The Astronomical Journal, 166:194 (15pp), 2023 November https://doi.org/10.3847/1538-3881/acfc46
© 2023. The Author(s). Published by the American Astronomical Society.
OPEN ACCESS

Accelerating Polyhedral Discrete Element Method with CUDA

Tongge Wen and Xiangyuan Zeng[1]
School of Automation, Beijing Institute of Technology, Beijing 100081, People's Republic of China; zeng@bit.edu.cn
Received 2023 April 25; revised 2023 September 11; accepted 2023 September 20; published 2023 October 16

Abstract

This paper presents an efficient CUDA-based implementation of a nonspherical discrete element method where irregular particles are described by using polyhedrons. Two strategies are employed to exploit the parallelism of the numerical method. One is to perform contact detection based on the contact pair level instead of the traditional particle level. The second is to reduce the computational burden of each kernel function by allocating thread blocks reasonably. Contact detection between potential contact pairs is the most complicated, time-consuming, and essential process for the polyhedral discrete element method. The linear bounding volume hierarchies are introduced to fix this issue. The hierarchies of the bounding volume tree are organized in a spatially coherent way. Such a structure can minimize branch divergence and is very suitable for parallel implementation with GPU. Two numerical examples are presented to show the performance of the code. It is found from the scenario of two sphere collision that improving the mesh resolution of polyhedral particles can reduce the computational error while slowing down the computational speed correspondingly. A trade-off must be made between accuracy and efficiency. The other example of self-gravitating aggregation demonstrates the code is convergent, stable, and highly efficient. Particularly, with a mainstream GPU, the proposed method easily performs hundreds of times faster than the serial CPU code that does the same function.

Unified Astronomy Thesaurus concepts: Astronomical simulations (1857); N-body problem (1082)

1. Introduction

The discrete element method (DEM) has been widely used in planetary science and modern astrodynamics (Richardson 2000; Michel et al. 2001; Yu et al. 2014). The reason can be summarized as follows. First, emerging evidence shows that kilometer-sized asteroids may be gravitational aggregates of low tensile strength and can be categorized into granular matter (Richardson et al. 2009). Such a matter form is well suited to study by DEM since it is good at dealing with the dynamic process of large-scale particle interactions (Zhang & Michel 2021). Second, some speculations and theories can only be verified through numerical simulations. The morphology evolution of asteroids, which have experienced a progressive process for millions of years, is hard to observe due to current human science and technology limitations. Instead, DEM could offer a plausible explanation of the formation and evolution process of asteroids. For instance, DEM can simulate the disruption and reaccumulation process of small bodies, revealing the origin of small body families (Michel et al. 2001; Richardson et al. 2009; Sánchez & Scheeres 2011; Schwartz et al. 2012; Michel & Richardson 2013; Schwartz et al. 2018). It can also be used to study the influence of external factors, such as the tidal force and solar radiation pressure, on the shape and state evolution of small bodies (Asphaug & Benz 1996; Cotto-Figueroa et al. 2015; Zhang & Michel 2020). The geological activity of small bodies can be simulated through DEM, such as the avalanche, landslide, evolution of ejecta et al. (Yu et al. 2014). Third, it is challenging to create the vacuum and microgravity environment of asteroids artificially. The experimentally produced reduced-gravity environment cannot maintain temporal stability and is expensive. Plus, the reduced-gravity levels are similar to that of Martian or Lunar and cannot reach the microgravity levels identical to asteroids (Güttler et al. 2013). Numerical simulation using DEM provides a preferable option to solve the above problems.

In most studies, particles are treated as spheres to reduce computational complexity and improve computational efficiency (Richardson et al. 2009; Sánchez & Scheeres 2011). Nevertheless, most realistic particles are not perfectly spherical (Wen et al. 2020). Simplifications of particle shapes are inappropriate for problems where the particle shape plays a significant role. Previous studies also demonstrated that findings obtained from spherical particles could not be readily extrapolated to nonspherical particle systems (Lu et al. 2015). Being aware of this problem, nonspherical DEM is developed rapidly in recent years, especially in geotechnical engineering (Liu et al. 2020; Xie et al. 2020; Zhan et al. 2021). The study of nonspherical DEM in planetary science is just getting started. Sánchez et al. (2021) developed a numerical implementation of nonspherical particles in the open-source software LMGG90. The amount of nonspherical particles used in the simulations reached 8000; therefore, the impulse-based contact model is adopted to promote the computing speed, yet it cannot accurately capture the particle deformation in dense regimes. Ferrari et al. (2017, 2020) proposed a GPU-based code for the *N*-body contact gravitational and dynamics, which features force-based and impulse-based contact models. They employed the Gilbert–Johnson–Keerthi (GJK) algorithm to detect contact between nonspherical particles. The impulse-based contact model is also used in their study to lighten the computational burdens in the GPU-based *N*-body numerical simulations. Nevertheless, it should be noted that the GJK algorithm is only applicable to convex bodies. Additional algorithms are required to partition nonconvex bodies into convex bodies. Moreover,

[1] Professor, Member AIAA.

Original content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

implementing the GJK algorithm needs to manage memory dynamically, which increases the program complexity (Liu et al. 2022).

In our recent study, a CPU-based code suite has been developed to solve the dynamical evolution of the nonspherical granular system (Zeng et al. 2022). The particle shapes have no restrictions where the force-based contact model was adopted. Although our code is stable and effective in handling the contact between irregularly shaped particles, it takes about 70–100 iterative steps to run a single contact between two intersecting particles. To make matters worse, the algorithm dealing with contact detection is complicated. These two superimposed reasons lead to poor computational efficiency of the previous method. As an attempt, the OpenMP was applied to parallelize the code, but the speed boost is very limited. For example, we used a computer built by Intel(R) Xeon(R) Gold 6138 CPU @ 2.00 GHz with 16 GB RAM to run the code. During simulations, 16 threads were enabled to accelerate the code. It took approximately 10 days for 1440 particles to settle in a sample box. It seems impossible to perform large-scale simulations with such computing speed since the time cost is obviously unaffordable. Trial and error costs also increase dramatically. As a result, the computational efficiency severely limits the application scope of the code.

Fortunately, compute unified device architecture (CUDA) provides a feasible alternative to improve computational efficiency. It is a general-purpose computing platform and programming model introduced by the NVIDIA GPU accelerator, providing a development environment for creating high-performance GPU-accelerated applications. GPU possesses numerous lightweight computing cores (usually thousands or tens of thousands) to achieve high computing performance. It fulfills parallelization by assigning the number of threads, blocks, and grids to kernel functions. Note that the *kernel function* mentioned in this paper refers to the functions that run on the GPU and is launched by the CPU. The kernel function dictates which data each GPU thread accesses and what computation is performed. GPU provides a hardware foundation to improve the computing speed of nonspherical DEM. This is exactly the motivation of this paper: developing a GPU-based code to support the large-scale numerical simulation of the polyhedral DEM (PolyDEM). Since the particle shapes are described using polyhedral meshes, the code is named PolyDEM.

In the previous CPU-based code, we performed the neighbor search, contact detection, and integration based on the particle level. The bounding volume hierarchies (BVH) are employed to deal with contact between nonspherical particles, in which the construction, update, and query of the bounding volume (BV) tree are implemented iteratively (Hippmann 2004). However, this numerical implementation is unsuitable for GPU architecture since the program nesting is too deep. To further excavate concurrency, a brand-new GPU-based code, PolyDEM, is developed in this paper, which is more suitable for the GPU environment. PolyDEM inherits the advantages of the CPU-based code. By employing the force-based contact model, the code can accurately track the transmission of forces through particles. Contact between particles in a granular medium is not instantaneous. Consequently, the force-based contact model is more consistent with the actual physical process (Sánchez & Scheeres 2011; Schwartz et al. 2012). The particles can owe arbitrary shapes by using the polyhedral meshes to describe the particle shapes. The difference is that PolyDEM has finer-grained parallelism than previous CPU-based code. The operations are performed based on contact pair levels. The complicated tasks in the original CPU-based code are now partitioned into several kernel functions to execute on GPU. By organizing the thread blocks, the computational burdens in each kernel function are greatly reduced. Furthermore, the linear bounding volume hierarchies (LBVH) technique is adopted to fulfill the contact detection between polyhedral particles, where the operations of the BV tree are accomplished in a traversal way (Karras & Aila 2013). Such an implementation would minimize the branch divergence and is preferable for GPU architecture. In the remaining part of the paper, the basic theory of PolyDEM is briefly introduced in Section 2, including the dynamical equations, contact model, and integrator. The structure of the PolyDEM, GPU acceleration, and detailed implementation processes are described in Section 3. The accuracy, efficiency, and stability of the code are demonstrated by presenting two numerical examples in Section 4. Particularly, unlike the impulse-based contact model used in previous studies, we employ the force-based contact model to simulate the accretion process between nonspherical particles. The PolyDEM still achieves high computational efficiency and numerical stability. Section 5 summarizes the paper.

2. Theory of PolyDEM

2.1. Governing Equations

In PolyDEM, the governing equations of an individual particle i can be written as (Richardson 2000)

$$\begin{cases} m_i \boldsymbol{a}_i = \boldsymbol{F} + \sum\limits_{j \in \mathbb{C}_\alpha} \boldsymbol{F}_{cij} \\ \boldsymbol{J}_i \dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times (\boldsymbol{J}_i \boldsymbol{\omega}_i) = \boldsymbol{M} + \sum\limits_{j \in \mathbb{C}_\alpha} \boldsymbol{M}_{cij} \end{cases}. \tag{1}$$

In Equation (1), the motion of an individual particle is decomposed into translational and rotational motion. On the left-hand side of the equal sign, the vector $\boldsymbol{a}_i$ denotes the acceleration of the particle. The vector $\boldsymbol{\omega}_i$ denotes the angular velocity. The symbols m_i and J_i denote the mass and moment of inertia of the particle. On the right-hand of the equal sign, the vectors $\boldsymbol{F}$ and $\boldsymbol{M}$ are the external force and moment resultants (for example, the gravitational forces and torques), respectively. The vectors $\boldsymbol{F}_{cij}$ and $\boldsymbol{M}_{cij}$ denote the contact force and torque from the neighboring particle j on particle i. The symbol $\mathbb{C}_\alpha$ denotes the collection of all particles that adjoins the particle i. The translational motion of the particle (the first row in Equation (1)) is expressed in the inertial frame, while the rotational motion (the second row in Equation (1)) is expressed in the body-fixed frame attached to the particle i. Then, the dynamical evolution of the granular system can be tracked by applying Equation (1) to each particle.

In PolyDEM, the geometrical representation of a nonspherical particle is approximated as a polyhedron using surface triangular meshes. Figure 1 illustrates the polyhedral particles with different mesh solutions. Describing the nonspherical particles using polyhedral mesh has three advantages (Zhan et al. 2021): (1) Polyhedral mesh is the most universal way to digitally describe three-dimensional particles, regardless of acquired techniques, such as using three-dimensional computerized tomograpgy/laser scanning or computer software. (2)

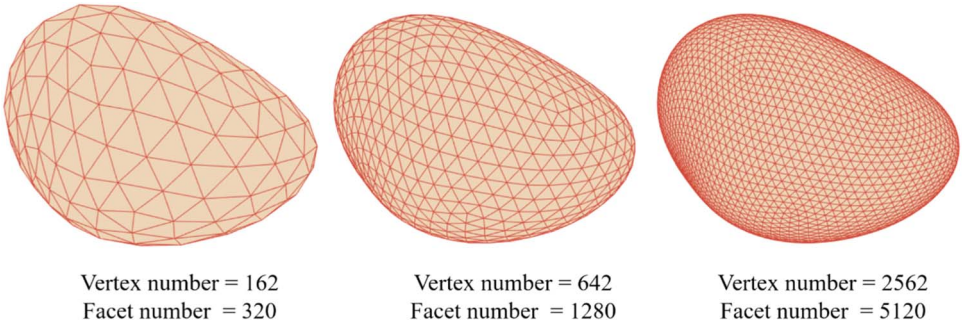


Figure 1. Irregular-shaped particles with different mesh resolutions.

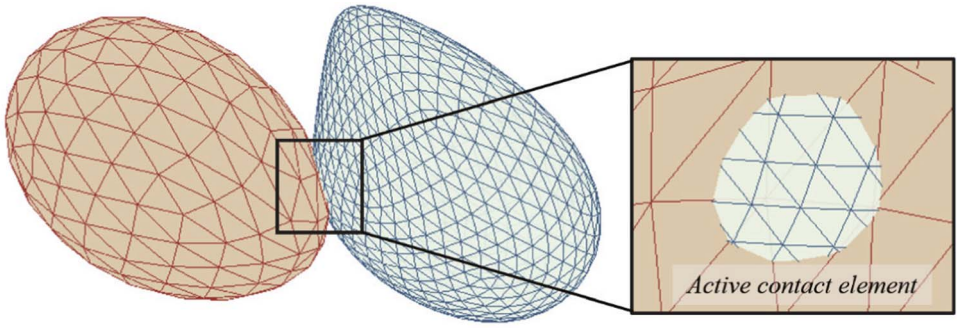


Figure 2. Locally intersecting triangular facets between two polyhedral particles.

The polyhedral mesh has high versatility. Arbitrarily complex shapes can be represented by using the unified polyhedral mesh, which significantly simplifies the preprocessing for data preparation. Additionally, if one wants to update, simplify, reconstruct, smooth, or deform the mesh, there have been abundant open-source algorithms to manipulate the polyhedral mesh. (3) The polyhedral mesh is flexible. It can adapt to different precision and efficiency requirements by choosing different mesh resolutions, which is quite essential in the numerical simulations and will be elaborated on in Section 4.1.

The surfaces of the particles are all discretized into polyhedral meshes. Thus, whether each two particles are in contact depends on whether their triangular facets intersect. Figure 2 illustrates the contact scenario between two polyhedral particles. The locally intersecting triangular facets between two triangular meshes are referred to as active contact elements' (the inset in Figure 2). The mesh with the higher spatial resolution is utilized to compute the contact force. For each active contact element, the normal and tangential contact force can be ascertained by using the Hertz contact model and Coulomb friction model as (Hertz 1881; Cheng et al. 2017, 2018)

$$\begin{aligned} F_{nk} &= \frac{4}{3}\sqrt{r_{\text{eff}}}E_{\text{eff}}\delta_{nk}^{3/2} - C_{nk}u_{nk};\\ F_{sk} &= \min\left(\mu F_{nk},\ 8\sqrt{r_{\text{eff}}\delta_{nk}}G_{\text{eff}}\delta_{sk} - C_{sk}u_{sk}\right) \end{aligned} \tag{2}$$

where the variable δ_{nk} indicates the mutual compression of each active contact element (i.e., intersecting triangular facets; see Figure 2), and δ_{sk} corresponds to the tangential relative displacement. The variables C_{nk} and C_{sk} denote the damping coefficient in the normal and tangential directions, respectively, and u_{nk} and u_{sk} are the relative velocities in normal and tangential directions. The effective radius r_{eff} is defined as half of the harmonic average of two contacting bodies' equivalent spheres (i.e., the sphere has the same volume as the body). The effective Young's modulus E_{eff} and effective shear modulus G_{eff} are defined as half of the average of the two contacting bodies' modified Young's modulus and modified shear modulus. Readers can refer to Wen et al. (2023), Zeng et al. (2022) for the calculations of δ_{nk} and δ_{sk}, as well as more details of the contact model.

Assume the total number of the active contact elements for a pairing of contacting bodies is N. The resulting contact force acting on the two contact bodies, i and j, yields

$$\boldsymbol{F}_{ci} = \frac{\sum_{k=1}^{N}(F_{nk}\boldsymbol{n}_k + F_{sk}\boldsymbol{s}_k)}{N}, \quad \boldsymbol{F}_{ci} = -\boldsymbol{F}_{cj} \tag{3}$$

where the triangular mesh of the body i is used to generate the active contact elements and calculate the contact force.

2.2. Motion Integration

The second-order leap-frog integration is adopted to solve the dynamical equations of the granular system presented in Equation (1) to make a trade-off between computational accuracy and efficiency. On the one hand, the leap-frog integration is symplectic conservation. Its stability in handling intensive contact scenarios has been validated in previous studies. On the other hand, the leap-frog integration scheme does not require iteration in a single time step; therefore, the computational cost is low. The "kick-drift-kick" scheme is applied to solve the translational motion of the particle from step m to step $m+1$ (Schwartz et al. 2012):

$$\begin{aligned} \boldsymbol{v}_{m+1/2} &= \boldsymbol{v}_m + \boldsymbol{a}_m\frac{\Delta t}{2};\\ \boldsymbol{r}_{m+1} &= \boldsymbol{r}_m + \boldsymbol{v}_{m+1/2}\Delta t;\\ \boldsymbol{v}_{m+1} &= \boldsymbol{v}_{m+1/2} + \boldsymbol{a}_{m+1}\frac{\Delta t}{2} \end{aligned} \tag{4}$$

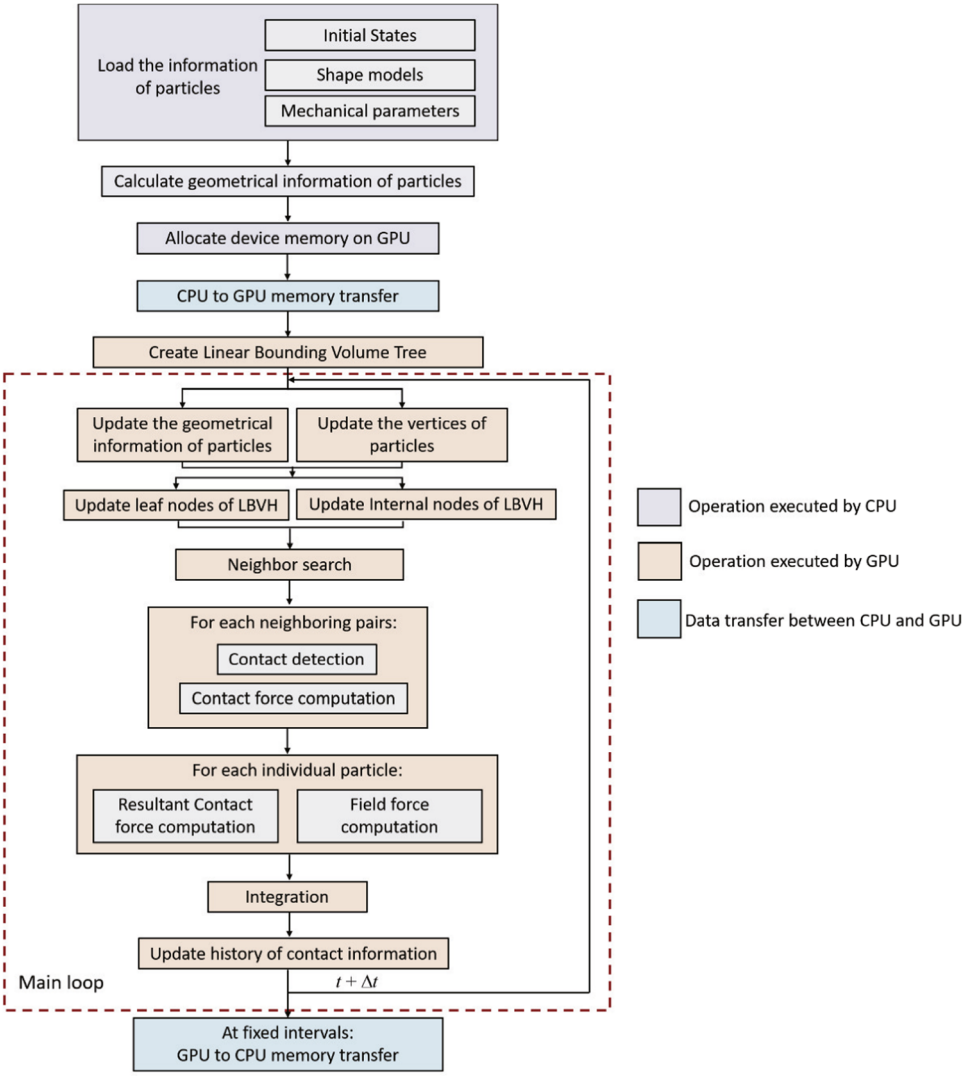


Figure 3. Flowchart of the simulation process in PolyDEM.

where $\boldsymbol{v}$ and $\boldsymbol{r}$ are the velocity and position of the particle, and the subscript denotes the step number. The symbol Δt represents the step size in the numerical simulation. When solving the rotational motion of the particle, the quaternion $\boldsymbol{q}$ is used to track the attitude of the particle. Consequently, the corresponding iterative formulas from step m to step $m+1$ are

$$\begin{aligned} \boldsymbol{\omega}_{m+1} &= \boldsymbol{\omega}_m + \dot{\boldsymbol{\omega}}_{m+1}\Delta t; \\ \boldsymbol{q}_{m+1} &= \boldsymbol{q}_m + \boldsymbol{f}(\boldsymbol{q}_m)\Delta t + \boldsymbol{f}'(\boldsymbol{q}_m)\frac{\Delta t^2}{2}. \end{aligned} \tag{5}$$

The variable $\boldsymbol{f}(\boldsymbol{q}_m)$ in Equation (5) is defined as

$$f(\boldsymbol{q}_m) = \frac{1}{2}\begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{6}$$

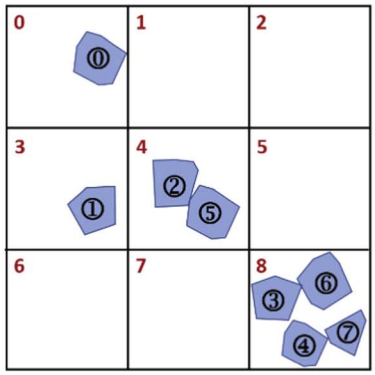
where ω_x, ω_y, ω_z are three components of the angular velocity, and q_0 to q_3 correspond to the components of the quaternion $\boldsymbol{q}$. The variable $\boldsymbol{f}'(\boldsymbol{q}_m)$ defines the first derivative of $\boldsymbol{f}(\boldsymbol{q}_m)$ versus time.

3. GPU Acceleration and Numerical Implementation

3.1. PolyDEM Frame on GPU

The GPU framework is designed for asynchronous computing, which is not a standalone platform but a coprocessor to a CPU. The computing structure is divided into the host (CPU and its memory) and device (GPU and its memory), where GPU must operate in conjunction with a CPU-based host. In PolyDEM, the CPU works as a process controller. Since data transmission is a GPU computing bottleneck, the PolyDEM reduces the data transfer between the host and device as much as possible. CPU is only responsible for loading and saving data, initializing the settings of the simulation parameters, and driving the GPU computing kernels to perform computations in a designed order. All of the computational operations are executed on GPU. Such an architecture can fully utilize the computational power of the combined CPU + GPU system. The overall flowchart of PolyDEM is depicted in Figure 3.

In Figure 3, the CPU loads the initial states, shape models, and mechanical parameters of the particles from external files. The initial conditions include information about the positions, velocities, quaternions, and angular velocities; the shape models include the topological information of vertices and



Before sorting:

ParticleIndex	0	1	2	3	4	5	6	7
ParticleHash	0	3	4	8	8	4	8	8

After sorting:

ParticleIndex	0	1	2	5	3	4	6	7
ParticleHash	0	3	4	4	8	8	8	8

CellHash	0	1	2	3	4	5	6	7	8
CellStart	1	-	-	2	3	-	-	-	5
CellEnd	1	-	-	2	4	-	-	-	8

Potential contact pairs:

AList	8	2	3	3	4	4	5	6	7
BList	8	5	4	6	3	7	2	3	4

Figure 4. The schematic diagram of the uniform grid algorithm.

triangular facets; the mechanical parameters define the material properties of particles. After all information is loaded, the CPU pre-calculates the geometrical information of the particles, which will be applied to later computations. The geometrical information includes the normal and centroid vectors of each triangular facet of a particle. Then, the device memory is allocated, and all previous information is transferred into GPU.

The flowchart patched in orange color illustrates the operations executed by the GPU, which is the key to accelerating the code and is the focus of this study. In the traditional framework of DEM, the main process can be divided into four parts: update, neighbor search, contact detection, and integration. We also inherit the framework of the traditional DEM. At the beginning of each iteration, the geometrical information (namely the normal and centroid vectors of particles' triangular facets) and vertices of the particles are mapped into the inertia frame according to the calculations of the previous iteration. These two functions can be performed simultaneously. After these two steps are accomplished, the leaf and internal nodes of the linear bounding volume tree (LBVH) are updated based on the geometrical information and vertices of the particles. The most complicated and time-consuming operation in PolyDEM is to compute the interaction between particles. To precisely handle the contact between nonspherical particles, the LBVH technique is employed in the contact detection phase. In this section, we exhibit the overall flowchart, aiming to give a clear explanation of how the code works. Details about the code implementation can be found in Sections 3.2–3.4.

3.2. Neighbor Search

The simplest conceivable method is to directly check the contact between each pair of particles; however, this method would result in unacceptable calculation effort once the numerous particles are employed. To improve the execution speed of the program and reduce the computational load, the operation of neighbor search is introduced before detecting contact to determine the neighboring particles quickly. It is also the common practice of DEM. The uniform grid method is utilized in PolyDEM to search for neighboring particles (Peng et al. 2019). Figure 4 uses a simple example to show the steps.

The uniform grid method subdivides the computational domain into a grid of uniformly sized cells. The circumscribed sphere is computed for each individual nonspherical particle. The cell size of the grid must be greater than the diameter of the maximum circumscribed sphere of the particle, ensuring nonneglect of potential contact. Each cell of the uniform grid corresponds to a hash value, mapping the three-dimensional grid into a one-dimensional array. The number in the top left corner of the cell in Figure 4 (red) labels the hash value for each cell. Hash values of particles can also be computed according to their spatial positions. The particles with the same hash values indicate that they reside in the same cell. Hence, for each particle, we only need to examine its own located cell and 26 surrounding cells (27 cells in total) to identify the potential neighboring particles. Four kernel functions are used to accomplish this process:

(i) Calculate the hash values of each individual particle. The detailed formulas to calculate the hash value can be found in Zeng et al. (2022), which is omitted here for simplicity.
(ii) Sort particles according to their hash values. In this kernel function, two arrays, *ParticleIndex* and *ParticleHash*, are employed to record the original indices and hash values of the particles. The dimensions of these two arrays are equal to the number of particles. Then, the particles are sorted based on their hash values, and the order of the *ParticleIndex* array is also adjusted accordingly, as illustrated in Figure 4.
(iii) Aided by the sorted *ParticleHash* array, two arrays, *CellStart* and *CellEnd*, are employed to record the index of the beginning and ending particles in each cell. The dimensions of these two arrays are equal to the number of the three-dimensional uniform grid. Using the index recorded in *CellStart* and *CellEnd* arrays makes it convenient to ascertain the number of particles in each cell. By visiting the sorted *ParticleIndex* array, it is also convenient to determine which particle resides in the cell. For example, in the cell with a hash value of 8 in Figure 4, *CellStart*[8] and *CellEnd*[8] correspond to the numbers 5 and 8, respectively. It means this cell owes four particles. In the sorted *ParticleIndex* array, the fifth to eighth elements reside in this cell. In this way, we do not record all particle indexes for each cell, thus reducing memory overhead.
(iv) For each particle, the 27 cells are traversed. Suppose the circumscribed spheres of two particles overlap each other. In that case, the indexes of the two particles are identified as potential contact pairs and stored in two arrays, *AList* and *BList*, respectively. The elements *AList*[0] and *BList*[0] store the total number of the potential contact pairs, as

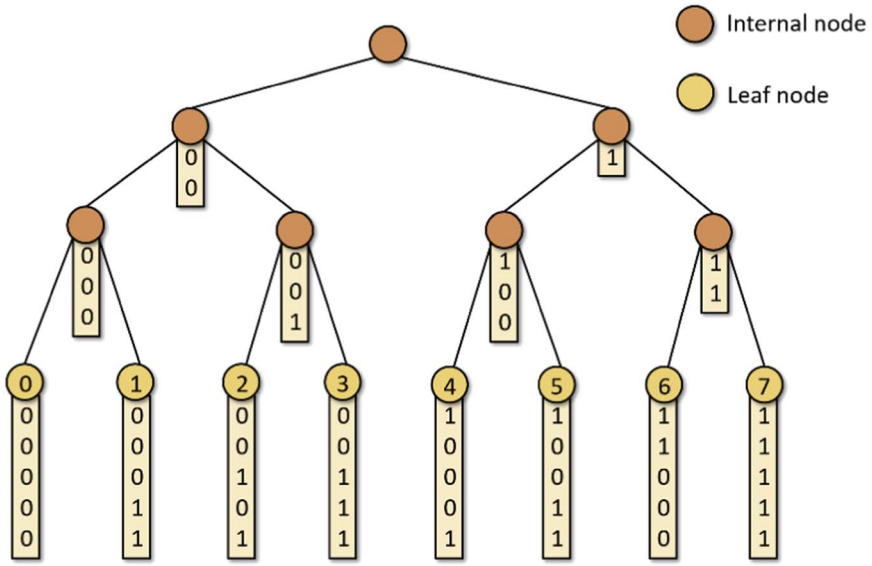


Figure 5. The hierarchies of the BV tree generated by using Morton code.

shown in Figure 4. Consequently, based on these two arrays, we are able to perform parallel computation based on the contact pairs in the subsequent process.

3.3. Contact Detection by Using LBVH

Contact detection between nonspherical particles is carried out by using BVH in the previous CPU code. We use the recursive-tree method in the CPU version to find the overlapped BVs. It is unsuitable for the GPU framework since GPU is much better at handling large numbers of parallel tasks with low complexity. Implementing such a complicated program cannot make the GPU at full compacity. Additionally, the method requires constant allocation and release of memory. The dynamic memory management on GPU is quite cumbersome. The execution time slows down significantly as well. For these reasons, we have abandoned the recursive-tree approach on the GPU platform. Instead, we adopt the LBVH, which is the depth-first method and can use the full performance of the GPU. Although the GPU-LBVH and previous CPU-BVH both create BVs surrounding the body to detect contact, the workflows of these two algorithms are radically different. The following contents illustrate the implementations of LBVH on the GPU.

Reviewing Figure 3, the BV tree for each particle is created first before the main loop of the program. In the main loop, since the topological information of the particle shape model is not changed, we do not need to reconstruct the hierarchical tree at each time step. It only needs to update the BVs according to the position and attitude of particles at each time step. After the neighbor search phase, all potential contact pairs are identified. LBVH is used to precisely determine the contact regions between nonspherical particles and calculate the contact force.

The first assignment is to construct the BV tree. In LBVH, the construction of the BV tree is reduced to a sorting problem by using the Morton code. The Morton code can map the three-dimensional data to one-dimension and preserve the locality of the data points at the same time. It defines a space-filling curve where the data points owning neighboring coordinates have close Morton codes. Readers may find the computations of the Morton code in Karras & Aila (2013).

The shape model of each particle consists of vertices and triangular facets. Each particle's Morton code for each triangular facet is computed according to its centroid coordinates. The centroid coordinates are sorted in increasing order of their Morton code. The corresponding triangular facets are also ordered in a spatially coherent way. Then, it is simple to construct the BV tree based on ordered Morton codes. The pseudocode is shown in Karras & Aila (2013). Here, we briefly describe the algorithm. Figure 5 is used as an auxiliary to explain the algorithm.

The Morton codes are represented by using binary strings. Binary search is employed to find the first different bit of the Morton codes. Then, all BVs with the highest bit equal to zero are divided into the left child tree of the root node; correspondingly, all BVs with the highest bit equal to 1 are divided into the right child tree of the root node. Similarly, the same method is applied to recursively split the current leaf nodes into the child trees in the next level. The BV tree is generated until the final bit of the Morton code is visited.

In the main loop, the BVs of the leaf and internal nodes should be updated at each time step. A shape model of a particle consisting of N_v vertices and $N_f = 2N_v - 4$ facets have N_f leaf nodes and $N_f - 1$ internal nodes. The BVs of the leaf nodes are updated by finding the maximum and minimum coordinates of the triangular facets. The internal nodes are updated by using a bottom-up reduction algorithm. Before entering the main loop, the hierarchical structure of the BV tree is pre-generated for each particle; in other words, we already know the parent and child nodes for each node. Therefore, we assign a single leaf node per thread and proceed to the root node. If the BV of the given node is not calculated, the thread finds the BVs of its child nodes and calculates their union, with an atomic flag indicating that the BV of the given node is already computed. If the BV of the given node is already calculated, the thread proceeds to find its parent and calculate the BV of its parent. This atomic flag is quite useful. It can avoid the duplicate calculation of a node and ensure the node is not processed before its children are processed.

In the flowchart of Figure 3, contact detection between potential contact pairs is performed after the neighbor search operation. Recursive searching for intersected BVs between

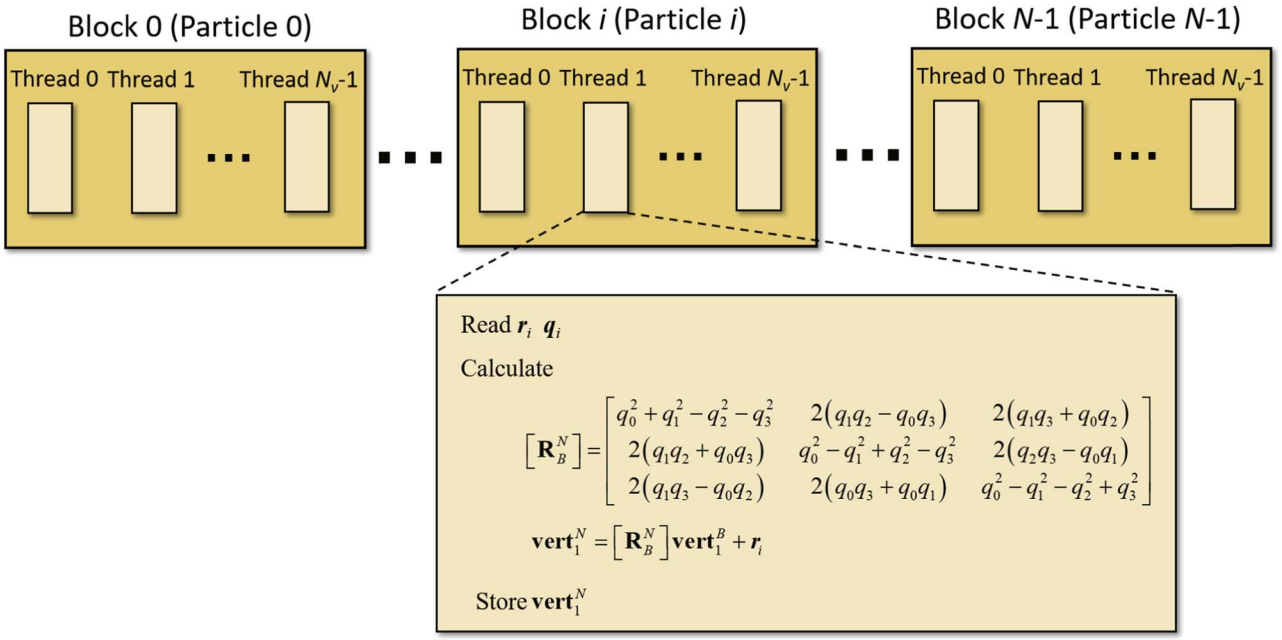


Figure 6. Thread block allocation for updating the particle vertices.

potential contact pairs is an efficient method on the CPU, but it would lead to high execution divergence on GPU. The strategy of independent traversal is applied to fix this issue. For each thread, we check whether each leaf node of the particle from *AList* (represented as particle *A* for simplicity) overlaps with the node of the particle from *BList* (represented as particle *B*). Note that *AList* and *BList* are recorded in the neighbor search step (see Figure 4). The algorithm first checks the overlap between the leaf node from *A* and the children of the current node from *B* and reports an intersection if one of the child nodes from *B* is also a leaf node. If the current node from *B* is an internal node, but it reports overlap with the leaf node from *A*, the algorithm continues iterating. If the current node only has one child report overlap, its child is set as the current node to start over. If both two child nodes report overlap, the left child is set as the current node to perform the same operation, while the right child is pushed into the stack. A node is popped off the stack if there are no children to be traversed. The loop ends once the value popped off of the stack is null. After this process, it reports the intersection between two leaf BVs, which is the minimum unit of the BV tree. The above steps are executed for each leaf node of particle *A*. Particle *A* owns N_f triangular facets, meaning it also has N_f leaf nodes. Therefore, contact detection steps will be executed for N_f times to verify contact between a pair of particles. Generally speaking, an irregular particle has hundreds of triangular facets. For each iterative step, the function for the contact detection step would perform $N \times N_f$ times, where N denotes the number of potential contact pairs. The calculative scale seems to be great. Nevertheless, GPU has thousands of CUDA cores to support large-scale computations. The algorithm to perform contact detection is decomposed into fine granularity and is highly parallel by organizing the thread blocks on the GPU. It can fully utilize GPU computing resources and is computationally efficient, which will be elaborated on in Section 3.4. Then, the algorithm queries the triangular facets enveloped by the leaf BVs for overlap. The corresponding algorithm is consistent with the previous CPU code and is omitted here for simplicity (see Wen et al. 2023; Zeng et al. 2022 for more details). Finally, the contact force and torque of a pair of intersected triangular facets, i.e., active contact elements, can be obtained through Equation (2). The resultant contact force and torque between the contact pairs can be obtained by summing up the calculated force and torque from the triangular facets, as presented in Equation (3). Once the contact force and torque between the potential contact pairs are calculated, we use another kernel function to obtain the resultant contact force and torque on each individual particle.

3.4. Organization of Thread Blocks in PolyDEM

Except for the algorithm, the organization of thread blocks is another essential factor in determining the kernel function performance, as it determines the parallel granularity. We attempted to allocate a single particle for each thread in the early state of PolyDEM development. Nevertheless, such an organization strategy leads to too many loops in a single thread and risk processors idle even though there is plenty of work to be processed. To effectively utilize the available hardware and squeeze the computing power of the GPU, we excavate finer-grained parallelism in PolyDEM by configuring reasonable threads and blocks. The following are detailed implementations.

Reviewing the flow diagram in Figure 3, the first step in the main loop is to update the geometrical information and vertices of particles according to their position and attitude information. We use a one-dimensional grid with one-dimensional blocks while driving these two kernel functions. By considering the data-access locality in the PolyDEM, a block is used to manage a particle. Each block uses a thread to manipulate a vertex or a triangular facet of the particle. Therefore, the grid size is naturally set to the total number of particles. The block sizes are set to the vertex number of particle shape models when updating the vertices and are set to the facet number of particle shape models when updating the geometrical information. Figure 6 illustrates the thread block allocation for the kernel function updating vertex information. In Figure 6, block i is used to manage particle i. In block i, N_v threads correspond to

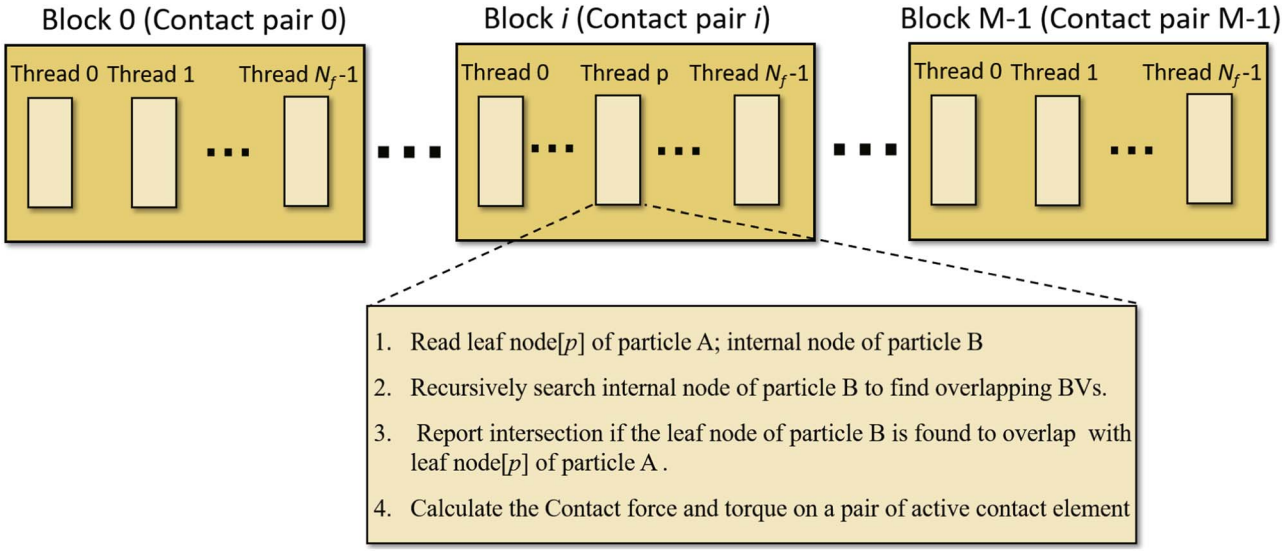


Figure 7. Thread block allocation for computing contact force and torque between potential contact pairs.

N_v vertices. Each thread in the same block i reads the position $\boldsymbol{r}_i$ and quaternion $\boldsymbol{q}_i$ of the particle, then calculates the vertex coordinates in the inertia frame by using the direction cosine matrix $[\boldsymbol{R}_B^N]$. The symbols vert_1^N and vert_1^B represent the vertex of the particle mapped in the inertia frame and body-fixed frame, respectively.

The same strategy is employed here to update the leaf and internal nodes of LBVH. Therefore, each thread is assigned very few operations, which enables parallel computation at a much finer level of granularity. All potential contact pairs are found and recorded in two arrays after the neighboring search. In the contact detection phase, the block number is equal to the number of potential contact pairs. We assign a contact pair per block, as shown in Figure 7. A block has N_f threads corresponding to the leaf node number of particle A. A thread is used to manage a leaf node of particle A. After the contact force between potential contact pairs is computed, we employ the atomic operation in the kernel function to obtain the resultant contact force and torque on each particle.

In general, two strategies are proposed to accelerate the computation speed: (1) Parallel the contact detection at the contact pair level rather than the particle level. (2) By organizing the thread blocks, the tasks of the kernel functions are decomposed into smaller granularity to exploit the code's parallelism fully.

4. Performance Evaluation and Discussions

This section presents two numerical examples to demonstrate the performance of PolyDEM. The first case is the collision between two spheres to check the precision of the code. The second is the self-gravitating aggregation of the rubble-pile asteroid to check its convergence property and computational efficiency.

4.1. Collision between Two Rigid Spheres

During the numerical implementations, multihierarchies of grids are utilized to improve computational efficiency. The uniform grid method is employed to search neighboring particles; the LBVH is employed to determine the active contact elements between two contacting bodies. In this section, we use spherical-shaped particles as an example to illustrate the differences in the code implementations between the spherical DEM code and the PolyDEM code.

In the PolyDEM, the shapes of particles are represented by surface triangular meshes. Hundreds of triangular facets are employed to record the shape characteristics of a spherical particle. Then, the LBVH is constructed based on the stored vertices and facets, aiming to detect the contact between the two particles. LBVH provides a hierarchical representation that could split the whole mesh into certain levels. The root node of the BV tree contains the entire triangular facets. Then, the BV tree grows from the root node: the root node is recursively partitioned into two separate parts that are enclosed by smaller BVs. These two separate parts will then keep partitioning until each small BV only enclose a triangular facet, which refers to the leaf node of the BV tree. Figure 8 presents a BV hierarchy of a spherical particle, which has 162 vertices and 320 facets. In the code, the data structure of the BV tree includes the maximum and minimum coordinates of current BVs, together with two pointers to the next level of LBVH; therefore, the occupied memory increase as the vertices and facet numbers of the mesh increases. Table 1 shows the memory occupied by spherical particle LBVH with different mesh resolutions. The first and second rows show the vertex and facet numbers of four models. The third and fourth rows show the depth and node number of the BV tree based on these four different models. The last row shows the minimum memory consumption of the BV tree in the computer. Beyond that, PolyDEM also stores the vertices, facets, normal vectors, and centroid vectors of facets for each particle. It can be seen that memory consumption increases rapidly with the increase in mesh resolution. Nevertheless, in the spherical DEM, only the radii of the spherical particles need to be recorded. Comparing the radii of the two spherical particles and the distance between their mass centers makes it convenient to know whether the two particles overlap. We only use 8 Bytes to store the radii of the sphere. Although the LBVH has high computational complexity and memory consumption, it could manage the different contact scenarios between irregular-shaped particles, which recuperated some disadvantages of the spherical DEM.

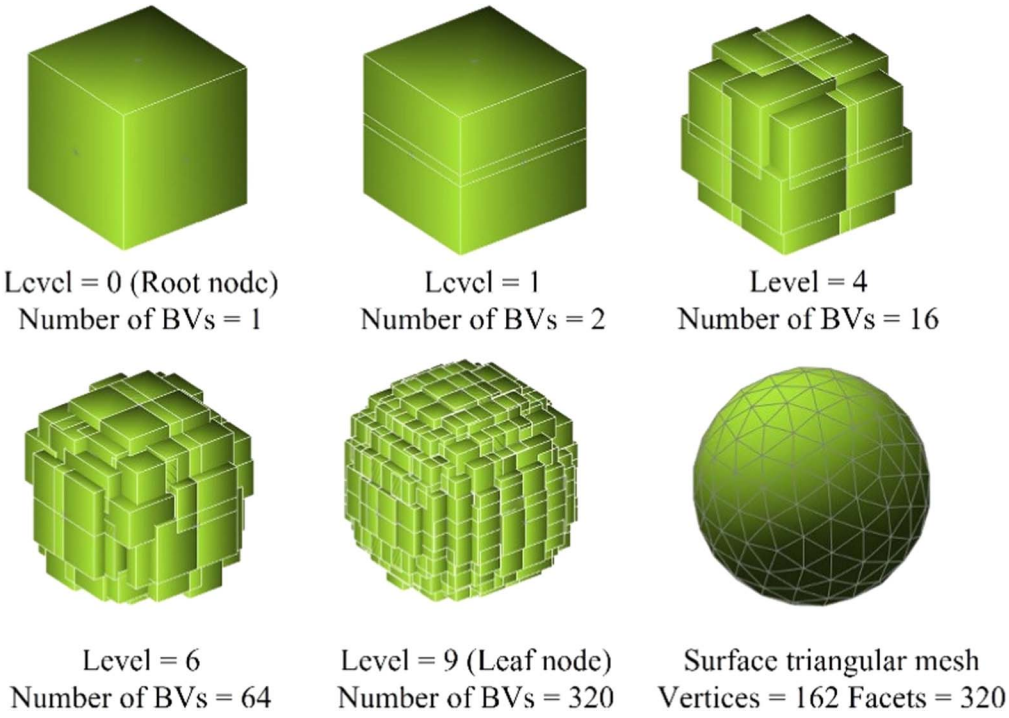


Figure 8. Illustrative levels of the LBVH of a sphere.

Table 1
BV tree of Spherical Particles with Different Mesh Resolutions

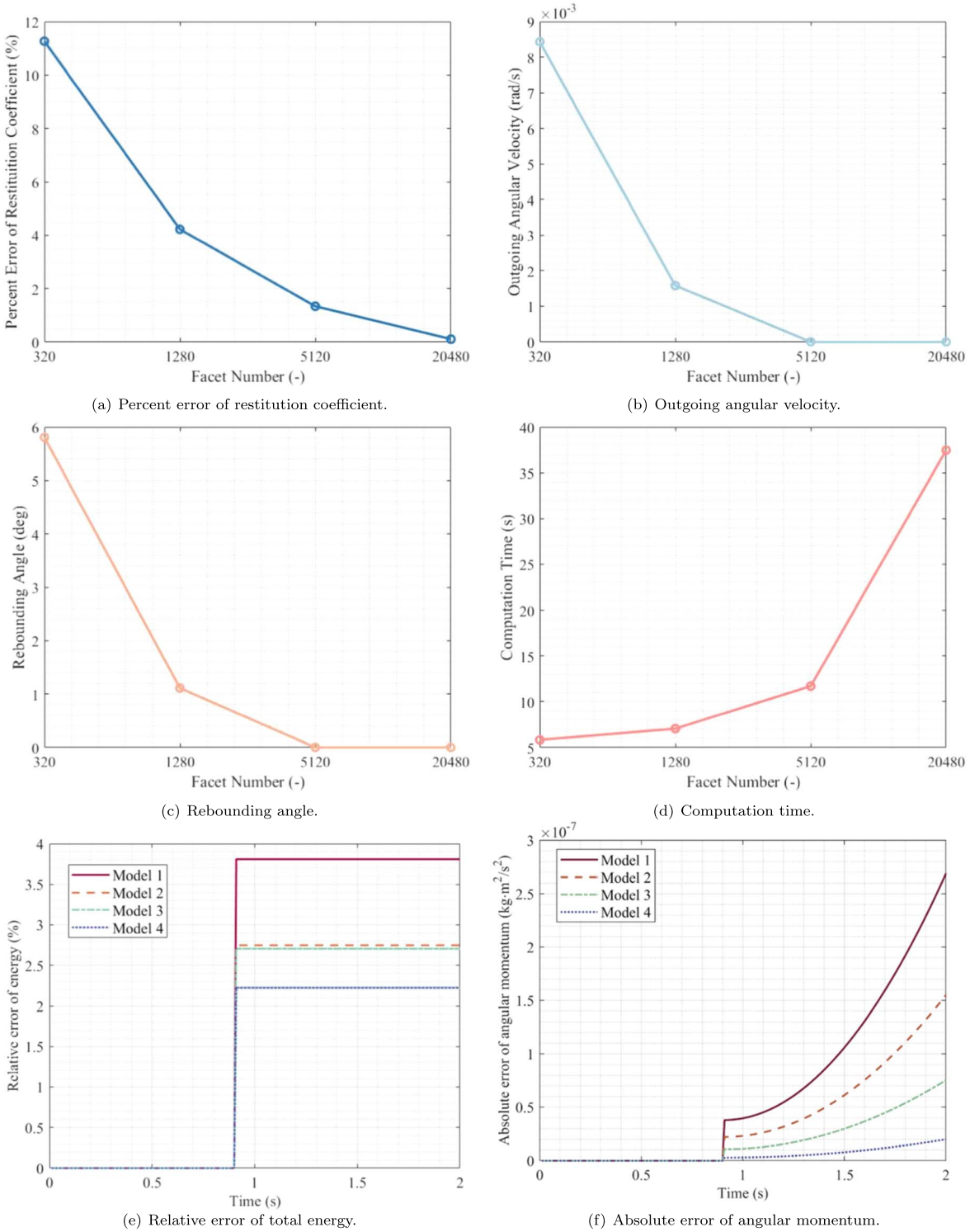
Index	Model 1	Model 2	Model 3	Model 4
Vertex number	162	642	2562	10,242
Facet number	320	1280	5120	20,480
BV tree depth	9	11	13	15
Node number	639	2559	10,239	40,959
Minimum memory consumption (bytes)	31,950	127,950	511,950	2,047,950

Then, we assess how the mesh resolution affects the computational accuracy using the same strategy as Zhan et al. (2021). Two spheres, each 0.05 cm in radius, are placed along the z-axis with a distance of 5 cm between their mass centers. Sphere A is released and drops freely with zero velocity under the gravity acceleration of 0.1 m s^{-2}. Sphere B is fixed. Both spheres are represented by using the surface triangular meshes with the same vertex and facet number. The restitution coefficient is set to 1.0. The damping and frictional forces are neglected. Such a case corresponds to a completely elastic collision. In an ideal scenario, sphere A would rebound vertically after the two spheres interact and return to the initial releasing position. We use this criterion to check whether the computation is accurate. Four simulation groups are performed, where the spheres have different mesh resolutions, as shown in Table 1 (Model 1 to Model 4). The orientations of both spheres are randomly initialized, and the tests are repeated 5 times for each group of numerical simulation to consider the influence of random alignments. Three indicators are used to assess the computational accuracy of different mesh models: the percent error of restitution coefficient, angular velocity, and rebounding angle. Figure 9 summarizes the average values of each indicator.

The restitution coefficient is calculated by using the velocity of sphere A before and after the collision. In Figure 9(a), the percent error of restitution coefficient is given by $e = |\|\boldsymbol{u}_{\text{out}}|/|\boldsymbol{u}_{\text{in}}| - 1|$, where the variables $\boldsymbol{u}_{\text{out}}$ and $\boldsymbol{u}_{\text{in}}$ are the outgoing and ingoing velocity, respectively. In Figure 9(c), the rebounding angle defines the deviation angle after sphere A rebounds and returns to the maximum height. Theoretically, the motion of the two spheres is a central and utterly elastic collision. The restitution coefficient should equal 1.0, and the outgoing angular velocity and rebounding angle should be zero. Figure 10 shows the motion trajectory of sphere A when Model 1 is adopted, where the rebounding angle is labeled as δ in Figure 10. It can be seen that the rebound trajectory of sphere A deviates from the freefall trajectory. Moreover, the angular velocity of sphere A is not equal to 0, according to Figure 9(b). The phenomenon can be interpreted as follows: The contact force is calculated through active contact elements, i.e., the intersection of triangular facets between two spheres. When the mesh resolution is low, the obtained contact force is not exactly along the z-axis, which induces the rotation of the sphere and a deviation of the rebound trajectory. The numerical error is obviously improved when the mesh resolution is increased. When the facet number reaches 20480, the numerical error is eliminated. However, it can be seen in Figure 9(b) that the computational time also increases rapidly. It has been argued that the error in the single particle level may not be significant on the particle assemblies. Therefore, we should make a trade-off between computational accuracy and efficiency when choosing the mesh models, and it is allowed to lose a little precision to ensure efficient computation.

Figures 9(e) and (f) only exhibit sphere A's energy and angular momentum error evolution since sphere B is constrained during the whole process. The relative error of sphere A's energy is computed as follows:

$$E_{\text{err}} = \frac{\left|\frac{1}{2}m\left|\boldsymbol{v}\right|^2 + \frac{1}{2}\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{J}\boldsymbol{\omega} + mgh - mgh_0\right|}{mgh_0}. \tag{7}$$



(a) Percent error of restitution coefficient.

(b) Outgoing angular velocity.

(c) Rebounding angle.

(d) Computation time.

(e) Relative error of total energy.

(f) Absolute error of angular momentum.

Figure 9. Indicators to assess the computational accuracy of different mesh models.

The symbols m and $\boldsymbol{J}$ denote the mass and inertia matrix of the sphere, $\boldsymbol{v}$ and $\boldsymbol{\omega}$ denote the velocity and angular velocity, and g is the gravitational acceleration. The symbols h and h_0 represent the height of sphere A during its motion and the height at the initial moment, respectively. Since the initial radial velocity and angular velocity are set to zero, then the initial angular momentum should also be zero. Therefore, Figure 9(e) shows the evolution of angular momentum in the form of absolute error. Before the two spheres interact with each other, the error curves are approximately zero. Jump points appear in the error curves after the two spheres interact with each other, as shown in Figures 9(e) and (f). This phenomenon indicates that the energy and angular momentum are not strictly conserved. Numerical errors and mesh discretization (discontinuous sphere surface) deduce this phenomenon. However, the errors in energy and angular

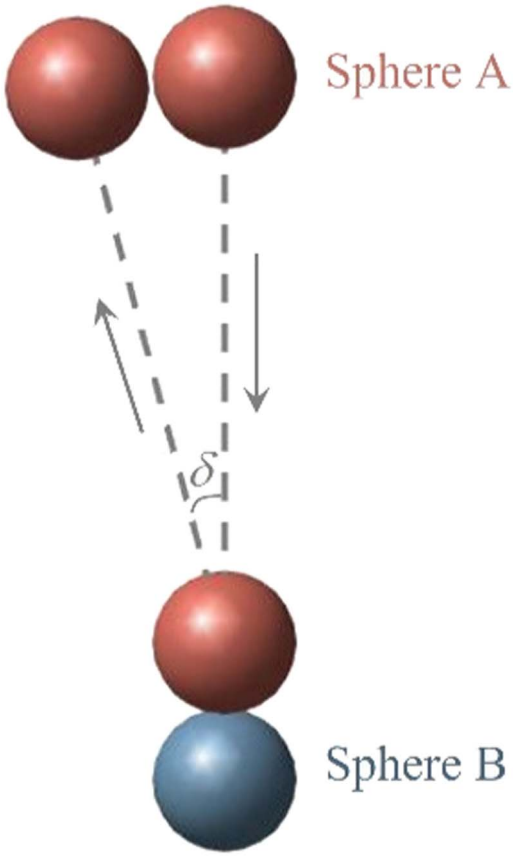


Figure 10. Motion trajectory of sphere A when Model 1 is adopted.

momentum are both at low levels. Furthermore, the errors in energy and angular momentum both decrease as the mesh resolutions improve, as shown in both Figures 9(e) and (f).

4.2. Simulation of Self-gravitating Aggregation

This section presents the gravitational accretion process of a rubble-pile asteroid by using PolyDEM. The aggregation process features both gravitational and collisional dynamics. As presented in Section 2.1, the force-based contact model is applied. Although the computational efficiency is lower than that of the impulse-based model, the force-based simulation is more suitable in regimes where the particles are in steady, long-lasting contact. The brute force method computes the mutual gravitational interactions between particle assemblies. The total gravitational force on particle i due to its interaction with the other $N-1$ particles can be given by Barnes (2012)

$$\boldsymbol{F}_g = Gm_i \sum_{1 \leqslant j \leqslant N} \frac{m_j \boldsymbol{r}_{ij}}{(|\boldsymbol{r}_{ij}|^2 + \varepsilon^2)^{3/2}} \tag{8}$$

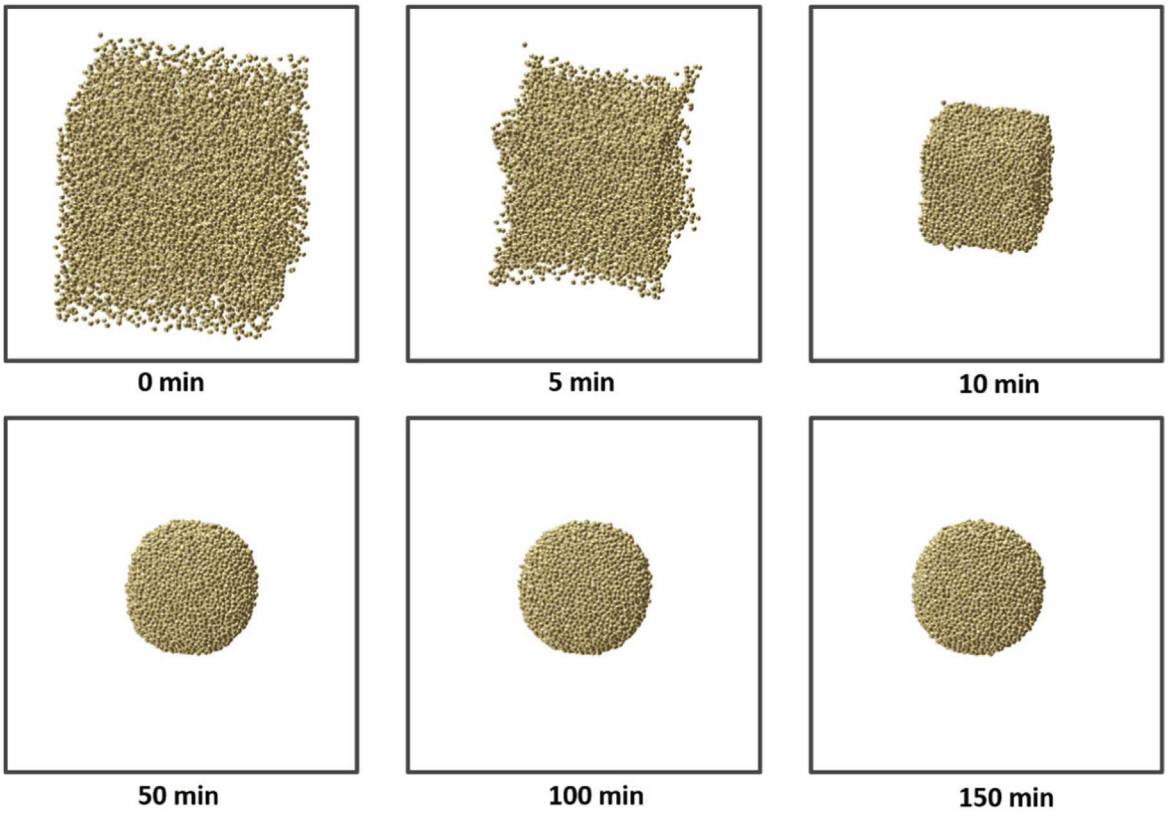
in which G defines the gravitational constant, and m_i and m_j define the masses of the particles i and j. The vector $\boldsymbol{r}_{ij}$ denotes the relative position between i and j. Introducing the softening factor ε^2 has two benefits: on the one hand, the softening factor can avoid singularity in computing $\boldsymbol{F}_g$ when $i = j$, since $\boldsymbol{F}_g = 0$. We do not need to exclude the self-gravity of individual particles, which would reduce the branch divergence of GPU and accelerate the computing speed. On the other hand, the softening factor limits the magnitude of the gravitational force between two very close particles and also avoids the singularity of the numerical integration. While the computational complexity of the brute force method reaches $O(N^2)$, the preponderance of GPU is exact in handling simple intensive computing tasks. Nyland et al. (2009) already gave the complete CUDA solution of the brute force method. Therefore, we directly integrated this routine into the PolyDEM.

In this numerical example, three sets of different numbers of particle assemblies are considered to discuss the computational efficiency of PolyDEM. The numbers of particles are 104, 1040, and 10400, respectively. To avoid undesirable effects induced by anisotropic physical properties of particles, the density of particles in these three groups are all 3.0 kg m^{-3}, and the shapes keep identical as well. The shape models of the polyhedron particles have 162 vertices and 320 triangular facets. The radii (i.e., the equivalent sphere of the polyhedron particles) are 84.93, 39.42, and 18.30 m, respectively. Such a parameter setting could guarantee the same total mass of the particle assemblies. For all simulations presented in this paper, Young's modulus is set to 1.6 MPa, the restitution coefficients are set to 0.2, and the friction coefficients are set to 0.5. The time step is fixed to 0.1 s to guarantee the numerical stability of the integrator.

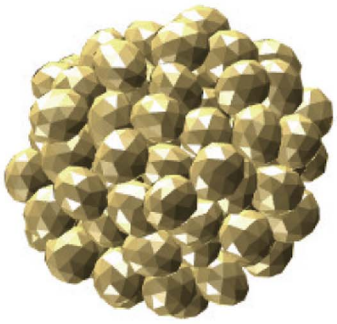
Previous studies have shown that the results are heavily dependent on the initial conditions of particles. To foster the formation of aggregates, the initial radial velocities of the particles are set to zero, and the initial angular velocities range from 0.0 to 1.732×10^{-3} rad s^{-1}. The accretion process for the 10400-particle assembly is exhibited in Figure 11(a). We obtain a similar pattern as presented by Sánchez et al. (2021). Particles accrete together relying on purely self-gravitational interactions. During evolution, particles come into contact with each other as they approach each other. Particles constantly configure their positions and attitude under the combined influence of gravitational and contact interactions. The shape of the aggregation gradually evolves from a cube into a sphere. Part of the energy is dissipated through interactions of particles. Finally, the particle system reaches a steady-state equilibrium in the form of a spherical cluster. In Figures 11(b) and (c), the 104 and 1040 particle assemblies also accrete into spherical clusters at 150 minutes.

Figure 12 presents the contact force chain of 10400-particle packing. It can be seen that the interparticle contact is dense. The interlocking particles jointly constitute dense chains of contact force. The magnitude and orientations of the contact force of the particle aggregates formed by self-gravity influence are evenly distributed. Figure 13 presents the statistics of the contact force magnitude for three particle assemblies. The variable $\overline{F}_c$ denotes the normalized contact force, which is calculated by using $\overline{F}_c = F_c / \langle F_c \rangle$. The variable $\langle F_c \rangle$ indicates the average magnitude of contact force. In Figure 13, the magnitude of contact force is classified into three categories, which correspond to weak, medium, and strong force chains. Although the numbers of the three particle assemblies are different, the distribution law of force chain magnitude is quite similar. Almost 90% of force chains fall into the range of medium force chains. Few force chains belong to weak or strong force chains.

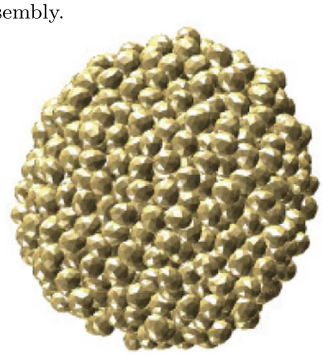
Figure 14 presents distributions of contact force orientations in the form of polar diagrams. The angular coordinate represents the orientation of the contact force, and the radial coordinate represents the probability distribution of the contact force. In Figure 14, the contact force orientations present



(a) Accretion evolution of 10400-particle assembly.



(b) Aggregates of 104-particle assembly at 150 min.



(c) Aggregates of 1040-particle assembly at 150 min.

Figure 11. Shapes of aggregates of three numerical simulations.

homogenous properties when the particle number is large. The 10400-particle assembly has the most uniform distributions of contact force orientations. For 104-particle assembly, the polar diagram presents nonuniform characteristics in radial coordinates. Despite the radial coordinates being different for 104-particle assembly, the angular coordinates almost cover 360°. Since the particle number is limited, the statistical characters of the 104-particle assembly are not representative enough.

Figure 15(a) illustrates the total momentum evolution of 10400-particle aggregate during the accretion process. Figure 15(b) corresponds to the relative error of angular momentum. The initial velocities of all particles are set to zero. Theoretically, the momentum of particle assemblies should keep zero during the whole process. The angular momentum should keep constant as well. Nevertheless, momentum fluctuates around zero; the angular momentum errors also drift over time. These errors result from three sources: The first is the discretized mesh description of irregular-shaped particles. Such approximation would lead to an error in contact position and penetration calculations, bringing additional angular velocity for the particles, as shown in Section 4.1. Second, the integration method for the angular velocity (see Equation (5)) is not momentum-conserved, which also contributes to the drift of the angular momentum. Lastly, the floating-point number is used to store and operate decimals in computers. Although we use the double-precision floating number in PolyDEM to decrease the numerical errors as much as possible, the effective number of bits is only 15 or 16. Since the total mass of the particle assembly is great ($\sim 8.0052 \times 10^{11}$ kg), the tiny errors in decimals are also magnified. The double-precision floating number arithmetic still produces the significant error. We attempted to decrease the integration step, but still, the error in the momentum and angular momentum cannot be eliminated entirely. Although

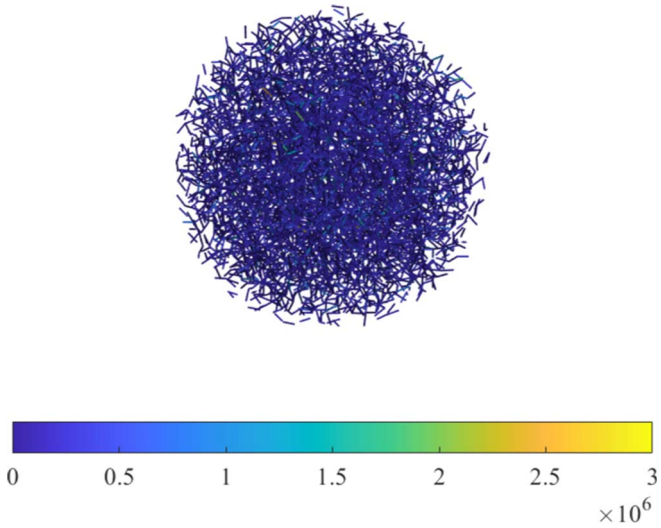


Figure 12. Force chain of 10400-particle aggregates.

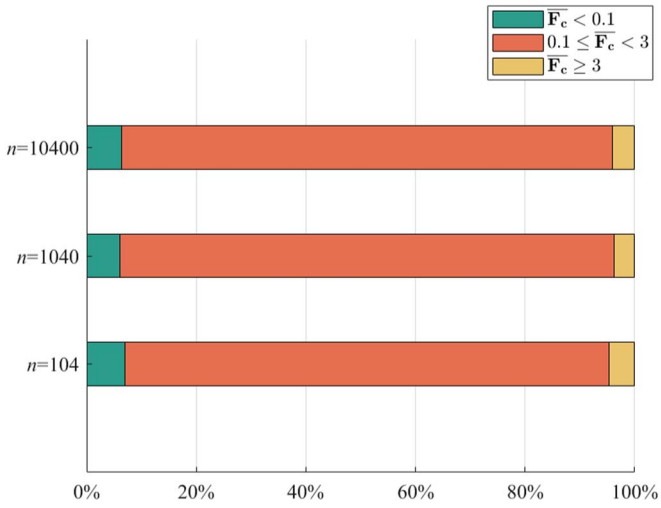


Figure 13. Statistics of contact force magnitude.

errors in numerical calculations are inevitable, they are controlled at a low level and are acceptable for current research.

Finally, numerical details of the time spent on PolyDEM are discussed. Results presented in this paper were obtained using a GPU server, and the hardware configuration of our computing platform is shown as follows:

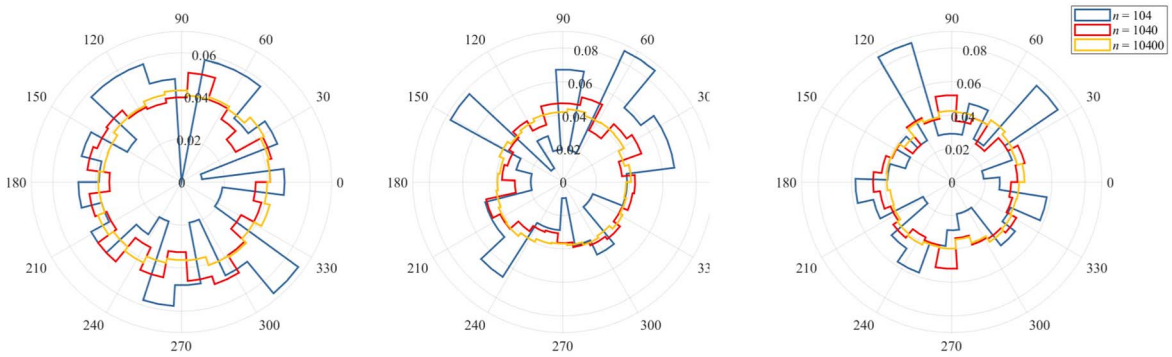
(i) Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz;
(ii) NVIDIA Tesla P100-PCIE-16GB.

The computational time is 13 minutes 53 s, 23 minutes 32 s, and 3 hr 4 minutes for 104-particle, 1040-particle, and 10400-particle assemblies. The computing time based on GPU is proportional to the square of the particle numbers. To further check the acceleration of PolyDEM, the single-core CPU-based code is used to simulate the accretion of the 10400-particle assembly. The implementation of the CPU-based code follows the same algorithms described in Zeng et al. (2022). The CPU simulations are performed using Intel(R) Xeon(R) Gold 6138 CPU @2.0 GHz. We only execute the first 100 computational steps since the single-core CPU-based code is very slow with such a large number of particles. It shows that the computing speed of PolyDEM is about 688 times faster than that of CPU-based code, which is improved quite remarkably. We also use the command-line tool *nvprof* to analyze GPU resource consumption, and the consumption of each module is summarized in Figure 16. In terms of calculation time, the kernel function for calculating the contact force between contact pairs takes up the most resources. This is in line with expectations since the contact detection between polyhedral particles is a very complicated problem. The numerical examples presented here lie in the dense contact regime, leading to high computational cost on contact detection. Furthermore, we notice that the kernel function for calculating the mutual gravity only occupies 2.61% of the total time, indicating that the brute force method is still very efficient on the GPU platform.

5. Conclusions

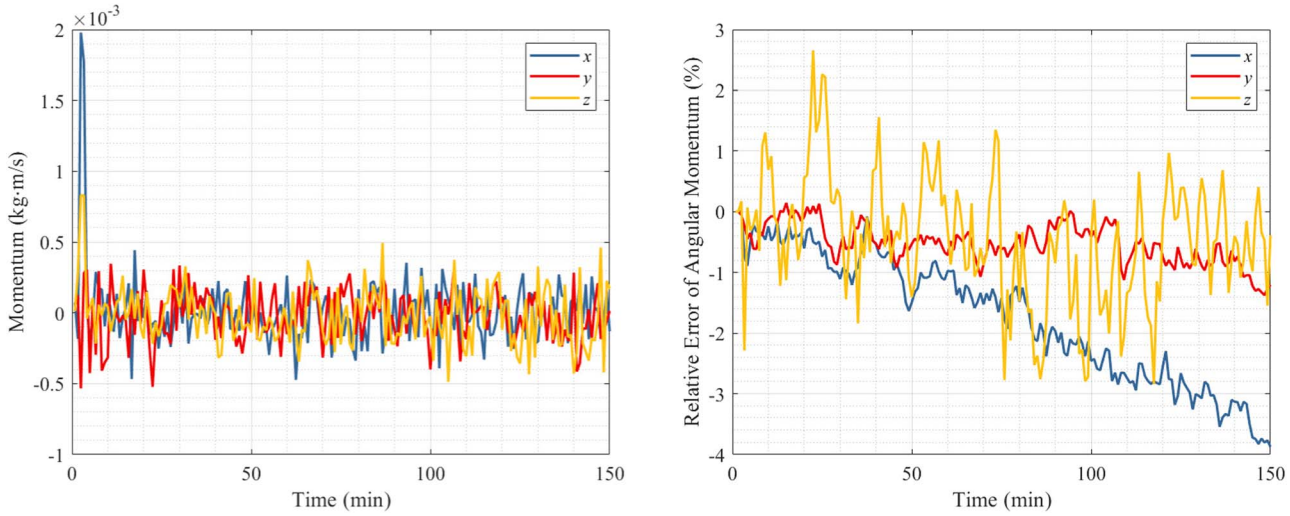
This paper designs a GPU parallel method to accelerate the PolyDEM. Two strategies are proposed to enhance the computational efficiency: (1) Parallel contact detection and contact force calculation at the contact pair level rather than the particle level. The contact detection and contact force calculation of each particle is decomposed into several kernel functions to avoid deep nesting of the program. (2) Reduce the loop numbers in kernel functions by organizing the thread blocks reasonably. Both two strategies transform complicated tasks into simple tasks, which can implement fine-grain parallelism effectively and help hide communication latency. The LBVH technique is introduced in the PolyDEM for the most time-consuming, complex, and essential contact detection process. The construction, update, and query of the BVs can be realized by using traversal. In the CPU-based code, the traditional BVH uses an iterative way to accomplish the above tasks. It is unsuitable for GPU architecture since it would cause severe branch divergence and slow down the computing speed of the program. Nevertheless, the traversal of LBVH is able to minimize the branch divergence and enable the best performance of the GPU.

Two benchmarks are used for the verification of the PolyDEM. The influence of mesh resolution on computational efficiency and accuracy is discussed in the first scenario by simulating the collision between two rigid spheres. The results show that the fine mesh resolution will increase the precision but decrease the computing speed. One should make a trade-off between the computational efficiency and accuracy. The self-gravitating aggregation scenario of a rubble-pile asteroid is carried out as the second test by varying the number of particles. Numerical results show that the accretion process of particles is in agreement with previously published research. The particles aggregate into sphere clusters due to the pressure gradient of self-gravity. The internal force chains are interlaced and evenly distributed in the aggregation. In summary, PolyDEM can easily achieve hundreds of times more acceleration than that of a single-core CPU-based code. Even though the force-based contact model is employed, PolyDEM can achieve high computational efficiency and numerical stability as well. The promotion of computing speed makes it possible for large-scale numerical simulations by expanding its potential applications of PolyDEM. In the future, the PolyDEM is expected to be applied to the high-fidelity simulation of the planetary granular system and provide new insight into irregular-shaped granular matters.



(a) xoy plane (b) yoz plane (c) xoz plane

Figure 14. Distributions of contact force orientation.



(a) Momentum. (b) Relative error of angular momentum

Figure 15. Evolution of momentum and angular momentum error during the accretion process.

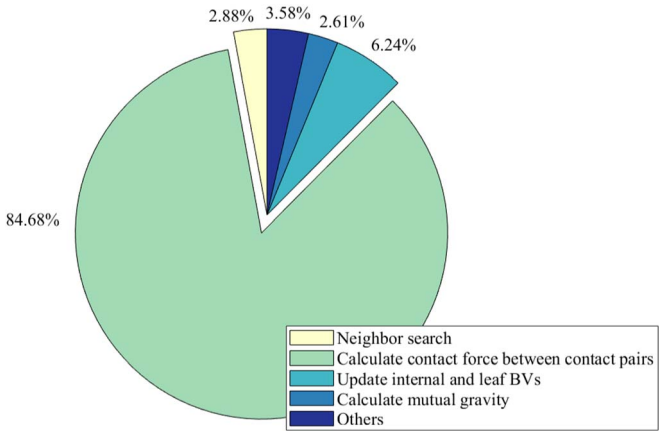


Figure 16. Resource consumption of each module on GPU.

Acknowledgments

Thoughtful discussions on the GPU implementations from Dr. Guangyu Liu at Tsinghua University and Dr. Chong Peng at ESS Engineering Software Steyr GmbH are appreciated. The authors would like to acknowledge the support from the National Natural Science Foundation of China (Nos. 12222202 11972075).

ORCID iDs

Tongge Wen https://orcid.org/0000-0001-9883-7025

References

Asphaug, E., & Benz, W. 1996, Icar, 121, 225
Barnes, J. E. 2012, MNRAS, 425, 1104
Cheng, B., Yu, Y., & Baoyin, H. 2017, NatSR, 7, 10004
Cheng, B., Yu, Y., & Baoyin, H. 2018, PhRvE, 98, 012901
Cotto-Figueroa, D., Statler, T. S., Richardson, D. C., & Tanga, P. 2015, ApJ, 803, 25
Ferrari, F., Lavagna, M., & Blazquez, E. 2020, MNRAS, 492, 749
Ferrari, F., Tasora, A., Masarati, P., & Lavagna, M. 2017, Multibody Syst. Dyn., 39, 3
Güttler, C., von Borstel, I., Schräpler, R., & Blum, J. 2013, PhRvE, 87, 044201
Hertz, H. 1881, Reine Angew. Mathematik, 92, 156
Hippmann, G. 2004, Multibody Syst. Dyn., 12, 345
Karras, T., & Aila, T. 2013, in HPG '13: Proc. 5th High-Performance Graphics Conf. (New York: ACM), 89
Liu, G. Y., Xu, W. J., Zhou, Q., & Zhang, X. L. 2022, Int. J. Geomech., 22, 12
Liu, S., Chen, F., Ge, W., & Ricoux, P. 2020, J. Partic., 49, 65
Lu, G., Third, J., & Müller, C. 2015, ChEnS, 127, 425
Michel, P., Benz, W., Tanga, P., & Richardson, D. C. 2001, Sci, 294, 1696
Michel, P., & Richardson, D. C. 2013, A&A, 554, L1

Nyland, L., Harris, M., & Prins, J. 2009, GPU Gem, 3, 677
Peng, C., Wang, S., Wu, W., et al. 2019, Acta Geotech., 14, 1269
Richardson, D. 2000, Icar, 143, 45
Richardson, D., Michel, P., Walsh, K., & Flynn, K. 2009, P&SS, 57, 183
Sánchez, P., Renouf, M., Azéma, E., Mozul, R., & Dubois, F. 2021, Icar, 363, 114441
Sánchez, P., & Scheeres, D. J. 2011, ApJ, 727, 120
Schwartz, S. R., Michel, P., Jutzi, M., et al. 2018, NatAs, 2, 379
Schwartz, S. R., Richardson, D. C., & Michel, P. 2012, Granular Matter, 14, 363
Wen, T., Zeng, X., Circi, C., & Gao, Y. 2020, JGCD, 43, 1269
Wen, T., Zeng, X., Li, Z., & Zhang, Y. 2023, P&SS, 226, 105634
Xie, C., Song, T., & Zhao, Y. 2020, J. Pow. Tech., 368, 253
Yu, Y., Richardson, D. C., Michel, P., Schwartz, S. R., & Ballouz, R. L. 2014, Icar, 242, 82
Zeng, X., Wen, T., Yu, Y., Cheng, B., & Qiao, D. 2022, Icar, 387, 115201
Zhan, L., Peng, C., Zhang, B., & Wu, W. 2021, J. Pow. Tech., 377, 760
Zhang, Y., & Michel, P. 2020, A&A, 640, A102
Zhang, Y., & Michel, P. 2021, Astrodynamics, 5, 293