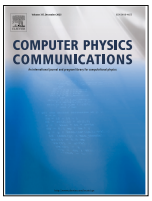
Computer Physics Communications 320 (2026) 109942

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc



Computer Programs in Physics

TinyDEM: Minimal open granular DEM code with sliding, rolling and twisting friction

Roman Vetter *

Department of Biosystems Science and Engineering, ETH Zürich, Schanzenstrasse 44, 4056, Basel, Switzerland

ARTICLE INFO

Dr. J Ballantyne

Keywords:
Particles
Granular media
Discrete element method
Friction
Quaternion

ABSTRACT

This article introduces TinyDEM, a lightweight implementation of a full-fledged discrete element method (DEM) solver in 3D. Newton's damped equations of motion are solved explicitly for translations and rotations of a polydisperse ensemble of dry, soft, granular spherical particles, using quaternions to represent their orientation in space without gimbal lock. Particle collisions are modeled as inelastic and frictional, including full exchange of torque. With a general particle-mesh collision routine, complex rigid geometries can be simulated. TinyDEM is designed to be a compact standalone program written in simple C++11, devoid of explicit pointer arithmetics and advanced concepts such as manual memory management or polymorphism. It is parallelized with OpenMP and published freely under the 3-clause BSD license. TinyDEM can serve as an entry point into classical DEM simulations or as a foundation for more complex models of particle dynamics.

PROGRAM SUMMARY
Program Title: TinyDEM
CPC Library link to program files: https://doi.org/10.17632/yv68h6nn9c.1
Licensing provisions: BSD 3-clause
Programming language: C++11
Supplementary material: Videos 1–6
Nature of problem:
Dynamics and statics of polydisperse ensembles of visco-elastic, frictional, non-adhesive spherical particles (such as in granular media) in 1D, 2D and 3D. All three modes of torque exchange (sliding, rolling and twisting) are modeled with slip-stick Coulomb friction.
Solution method:
The discrete element method is used to solve Newton's damped equations of motion for particle translations and rotations with the semi-implicit Euler scheme. Quaternions are used to represent particle orientations. For efficient collision detection, a linked cell list is used. A static geometrical environment can be defined with a discrete mesh. The program is parallelized with OpenMP for shared-memory systems.
Additional comments including restrictions and unusual features:
The source code is exceptionally compact, consisting of only about 600 commented lines in two files—a header and a source file. With no dependencies, it is highly portable and accessible, making it also suited for educational purposes.

1. Introduction

This paper does not introduce new concepts or methods. It does not solve a long-standing or particularly challenging problem, nor does it allow to simulate something that was impossible before. It is not concerned with relativistic or quantum physics. Instead, this article has something to offer that is rarely found in published computer programs: It introduces a lean, easy-to-read standalone implementation of an elementary ingredient for granular (macroscopic) particle simulations, named TinyDEM. TinyDEM is unique not in *what* it does, but in *how* it

* Corresponding author.
E-mail address: vetterro@ethz.ch

https://doi.org/10.1016/j.cpc.2025.109942
Received 18 July 2025; Received in revised form 9 October 2025; Accepted 6 November 2025
Available online 19 November 2025

does it. If you are a student or developer of own simulations of particle dynamics, looking for a simple DEM program to use, or just interested in a neat coding example, read on.

The discrete (or disjunct) element method (DEM) was pioneered by Cundall and Strack who developed the first 2D implementation named BALL in the 1970s [1]. In the DEM, a granular medium is represented by an ensemble of spherical or non-spherical particles that move, rotate and collide with each other, exchanging forces and torques in the process. The first open 3D granular DEM implementation with a source code that is still available today appears to be YADE-DEM from 2008, which is integrated in the multi-purpose physics simulation framework YADE [2]. Numerous other codes with varying degrees of complexity and capability followed, published mostly under a GNU General Public or BSD license. They are reviewed in Table 1. It is fair to say that a good selection of powerful open-source DEM programs are available nowadays, at least for a technically versed audience. Nine popular ones of them were recently compared and benchmarked [3], revealing largely good agreement between model predictions, despite occasionally varying implementation details.

Why introduce yet another one, then, one might ask. From the code volume quantification in Table 1, it becomes evident that, although writing good and versatile DEM code is not rocket science, it is usually still a considerable endeavor. There is no complete classical DEM code available with under 10,000 lines of code, which is easily doable in principle, as will be shown here. Fig. 1 shows this graphically: Published DEM frameworks often exceed 100,000 lines of code (including whitespace and comments), even when the friction model used for particle collisions is incomplete. While this is certainly in part because DEM code is often embedded in larger multiphysics frameworks whose functionality far exceeds that of just the DEM part, it sets the hurdle to start new computational DEM projects unnecessarily high. Large frameworks can be difficult to install, operate, edit, and maintain. Only two open DEM codes (ppohDEM and Blaze-DEM) have a code volume lower than 10,000 lines.

Fig. 1 highlights an additional limitation of the current body of open-source DEM frameworks: Most of them model frictional collisions only partially. Appropriate representation of particle rotations and friction is important in crack propagation [31], soil mechanics simulations [32], mixing of grains [33], collapse of granular assemblies [34], and to measure stress-strain relations of granular materials [28]. Out of 25 reviewed codes, only five model all three modes of frictional torque exchange: sliding, rolling, and twisting. Ten include only sliding friction,

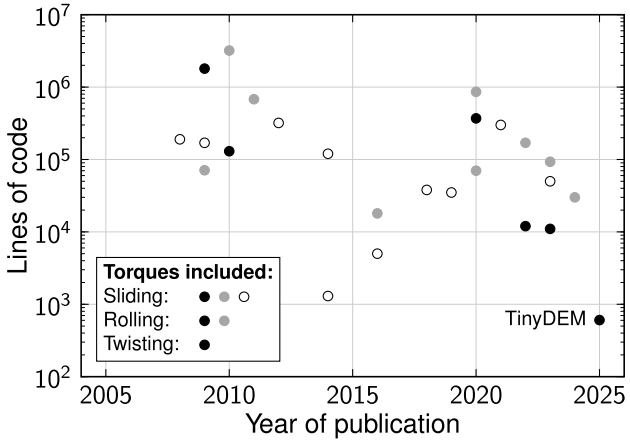


Fig. 1. Volume and level of detail of the contact model of published open-source soft-sphere DEM codes. Code volume is measured in number of lines as detailed in Table 1, and includes features beyond the DEM part, where applicable. Codes represented by a filled dot have all three modes of torque included in their friction model, gray dots lack the twisting friction, and empty dots lack both rolling and twisting friction.

some of them limited to history-independent kinetic friction, making them unsuited for the simulation of particle configurations in static equilibrium. Both codes below 10,000 lines lack rolling and twisting friction models.

There is clear value in simple standalone code—be it for education, method development, or other purposes, especially when code reusability is a concern. Several other lightweight programs that focus on doing one thing very well have demonstrated this paradigm: TinyFSM [35], SVL [36], jsmn [37], TinyXML-2 [38], tinyMD [39], MMM1D [40], PolyHoop [41], CImg [42], and the suite of mini-apps contained in the Mantevo project [43] are examples of minimalistic code that can be useful in many situations, from small student projects to large software libraries. They are often published under permissive licenses, which enhances their usefulness.

In this spirit, this work introduces TinyDEM, an exceptionally compact, portable and easy-to-use standalone DEM simulator that includes all three modes of frictional torque exchange, and an accurate representation of particle orientation with quaternions. TinyDEM is released under the BSD 3-clause license that permits both commercial and non-commercial use. It consists of only about 600 lines of commented C++11 code, has no dependencies, and is parallelized with OpenMP, requiring no access to high-performance computer networks or GPUs. The source code is designed not only for compactness and readability, but also to be as direct and instructive as possible, containing no magical numbers, numerical tolerances, explicit pointer arithmetics, manual memory management or polymorphism beyond what the standard template library provides. A unified collision routine handles both particle-particle and particle-mesh contacts with the same general code.

This article serves as TinyDEM's comprehensive documentation: All essential program features are described in the text, and in turn, all physics described here can be found in this form in the source code. After a detailed model description, some classical example simulations are showcased, and instructions for installation, usage and postprocessing are given. Finally, a computational benchmark is provided as a basis for runtime and memory requirement estimation in the user's specific scenario.

2. Particle model

2.1. Equations of motion

Like many other soft-sphere DEM codes, TinyDEM solves the damped Newton–Euler equations for translation and rotation. Each particle $i = 1, \dots, N$ in the ensemble carries its own position vector $\vec{x}_i$, velocity $\vec{v}_i$, orientation $\vec{q}_i$ (discussed later), angular velocity $\vec{\omega}_i$, and radius R_i. Particles move according to [33]

$$m_i \frac{d\vec{v}_i}{dt} = \sum_{j \neq i} \left(\vec{f}_{\mathrm{n}}^{ij} + \vec{f}_{\mathrm{s}}^{ij} \right) + m_i \vec{g} - c_i m_i \vec{v}_i \tag{1}$$

$$I_i \frac{d\vec{\omega}_i}{dt} = \sum_{j \neq i} \left(\vec{\tau}_{\mathrm{s}}^{ij} + \vec{\tau}_{\mathrm{r}}^{ij} + \vec{\tau}_{\mathrm{t}}^{ij} \right) - \gamma_i I_i \vec{\omega}_i \tag{2}$$

where $m_i = \frac{4}{3}\pi\rho_i R_i^3$ is the particle mass, $I_i = \frac{2}{5} m_i R_i^2$ the moment of inertia. In general, the vector $\vec{g}$ can be any external acceleration field, but typically it represents the gravitational acceleration.

The last terms in Eqs. (1) and (2) are often used to enhance numerical stability or to introduce energy dissipation that slows down particle motion to drive the ensemble toward a static equilibrium with linear and angular damping rates c_i and γ_i. A convenient way to avoid having to tune these parameters individually is to assume drag in a (real or virtual) viscous fluid at low Reynolds numbers. Stokes's law [44] then relates both of them to a single material parameter, the dynamic viscosity η of the fluid:

$$c_i = \frac{6\pi\eta R_i}{m_i} = \frac{9\eta}{2\rho_i R_i^2}, \tag{3}$$

Table 1
Overview of open-source DEM programs. Only models that include frictional collisions of soft spherical granular particles in 3D are listed. Number of code lines, where available, are rounded and based on latest available versions at the time of research (January 2025), including comments, whitespace, and supplied dependencies, but excluding dependencies that are not supplied with the code. Where available, the year of publication of the accompanying paper is used; otherwise, the approximate introduction year of the core DEM functionality is taken. OS: Open source; HM: Hertz–Mindlin; DMT: Derjaguin–Muller–Toporov; JKR: Johnson–Kendall–Roberts; HKK: Hertz–Kuwabara–Kono.

Year	Name	Contact model	Implementation	OS	License	Ref.
2005	—	Hooke/HM/HKK, static & kinetic Coulomb sliding	Modified MD code "DL_POLY_2" in Fortran 90, MPI	partial (printed)	—	[4]
2008	YADE	HM, Mohr–Coulomb sliding	190,000 lines of C++/Python, OpenMP, several dependencies (boost, eigen, Qt, freeglut3 etc.)	yes	GPL 2	[2]
2009	ESyS-Particle	Hooke/HM, static & kinetic Coulomb sliding	170,000 lines of C++/Python, MPI, some dependencies (boost)	yes	Apache 2.0	[5]
2009	LAMMPS	Hooke/HM/DMT/JKR, static & kinetic Coulomb friction (sliding, rolling & twisting)	1.8 mio. lines of C++, MPI	yes	GPL 2	[6]
2009	MechSys	Hooke, static & kinetic Coulomb friction (sliding, rolling)	71,000 lines of C++, OpenMP, CUDA, several dependencies (boost, GSL, LAPACK etc.)	yes	GPL 3	[7]
2010	Woo	Various, static & kinetic Coulomb friction (sliding, rolling & twisting)	Fork of YADE, 130,000 lines of C++/Python, OpenMP, several dependencies (boost, eigen etc.)	yes	GPL 2	[8]
2010	Kratos	Various, static & kinetic Coulomb sliding, kinetic rolling	3.2 mio. lines of C++/Python, OpenMP, MPI, dependency on boost	yes	BSD-3	[9]
2011	LIGGGHTS	Hooke/HM, static & kinetic Coulomb sliding & rolling	680,000 lines of C++, MPI	yes	GPL 2	[10]
2012	MFIX-DEM	Hooke/HM, static & kinetic Coulomb sliding	320,000 lines of Fortran 90/Python (Conda), OpenMP; numerous dependencies	registration & approval required	—	[11]
2014	ppohDEM	Hooke, static & kinetic Coulomb sliding	Single file, 1300 lines of Fortran 90, OpenMP, MPI	yes	CPC non-profit	[12]
2014	GranOO	Hooke, kinetic Coulomb sliding	120,000 lines of C++/Python, several dependencies (boost, zlib, eigen etc.)	yes	GPL 3	[13]
2016	Blaze-DEM(GPU)	Hooke, kinetic Coulomb sliding	5000 lines of C++, CUDA	yes	BSD-3	[14,15]
2016	cemfDEM	HM, static & kinetic Coulomb sliding, kinetic rolling	18,000 lines of Fortran 90	yes	GPL 3	[16]
2018	ParaEllip3d	HM, static & kinetic Coulomb sliding	38,000 lines of C++, MPI, dependency on boost, eigen, qhull	yes	MIT, Apache 2.0	[17]
2019	OpenFPM	Hertz, static & kinetic Coulomb sliding	Reimplementation of [18] in 35,000 lines of C++, CPU & GPU parallelization, several dependencies (boost, OpenMPI, libhilbert etc.)	yes	BSD-3	[19]
2020	MercuryDPM	Various, static & kinetic Coulomb friction (sliding, rolling & twisting)	370,000 lines of C++14, Fortran & Python, MPI	yes	BSD-3	[20]
2020	MUSEN	Various, static & kinetic Coulomb sliding, kinetic rolling	70,000 lines of C++, CUDA, dependency on Qt, protobuf, zlib	yes	BSD-3	[21]
2020	Chrono::Granular	Various, static & kinetic Coulomb sliding, kinetic rolling	860,000 lines of C++, CUDA	yes	BSD-3	[22,23]
2021	SudoDEM	Identical to YADE	Derived from YADE (same dependencies), 300,000 lines of C++, Python	yes	GPL 3	[24]
2022	Lethe-DEM	Hooke/HM/JKR/DMT, static & kinetic Coulomb sliding, kinetic rolling	170,000 lines of C++/Python, MPI, several dependencies (deal.II, numdiff, p4est, trilinos, METIS)	yes	Apache 2.0, LGPL 2.1	[25]
2022	DEMBody	HM, static & kinetic Coulomb friction (sliding, rolling & twisting)	12,000 lines of Fortran 90, OpenMP	registration & approval required	non-commercial agreement	[26]
2023	PhasicFlow	Hooke/HM/HKK, static & kinetic Coulomb sliding, kinetic rolling	93,000 lines of C++, OpenMP, CUDA, some dependencies (Kokkos, tbb)	yes	GPL 3	[27]
2023	—	JKR, static & kinetic Coulomb friction (sliding, rolling & twisting)	Builds on MechSys; 11,000 lines of C++, OpenMP	yes	GPL 3	[28]
2023	CP3d	Hooke/HM, static & kinetic Coulomb sliding	50,000 lines of Fortran 95, MPI, dependency on FFTW	yes	MIT	[29]
2024	DEM-Engine	HM, static & kinetic Coulomb sliding, kinetic rolling	Builds on Chrono; 30,000 lines of C++ (not counting Chrono), CUDA	yes	BSD-3	[30]

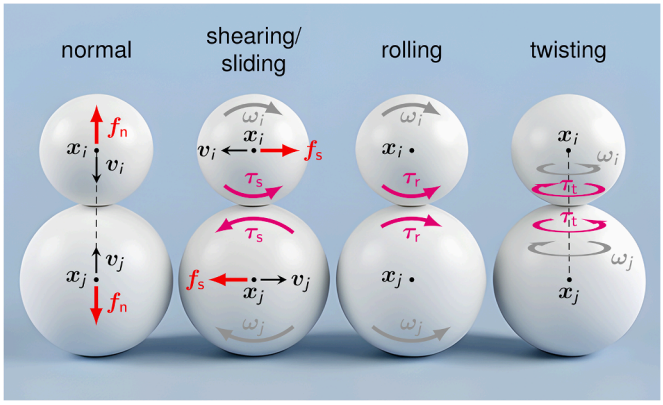


Fig. 2. The four particle-particle contact modes. Black arrows indicate linear motion, gray arrows indicate rotation, red arrows indicate resulting forces, magenta arrows indicate resulting torques. Directions are exemplary and can also be reversed. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

$$\gamma_i = \frac{8\pi\eta R_i^3}{I_i} = \frac{10}{3}c_i. \tag{4}$$

The sums in Eqs. (1) and (2) run over all other objects j that particle i interacts with. These include other particles, but also other geometric entities that may act as confining boundaries or obstacles. How nearby objects are found efficiently, such that this sum does not translate to a loop over all objects in the code, will be discussed further below. $\vec{f}^{ij}$ are the interaction forces and $\vec{\tau}^{ij}$ the torques exchanged by a pair of colliding objects. Subscript letters denote the mode or directionality of this interaction: normal (n), shearing or sliding (s), rolling (r) and twisting (t) (Fig. 2).

2.2. Inelastic contact model

To model the forces of dry and non-adhesive contact between elastic bodies, one needs to pick one of essentially two types of models: A simple choice is to use linear spring forces (Hooke's law), but a more realistic option is to make the repulsive force nonlinear in the indentation depth δ (as in Hertzian contact). This choice affects the behavior of particle ensembles qualitatively [45], and most available DEM codes implement at least one of the two (Table 1). For Hertzian contact, Mindlin derived the tangential (shear) forces [46,47], and the combination of both became known as the Hertz–Mindlin model. TinyDEM implements the visco-elastic Hertz–Mindlin contact model with Coulomb friction.

In the following, the expressions for the interaction forces and torques are provided with the particle pair superscript ij omitted for better readability. The sign convention is such that vectorial quantities refer to the property (position, velocity, etc.) of body i relative to that of body j, or an effect (force, torque) on body i due to interaction with body j. In brief, the normal and shear forces are computed as

$$\vec{f}_\mathrm{n} = -k_\mathrm{n}\vec{u}_\mathrm{n} - c_\mathrm{n}\vec{v}_\mathrm{n}, \tag{5}$$

$$\vec{f}_\mathrm{s} = -k_\mathrm{s}\vec{u}_\mathrm{s} - c_\mathrm{s}\vec{v}_\mathrm{s}, \tag{6}$$

$$\delta = R_i + R_j - d \geq 0, \tag{7}$$

$$\vec{d} = \vec{x}_i - \vec{x}_j, \quad d = \|\vec{d}\|, \tag{8}$$

$$\hat{\vec{n}} = \frac{\vec{d}}{d}, \tag{9}$$

$$\vec{u}_\mathrm{n} = -\delta\hat{\vec{n}}, \tag{10}$$

where $\vec{u}_\mathrm{n}$, $\vec{v}_\mathrm{n}$ are the vectors of relative normal position and velocity, and $\vec{u}_\mathrm{s}$, $\vec{v}_\mathrm{s}$ are the integrated displacement of static shearing friction and shear velocity, respectively, as defined further below. To satisfy Coulomb's friction law, the shearing force vector (Eq. (6)) is clipped to

$$\|\vec{f}_\mathrm{s}\| \leq \mu_\mathrm{s}\|\vec{f}_\mathrm{n}\| \tag{11}$$

for a specified sliding friction coefficient μ_s.

For two linearly elastic spherical particles, or a sphere and a half-space, the Hertz–Mindlin model provides the normal and shear stiffness coefficients as

$$k_\mathrm{n} = \frac{4}{3}E^* a, \tag{12}$$

$$k_\mathrm{s} = 8G^* a, \tag{13}$$

$$a = \sqrt{R^*\delta} \tag{14}$$

with effective Young's modulus, shear modulus, and radius

$$E^* = \left(\frac{1-\nu_i^2}{E_i} + \frac{1-\nu_j^2}{E_j}\right)^{-1}, \tag{15}$$

$$G^* = \left(\frac{2-\nu_i}{G_i} + \frac{2-\nu_j}{G_j}\right)^{-1}, \quad G_i = \frac{E_i}{2(1+\nu_i)}, \tag{16}$$

$$R^* = \left(\frac{1}{R_i} + \frac{1}{R_j}\right)^{-1}. \tag{17}$$

Note the depth-dependency of the spring coefficients via a, hence the nonlinearity of the contact law. a is the radius of the contact area between two Hertzian spheres, which is not to be confused with the radius of overlap between two rigid spheres (Fig. 3). Since real collisions are usually inelastic, damping terms are added in Eqs. (5) and (6). The normal and shear coefficients are given by

$$c_\mathrm{n} = \sqrt{5m^* k_\mathrm{n}}\,\beta, \tag{18}$$

$$c_\mathrm{s} = \sqrt{\frac{1}{6}m^* k_\mathrm{s}}\,\beta,. \tag{19}$$

with effective mass and damping factor

$$m^* = \left(\frac{1}{m_i} + \frac{1}{m_j}\right)^{-1}, \tag{20}$$

$$\beta = \frac{1}{\sqrt{1+(\pi/\ln e_\mathrm{n})^2}}. \tag{21}$$

The factor $\sqrt{5}$ in Eq. (18) originates from for the exponent 3/2 in the distance dependency of the Hertzian normal force [48]. For Hookean contact, that factor would be 2 [49]. e_n is the coefficient of normal restitution, defined as the normal velocity ratio after (') and before contact ($\vec{v}'_\mathrm{n} = -e_\mathrm{n}\vec{v}_\mathrm{n}$ with $0 \leq e_\mathrm{n} \leq 1$). A coefficient of tangential restitution e_s can be defined analogously ($\vec{v}'_\mathrm{s} = e_\mathrm{s}\vec{v}_\mathrm{s}$, with $-1 \leq e_\mathrm{s} \leq 1$ [50]), but rather than being a constant, it depends on the collision velocity [51,52] and angle [53]. A general, physically accurate expression for a damping coefficient $c_\mathrm{s}(e_\mathrm{s})$ with constant e_s is therefore impossible to formulate, and some DEM implementations just set $c_\mathrm{s} = c_\mathrm{n}$ [19,28,33,54–58] or use fixed ratios $c_\mathrm{s}/c_\mathrm{n}$ [11,59,60]. Here, a heuristic approach it taken instead: Eq. (19) is used with a factor $\sqrt{1/6}$ determined numerically in a direct central collision with pure sliding ($\vec{u}_\mathrm{s} = \vec{0}$), in such a way that $0 \leq e_\mathrm{n} = e_\mathrm{s} \leq 1$ over the entire range of values. With $\vec{u}_\mathrm{s}$ included, the tangential contact velocity can then reverse (corresponding to $e_\mathrm{s} < 0$) in collisions with sufficient static friction, as it should be [53]. It appears that this expression for c_s has not occurred in the DEM literature before.

For simplicity, TinyDEM assumes that all particles have identical material properties ($E_i \equiv E$, $\nu_i \equiv \nu$, $\rho_i \equiv \rho$). Some of the above expressions therefore simplify:

$$E^* = \frac{E}{2(1-\nu^2)}, \tag{22}$$

$$G^* = \frac{E}{4(2-\nu)(1+\nu)}, \tag{23}$$

$$m^* = \frac{4}{3}\pi\rho\left(\frac{1}{R_i^3} + \frac{1}{R_j^3}\right)^{-1}, \tag{24}$$

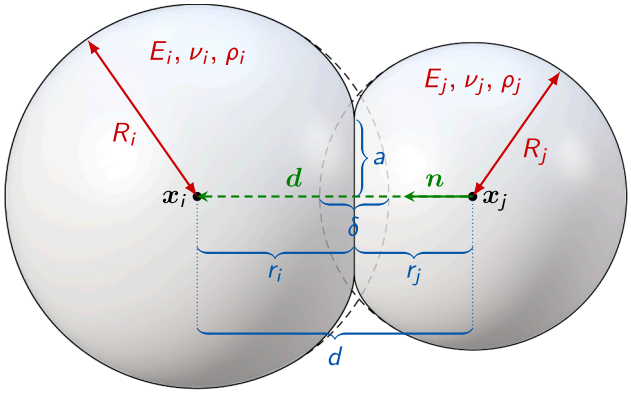


Fig. 3. Particle-particle contact. Fundamental particle properties in red, geometric contact parameters (distances) in blue, normal vectors in green. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

$$c_\mathrm{s} = c_\mathrm{n}\sqrt{\frac{1-\nu}{10(2-\nu)}}. \tag{25}$$

Note that the ratio between tangential and normal damping $c_\mathrm{s}/c_\mathrm{n}$ is confined to the range $(0.18, 0.26)$ for a collision between two equal particles here, for any Poisson ratio $\nu \in (-1, 0.5]$.

TinyDEM handles particle contact with static walls and other geometric primitives (detailed further below), all by the same routine. For a collision between a particle i and a static object j, the latter is assumed to be infinitely flat, stiff and heavy ($R_j = \infty$, $E_j = \infty$, $m_j = \infty$), such that $R^* = R_i$, $m^* = m_i$, $E^* = E_i/(1-\nu_i^2)$, and $G^* = G_i/(2-\nu_i)$.

To obtain the normal and tangential velocity at the contact point for Eqs. (5) and (6), the total relative velocity $\vec{v} = \vec{v}_\mathrm{n} + \vec{v}_\mathrm{s}$, is decomposed as

$$\vec{v} = \vec{v}_i - \vec{v}_j + \hat{\vec{n}} \times (r_i\vec{\omega}_i - r_j\vec{\omega}_j), \tag{26}$$

$$\vec{v}_\mathrm{n} = \left(\vec{v}\cdot\hat{\vec{n}}\right)\hat{\vec{n}}, \tag{27}$$

$$\vec{v}_\mathrm{s} = \vec{v} - \vec{v}_\mathrm{n}, \tag{28}$$

where $\vec{v}_i$, $\vec{\omega}_i$ and $\vec{v}_j$, $\vec{\omega}_j$ are the linear and angular velocities of bodies i and j at and about their centers of mass, respectively. Walls are assumed to be stationary ($\vec{v}_j = \vec{0}$, $r_j\vec{\omega}_j = \vec{0}$). The radii

$$r_i = R_i - \delta/2, \tag{29}$$

$$r_j = R_j - \delta/2 \tag{30}$$

measure the distance from the centers of mass of bodies i and j to the mutual contact point (Fig. 3).

The form of the torques in Eq. (2) has been the subject of considerable debate in the literature [61–64]. Rolling resistance is a complex subject with several different models that have been proposed [61,62]. Of the three torques, the twisting resistance is often considered the least important, and therefore neglected in many codes [57,65]. However, both rolling and twisting friction can be required to reproduce experimentally generated granular packings [57]. In TinyDEM, the sliding, rolling and twisting torques are implemented as [66,67]

$$\vec{\tau}_\mathrm{s} = r_i\vec{f}_\mathrm{s} \times \hat{\vec{n}}, \tag{31}$$

$$\vec{\tau}_\mathrm{r} = R^*\hat{\vec{n}} \times \vec{f}_\mathrm{r}, \tag{32}$$

$$\vec{\tau}_\mathrm{t} = f_\mathrm{t}R^*\hat{\vec{n}}, \tag{33}$$

with linearly viscoelastic (Kelvin–Voigt) rolling and twisting forces:

$$\vec{f}_\mathrm{r} = -k_\mathrm{r}\vec{u}_\mathrm{r} - c_\mathrm{r}\vec{v}_\mathrm{r}, \tag{34}$$

$$f_\mathrm{t} = -k_\mathrm{t}u_\mathrm{t} - c_\mathrm{t}v_\mathrm{t}, \tag{35}$$

where $\vec{u}_\mathrm{r}$ and u_t are the integrated displacements of static rolling and twisting friction, respectively, defined later. The relative rolling and twisting velocities can be obtained as [62,66]

$$\vec{v}_\mathrm{r} = \left(\frac{1}{r_i} + \frac{1}{r_j}\right)^{-1} \vec{\omega} \times \hat{\vec{n}}, \tag{36}$$

$$v_\mathrm{t} = R^*\vec{\omega}\cdot\hat{\vec{n}}, \tag{37}$$

$$\vec{\omega} = \vec{\omega}_i - \vec{\omega}_j, \tag{38}$$

which also applies for particle-wall collisions ($r_j = \infty$).

It has been suggested that the twisting resistance parameters in Eqs. (34) and (35) can be linked to those for sliding through integration of the frictional stress over the contact area [67]. In the notation used here, these relationships would read $k_\mathrm{t} = bk_\mathrm{s}$, $c_\mathrm{t} = bc_\mathrm{s}$ where $b = \delta/(2R^*)$ is the relative indentation depth. Similarly, the coefficients for rolling resistance may be related to the normal contact parameters through a dimensionless shape parameter [65] which, when interpreted as the particle deformation from indentation, result in $k_\mathrm{r} = bk_\mathrm{n}/2$, $c_\mathrm{r} = bc_\mathrm{n}/2$. These expressions lead to relatively small stiffness and damping parameters for rolling and twisting and in practical tests with TinyDEM, prolonged oscillations with unrealistically large amplitudes were observed with them, unless substantial damping is used. Alternatively, rolling resistance can be related to the visco-elastic properties of the particles, and the coefficient of restitution that results from a collision with a given impact velocity [68]. Iwashita and Oda [69] noted that "there is no rational reason, unfortunately, to choose a specific value as the rolling stiffness k_r". Other DEM codes use fixed ratios between the coefficients for sliding, rolling and/or twisting [66] or set them equal [57,69]. TinyDEM follows that heuristic by setting $k_\mathrm{r} = k_\mathrm{n}$, $k_\mathrm{t} = k_\mathrm{s}$, $c_\mathrm{r} = c_\mathrm{n}$, $c_\mathrm{t} = c_\mathrm{s}$.

Analogously to the sliding friction force (Eq. (11)), also the rolling and twisting friction forces (Eqs. (34) and (35)) are clipped to satisfy Coulomb's law, each with their own friction coefficients:

$$\|\vec{f}_\mathrm{r}\| \le \mu_\mathrm{r}\|\vec{f}_\mathrm{n}\|, \tag{39}$$

$$|f_\mathrm{t}| \le \mu_\mathrm{t}\|\vec{f}_\mathrm{n}\|. \tag{40}$$

The Coulomb friction coefficients for sliding and rolling, μ_s and μ_r, are independent material parameters, but the one for twisting can be related to sliding as [67]

$$\mu_\mathrm{t} = \frac{2}{3}\mu_\mathrm{s}, \tag{41}$$

because the twisting motion is essentially rotational sliding.

With these expressions, the inelastic contact model is completely defined and parameterized, with exception of the calculation of the relative displacements in shear, rolling and twisting resistance, $\vec{u}_\mathrm{s}$, $\vec{u}_\mathrm{r}$, u_t (i.e., the static friction history). These are initialized to zero at the beginning of a new collision and then updated in every timestep as detailed in the next section.

2.3. Time integration

To evolve the particle ensemble in time, the equations of motion are solved with the semi-implicit Euler method. Apart from its simplicity, it has the advantage of being symplectic, i.e., preserving the Hamiltonian when all dissipative forces are disabled ($e_\mathrm{n} = 1$, $\mu_\mathrm{s} = \mu_\mathrm{r} = \mu_\mathrm{t} = \eta = 0$). From the linear and angular accelerations of all particles,

$$\vec{a}_i = \frac{1}{m_i}\sum_{j\neq i}\left(\vec{f}_\mathrm{n}^{ij} + \vec{f}_\mathrm{s}^{ij}\right) + \vec{g} - c_i\vec{v}_i, \tag{42}$$

$$\vec{\alpha}_i = \frac{1}{I_i}\sum_{j\neq i}\left(\vec{\tau}_\mathrm{s}^{ij} + \vec{\tau}_\mathrm{r}^{ij} + \vec{\tau}_\mathrm{t}^{ij}\right) - \gamma_i\vec{\omega}_i, \tag{43}$$

the velocities, positions and angular velocities are updated according to

$$\vec{v}_i \leftarrow \vec{v}_i + \Delta t\,\vec{a}_i, \tag{44}$$

$$\vec{x}_i \leftarrow \vec{x}_i + \Delta t\,\vec{v}_i, \tag{45}$$

$$\vec{\omega}_i \leftarrow \vec{\omega}_i + \Delta t\,\vec{\alpha}_i. \tag{46}$$

where $\Delta t > 0$ is the finite timestep. Note that the order in which Eqs. (44) and (45) are evaluated matters: To be symplectic, the position updates (Eq. (45)) must use the already updated velocities (Eq. (44)).

The actual orientation of spherical particles does not necessarily need to be tracked, as all that is needed to evaluate the forces are angular velocities, not angles. Nevertheless, for various purposes including visualization (or to generalize the code to non-spherical particles), it can be useful to store and update particle orientations, too. Updating them using Euler angles would lead to the infamous gimbal lock, the inability to rotate in certain directions when two rotational axes coincide. A simple and robust way to avoid this problem is to use unit quaternions to represent particle orientations, although they are not the only possible remedy [70]. A quaternion can be written as $\hat{q} = (q, \vec{q})$ where q is the scalar part and $\vec{q}$ the vectorial part. Numerous algorithms exist to integrate quaternions in time. TinyDEM uses the synchronous version of a recently introduced, highly accurate method called SPIRAL [71]:

$$\hat{q}_i \leftarrow \hat{q}_i \left(\cos\varphi_i,\ \sin\varphi_i \frac{\vec{\omega}_i}{\|\vec{\omega}_i\|} \right) \left(\cos\theta_i,\ \sin\theta_i \frac{\vec{\alpha}_i}{\|\vec{\alpha}_i\|} \right) \tag{47}$$

where

$$\varphi_i = \frac{\Delta t}{2} \|\vec{\omega}_i\|, \qquad \theta_i = \left(\frac{\Delta t}{2}\right)^2 \|\vec{\alpha}_i\|. \tag{48}$$

The triple product in Eq. (47) is computed from left to right as two consecutive Hamilton products:

$$\hat{p}\hat{q} = \left(pq - \vec{p}\cdot\vec{q},\ p\vec{q} + q\vec{p} + \vec{p}\times\vec{q}\right). \tag{49}$$

Note that these products are computed only if $\|\vec{\omega}_i\| > 0$ and $\|\vec{\alpha}_i\| > 0$, respectively. Moreover, note that the quaternion must be updated before the angular velocity, i.e., before Eq. (46) is evaluated.

For reasonable numerical stability and accuracy with explicit single-step time integration, the size of the timestep usually needs be chosen well below the period of the fastest harmonic oscillator in the Newtonian system: $\Delta t \ll \sqrt{m/k}$, where m is a typical (small) mass and k a typical (large) stiffness. A good approach is to base the size of the timestep on the propagation of elastic waves across particles, which need to be resolved with sufficient resolution [72,73]. In TinyDEM, a value of

$$\Delta t = R_{\min} \sqrt{\rho \frac{1-\nu^2}{E}} \tag{50}$$

is used by default, resolving a head-on normal collision of two equal particles with a maximum relative indentation depth $\delta/(2R^*) = 10\,\%$ with a dozen timesteps at $e_\mathrm{n} = 1/2$. $R_{\min}$ is the specified minimum particle size in the (possibly polydisperse) ensemble. For particles with a radius of 1 cm, a mass density of 1 g/cm^3, and a Young's modulus of 1 MPa, the timestep will be about 0.3 ms, and the simulation of a minute physical time thus requires about 200,000 timesteps. Smaller Δt values may be set by the user if higher accuracy is desired, for example to accurately resolve stiff forces that can result from large friction coefficients.

In the evaluation of the frictional forces (Eqs. (6), (34) and (35)), the displacement of the static friction springs,

$$\vec{u}_\mathrm{m}(t) = \int_{t_0}^{t} \vec{v}_\mathrm{m}(t')\,dt', \tag{51}$$

(integrated in the co-rotated frame since the time of first contact t_0), needs to be updated for each contact, for each friction mode $\mathrm{m} \in \{\mathrm{s}, \mathrm{r}, \mathrm{t}\}$ (sliding, rolling and twisting). From the previous timestep, the sliding and rolling displacements are first projected back onto the tangent plane perpendicular to the current unit normal vector $\hat{\vec{n}}$, and scaled back to their original length unless zero [66]:

$$\vec{u}_\mathrm{m} \leftarrow \|\vec{u}_\mathrm{m}\| \frac{\vec{u}_\mathrm{m} - \left(\vec{u}_\mathrm{m}\cdot\hat{\vec{n}}\right)\hat{\vec{n}}}{\left\|\vec{u}_\mathrm{m} - \left(\vec{u}_\mathrm{m}\cdot\hat{\vec{n}}\right)\hat{\vec{n}}\right\|}, \quad \mathrm{m} = \mathrm{s}, \mathrm{r}. \tag{52}$$

Note that for the twisting friction, a projection onto the current normal is not needed, as it is readily expressed as a scalar torque about $\hat{\vec{n}}$. The frictional forces are then evaluated using these updated spring displacements. If the Coulomb inequality (Eqs. (11), (39) and (40)) is satisfied, friction is in the static regime, and the displacement is incremented for the next timestep as

$$\vec{u}_\mathrm{m} \leftarrow \vec{u}_\mathrm{m} + \Delta t\, \vec{v}_\mathrm{m}, \quad \mathrm{m} = \mathrm{s}, \mathrm{r}, \mathrm{t}. \tag{53}$$

Otherwise, i.e., in the dynamic friction regime, the spring is set to the length that fulfills Coulomb's condition equally [66]:

$$\vec{f}_\mathrm{m} \leftarrow \mu_\mathrm{m} \|\vec{f}_\mathrm{n}\| \frac{\vec{f}_\mathrm{m}}{\|\vec{f}_\mathrm{m}\|}, \tag{54}$$

$$\vec{u}_\mathrm{m} \leftarrow -\frac{1}{k_\mathrm{m}} \left(\vec{f}_\mathrm{m} + c_\mathrm{m}\vec{v}_\mathrm{m}\right), \quad \mathrm{m} = \mathrm{s}, \mathrm{r}, \mathrm{t}, \tag{55}$$

forgetting about any potential previous history of $\vec{u}_\mathrm{m}$. This ensures a continuous force transition across the switch from dynamic to static friction. Note that Eqs. (51), (53), (54) and (55) are to be interpreted in scalar form for twisting (u_t, v_t, f_t in place of $\vec{u}_\mathrm{t}$, $\vec{v}_\mathrm{t}$, $\vec{f}_\mathrm{t}$). The procedure is applied for each of the three friction modes m individually, not together, such that they can be in static or kinetic state independently from each other. The static and kinetic friction modes are assumed to have the same friction coefficients here—a simplification commonly made in DEM codes that implement Coulomb friction, such as [10,57,60,66].

Table 2
Complete list of program parameters. In the parameter dimension, M represents mass, L length, T time. Default values create a quick simulation of a small polydisperse ensemble of soft particles falling under gravity through an hourglass (Fig. 5A).

Symbol	Default value	Requirements	Dimension	Description
Geometric parameters				
$R_{\min}$	0.002	> 0	L	Minimum particle radius
$R_{\max}$	0.01	$\geq R_{\min}$	L	Maximum particle radius
R_{mesh}	$\{0.002, 0.02, 0.01\}$	≥ 0	L	Mesh radii (can be a list of values)
$\vec{s}_{\min}$	$(-0.1, -0.1, 0.3)$	—	L	Minimum coordinate of particle spawn box
$\vec{s}_{\max}$	$(0.1, 0.1, 0.4)$	$\geq \vec{s}_{\min}$	L	Maximum coordinate of particle spawn box
Material parameters				
ρ	10^3	> 0	M/L^3	Mass density
E	10^6	> 0	M/LT^2	Young's modulus
ν	0.3	$\in (-1, 0.5]$	—	Poisson's ratio
e_n	0.5	$\in [0, 1]$	—	Coefficient of normal restitution
μ_s	0.3	≥ 0	—	Coulomb friction coefficient for sliding
μ_r	0.3	≥ 0	—	Coulomb friction coefficient for rolling
μ_t	Eq. (41)	≥ 0	—	Coulomb friction coefficient for twisting
η	0	≥ 0	M/LT	Dynamic viscosity for Stokes drag
Kinetic parameters				
$\vec{v}_0$	$(0, 0, -1)$	—	L/T	Initial linear particle velocity
$\vec{\omega}_0$	$(0, 0, 0)$	—	1/T	Initial angular particle velocity
$\vec{g}$	$(0, 0, -9.81)$	—	L/T^2	Gravitational acceleration
Simulation parameters				
Δt	Eq. (50)	> 0	T	Time increment
$N_{\max}$	1000	≥ 0	—	Maximum number of particles
N_{frames}	100	≥ 0	—	Number of output frames
N_{steps}	500	≥ 0	—	Number of timesteps per frame
N_{spawn}	100	≥ 0	—	Number of particle spawn attempts per timestep

3. Implementation

TinyDEM was designed following a min-max coding philosophy, providing as much typical core DEM functionality (for a certain set of common use cases) with as little and as comprehensible code as possible, with goals like leanness, ease of maintenance, and portability in mind. One example manifestation of this is the floating-point precision used in all calculations, which can be changed with a single `typedef`. The preset precision is `double`. The entire program consists of only two C++

files, `tinydem.hpp` and `tinydem.cpp`, totaling in about 600 lines of commented code. For maximum accessibility, variables are largely named as written in this paper. The header file contains only generic and minimal data structures, distance calculation and I/O routines, and normally need not be modified by the user. The source file contains the physical model with all its parameters (Table 2), the simulation loop, etc., which may be altered by the user to simulate a specific scenario.

3.1. Particle generation

TinyDEM supports two ways of generating particles: Through an initial import from a CSV file (see usage instructions below), or through dynamically spawning of new particles within a defined spatial region. To use the second method, a spawn box can be defined via its lower and upper corner vertices, $\vec{s}_{\min}$ and $\vec{s}_{\max}$. Additionally, a number of particle spawn attempts per timestep, N_{spawn}, can be specified. In each attempt, a random particle center position is drawn uniformly within the spawn box, a random particle radius is drawn uniformly in the range $[R_{\min}, R_{\max}]$, and unless this would result in an overlap with other particles or the mesh, the particle is generated with initial velocity $\vec{v}_0$ and angular velocity $\vec{\omega}_0$ there. Note that in case of failure to spawn the particle at the drawn location, the assigned random radius is carried over to the next particle spawn attempt (possibly during the next timestep) to avoid biasing the particle size distribution toward smaller particles that have a higher chance of filling small gaps. Particles are spawned over time only until a specified maximum number $N_{\max}$ is reached.

TinyDEM can be run in pure 2D mode in any plane (or multiple planes) parallel to any of the three Cartesian planes, or in 1D mode along any line (or multiple lines) parallel to the Cartesian axes. As long as all forces, torques and velocities are restricted to such a plane or line, particles will remain confined to it. All that is needed for this is that the initial conditions (including the particle import file and particle spawn region) are conforming, and that the mesh contains no elements positioned such that they can collide with the particles in an oblique direction, pushing them out of their plane or line.

Note that, when generating particles with the spawn region method, only the particles' center points, not their entire volumes, are restricted to lie within the specified spawn region. This is to enable lower-dimensional (2D or 1D) spawn regions also for polydisperse ensembles by setting a subset of the x-, y- or z-components of the spawn box vertices $\vec{s}_{\min}$ and $\vec{s}_{\max}$ equal.

3.2. Particle and collision data

For a better overview, Table 3 lists all properties stored per particle object. Note that the contact forces and torques are antisymmetric,

$$\vec{f}_{\text{n}}^{ij} = -\vec{f}_{\text{n}}^{ji}, \tag{56}$$

$$\vec{f}_{\text{s}}^{ij} = -\vec{f}_{\text{s}}^{ji}, \tag{57}$$

$$\vec{\tau}_{\text{r}}^{ij} = -\vec{\tau}_{\text{r}}^{ji}, \tag{58}$$

$$\vec{\tau}_{\text{t}}^{ij} = -\vec{\tau}_{\text{t}}^{ji}, \tag{59}$$

except the sliding torque, for which the relationship

$$r_j \vec{\tau}_{\text{s}}^{ij} = r_i \vec{\tau}_{\text{s}}^{ji} \tag{60}$$

holds [66]. One may therefore iterate over the ordered body pairs $i < j$ only once, computing the forces and torques on both of them using these relationships from a single interaction, or over the unordered pairs $i \neq j$ twice, reevaluating them from both of the particles' perspectives. In TinyDEM, the single-evaluation approach is taken, exploiting Eqs. (56)–(60), as it is more efficient.

Each particle i holds a list c of contact data structures for collisions with particles j with $i < j$. That way, since $j > 0$ for particle-particle collisions by this principle, non-positive indices $j \leq 0$ are free to be used to label collisions with mesh elements. Each entry in c of particle i holds the index j of the contact partner, the static friction spring displacements, and a Boolean flag $\psi \in \{0, 1\}$ indicating whether the collision is still active in the current timestep. During the dense phase of collision detection (detailed below), all active collisions are marked ($\psi = 1$). Once all collisions involving a particle i have been handled, all inactive contact objects ($\psi = 0$) of that particle that are still present from the previous timestep are destroyed and the flag of all remaining (i.e., active) ones is reset for the next iteration ($\psi = 0$).

Table 3
Summary of particle-specific variables implemented. Particle indices are omitted for readability. Particle properties not listed (such as elastic moduli and mass densities) are shared among all particles and thus do need not be stored per particle. In the dimension, L represents length, T time.

Symbol	Type	Dimension	Description
Particle-specific variables			
R	scalar	L	radius
$\vec{x}$	vector	L	position
$\vec{v}$	vector	L/T	velocity
$\vec{a}$	vector	L/T^2	acceleration
q	scalar	—	scalar part of quaternion
$\vec{q}$	vector	—	vectorial part of quaternion
$\vec{\omega}$	vector	1/T	angular velocity
$\vec{\alpha}$	vector	1/T^2	angular acceleration
n	integer	—	index of next particle in cell
c	list	—	list of contacts (entries below)
Contact-specific variables (per entry of c above)			
j	integer	—	index of contact partner
$\vec{u}_{\text{s}}$	vector	L	sliding spring displacement
$\vec{u}_{\text{r}}$	vector	L	rolling spring displacement
u_{t}	scalar	L	twisting spring displacement
ψ	Boolean	—	flag marking active collisions

3.3. Collision detection

To handle particle collisions efficiently with linear time complexity in the number of particles N, it is essential to avoid unnecessary long-distance collision checks. TinyDEM uses regular space partitioning with linked cell lists [74]. In each timestep, the bounding box of the entire particle ensemble is calculated and discretized into cubic cells with an edge length equal to the specified maximum particle diameter, $2R_{\max}$. This ensures that all potential contact candidates can be found within the local Moore neighborhood of up to 27 cells about each particle. Only particle pairs within this neighborhood are then tested for overlap, jumping from one particle to the next one in the same cell (Table 3).

Note that the memory used by the spatial grid scales with its volume. Binary space partitioning with a tree data structure would prevent this, but is a complexity eschewed here for simplicity. Therefore, it is generally advisable to spatially confine the ensemble by defining a global bounding box with the mesh functionality detailed below to prevent excessive memory usage in simulations where particles would otherwise vastly separate.

In addition to collisions among particles, TinyDEM supports contact with arbitrarily shaped static meshes, such as walls or other obstacles. They are unlimited in number, may be connected or disjoint, and can consist of polygonal elements with one to four vertices. Thus, any discrete object whose surface consists of points (Fig. 4A), edges (Fig. 4B), triangles (Fig. 4C) and rectangles (Fig. 4D) can be used to define a geometrical environment (Fig. 4E) for the particle simulation. Like the particles, each mesh element j can have its own radius $R_j \geq 0$, and their edges and vertices are rounded off accordingly, i.e., a mesh vertex is effectively treated as a sphere, an edge as a spherocylinder, etc. The geometrical part of the particle-mesh collision detection thus reduces to four types of primitive subproblems: computing the distance vector between a point and 1) another point; 2) a line segment; 3) a triangle;

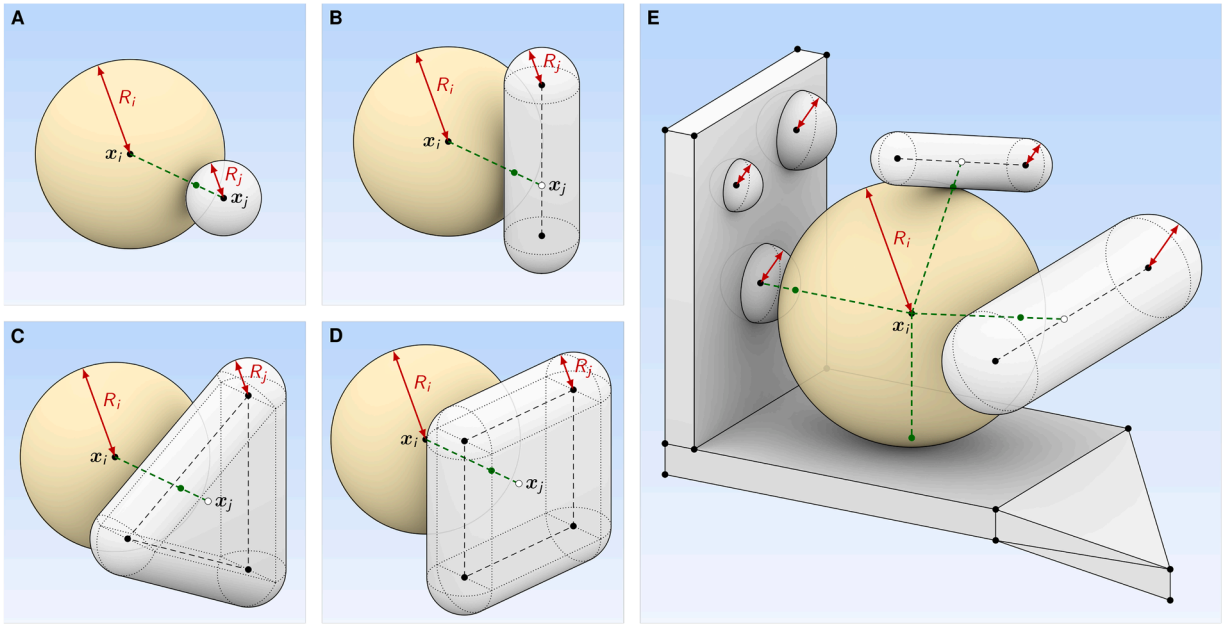


Fig. 4. Particle-mesh collisions. Collisions between particles (yellow) and four different mesh primitives (white) are implemented: a point (A), line segment (B), triangle (C), and rectangle (D). With an optional mesh "radius" $R_j \geq 0$ (red), each mesh element j can be given an individual thickness with rounded-off edges. The contact point is shown in green, the closes point of approach on the mesh (which may also lie on the dashed edges of the mesh primitive) in white. By combining these elements, complex mesh geometries can be formed (E), allowing for concurrent contacts of a particle with multiple mesh elements, each having their own "radius". Mesh vertices are shown as black dots. The triangular and rectangular mesh elements have $R_j = 0$ in E, hence the sharp edges. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

and 4) a rectangle (Fig. 4). For all of these, simple direct algorithms are commonly available [75] and implemented in TinyDEM.

A rather efficient way to implement the broad phase of collision detection between particles and mesh elements is to precompute each mesh element's axis-aligned bounding box, and then dynamically determine the indices of the block of grid cells it overlaps with. The dense phase of actual distance calculations can then be limited to the (usually small fraction of) particles that lie in those cells ± 1. Mesh elements that are far from any particle are thus not entering the dense phase at all. This leads to a time complexity for collisions between the N particles and M mesh elements that is linear in M and sub-linear in N, unless the mesh densely fills the space occupied by the particle ensemble.

A subtle difficulty arises when a particle moves from a collision with one mesh element to another, for instance across the internal shared edge between two coplanar triangles making up a planar wall. For a period proportional to $R_j/\|\vec{v}_i\|$, the particle will experience a repulsive force from both elements, unless dedicated countermeasures are taken. Such countermeasures are a non-trivial task, though: It could well be that the particle is righteously in contact with multiple mesh elements at the same time (for example in the corner of a box, or rolling down an inclined furrow)—a case that would need to be distinguished from rolling or sliding across seams of a flat mesh surface, which is a single continuous collision event. This problem of "ghost collisions" is well-known in rigid-body simulations and has no known universal, computationally efficient solution [76]. To mitigate the error made in the contact force *magnitude*, weighting based on intersection areas can be used [77], but this does not ensure a correct force *direction*. A robust method based on Voronoi regions has been proposed [78], but requires multiple passes, mesh connectivity information, and a distinction between sphere-polygon contact locations (face, edge or vertex)—a degree of complexity intentionally avoided in TinyDEM. Instead, a simple approach is implemented: particles can collide with any number of mesh elements simultaneously. This avoids the complexity and cost of continuous collision detection algorithms at the expense of losing contact information (Table 3) when a contact crosses over from one mesh element to another. Avoiding the ghost collisions across element borders as much as possible is one of the reasons why rectangular mesh elements are implemented here, even though they could also be represented by two joined triangles. The user is advised that ghost bumps can occur in the provided implementation, and that meshes should be as coarse as possible to reduce them to a minimum.

4. Example simulations

To demonstrate the range of potential applications of TinyDEM, six classical simulations are briefly showcased in Fig. 5.

The first scenario presents the default parameter setting used when TinyDEM is run out of the box. A small polydisperse ensemble of 1000 soft particles is simulated, spawning and dropping under the effect of gravity into an houglass-like container with a cylindrical and a spherical obstacle partially blocking the throat (Fig. 5A, Video 1). In the simulated period of about three physical seconds, the particles fall through to the bottom or get jammed, depending on randomness in particle initialization and in the race of threads in parallel execution. No viscous drag is used; the particles come to rest only through inelastic and frictional collisions with each other and with the mesh. The complete list of simulation parameters is given in Table 2. The simulation takes about half a minute on a 2.3 GHz Intel Core i9 processor with 8 threads.

The purpose of the second example is to demonstrate the suitability of TinyDEM to simulate phenomena involving large numbers of particles, despite its simple implementation. A stream of equal grains is dropped into a separator funnel that dyes them with two different colors (Fig. 5B, Video 2). Additional grains are dynamically added to the stream as room becomes available inside the spawn region at the top. The two sub-streams then fall onto chutes that merge them again, resulting in a binary dispersion pattern in the particle bed below. Collisions are frictional and inelastic, but no viscous drag is used. The simulation was run on an 8-core CPU and terminated when $N_{\max} = 10$ million particles were reached.

As a third scenario, a marble run is simulated to showcase a somewhat more complex geometric obstacle course. TinyDEM does not pose any limits to the complexity of the static mesh, and is therefore suited

also for more complicated geometric setups than shown here. With 40 texturized marbles completing the run, this example also shows the quaternion-based particle rotation in action (Fig. 5C, Video 3). The marbles are modeled without rolling friction and increased coefficient of restitution, but with moderate viscous drag.

The fourth simulation demonstrates a classical test of static friction in DEM codes. Confined by two parallel vertical plates separated by 11 grain diameters, a stream of 100,000 monodisperse grains piles up into a triangular heap (Fig. 5D, Video 4). Due to static friction, both sides form a stable linear slope (up to a logarithmic deviation at the bottom ends, which is a boundary effect [79]). Greater angles produce landslides that flatten the slope, smaller angles let additional grains stack up to steepen it. Large friction coefficients ($\mu_s = \mu_r = 3$) are used to produce a challenging simulation with steep fronts. The angle of repose θ yields an effective Coulomb friction coefficient $\mu_{eff} = \tan\theta$ that is considerably smaller than μ_s, because the grains can rotate. Here, $\mu_{eff} \approx 1.15$ ($\theta \approx 49°$) is observed. Note that also the wall separation affects the slope [80,81]. To accurately resolve the stiff static friction forces, a tenfold lower timestep is used than set by default via Eq. (50).

A classical use case of the DEM is the study of jamming and unjamming [82]—a good opportunity to showcase TinyDEM's 2D simulation mode. Although particles also jam in the default 3D setup (Fig. 5A), the behavior is better observable in 2D. About 4500 equally sized particles are spawned above three layers of lined-up funnels with narrowing throats. Under their own weight, particles spontaneously bridge the orifices (Fig. 5E, Video 5). This form of arching requires the throats to be no more than about 5 particle diameters wide at moderate friction [83], but stronger friction widens that critical size [84]. The shown simulation does not use viscous drag but considerable friction ($\mu_s = \mu_r = 1$) to promote jamming even with throat diameters of 6–10 particles. Although only anecdotal here, the formed arches are seen to have an aspect ratio (height to semi-width) of about one on average, consistent with previous reports [85]. Similar to the previous piling example, a 5-fold reduced timestep is used for finer resolution of the static friction dynamics.

Finally, a showcase of another extensively studied phenomenon in granular media: size-induced particle segregation (Fig. 5F, Video 6). 30,000 beads are placed in a cylindrical container with an inner radius of 39 bead radii R_{min}. Seven of them are ten times larger ($R_{max} = 10R_{min}$, blue balls) but have the same material properties (mass density etc.), and are randomly placed at the very bottom, buried by the smaller beads. Then, the base is harmonically vibrated in vertical direction with amplitude $A = R_{min}$ and frequency f such that the peak acceleration $A(2\pi f)^2$ is equal to $3g$, a commonly used value that promotes quick segregation [86]. In this agitated bath, the larger balls are convected upward to the top over time, against one's intuition that the heavier particles should settle at the bottom. This size segregation is facilitated by particle-wall friction, while particle-particle friction has no substantial effect [87,88]. In the simulation shown here, $\mu_s = \mu_r = 0.3$ is used for both types of contact.

5. Computational performance and resource usage

TinyDEM is not designed to max out high-performance computing infrastructure, but offers competitive performance on multi-core CPUs nevertheless. All loops over particle and mesh elements in the timestepping function are parallelized with OpenMP, with exception of the construction of the linked cell list, due to speedup limitations [89].

To measure the computational performance, a controlled and easily reproducible setting involving many particle contacts was chosen. N particles were placed in a cubic container whose edge length scales as

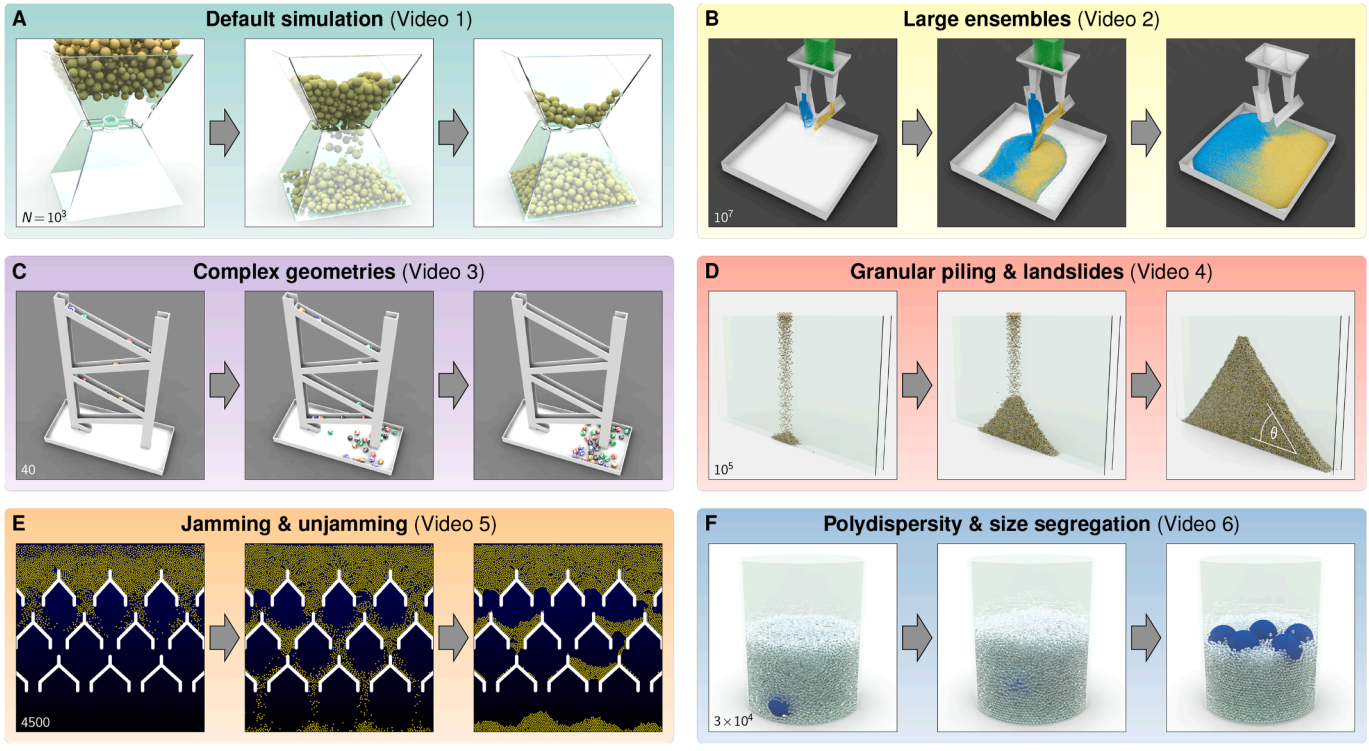


Fig. 5. Example simulations. Particle numbers N are given in the lower left corners. All six simulations have accompanying supplementary videos as labeled. A: The default simulation as preconfigured in the supplied code, a polydisperse set of particles falling through an hourglass with additional obstacles at the throat. B: Binary mixture of two streams consisting of 10 million grains. C: Marble run to demonstrate particle rotations and a somewhat more complex mesh geometry. D: A quasi-2D pile of strongly frictional grains between two vertical plates. θ is the angle of repose. E: Jamming in 2D. Frictional particles fall through three layers of funnels with neck sizes of 10, 8 and 6 particle diameters, from top to bottom. F: Particle size segregation through vibration of the base of a cylindrical container. Seven balls (blue, initially placed on the ground) have a ten-fold diameter than the rest of the beads, but equal material properties. Images show the gradual upward convection of the larger balls after 0, 83 and 175 vibration cycles. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

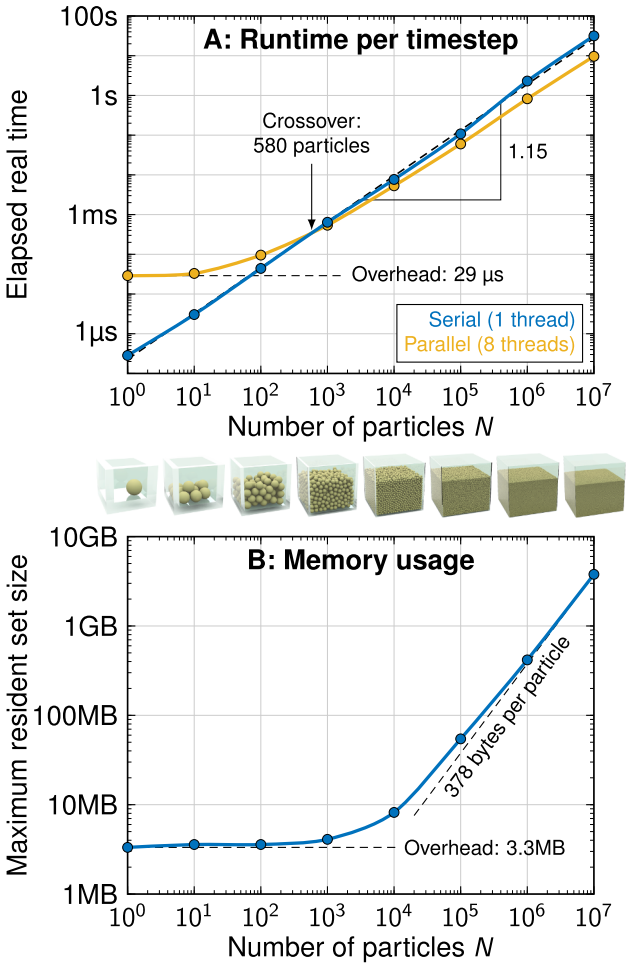


Fig. 6. Computational performance. A: Scaling of the runtime per timestep with the number of particles. The dashed black slope is a fitted power law with annotated exponent. B: Scaling of the maximum memory used with the number of particles. Benchmarks were performed under Red Hat Enterprise Linux 9.4 on a 2019 Intel Xeon Gold 6244 CPU (3.60 GHz, 8 cores). TinyDEM was compiled using GCC 11.2.0 with optimization level 3.

$N^{1/3}$, and allowed to settle under gravity. In this densely packed system, the wallclock time to perform one timestep was measured (averaged over at least a minute to buffer fluctuations) in serial mode and using 8 threads, for up to 10 million particles. At $N = 1000$ particles, one timestep took about 0.64 ms on a 3.6 GHz Intel Xeon Gold 6244 CPU, which is more than an order of magnitude faster than the fastest Fortran implementation tested in 2006 on an 2.4 GHz AMD Athlon CPU in a similar benchmark [90]. This might just about reflect the performance differences between these CPUs. Although all code components scale at most linearly with N, slightly superlinear serial time complexity is observed in practice (Fig. 6A), presumably due to caching and memory bandwidth limitations: runtime $\sim N^{\sigma}$ with $\sigma = 1.154 \pm 0.012$ (S.E.). Similar superlinear scaling (although with larger exponent $\sigma = 1.33$) was previously found for comparable memory-bound vertex simulations [41,91]. In parallel execution, a crossover from parallelization overhead to speedup was observed at about 580 particles (interpolated). The number of particle updates per wall clock second—a quantity that became known as the Cundall number [92]—ranges from $3.5{\times}10^6$ to $3.2{\times}10^5$ in serial execution in this measurement.

In addition to the runtime, in can be useful to have an estimate of the expected memory requirements of a simulation. Fig. 6B shows the maximum memory occupied by TinyDEM in the same set of simulations. Since scenarios with fewer particle contacts require less memory (Table 3), the memory used in this high-density benchmark can be considered an upper bound for other reasonable scenarios, unless the ensemble is spread out enough to let the spatial partitioning grid eat up considerably more memory. The used memory was observed to transition from a constant overhead to linear scaling between about $N = 10^3$ to 10^5 particles. In the largest tested simulation, 378 bytes were used per particle. An ensemble of a million particles requires about 400 MB to simulate.

6. Usage instructions

In the spirit of [41], TinyDEM was developed to be highly portable and only requires a C++11 compiler. For multithreading support, the OpenMP 3.1 specification must be supported, which is the case in ICC since v12.1 (2011), in GCC since v4.7 (2012), and in Clang/LLVM since v3.7 (2015). To compile it, run

```
g++ -fopenmp -O3 -o tinydem tinydem.cpp
```

or an equivalent command. Omitting the `-fopenmp` option compiles TinyDEM for serial execution. To run a simulation, set the desired parameters in lines 8–31 of `tinydem.cpp`, compile it, and execute the binary by typing

```
OMP_NUM_THREADS=8 ./tinydem mesh.off input.csv output
```

or similar. If no number of threads is specified, the selection of a suitable number is delegated to OpenMP.

The three command-line arguments are optional. To use later arguments but skip previous ones, empty strings (`''`) can be used. With the first argument, the path to a text file in GeomView's Object File Format (OFF) [93] can be given to define the mesh geometry. At the end of each face specification in the OFF file, an optional integer index can be specified. While this is intended as a colormap index in the OFF specification, TinyDEM uses it to specify the index in the list of mesh radii R_{mesh} (see the supplied default `mesh.off` file). If omitted, this index is set to 0, such that the first entry in R_{mesh} is the default mesh radius. The second argument is the path to a comma-separated value (CSV) file specifying the particles to start the simulation with. The format is identical to the output files generated by the program itself. For an example, run the default simulation. If none is given, the simulation starts with no particles, but still generates them with the spawning method described earlier. The third argument specifies an output directory. If unspecified, a folder named `output` is generated in the current working directory.

TinyDEM produces a time series of minimal CSV files for maximum compatibility. Existing output is overwritten without prompt. The number of files written is $1 + N_{\text{frames}}$, where N_{frames} is user-specified (Table 2) and the extra 1 is the initial state. Between frames, N_{steps} timesteps of size Δt are performed, such that the total simulated physical time is given by $N_{\text{frames}} N_{\text{steps}} \Delta t$.

The simulated particle ensemble can be visualized in ParaView, for example. Open the CSV file series and apply the TableToPoints filter to it, using the x, y and z columns, with the "Keep All Data Arrays" option enabled. The particles can then be rendered either using the fast "Point Gaussian" representation with the radius R selected in "Use Scalar Array", or using the 3D Glyphs representation with Sphere as the "Glyph Type", Magnitude as "Scale Mode" and R as the "Scale Array". To also apply the quaternion rotation to the rendered particles (for instance to show their orientation with an Arrow glyph), apply a Programmable Filter after the TableToPoints filter with the following Python code:

```
import numpy as np
q = [inputs[0].PointData["q%s"%i] for i in range(4)]
output.PointData.append(np.stack(q, axis=1), "q")
```

This makes the quaternion available as "Orientation Vectors" in ParaView's 3D Glyphs.

7. Conclusion

Since about one and a half decades, open-source DEM codes have become almost a commodity. Table 1 and Fig. 1 may help researchers pick the best code for their use case, but they also provide a historical overview. With a median number of lines of code beyond 90,000, however, the more than 20 available programs are complex and heavy, making it difficult to adapt them to one's own particular needs. Is writing capable custom DEM code a major undertaking, then, that the interested researcher or student should shy away from? Is it necessary to fine-tune numerous model parameters to obtain consistent physical behavior? It is not, as TinyDEM demonstrates.

TinyDEM is a lightweight standalone program to simulate frictional granular media that stands out of the crowd of existing DEM codes by combining two uncommon features:

- *Accessibility.* TinyDEM consists of only just above 600 lines of compact, simple, commented C++ code in two files—a source and a header file. It is easy to handle, maintain, adjust or extend. Without dependencies, it is exceptionally independent and portable.
- *Comprehensive constitutive parameterization.* All three modes of frictional inelastic contacts (sliding, rolling and twisting) are implemented, and the latest constitutive relationships are used to simplify the parameterization to an intuitive set of fundamental material properties (Table 2). No effective modeling parameters need to be guessed or gauged.

TinyDEM is the only publicly available, full-fledged DEM program below 1000 lines of code, the only one below 10,000 lines that includes all three modes of torque exchange, and the only such program below 100,000 lines that is published under a permissive free software license without copyleft. All algorithms are direct and there are no numerical tolerances, magic numbers or similar. With its lean standalone implementation, maximal portability, and permissive 3-clause BSD license, it can serve as a basis for various DEM projects in research and commercial application, and is suited also for educational purposes. As the review of previous implementations revealed, comparable DEM codes are orders of magnitude more voluminous than TinyDEM and many of them have numerous dependencies. While this may not be a problem for some simulation projects, it can be a hindrance for others. In these cases, TinyDEM offers an alternative.

Several possible extensions of the present code are easy to recognize. While a generalization to non-spherical particles (for example through rigid aggregates of spheres [94] or other representations [95]) would require rather deep modifications to the code, extending the contact model by adhesion (typically with either the DMT or the JKR model, see [96] for a review, and LAMMPS for an elegant implementation) is relatively straightforward. Other common DEM features that were left out here for simplicity are periodic boundary conditions, different classes of particles and mesh elements with different material properties, moving and rotating meshes, collisions with more primitive shapes (cylinders, cones, ellipsoids etc.), more flexible particle spawn methods, particle removal, different static and dynamic friction coefficients, a selection of constitutive models to choose from, a parameter file parser, etc.

On the numerical side, certain room exists for performance improvements. The present lean implementation is largely memory-bound, implying that it could likely benefit from parallelization for distributed memory systems with MPI. Moreover, more elaborate bookkeeping of nearby particle pairs—for example with Verlet lists [97] rather than the naive reconstruction of linked cells in every timestep—could potentially speed up some simulations. For dilute or widely dispersed ensembles, binary spatial partitioning with a tree data structure will be more memory efficient than the regular grid used here, at the cost of a more involved implementation. Combined with mesh-aligned bounding boxes or voxelization of large slanted mesh elements, collision detection could potentially be sped up considerably. Such extensions are, however, beyond the scope of this work.

Data availability

No data was used for the research described in the article.

CRediT authorship contribution statement

Roman Vetter: Writing – original draft, Visualization, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Part of this work was funded by ETH Zürich through ETHIIRA Grant no. ETH-03 10-3.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.cpc.2025.109942.

References

[1] P.A. Cundall, O.D.L. Strack, A discrete numerical model for granular assemblies, Géotechnique 29 (1979) 47–65. https://doi.org/10.1680/geot.1979.29.1.47
[2] J. Kozicki, F.V. Donzé, A new open-source software developed for numerical simulations using discrete modeling methods, Comput. Methods Appl. Mech. Engrg. 197 (2008) 4429–4443. https://doi.org/10.1016/j.cma.2008.05.023
[3] M. Dosta, D. Andre, V. Angelidakis, R.A. Caulk, M.A. Celigueta, B. Chareyre, J.-F. Dietiker, J. Girardot, N. Govender, C. Hubert, R. Kobyłka, A.F. Moura, V. Skorych, D.K. Weatherley, T. Weinhart, Comparing open-source DEM frameworks for simulations of common bulk processes, Comput. Phys. Commun. 296 (2024) 109066. https://doi.org/10.1016/j.cpc.2023.109066
[4] M. Dutt, B. Hancock, C. Bentham, J. Elliott, An implementation of granular dynamics for simulating frictional elastic particles based on the DL_POLY code, Comput. Phys. Commun. 166 (2005) 26–44. https://doi.org/10.1016/j.cpc.2004.10.006
[5] Y. Wang, P. Mora, The ESyS_Particle: A New 3-D Discrete Element Model with Single Particle Rotation, vol. 119 of *Lecture Notes in Earth Sciences*, Springer, 2009, pp. 183–228. https://doi.org/10.1007/978-3-540-85879-9_6
[6] C. Kloss, C. Goniva, Granular Simulations in LAMMPS: New Key Features and Perspectives, 2010, https://www.lammps.org/workshops/Feb10/Christoph_Kloss/granular.pdf.
[7] S.A. Galindo-Torres, MechSys: Multi-physics simulation library, 2009, https://mechsys.nongnu.org.
[8] V. Šmilauer, M. Kotrč, Woo, 2010, https://github.com/woodem/woo.
[9] P. Dadvand, R. Rossi, E. Oñate, An object-oriented environment for developing finite element codes for multi-disciplinary applications, Arch. Computat. Methods Eng. 17 (2010) 253–297. https://doi.org/10.1007/s11831-010-9045-2
[10] C. Kloss, C. Goniva, LIGGGHTS - Open Source Discrete Element Simulations of Granular Materials Based on LAMMPS, vol. 2, John Wiley & Sons, Ltd, 2011, pp. 781–788. https://doi.org/10.1002/9781118062142.ch94
[11] 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
[12] D. Nishiura, M.Y. Matsuo, H. Sakaguchi, ppohDEM: computational performance for open source code of the discrete element method, Comput. Phys. Commun. 185 (2014) 1486–1495. https://doi.org/10.1016/j.cpc.2014.02.014
[13] D. André, J.-L. Charles, I. Iordanoff, J. Néauport, The GranOO workbench, a new tool for developing discrete element simulations, and its application to tribological problems, Adv. Eng. Softw. 74 (2014) 40–48. https://doi.org/10.1016/j.advengsoft.2014.04.003
[14] 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
[15] N. Govender, D.N. Wilke, S. Kok, Blaze-DEMGPU: modular high performance DEM framework for the GPU architecture, SoftwareX 5 (2016) 62–66. https://doi.org/10.1016/j.softx.2016.04.004
[16] H.R. Norouzi, R. Zarghami, R. Sotudeh-Gharebagh, N. Mostoufi, Coupled CFD-DEM Modeling: Formulation, Implementation and Application to Multiphase Flows, Wiley, 2016.
[17] B. Yan, R.A. Regueiro, A comprehensive study of MPI parallelism in three-dimensional discrete element method (DEM) simulation of complex-shaped granular particles, Comp. Part. Mech. 5 (2018) 553–577. https://doi.org/10.1007/s40571-018-0190-y

[18] J.H. Walther, I.F. Sbalzarini, Large-scale parallel discrete element simulations of granular flow, Eng. Comput. 26 (2009) 688–697. https://doi.org/10.1108/02644400910975478

[19] P. Incardona, A. Leo, Y. Zaluzhnyi, R. Ramaswamy, I.F. Sbalzarini, OpenFPM: a scalable open framework for particle and particle-mesh codes on parallel computers, Comput. Phys. Commun. 241 (2019) 155–177. https://doi.org/10.1016/j.cpc.2019.03.007

[20] 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

[21] M. Dosta, V. Skorych, MUSEN: an open-source framework for GPU-accelerated DEM simulations, SoftwareX 12 (2020) 100618. https://doi.org/10.1016/j.softx.2020.100618

[22] C. Kelly, N. Olsen, D. Negrut, Billion degree of freedom granular dynamics simulation on commodity hardware via heterogeneous data-type representation, Multibody Syst. Dyn. 50 (2020) 355–379. https://doi.org/10.1007/s11044-020-09749-7

[23] L. Fang, R. Zhang, C. Vanden Heuvel, R. Serban, D. Negrut, Chrono::GPU: an open-source simulation package for granular dynamics using the discrete element method, Processes 9 (2021) 1813. https://doi.org/10.3390/pr9101813

[24] S. Zhao, J. Zhao, SudoDEM: unleashing the predictive power of the discrete element method on simulation for non-spherical granular particles, Comput. Phys. Commun. 259 (2021) 107670. https://doi.org/10.1016/j.cpc.2020.107670

[25] S. Golshan, P. Munch, R. Gassmöller, M. Kronbichler, B. Blais, Lethe-DEM: an open-source parallel discrete element solver with load balancing, Comp. Part. Mech. 10 (2023) 77–96. https://doi.org/10.1007/s40571-022-00478-6

[26] B. Cheng, DEMBody User Manual (in Chinese, Tsinghua University), 2022.

[27] 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

[28] Y.-C. Qian, R.-R. Cai, L.Z. Zhang, A spheropolyhedral-based discrete element lattice Boltzmann method for simulation of non-spherical adhesive particulate flow, Comput. Phys. Commun. 291 (2023) 108809. https://doi.org/10.1016/j.cpc.2023.108809

[29] Z. Gong, Z. Wu, C. An, B. Zhang, X. Fu, CP3d: a comprehensive Euler-Lagrange solver for direct numerical simulation of particle-laden flows, Comput. Phys. Commun. 286 (2023) 108666. https://doi.org/10.1016/j.cpc.2023.108666

[30] R. Zhang, B. Tagliafierro, C. Vanden Heuvel, S. Sabarwal, L. Bakke, Y. Yue, X. Wei, R. Serban, D. Negrut, Chrono DEM-engine: a discrete element method dual-GPU simulator with customizable contact forces and element shape, Comput. Phys. Commun. 300 (2024) 109196. https://doi.org/10.1016/j.cpc.2024.109196

[31] Y. Wang, P. Mora, Modeling wing crack extension: implications for the ingredients of discrete element model, Pure Appl. Geophys. 165 (2008) 609–620. https://doi.org/10.1007/s00024-008-0315-y

[32] W.F. Oquendo, J.D. Muñoz, A. Lizcano, Influence of rotations on the critical state of soil mechanics, Comput. Phys. Commun. 182 (2011) 1860–1865. https://doi.org/10.1016/j.cpc.2010.11.018

[33] B. Remy, J.G. Khinast, B.J. Glasser, Discrete element simulation of free flowing grains in a four-bladed mixer, AIChE J. 55 (2009) 2035–2048. https://doi.org/10.1002/aic.11876

[34] A.S.J. Suiker, N.A. Fleck, Frictional collapse of granular assemblies, J. Appl. Mech. 71 (2004) 350–358. https://doi.org/10.1115/1.1753266

[35] A. Burri, TinyFSM, https://github.com/digint/tinyfsm.

[36] A.J. Wilmottt, Simple Vector Library, https://www.cs.cmu.edu/%7Eajw/doc/svl.html.

[37] S. Zaitsev, jsmn, https://github.com/zserge/jsmn.

[38] L. Thomason, TinyXML-2, https://github.com/leethomason/tinyxml2.

[39] R.R.L. Machado, J. Schmitt, S. Eibl, J. Eitzinger, R. Leißa, S. Hack, A. Pérard-Gayot, R. Membarth, H. Köstler, tinyMD: mapping molecular dynamics simulations to heterogeneous hardware using partial evaluation, J. Comput. Sci. 54 (2021) 101425. https://doi.org/10.1016/j.jocs.2021.101425

[40] R. Vetter, J.O. Schumacher, Free open reference implementation of a two-phase PEM fuel cell model, Comput. Phys. Commun. 234 (2019) 222–234. https://doi.org/10.1016/j.cpc.2018.07.023

[41] R. Vetter, S.V.M. Runser, D. Iber, PolyHoop: soft particle and tissue dynamics with topological transitions, Comput. Phys. Commun. 299 (2024) 109128. https://doi.org/10.1016/j.cpc.2024.109128

[42] D. Tschumperlé, The CImg library, in: IPOL 2012 Meeting on Image Processing Libraries, Cachan, France, 2012. https://hal.science/hal-00927458.

[43] M.A. Heroux, D.W. Doerfler, P.S. Crozier, J.M. Willenbring, H.C. Edwards, A. Williams, M. Rajan, E.R. Keiter, H.K. Thornquist, R.W. Numrich, Improving Performance via Mini-applications, Technical Report SAND2009-5574, Sandia National Laboratories, 2009. https://mantevo.github.io/pdfs/MantevoOverview.pdf.

[44] L.D. Landau, E.M. Lifshitz, Fluid Mechanics, Pergamon Press, 2 ed., 1987. https://doi.org/10.1016/C2013-0-03799-1

[45] C.S. O'Hern, L.E. Silbert, A.J. Liu, S.R. Nagel, Jamming at zero temperature and zero applied stress: the epitome of disorder, Phys. Rev. E 68 (2003) 011306. https://doi.org/10.1103/PhysRevE.68.011306

[46] R.D. Mindlin, Compliance of elastic bodies in contact, J. Appl. Mech. 16 (1949) 259–268. https://doi.org/10.1115/1.4009973

[47] R.D. Mindlin, H. Deresiewicz, Elastic spheres in contact under varying oblique force, J. Appl. Mech. 20 (1953) 327–344. https://doi.org/10.1115/1.4010702

[48] D. Antypov, J.A. Elliott, On an analytical solution for the damped Hertzian spring, EPL 94 (2011) 50004. https://doi.org/10.1209/0295-5075/94/50004

[49] E.H. Jakubowski, Dynamic Formulation of Coefficient of Restitution, Technical Report SA-TR20-2811, Springfield Armory, Springfield, Massachusetts, 1964. https://apps.dtic.mil/sti/pdfs/AD0603711.pdf.

[50] J. Schäfer, S. Dippel, D.E. Wolf, Force schemes in simulations of granular materials, J. Phys. I France 6 (1996) 5–20. https://doi.org/10.1051/jp1:1996129

[51] N.V. Brilliantov, F. Spahn, J.-M. Hertzsch, T. Pöschel, Model for collisions in granular gases, Phys. Rev. E 53 (1996) 5382–5392. https://doi.org/10.1103/PhysRevE.53.5382

[52] T. Schwager, V. Becker, T. Pöschel, Coefficient of tangential restitution for viscoelastic spheres, Eur. Phys. J. E 27 (2008) 107–114. https://doi.org/10.1140/epje/i2007-10356-3

[53] S. Luding, Collisions & contacts between two particles, in: H.J. Herrmann, J.P. Hovi, S. Luding (Eds.), Physics of Dry Granular Media, 350 of *NATO ASI Series*, Springer, 1998, pp. 285–304. https://doi.org/10.1007/978-94-017-2653-5_20

[54] Y. Tsuji, T. Tanaka, T. Ishida, Lagrangian numerical simulation of plug flow of cohesionless particles in a horizontal pipe, Powder Technol. 71 (1992) 239–250. https://doi.org/10.1016/0032-5910(92)88030-L

[55] Y.C. Zhou, B.D. Wright, R.Y. Yang, B.H. Xu, A.B. Yu, Rolling friction in the dynamic simulation of sandpile formation, Phys. A Stat. Mech. Appl. 269 (1999) 536–553. https://doi.org/10.1016/S0378-4371(99)00183-1

[56] L.E. Silbert, G.S. Grest, J.W. Landry, Statistics of the contact network in frictional and frictionless granular packings, Phys. Rev. E 66 (2002) 061303. https://doi.org/10.1103/PhysRevE.66.061303

[57] A.P. Santos, D.S. Bolintineanu, G.S. Grest, J.B. Lechman, S.J. Plimpton, I. Srivastava, L.E. Silbert, Granular packings with sliding, rolling, and twisting friction, Phys. Rev. E 102 (2020) 032903. https://doi.org/10.1103/PhysRevE.102.032903

[58] Y.-C. Qian, R.-R. Cai, L.Z. Zhang, Numerical simulation of mixed aerosols deposition behavior on cylindrical cross fibers, Adv. Powder Technol. 33 (2022) 103849. https://doi.org/10.1016/j.apt.2022.103849

[59] L.E. Silbert, D. Ertaş, G.S. Grest, T.C. Halsey, D. Levine, S.J. Plimpton, Granular flow down an inclined plane: Bagnold scaling and rheology, Phys. Rev. E 64 (2001) 051302. https://doi.org/10.1103/PhysRevE.64.051302

[60] C. Ringl, H.M. Urbassek, A LAMMPS implementation of granular mechanics: inclusion of adhesive and microscopic friction forces, Comput. Phys. Commun. 183 (2012) 986–992. https://doi.org/10.1016/j.cpc.2012.01.004

[61] J. Ai, J.-F. Chen, J.M. Rotter, J.Y. Ooi, Assessment of rolling resistance models in discrete element simulations, Powder Technol. 206 (3) (2011) 269–282. https://doi.org/10.1016/j.powtec.2010.09.030

[62] Y. Wang, F. Alonso-Marroquin, W.W. Guo, Rolling and sliding in 3-D discrete element models, Particuology 23 (2015) 49–55. https://doi.org/10.1016/j.partic.2015.01.006

[63] C. Zhao, C. Li, Influence of rolling resistance on the shear curve of granular particles, Phys. A Stat. Mech. Appl. 460 (2016) 44–53. https://doi.org/10.1016/j.physa.2016.04.043

[64] C. Zhao, Y. Luo, L. Hu, C. Li, Suitable rolling resistance model for quasi-static shear tests of non-spherical particles via discrete element method, Granul. Matter 20 (2018) 66. https://doi.org/10.1007/s10035-018-0837-7

[65] M. Jiang, Z. Shen, J. Wang, A novel three-dimensional contact model for granulates incorporating rolling and twisting resistances, Comput. Geotech. 65 (2015) 147–163. https://doi.org/10.1016/j.compgeo.2014.12.011

[66] S. Luding, Cohesive, frictional powders: contact models for tension, Granul. Matter 10 (2008) 235–246. https://doi.org/10.1007/s10035-008-0099-x

[67] J.S. Marshall, Discrete-element modeling of particulate aerosol flows, J. Comput. Phys. 228 (2009) 1541–1561. https://doi.org/10.1016/j.jcp.2008.10.035

[68] N.V. Brilliantov, T. Pöschel, Rolling friction of a viscous sphere on a hard plane, Europhys. Lett. 42 (1998) 511–516. https://doi.org/10.1209/epl/i1998-00281-7

[69] K. Iwashita, M. Oda, Rolling resistance at contacts in simulation of shear band development by DEM, J. Eng. Mech. 124 (1998) 285–292. https://doi.org/10.1061/(ASCE)0733-9399(1998)124:3(285)

[70] E.M.B. Campello, A description of rotations for DEM models of particle systems, Comput. Part. Mech. 2 (2015) 109–125. https://doi.org/10.1007/s40571-015-0041-z

[71] C.A. del Valle, V. Angelidakis, S. Roy, J.D. Muñoz, T. Pöschel, SPIRAL: an efficient algorithm for the integration of the equation of rotational motion, Comput. Phys. Commun. 297 (2024) 109077. https://doi.org/10.1016/j.cpc.2023.109077

[72] Y. Li, Y. Xu, C. Thornton, A comparison of discrete element simulations and experiments for 'sandpiles' composed of spherical particles, Powder Technol. 160 (2005) 219–228. https://doi.org/10.1016/j.powtec.2005.09.002

[73] S.J. Burns, P.T. Piiroinen, K.J. Hanley, Critical time step for DEM simulations of dynamic systems using a Hertzian contact model, Int. J. Numer. Methods Eng. 119 (2019) 432–451. https://doi.org/10.1002/nme.6056

[74] B. Quentrec, C. Brot, New method for searching for neighbors in molecular dynamics computations, J. Comput. Phys. 13 (1973) 430–432. https://doi.org/10.1016/0021-9991(73)90046-6

[75] C. Ericson, Real-Time Collision Detection, Morgan Kaufmann, San Francisco, San Francisco, 2005.

[76] I. Unity Software, Unity User Manual 1.0, Collision detection and welding overview, 2005, https://docs.unity3d.com/Packages/com.havok.physics@1.0/manual/collision-detection.html.

[77] F. Fleissner, T. Gaugele, P. Eberhard, Applications of the discrete element method in mechanical engineering, Multibody Syst. Dyn. 18 (2007) 81–94. https://doi.org/10.1007/s11044-007-9066-2

[78] P. Terdiman, Contact generation for meshes, 2015, https://www.codercorner.com/MeshContacts.pdf.

[79] J.J. Alonso, H.J. Herrmann, Shape of the tail of a two-dimensional sandpile, Phys. Rev. Lett. 76 (1996) 4911–4914. https://doi.org/10.1103/PhysRevLett.76.4911
[80] Y. Grasselli, H.J. Herrmann, On the angles of dry granular heaps, Physica A 246 (1997) 301–312. https://doi.org/10.1016/S0378-4371(97)00326-9
[81] Y.C. Zhou, B.H. Xu, A.B. Yu, P. Zulli, Numerical investigation of the angle of repose of monosized spheres, Phys. Rev. E 64 (2001) 021301. https://doi.org/10.1103/PhysRevE.64.021301
[82] R.P. Behringer, B. Chakraborty, The physics of jamming for granular materials: a review, Rep. Prog. Phys. 82 (2018) 012601. https://doi.org/10.1088/1361-6633/aadc3c
[83] I. Zuriguel, A. Garcimartín, D. Maza, L.A. Pugnaloni, J.M. Pastor, Jamming during the discharge of granular matter from a silo, Phys. Rev. E 71 (2005) 051303. https://doi.org/10.1103/PhysRevE.71.051303
[84] L. Pournin, M. Ramaioli, P. Folly, T.M. Liebling, About the influence of friction and polydispersity on the jamming behavior of bead assemblies, Eur. Phys. J. E 23 (2007) 229–235. https://doi.org/10.1140/epje/i2007-10176-5
[85] A. Garcimartín, I. Zuriguel, L.A. Pugnaloni, A. Janda, Shape of jamming arches in two-dimensional deposits of granular materials, Phys. Rev. E 82 (2010) 031306. https://doi.org/10.1103/PhysRevE.82.031306
[86] L.L. Zhao, Y.W. Li, X.D. Yang, Y. Jiao, Q.F. Hou, DEM study of size segregation of wet particles under vertical vibration, Adv. Powder Technol. 30 (2019) 1386–1399. https://doi.org/10.1016/j.apt.2019.04.019
[87] T. Elperin, E. Golshtein, Effects of convection and friction on size segregation in vibrated granular beds, Phys. A Stat. Mech. Appl. 247 (1997) 67–78. https://doi.org/10.1016/S0378-4371(97)00400-7
[88] J. Sun, F. Battaglia, S. Subramaniam, Dynamics and structures of segregation in a dense, vibrating granular bed, Phys. Rev. E 74 (2006) 061307. https://doi.org/10.1103/PhysRevE.74.061307
[89] R. Halver, G. Sutmann, Multi-threaded construction of neighbour lists for particle systems in OpenMP, in: R. Wyrzykowski, E. Deelman, J. Dongarra, K. Karczewski, J. Kitowski, K. Wiatr (Eds.), Parallel Processing and Applied Mathematics, 9574 of *Lecture Notes in Computer Science*, 2016, pp. 153–165. https://doi.org/10.1007/978-3-319-32152-3_15
[90] R. Balevičius, A. Džiugys, R. Kačianauskas, A. Maknickas, K. Vislavičius, Investigation of performance of programming approaches and languages used for numerical simulation of granular material by the discrete element method, Comput. Phys. Commun. 175 (2006) 404–415. https://doi.org/10.1016/j.cpc.2006.05.006
[91] S. Runser, R. Vetter, D. Iber, SimuCell3D: three-dimensional simulation of tissue mechanics with cell polarization, Nat. Comput. Sci. 4 (2024) 299–309. https://doi.org/10.1038/s43588-024-00620-9
[92] M.A. Hopkins, Discrete element modeling with dilated particles, Eng. Comput. 21 (2004) 422–430. https://doi.org/10.1108/02644400410519866
[93] M. Phillips, S. Levy, T. Munzner, GeomView Manual, The Geometry Center, University of Minnesota, 2007. Section 4.2.5, http://www.geomview.org/docs/geomview.pdf.
[94] N. Bell, Y. Yu, P.J. Mucha, Particle-based simulation of granular materials, in: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '05, 2005, pp. 77–86. https://doi.org/10.1145/1073368.1073379
[95] J. Zhao, S. Zhao, S. Luding, The role of particle shape in computational modelling of granular matter, Nat. Rev. Phys. (2023). https://doi.org/10.1038/s42254-023-00617-9
[96] E. Barthel, Adhesive elastic contacts: JKR and more, J. Phys. D Appl. Phys. 41 (2008) 163001. https://doi.org/10.1088/0022-3727/41/16/163001
[97] L. Verlet, Computer "Experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules, Phys. Rev. 159 (1967) 98–103. https://doi.org/10.1103/PhysRev.159.98