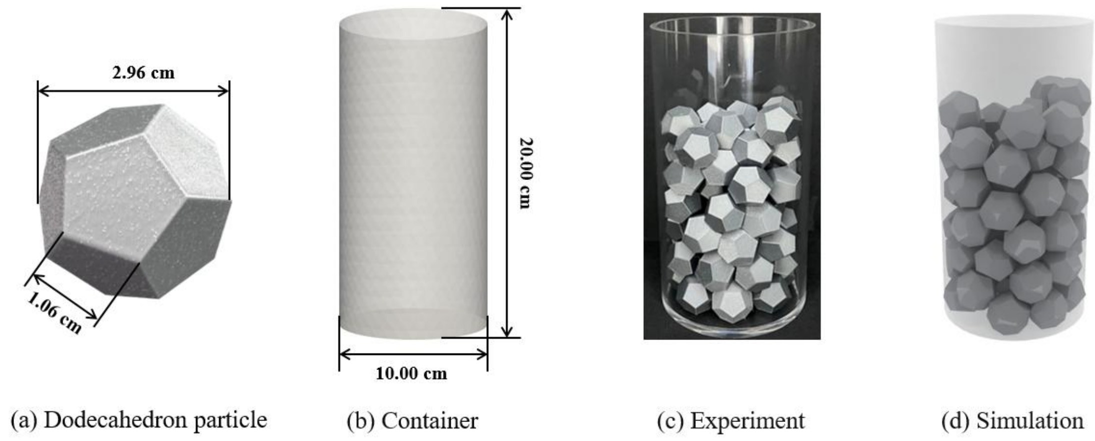


Fig. 13 Static confined packing of dodecahedron particle in a cylindrical container.

The final packing height in both the experiment and the simulation was approximately 16 cm. However, as shown in Fig. 13 (d), the DEM packing exhibited slightly larger interparticle voids than the experiment. This discrepancy may stem from the polyhedral particle model. In the simulation, surface effects are represented by a single friction coefficient. The 3D-printed particles, by contrast, have slightly rounded edges and rough surfaces, which encourage interlocking and result in denser packing. In addition, mild vibrations during loading further promote secondary rearrangements and increase the packing fraction. Despite these differences, the simulated configuration reproduces the main qualitative features observed experimentally, including the heterogeneous void distribution and anisotropic local structure.

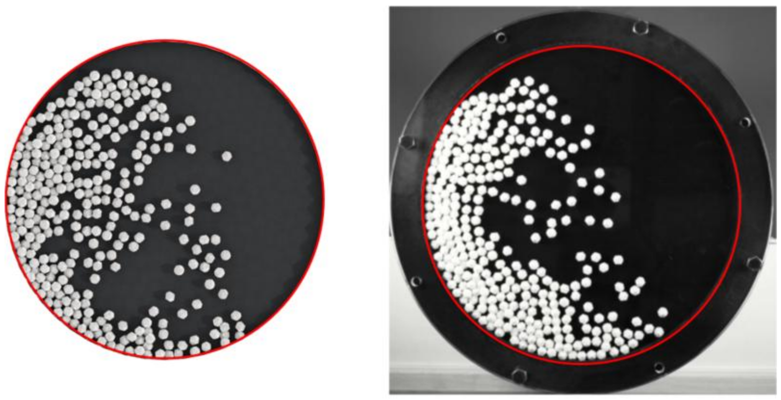
3.4 Quasi-2D rotating drum

A quasi-2D rotating drum setup was employed to assess the agreement between polyhedral DEM simulation and experiment in polyhedral particle motion behavior. The polyhedral particles were regular dodecahedra with a circumscribed-sphere diameter of approximately 1.0 cm, scaled down from the model shown in Fig. 13 (a). The drum thickness was approximately 1.1 cm, providing sufficient clearance for free particle translation and rotation. The high-speed imaging and drum operation parameters are summarized in Table 3, whereas the particle material properties were identical to those listed in Section 3.2, with the friction coefficient adjusted to 0.7 to achieve better agreement with the experimental velocity field.

Table 3 Operating parameters used in the quasi-2D rotating drum

Parameters	**Value**
Drum diameter, D (m)	3×10^{-1}
Drum thickness, L (m)	1.1×10^{-2}
Rotate speed, φ (rpm)	45
Sampling interval, t_s (s)	5×10^{-4}
Imaging resolution, (-)	960 × 960

Fig. 14 shows the results from simulation and experiment. The lifting and cataracting progress, as well as the overall flow pattern, were in close agreement. To further evaluate the agreement between the simulation and experiment, a Mask R-CNN-based instance segmentation network, coupled with a PTV tracking approach, was employed for particle detection and velocity field reconstruction [53]. The network was trained on synthetic images of polyhedral particles and then directly applied to the rotating drum DEM simulation images and experimental footage to obtain particle masks and centroids. The detection results are shown in Fig. 15.



(a) Simulation (b) Experiment

Fig. 14 Polyhedral DEM simulation and experiment.

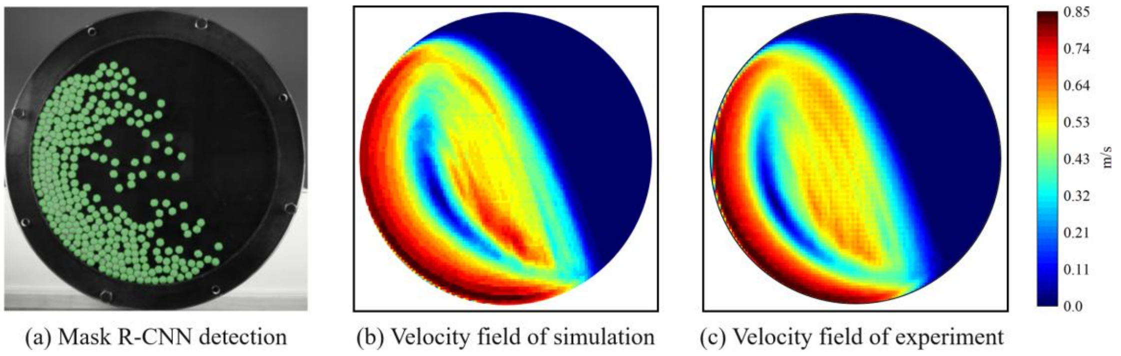


Fig. 15 Mask R-CNN non-sphere particle detection.

Fig. 15 (a) illustrates the performance of Mask R-CNN on the experimental polyhedral drum image. Although all particles were correctly detected, the predicted masks exhibited reduced accuracy at overlapping boundaries, which was likely caused by the lack of a hand-labeled dataset for fine-tuning. The reconstructed velocity fields through PTV approach are shown in Fig. 15 (b) and (c). Near the drum rim, particles are lifted by wall friction and exhibit higher speeds. In the outer layer the particles undergo free fall. Between these two regions lies a crescent-shaped quasi-static zone. The simulated and experimental velocity fields exhibit similar spatial distributions, with a correlation coefficient of R^2 = 0.8, indicating strong consistency. Minor discrepancies remained in the extent of the falling layer and in the maximum lift height.

3.5 Scalability analysis

In addition to accuracy and reliability, the scalability of the proposed multiple-GPU-based polyhedral DEM framework is examined through the strong and weak scaling analyses based on convex polyhedral packing simulations. The packing domain was fixed to 2 m along the Z-axis, while the X and Y axes were extended equally to form a large-scale particle system. Initially, particles were uniformly distributed within the domain and were randomly assigned orientations to ensure a homogeneous configuration. The wall mesh resolution was kept identical across all simulation cases, with the $\lambda \approx 1.8$. Fig. 16 shows the final packing state consisting of 20,000,000 polyhedral particles.

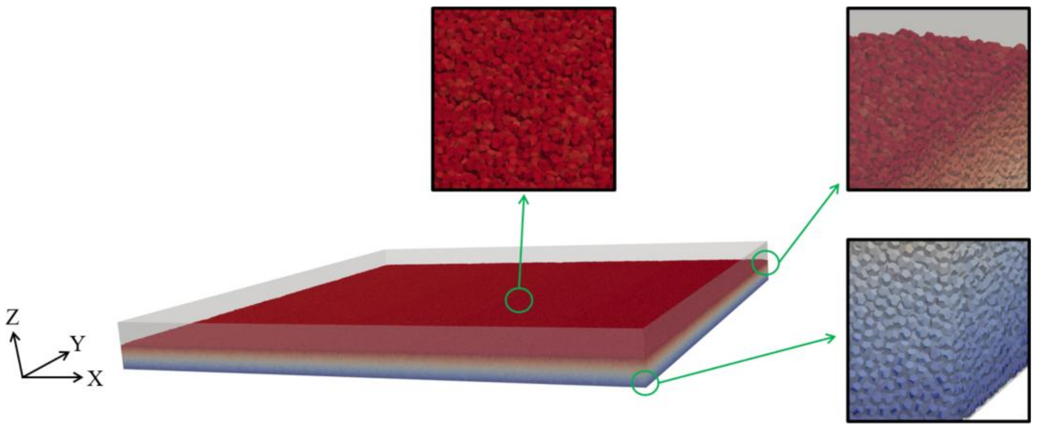


Fig. 16 Initial configurations of polyhedral particle packing.

For strong scaling, the speedup $s_s(N)$ and efficiency $\eta(N)$ are defined as:

$$s_s(N) = \frac{T_1}{T_N}, \tag{31}$$

and

$$\eta(N) = \frac{s_s(N)}{N}, \tag{32}$$

where T_1 and T_N represent the simulation times using one GPU and N GPUs, respectively. The strong scaling test evaluates the reduction in computational time for a fixed particle number problem as the number of GPUs increases. Results are shown in Fig. 17. The computational speedup increases almost linearly with the number of GPUs and the parallel efficiency remains above 85% up to 16 GPUs.

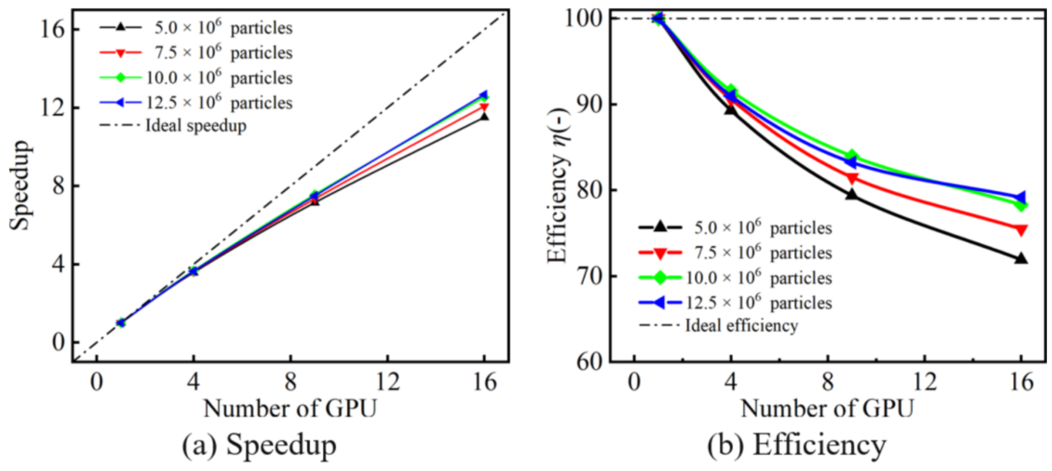


Fig. 17 Strong scaling test in polyhedral particle packing simulation.

In contrast, weak scaling evaluates whether runtime remains constant as the GPU count and the total particle number increase proportionally. The weak scaling speedup $s_w(N)$ is given by:

$$s_{\mathrm{w}}(N) = \frac{n \times T_1}{T_N}, \tag{33}$$

where each GPU handles a constant number of particles. The results of the weak-scaling test are shown in Fig. 18 (a), where the speedup curves remain close to the ideal line. These results demonstrate the good parallel scalability of the proposed framework. However, as shown in Fig. 18 (b), for the case with 5.0×10^6 particles per GPU, the parallel efficiency gradually decreases, dropping to approximately 85% at 16 GPUs. This reduction in efficiency can be attributed to the growing proportion of communication overhead relative to computation when the particle number per GPU decreases. In this case, the computational workload on each GPU is insufficient to fully utilize the available resources, and the communication latency between GPUs becomes more dominant.

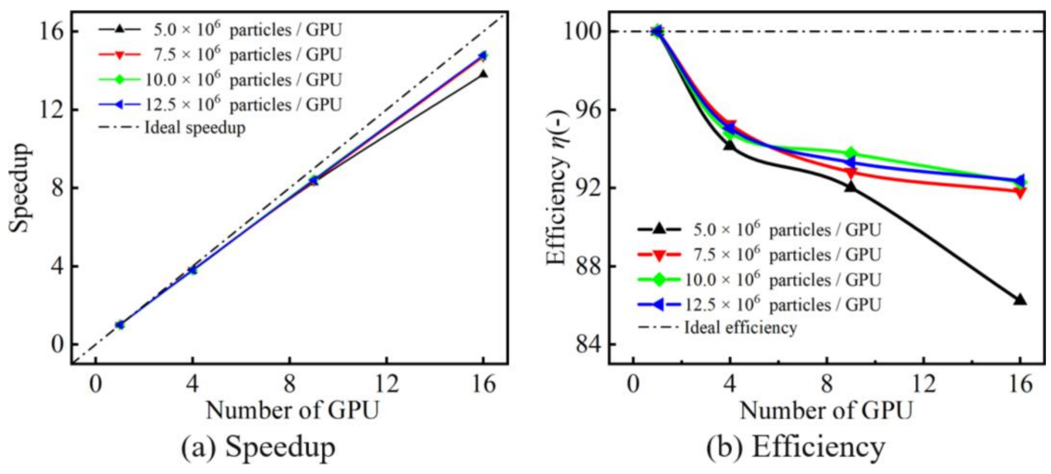


Fig. 18 Weak scaling test in polyhedral particle packing simulation.

4 Industrial-scale applications

Industrial production equipment often involves complex geometries. The proposed multiple-GPU-based polyhedral DEM framework was evaluated under realistic conditions using large-scale simulations of particle piling in a silo deposition and fixed bed.

4.1 Silo deposition

The geometry of the silo is illustrated in Fig. 19. The overall silo structure has a total height of 10.6 m. The top view shows a cylindrical cross-section with a diameter of 2.1 m. The bottom view highlights the hopper outlet, with an opening diameter of 0.67 m. To better reflect the actual system, the silo is modeled with corrugated sidewalls. The entire silo model consisted of approximately 150,000 triangular facets, and each particle interacted with about 300 neighboring facets during the computation, demanding substantial computational cost.

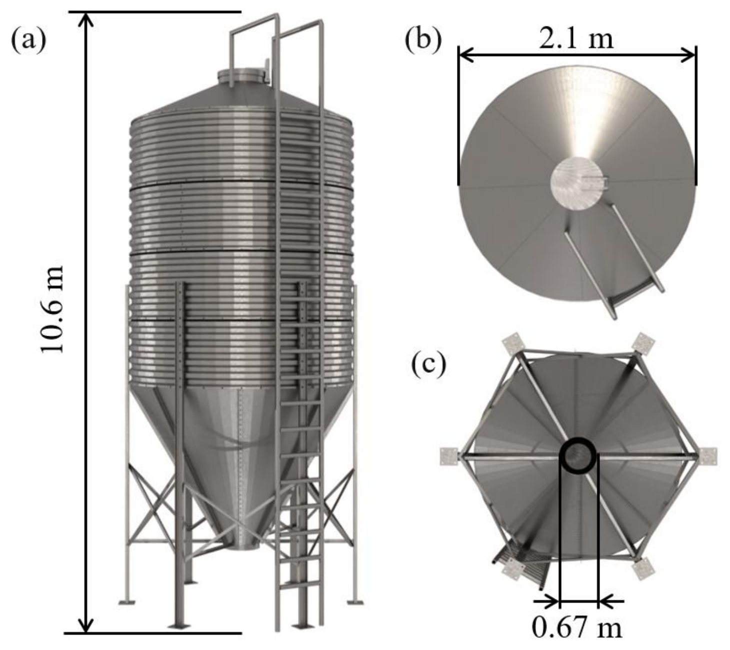


Fig. 19 Geometry of the silo: (a) overall structure; (b) top view; (c) bottom view.

The mineral particle is modeled as a convex polyhedron composed of 18 vertices, 45 edges, and 32 faces, as illustrated in Fig. 20 (a). During the simulation, particles were continuously fed from the top of the silo over a total duration of approximately 50 s, resulting in the formation of about 1,000,000 polyhedral particles within the silo. The overall packing configuration is presented in Fig. 20 (b). Due to geometric confinement, the particles at the bottom form a hopper-shaped packing structure, while those near the corrugated sidewalls exhibit a wavy arrangement that corresponds to the wall corrugation pattern. These results demonstrate that the proposed multiple-GPU-based polyhedral DEM framework can be effectively applied to large-scale particle packing simulations in industrial equipment with complex geometry.

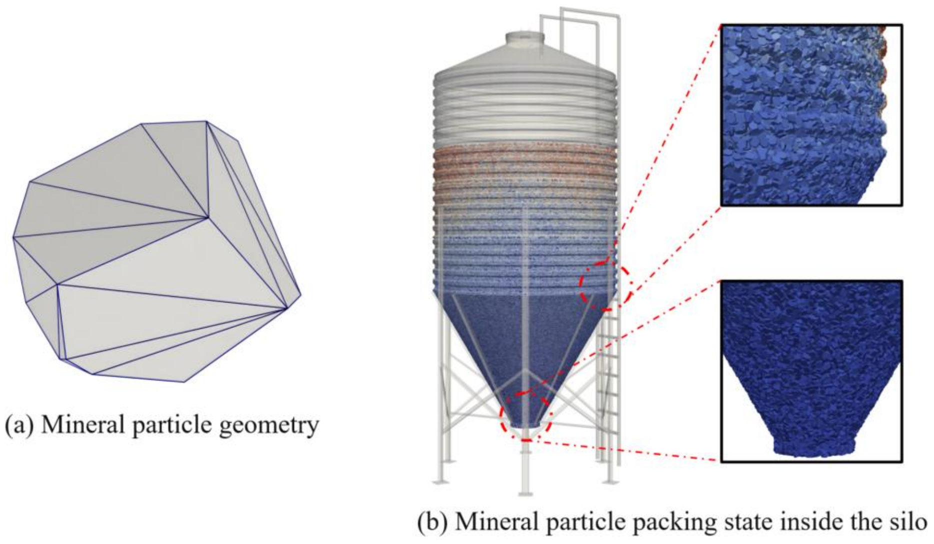
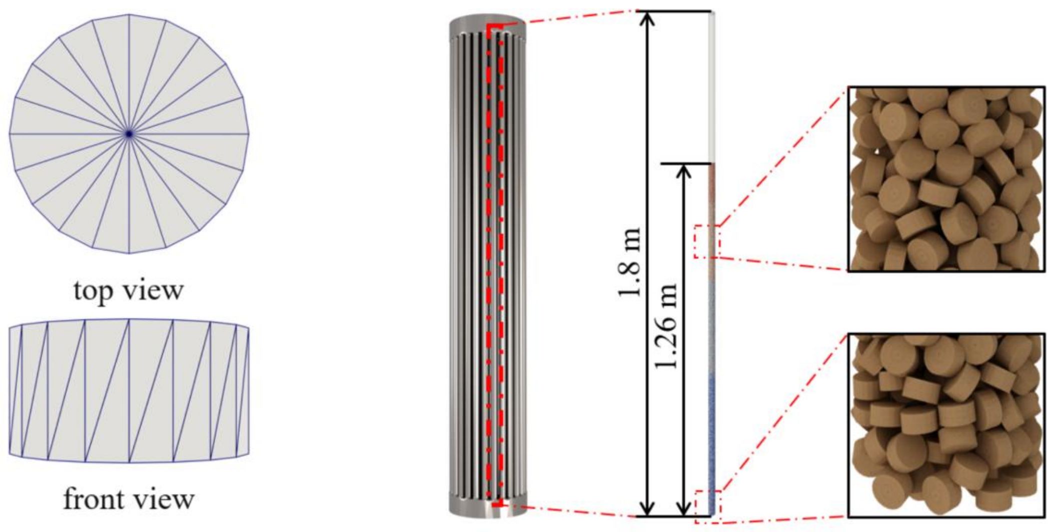

(a) Mineral particle geometry
(b) Mineral particle packing state inside the silo

Fig. 20 Mineral particle and industrial silo packing simulation.

4.2 Catalyst packing in fixed bed

The oxidation of *n*-butane to maleic anhydride typically employs a multiple-tubular fixed-bed reactor. For such a highly exothermic reaction, relatively narrow reaction tubes are generally used, usually on the order of a few centimeters in diameter. The pressure drop is dependent on the shape of the cylindrical catalyst particles, which are commonly cylindrical vanadium phosphorus oxide particles with a diameter of several millimeters. A key characteristic of this reactor is the small ratio of the reactor diameter to the particle diameter. Since the reaction tube diameter only allows for the accommodation of a few particles, irregular particle packing structures can lead to significant local inhomogeneities, which have a crucial impact on the distribution of reactant concentrations and temperatures within the reactor.

To effectively capture the packing structure of cylindrical catalyst particles inside the reactor tubes, the polyhedral DEM was employed to simulate the packing behavior of cylindrical catalysts in the fixed bed. As illustrated in Fig. 21 (a), the cylindrical catalyst particles with an outer diameter of 5.5 mm and a length of 5.5 mm. The reactor tube had an inner diameter of 2.5 cm, a total height of 1.80 m, and a packed bed height of 1.26 m. Cylindrical catalyst particles were sequentially dropped from the top of the reactor with random orientations and allowed to fall freely under gravity until a stable packed bed was formed. The feeding was terminated once the bed height reached 1.26 m, resulting in a non-uniform packing structure representative of the real reactor. Fig. 21 (b) illustrates the packing state in the fixed bed containing approximately 10,000 cylindrical catalyst particles.



(a) Cylindrical catalyst model (b) Packing state in the fixed bed

Fig. 21 Cylindrical catalyst particles packing in the fixed bed.

5 Conclusions

In this study, a multiple-GPU-based DEM simulation framework for large-scale polyhedral particle systems was developed. The framework integrates MPI with CUDA, enabling efficient execution of key computational modules, including domain decomposition, neighbor search, contact detection, and contact evaluation between polyhedral particles and walls. This design significantly improves computational efficiency and enables industrial-scale simulations of non-spherical particles.

For convex polyhedral particle contact, an energy-conserving contact model is employed. Contact detection is performed through a two-stage procedure combining bounding-sphere filtering and the GJK algorithm. After contact identification, overlap characteristics are evaluated via a dual-space transformation and convex-hull construction. Particle–wall interactions are computed on the triangular wall facets using Sutherland–Hodgman clipping to ensure the accurate contact forces. The framework was validated through numerical and experimental studies. In the particle–wall impact test, results agreed well with analytical solutions. The mesh-independence analysis of wall-contact forces exhibited stable numerical behavior. In static experiments, the packing height and structure of regular dodecahedra in a cylindrical container closely matched the measurements. Moreover, the velocity fields from the simulations were consistent with those reconstructed from rotating drum experiments through Mask R-CNN and PTV approach, confirming the effectiveness of the framework in reproducing granular dynamics. Scalability tests demonstrated strong parallel performance. Simulations involving up to 2×10^7 polyhedral particles on 16 GPUs achieved up to 14.8 times speedup. Additional simulations such as silo deposition and cylindrical catalyst particle packing in fixed bed, further demonstrate the reliability in industrial application.

This work offers practical guidance for parallelizing polyhedral DEM and scaling it to industrial-scale simulations on the multiple-GPU systems. Nevertheless, additional optimization is required to improve general applicability. In this context, future work will focus on extending the framework to arbitrary particle shapes, and develop localized optimization strategies that confine polyhedral overlap calculations to contacting triangular facets. To better reproduce experimental observations, optimization-based methods will be used to further calibrate collision parameters for non-spherical particles. Finally, integrating experimental data and artificial intelligence to establish generalized cohesive models for non-spherical particles, and developing coupled non-sphere particle–fluid simulations.

Acknowledgments

The authors gratefully acknowledge the financial support from the National Natural Science Foundation of China (Grant Nos. 22293024, 22293021 and 22408373), the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA29040200 and XDA0490102, the Special Research Assistant Funding Program of the Chinese Academy of Sciences. This work was carried out using the facilities at Huairou Interdisciplinary Research Center for Mesoscience, CAS-IPE. We sincerely thank Qingyuan Cao for his assistance in the scaling test.

References

[1] H.M. Jaeger, S.R. Nagel, R.P. Behringer, The Physics of Granular Materials, Phys. Today 49 (1996) 32–38. https://doi.org/10.1063/1.881494.

[2] G. Sun, J.R. Grace, The effect of particle size distribution on the performance of a catalytic fluidized bed reactor, Chem. Eng. Sci. 45 (1990) 2187–2194. https://doi.org/10.1016/0009-2509(90)80094-U.

[3] R. Agarwal, N. Yadav, Pharmaceutical Processing–A Review on Wet Granulation Technology, Int. J. Pharm. Front. Res. 1 (2011) 65–83.

[4] E. Ortega-Rivas, Bulk Properties of Food Particulate Materials: An Appraisal of their Characterisation and Relevance in Processing, Food Bioprocess Technol. 2 (2009) 28–44. https://doi.org/10.1007/s11947-008-0107-5.

[5] B. Kou, Y. Cao, J. Li, C. Xia, Z. Li, H. Dong, A. Zhang, J. Zhang, W. Kob, Y. Wang, Granular materials flow like complex fluids, Nature 551 (2017) 360–363. https://doi.org/10.1038/nature24062.

[6] S. Zhang, M. Zhao, W. Ge, C. Liu, Bimodal frequency distribution of granular discharge in 2D hoppers, Chem. Eng. Sci. 245 (2021) 116945. https://doi.org/10.1016/j.ces.2021.116945.

[7] P. Schall, M. Van Hecke, Shear Bands in Matter with Granularity, Annu. Rev. Fluid Mech. 42 (2010) 67–88. https://doi.org/10.1146/annurev-fluid-121108-145544.

[8] A. Hafez, Q. Liu, T. Finkbeiner, R.A. Alouhali, T.E. Moellendick, J.C. Santamarina, The effect of particle shape on discharge and clogging, Sci. Rep. 11 (2021) 3309. https://doi.org/10.1038/s41598-021-82744-w.

[9] P.A. Cundall, O.D.L. Strack, A discrete numerical model for granular assemblies, Geotechnique 29 (1979) 47–65. https://doi.org/10.1680/geot.1979.29.1.47.

[10] C.-Y. Lu, C.-L. Tang, Y.-C. Chan, J.-C. Hu, C.-C. Chi, Forecasting landslide hazard by the 3D discrete element method: A case study of the unstable slope in the Lushan hot spring district, central Taiwan, Eng. Geol. 183 (2014) 14–30. https://doi.org/10.1016/j.enggeo.2014.09.007.

[11] B. Blais, D. Vidal, F. Bertrand, G.S. Patience, J. Chaouki, Experimental Methods in

Chemical Engineering: Discrete Element Method—DEM, CJCE 97 (2019) 1964–1973. https://doi.org/10.1002/cjce.23501.

[12] F. Fleissner, T. Gaugele, P. Eberhard, Applications of the discrete element method in mechanical engineering, Multibody Syst. Dyn. 18 (2007) 81. https://doi.org/10.1007/s11044-007-9066-2.

[13] Y.T. Feng, Thirty years of developments in contact modelling of non-spherical particles in DEM: a selective review, Acta Mech. Sin. 39 (2023) 722343. https://doi.org/10.1007/s10409-022-22343-x.

[14] D. Fan, Y. Tang, P. Wang, Y. Li, C. Lian, A. Striolo, Y. Chen, Z. Lv, J. Li, S. Zhao, J. Bai, L. Zhou, P. Malgaretti, J. Zhu, D. Zhang, Crossover scaling of structural and mechanical properties in 3D assemblies of non-spherical, frictional particles, Commun. Phys. 8 (2025) 81. https://doi.org/10.1038/s42005-025-02009-0.

[15] D. Markauskas, R. Kačianauskas, A. Džiugys, R. Navakas, Investigation of adequacy of multi-sphere approximation of elliptical particles for DEM simulations, Granul. Matter 12 (2010) 107–123. https://doi.org/10.1007/s10035-009-0158-y.

[16] S. Wang, D. Liang, S. Ji, DEM study on mixing behaviors of concave-shaped particles in rotating drum based on level-set method, Powder Technol. 430 (2023) 118961. https://doi.org/10.1016/j.powtec.2023.118961.

[17] X. Gao, J. Yu, R.J.F. Portal, J.-F. Dietiker, M. Shahnam, W.A. Rogers, Development and validation of SuperDEM for non-spherical particulate systems using a superquadric particle method, Particuology 61 (2022) 74–90. https://doi.org/10.1016/j.partic.2020.11.007.

[18] M.V. Craveiro, A. Gay Neto, P. Wriggers, DEM simulations using convex NURBS particles, Comput. Part. Mech. 11 (2024) 1087–1118. https://doi.org/10.1007/s40571-023-00675-x.

[19] Y.T. Feng, K. Han, D.R.J. Owen, Energy-conserving contact interaction models for arbitrarily shaped discrete elements, Comput. Methods in Appl. Mech. Eng. 205–208 (2012) 169–177. https://doi.org/10.1016/j.cma.2011.02.010.

[20] R. Berger, C. Kloss, A. Kohlmeyer, S. Pirker, Hybrid parallelization of the LIGGGHTS open-source DEM code, Powder Technol. 278 (2015) 234–247.

https://doi.org/10.1016/j.powtec.2015.03.019.

[21] T. Weinhart, L. Orefice, M. Post, M.P. Van Schrojenstein Lantman, I.F.C. Denissen, D.R. Tunuguntla, J.M.F. Tsang, H. Cheng, M.Y. Shaheen, H. Shi, P. Rapino, E. Grannonio, N. Losacco, J. Barbosa, L. Jing, J.E. Alvarez Naranjo, S. Roy, W.K. Den Otter, A.R. Thornton, Fast, flexible particle simulations — An introduction to MercuryDPM, Comput. Phys. Commun. 249 (2020) 107129. https://doi.org/10.1016/j.cpc.2019.107129.

[22] R. Prat, T. Carrard, L. Amarsid, V. Richefeu, C. Doncecchi, P. Lafourcade, G. Latu, J.-M. Vanson, ExaDEM: a HPC application based on exaNBody targeting scalable DEM simulations with complex particle shapes, JOSS 10 (2025) 7484. https://doi.org/10.21105/joss.07484.

[23] R. Garg, J. Galvin, T. Li, S. Pannala, Open-source MFIX-DEM software for gas–solids flows: Part I—Verification studies, Powder Technol. 220 (2012) 122–137. https://doi.org/10.1016/j.powtec.2011.09.019.

[24] A. Podlozhnyuk, S. Pirker, C. Kloss, Efficient implementation of superquadric particles in Discrete Element Method within an open-source framework, Comput. Part. Mech. 4 (2017) 101–118. https://doi.org/10.1007/s40571-016-0131-6.

[25] J. Choquette, W. Gandhi, O. Giroux, N. Stam, R. Krashinsky, NVIDIA A100 Tensor Core GPU: Performance and Innovation, IEEE Micro 41 (2021) 29–35. https://doi.org/10.1109/MM.2021.3061394.

[26] S. Ji, L. Liu, Computational Granular Mechanics and Its Engineering Applications, Springer Singapore, Singapore, 2020. https://doi.org/10.1007/978-981-15-3304-4.

[27] J. Xu, H. Qi, X. Fang, L. Lu, W. Ge, X. Wang, M. Xu, F. Chen, X. He, J. Li, Quasi-real-time simulation of rotating drum using discrete element method with parallel GPU computing, Particuology 9 (2011) 446–450. https://doi.org/10.1016/j.partic.2011.01.003.

[28] J. Xu, P. Zhao, Y. Zhang, J. Wang, W. Ge, Discrete particle methods for engineering simulation: Reproducing mesoscale structures in multiphase systems, Resour. Chem. Mater. 1 (2022) 69–79. https://doi.org/10.1016/j.recm.2022.01.002.

[29] J. Xu, X. Liu, S. Hu, W. Ge, Virtual process engineering on a three-dimensional circulating fluidized bed with multiscale parallel computation, J. Adv. Manuf. Process. 1 (2019) e10014. https://doi.org/10.1002/amp2.10014.

[30] H.R. Norouzi, PhasicFlow: A parallel, multi-architecture open-source code for DEM simulations, Comput. Phys. Commun. 291 (2023) 108821. https://doi.org/10.1016/j.cpc.2023.108821.

[31] J.A. Anderson, J. Glaser, S.C. Glotzer, HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations, Comput. Mater. Sci. 173 (2020) 109363. https://doi.org/10.1016/j.commatsci.2019.109363.

[32] S. Wang, Y. Fan, S. Ji, Interaction between super-quadric particles and triangular elements andits application to hopper discharge, Miner. Eng. 339 (2018) 534–549. https://doi.org/10.1016/j.powtec.2018.08.026.

[33] N. Govender, R.K. Rajamani, S. Kok, D.N. Wilke, Discrete element simulation of mill charge in 3D using the BLAZE-DEM GPU framework, Miner. Eng. 79 (2015) 152–168. https://doi.org/10.1016/j.mineng.2015.05.010.

[34] N. Govender, R. Rajamani, D.N. Wilke, C.-Y. Wu, J. Khinast, B.J. Glasser, Effect of particle shape in grinding mills using a GPU based DEM code, Miner. Eng. 129 (2018) 71–84. https://doi.org/10.1016/j.mineng.2018.09.019.

[35] G.-Y. Liu, W.-J. Xu, A GPU-based DEM framework for simulation of polyhedral particulate system, Granul. Matter 25 (2023). https://doi.org/10.1007/s10035-023-01321-2.

[36] Y.T. Feng, An energy-conserving contact theory for discrete element modelling of arbitrarily shaped particles: Basic framework and general contact model, Comput. Methods in Appl. Mech. Eng. 373 (2021) 113454. https://doi.org/10.1016/j.cma.2020.113454.

[37] D. Guide, Cuda c programming guide, NVIDIA, July 29 (2013) 6.

[38] Y. He, A.E. Bayly, A. Hassanpour, F. Muller, K. Wu, D. Yang, A GPU-based coupled SPH-DEM method for particle-fluid flow with free surfaces, Powder Technol. 338 (2018) 548–562. https://doi.org/10.1016/j.powtec.2018.07.043.

[39] J. Zheng, X. An, M. Huang, GPU-based parallel algorithm for particle contact detection and its application in self-compacting concrete flow simulations, Comput. Struct. 112–113 (2012) 193–204. https://doi.org/10.1016/j.compstruc.2012.08.003.

[40] N. Govender, D.N. Wilke, S. Kok, R. Els, Development of a convex polyhedral discrete

element simulation framework for NVIDIA Kepler based GPUs, J. Comput. Appl. Math. 270 (2014) 386–400. https://doi.org/10.1016/j.cam.2013.12.032.

[41] N. Govender, D.N. Wilke, S. Kok, Collision detection of convex polyhedra on the NVIDIA GPU architecture for the discrete element method, Appl. Math. Comput. 267 (2015) 810–829. https://doi.org/10.1016/j.amc.2014.10.013.

[42] J.A. Anderson, C.D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics processing units, J. Comput. Phys. 227 (2008) 5342–5359. https://doi.org/10.1016/j.jcp.2008.01.047.

[43] N. Bell, J. Hoberock, Chapter 26 - Thrust: A Productivity-Oriented Library for CUDA, in: GPU Computing Gems Jade Edition, Morgan Kaufmann, Boston, 2012: p. 359. https://doi.org/10.1016/B978-0-12-385963-1.00026-5.

[44] E.G. Gilbert, D.W. Johnson, S.S. Keerthi, A fast procedure for computing the distance between complex objects in three-dimensional space, IEEE J. Robot. Automat. 4 (1988) 193–203. https://doi.org/10.1109/56.2083.

[45] J. Landauer, M. Kuhn, D.S. Nasato, P. Foerst, H. Briesen, Particle shape matters – Using 3D printed particles to investigate fundamental particle and packing properties, Powder Technol. 361 (2020) 711–718. https://doi.org/10.1016/j.powtec.2019.11.051.

[46] S. Wang, S. Ji, Computational Mechanics of Arbitrarily Shaped Granular Materials, Springer Nature Singapore, Singapore, 2024. https://doi.org/10.1007/978-981-99-9927-9.

[47] S.P. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.

[48] C.B. Barber, D.P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Trans. Math. Softw. 22 (1996) 469–483. https://doi.org/10.1145/235815.235821.

[49] L. Hu, G.M. Hu, Z.Q. Fang, Y. Zhang, A new algorithm for contact detection between spherical particle and triangulated mesh boundary in discrete element method simulations, Int. J. Numer. Methods Eng. 94 (2013) 787–804. https://doi.org/10.1002/nme.4487.

[50] Q. Zhou, W.-J. Xu, G.-Y. Liu, A contact detection algorithm for triangle boundary in GPU-based DEM and its application in a large-scale landslide, Comput. Geotech. 138 (2021) 104371. https://doi.org/10.1016/j.compgeo.2021.104371.

[51] Y. Feng, A generic energy-conserving discrete element modeling strategy for concave particles represented by surface triangular meshes, Int. J. Numer. Methods Eng. 122 (2021)

2581–2597. https://doi.org/10.1002/nme.6633.

[52] M. Kodam, R. Bharadwaj, J. Curtis, B. Hancock, C. Wassgren, Cylindrical object contact detection for use in discrete element method simulations, Part II—Experimental validation, Chem. Eng. Sci. 65 (2010) 5863–5871. https://doi.org/10.1016/j.ces.2010.08.007.

[53] J. Xu, S. Zhang, W. Ge, An efficient non-spherical particle tracking strategy based on deep-learning and simulation-experiment integration, Powder Technol. 468 (2026) 121681. https://doi.org/10.1016/j.powtec.2025.121681.