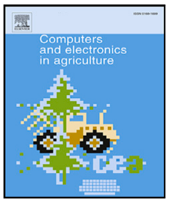
Contents lists available at ScienceDirect

Computers and Electronics in Agriculture

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



Development of a GPU-based DEM solver for parameter optimization in the simulations of soil-sweep tool interactions

Dániel Nagy [a], László Pásthy [b], Kornél Tamás [b,*]

[a] Department of Hydrodynamic Systems, Faculty of Mechanical Engineering, Budapest University of Technology and Economics, Muegyetem rakpart 3, Budapest, 1111, Hungary

[b] Department of Machine and Product Design, Faculty of Mechanical Engineering, Budapest University of Technology and Economics, Muegyetem rakpart 3, Budapest, 1111, Hungary

ARTICLE INFO

Keywords:
DEM
Soil simulation
GPU
Numerical optimization
Parameter fitting

ABSTRACT

The Discrete Element Method (DEM) is a powerful technique for simulating granular materials in agricultural applications, yet it is notoriously computationally and memory-intensive. A typical DEM simulation of soil-sweep tool interactions involves tens of thousands of particles, each of which may potentially interact with many others. This results in hundreds of thousands of interactions being computed at every time step, while the need for numerical stability often requires very small time steps. Despite these challenges, DEM holds significant promise to allow the design of more efficient agricultural tools. This paper introduces an in-house-developed modular library based on CUDA C++ for GPUs, aimed at accelerating DEM simulations using a single GPU. The library is designed to facilitate efficient memory usage, employing a per-thread approach where each GPU thread computes one discrete-element particle. To accelerate particle–particle contact searches, we implemented a cell-linked-list algorithm. Our library utilizes the Hertz–Mindlin contact model, which has been widely adopted in agricultural DEM applications. Validation of the code was performed through comparisons with commercial software. Using our software, experimental measurements of a sweep tool moving through sandy soil were replicated with high accuracy by employing differential evolution for parameter calibration, achieving these results using 38 912 particles and running 2 700 instances within 16 h on a single GPU.

1. Introduction

Enhancing soil quality and adopting sustainable management practices can not only boost food production but also assist in the conservation of natural resources. Despite soil tillage being a thousands of years old practice, its technology continues to advance, yielding new soil tillage tools, for instance, the narrow tillage tools used for conservation tillage (Odey, 2016). In contemporary times, there is a growing emphasis on environmentally friendly tillage practices (Pratibha et al., 2019; Busari et al., 2015). The primary objectives of tillage tool design include energy conservation, soil erosion control, and emissions reduction (Rádics and Jóri, 2010). However, designing and selecting the appropriate tillage tools pose significant challenges, given the variability in soil types.

Mathematical modeling of soil-sweep tool interactions presents opportunities to find theoretical solutions to the above-mentioned challenges, such as by optimizing sweep tools for increased energy efficiency. While analytical models offer insights into the draught force necessary to move a plough at a given speed (Gupta et al., 1989; Onwualu and Watts, 1998; Saunders et al., 2000), they provide only limited information. Although validated analytical models can facilitate the determination of optimal ploughing depth and speed (Godwin et al., 2007), they lack the capability for geometry optimization. Alternatively, the finite element method (FEM) can be employed to simulate soil behavior (Shen, 2017; Bentaher et al., 2013; Abo-Elnor et al., 2003). FEM enables the optimization of tool geometry and the simulation of soil movement around the cutting edge of the tool (Fielke, 1999). However, a drawback of FEM is its assumption of a continuous medium, which may not always align with real-world conditions. Moreover, FEM cannot adequately model soil deformation, the development of cracks, or the flow of soil particles (Asaf et al., 2007). Over the past two decades, the Discrete Element Method (DEM) has increasingly been adopted for soil simulations. DEM offers the advantage of accounting for soil heterogeneity and determining the flow of soil particles. It enables the simulation of various agricultural problems, including the determination of draught forces (Ucgul et al., 2017; Li et al., 2024), the exploration of cracks in the soil produced by sweep tools (Tamás

* Corresponding author.
E-mail addresses: dnagy@hds.bme.hu (D. Nagy), pasthy.laszlo@gt3.bme.hu (L. Pásthy), tamas.kornel@gt3.bme.hu (K. Tamás).

https://doi.org/10.1016/j.compag.2024.109482
Received 6 May 2024; Received in revised form 16 September 2024; Accepted 21 September 2024

et al., 2013), the movement of soil particles and seeds (Qi et al., 2019; Milkevych et al., 2018; Binelo et al., 2019) and rotary tillage (Huimin et al., 2016; Ucgul et al., 2018). Furthermore, DEM can be modified to take into account the presence of stem residues in the soil (Tamás and Bernon, 2021). The aforementioned studies mostly modeled large particles with a limited number of parameters.

Calibrating micromechanical DEM parameters is essential for ensuring physically accurate results, but this process requires the handling of a large number of parameters. For instance, Roessler and Katterfeld (2019) calibrated micromechanical DEM parameters of a cohesive bulk using 10 000 particles and 576 parameters (8 × 8 × 9). Similarly, Westbrink et al. (2021) conducted 10 648 DEM simulations with various micromechanical parameters to create a training dataset, with each dynamic simulation containing 29 681 particles. The primary challenge for achieving more accurate simulations with smaller particles is the runtime (Aikins et al., 2023), as typical soil-sweep tool simulations take over a day even when using larger particles (Tamás and Bernon, 2021). Enhancing the efficiency of DEM software would allow for higher resolution studies and the exploration of more parameters, potentially yielding new insights into soil-sweep tool interactions. To conduct such extensive numerical simulations, parallelization is crucial to reduce runtime. With current computing technology, large-scale simulations involving millions of particles are now feasible (Gan et al., 2016; Dosta and Skorych, 2020).

Central Processing Units (CPUs) historically used the *single instruction single data* (SISD) architecture, executing one operation on a single piece of data per machine cycle. Modern CPU architectures allow for parallelization through multiple cores and vectorization. Graphics Processing Units (GPUs) operate based on the principle of single instruction multiple thread (SIMT) on a large scale. This means they can execute a single instruction, such as multiplication or addition, across multiple pieces of data simultaneously using different threads, enabling more efficient processing of vast quantities of the same operation. In contemporary computing, there is a trend towards utilizing specialized tools like graphics and tensor processors (GPUs and TPUs). One of the most widely used development environments for GPUs is CUDA, an NVIDIA proprietary platform and parallel programming model which is exclusively compatible with NVIDIA GPUs (Karimi et al., 2010). Another common platform is OpenCL, which can operate in any shared memory parallel environment, including GPU and CPU clusters (Munshi, 2009). The primary advantages of CUDA are its higher speed and broader toolbox, largely attributed to the fact that both the programming model and the GPU were developed by NVIDIA, resulting in a high level of integration. The only notable disadvantage is its platform dependency. In this paper, the CUDA programming model will be utilized.

GPU-based DEM simulations have been around for several years, but in most cases, these are programs developed by a research group to address a specific problem that requires a large number of particles (Liu et al., 2020; Govender et al., 2019, 2015). Some GPU-based DEM solvers have been developed for general problems, such as Chrono::GPU, which can simulate tens of millions of particles (Fang et al., 2021), but the drawback of this package is that it can only use one type of material and all the particles must have the same size. Chrono DEM-Engine is a novel package that addresses the shortcoming of Chrono::GPU, in which clump-based elements of arbitrary sizes can be used (Zhang et al., 2024). A promising GPU accelerated DEM software is Musen (Dosta and Skorych, 2020), that is capable of tackling a wide range of problems from fracture processes (Hilarov et al., 2023) to modeling collisions between ships (Kraus et al., 2021) involving multiple materials. However it is not a pure GPU solver, as its contact detection is CPU-based. Altair EDEM® also has an option to run its calculations on a GPU, but the experience with that is limited. Although GPU-based soil simulations are described in the literature, they are limited to a few specific cases. GPUs have been utilized for simulating off-road tire-granular terrain interactions (Yang et al., 2020) and for modeling landslides (Zhang et al., 2023a; Zhou et al., 2021; Zhang et al., 2023b). The application of GPUs in agricultural DEM simulations has the potential to reduce runtime and enable efficient numerical optimization of tools. To date, however, based on our literature review there have been no examples of modeling the movement of an object (e.g., tillage tool) through a domain filled with particles and calculating the draught force in GPU-based DEM simulations. Currently, GPU-based DEM simulations are not widely used in agriculture, although, this approach could significantly accelerate simulations and contribute to a deeper understanding of soil-sweep tool interactions through more detailed studies involving smaller particles.

Our aim is to build a modular GPU-based DEM solver in CUDA C++, which can be reliably used to create large-scale soil simulations while operating faster than existing software thanks to the GPU acceleration. This would enable more effective parameter calibrations, addressing a major limitation in the current application of DEM in agriculture. The software should be capable of modeling moving objects at a desired velocity and calculating the resulting draught force, making it particularly suitable for agricultural applications. This capability sets our tool apart from current GPU-based DEM solvers, which lack this functionality, to the authors' knowledge. In addition to the creation of an effective DEM software application, this paper has a number of other objectives:

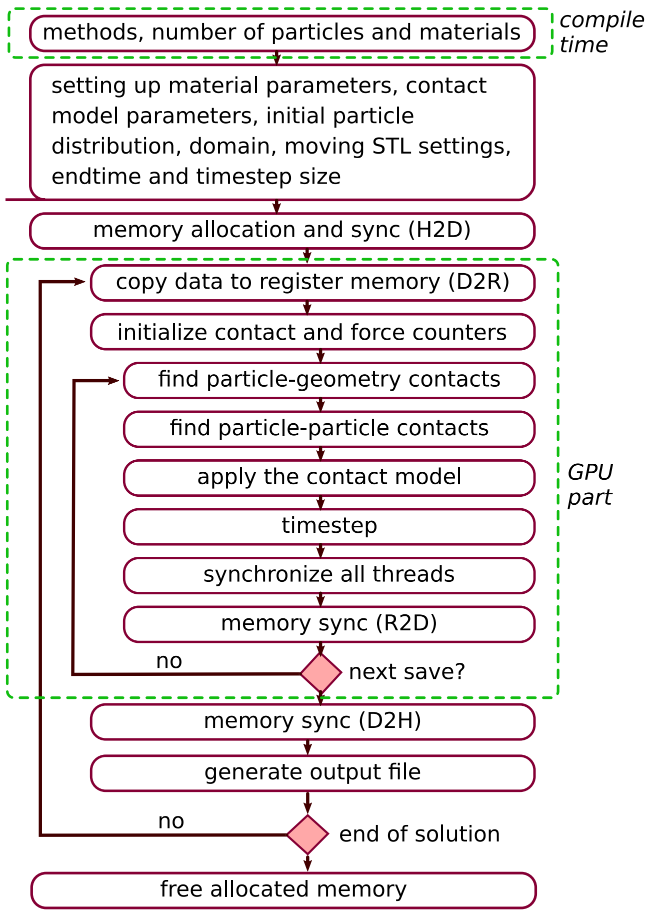
- Detailing the GPU specific implementation of the DEM algorithm and validating the code by comparison to commercial software. The per-thread approach we implemented, which assigns each particle to a GPU thread, is novel for GPU-based DEM and enables extremely high performance by means of low-level optimizations.
- Justifying the use of the Euler method for numerical time step in DEM, through simulations and theoretical considerations. The literature typically uses the first-order explicit Euler method (Kruggel-Emden et al., 2008), but this choice is not justified in most studies.
- Due to the significantly lower runtime offered by GPU acceleration, a further aim of this paper is to investigate parameter sensitivity in simulations of soil-sweep tool interaction, at a higher resolution than previous authors.
- The ultimate aim of this paper is to demonstrate that micromechanical DEM parameters can be calibrated to accurately reproduce soil-bin measurement results. Calibration is a lengthy process that requires several days of compute time (Mohajeri et al., 2020; Lubbe et al., 2022); our aim is to show that with GPUs it can be carried out much faster.

2. Materials and methods

2.1. Graphical processing units

The fundamental computational unit in a GPU is the thread, and there can be hundreds of millions of threads concurrently on a GPU. Unlike CPU threads, GPU threads typically handle smaller amounts of data, making them significantly lighter in terms of workload. Each thread on the GPU possesses its own identifier (index), enabling access to distinct data from memory based on this identifier. Moreover, each GPU has multiple Streaming Multiprocessors (SMs) that function independently. This is the highest hierarchical level, and these multiprocessors contain the computing units. GPU memory consists of global memory and L2 Cache, accessible to the entire GPU. Each multiprocessor has its own registers, local memory, L1 Cache, and user-programmable shared memory. Global memory, the highest memory level, is large but slow, with frequently accessed data automatically moved to the faster L1 Cache. Register memory, located near compute units, is exceptionally fast and user-programmable.

Threads are initially assigned to multiprocessors on a block-by-block basis, with each block containing multiple threads, a parameter which is adjustable by the user. Blocks are further subdivided into warps,



methods, number of particles and materials
compile time
setting up material parameters, contact model parameters, initial particle distribution, domain, moving STL settings, endtime and timestep size
memory allocation and sync (H2D)
copy data to register memory (D2R)
initialize contact and force counters
find particle-geometry contacts
find particle-particle contacts
apply the contact model
timestep
synchronize all threads
memory sync (R2D)
GPU part
no
next save?
memory sync (D2H)
generate output file
no
end of solution
free allocated memory

Fig. 1. The structure of GPUDEM. Each block represents an element of the program, with loops depicted by backward arrows. The green frame at the top is already computed at compile time, while the middle green frame indicates the part of the program running on the GPU. Notations: H2D — Host to Device (copy from CPU to GPU); D2H — Device to Host (copy from GPU to CPU); R2D — Register to Device (copy from GPU registers to GPU global memory); D2R — Device to Register (copy from GPU global memory to GPU registers).

each comprising 32 threads in current NVIDIA GPU architectures. A warp scheduler on each multiprocessor executes a warp when resources become available. Warps operate according to the SIMT structure, where each thread executes the same operation on different data simultaneously, resulting in substantial computational capacity compared to CPUs. However, this presents challenges with branching, which may potentially cause thread divergence, whenever individual threads need to execute different operations such as if–else conditions.

2.2. Program architecture

The program can handle an arbitrary number and size of particles, materials and boundary conditions, and at least one movable geometry, which is important for agricultural applications. Modularity is an essential charactieristic of the program to allow the use of different contact search algorithms, contact models and time step schemes and to allow for future additions to the program. Currently, the Hertz–Mindlin contact model is implemented with first and second order time step schemes. This in-house developed software is referred to as GPUDEM. The code has also been made open-source and available under the GPL-3.0 license.[1]

Table 1
Properties of the particle storage structure, stored in GPUDEM for each particle. Reciprocals of commonly used values are also stored to reduce runtime.

Notation	Dimension	Name
$\boldsymbol{u}$	m	Position vector
$\boldsymbol{v}$	m/s	Velocity vector
$\boldsymbol{a}$	m/s^2	Acceleration vector
$\boldsymbol{\omega}$	1/s	Angular velocity vector
$\boldsymbol{\beta}$	1/s^2	Angular acceleration vector
$\boldsymbol{F}$	N	Force vector
$\boldsymbol{M}$	N m	Torque vector
R	m	Particle radius
m	kg	Particle mass
θ	kg m^2	Moment of inertia
$\mathrm{m_{id}}$	–	Material index
$\mathrm{c_{id}}$	–	Cell index
R^{-1}		Radius^{-1}
m^{-1}		Mass^{-1}
θ^{-1}		Moment of inertia^{-1}

The complete GPUDEM program encompasses CPU, GPU, and compile-time computations. During the compilation of the program package, various parameters such as the number of particles and materials, the selected contact model, the contact search algorithm, and the time step scheme must already be determined. These parameters enable the optimization of the program code by the compiler. The primary functions of the CPU include reading initial conditions, setting material parameters, managing memory, and generating output files. In contrast, the GPU handles the computationally intensive tasks, including contact search, force calculation, and time stepping. The complete architecture is illustrated in Fig. 1. The subsequent sections present the GPU-specific implementation of the DEM algorithm, covering aspects such as particle storage in the GPU memory, particle–particle contact, particle–wall contact, force computation and temporal solution. Given the interrelated nature of computational, physical, and mathematical details, they are presented together, as they often cannot be disentangled.

The program utilizes the *perThread* approach, wherein a GPU thread is assigned to each particle. In Fig. 1, the number of threads initiated in parallel at the GPU kernel call (denoted by the dashed frame) matches the previously designated number of particles. However, the number of particles usually surpasses the count of CUDA cores in the GPU, resulting in not all threads being allocated computational resources simultaneously. Consequently, situations may arise where threads are not synchronized in the computation, potentially leading to non-physical outcomes. To circumvent this issue, the program has synchronization points at each time step, enabling threads to await each other's completion before proceeding.

2.3. Particles and their storage

In the following discussion, N_P represents the number of particles, and N_M indicates the number of materials. Every particle possesses the properties included in Table 1. Notably, one of the particle attributes is its material, which is stored in a distinct structure; particles merely store the associated material index. During a simulation, multiple material parameter sets can be designated, with each particle being assigned a corresponding material index, as mentioned earlier. The mass and moment of inertia of each particle are computed based on their specified density, given that the particles are spherical, thus fulfilling the equations

$$m = \frac{4}{3} \cdot R^3 \cdot \pi \cdot \rho \text{ and} \tag{1}$$

$$\theta = \frac{2}{5} \cdot m \cdot R^2. \tag{2}$$

The reciprocal of the radius, mass, and inertia of the particles are also stored to reduce computational demand, as these values are required in each time step. Division operations are more computationally

[1] Link to the repository: https://github.com/nnagyd/GPUDEM.

expensive than multiplication operations, requiring approximately an order of magnitude more compute time than multiplication (Granlund, 2012).

Moreover, for each possible material-material combination, equivalent material parameters such as the equivalent Young's modulus E^* and the equivalent shear modulus G^* are precalculated based on the Hertz–Mindlin model (Pasthy et al., 2022). Additionally, parameters such as the equivalent impact coefficient e^*, the equivalent sliding friction coefficient μ^*, the equivalent static friction coefficient μ_0^*, and the equivalent rolling friction coefficient μ_r^* can be specified for each combination.

Particle properties are initially stored in global memory and are then transferred to the much faster register memory at the onset of the GPU program segment. This memory layout is illustrated in the appendix in Fig. A.14. Efficient memory reading operations necessitate data being retrieved from a contiguous memory block. Since all the threads within a warp execute the same operation and a warp handles adjacent thread indices, read operations are performed in an aligned manner from a continuous memory block (see the coloring in Fig. A.14), which makes memory usage more efficient. As the description of each particle requires 7 vectors, and 8 scalars at minimum (see Table 1), storing one million particles requires 110.6 MB of data. This fits in the global memory of the GPU; however, for such large problems there is not enough register space, and register spills into higher-level memory occur, thus diminishing efficiency.

2.4. Particle–particle contact

Particle–particle contacts are relatively straightforward to detect due to their spherical nature. Two particles, indexed with i and j, are considered to be in contact if

$$R_i + R_j > d_{ij}, \tag{3}$$

where $d_{ij} = |\boldsymbol{u}_i - \boldsymbol{u}_j|$ represents the distance between the particles. The most straightforward algorithm for contact detection involves iterating through all the particles, computing the distance between each possible pair of particles, and determining whether they are in contact according to Eq. (3). This approach, known as the brute force method, leads to inefficiencies due to the extensive computation required to calculate the distance d_{ij}. This computational intensity arises from the fact that

$$d = \sqrt{(u_{x,i} - u_{x,j})^2 + (u_{y,i} - u_{y,j})^2 + (u_{z,i} - u_{z,j})^2}, \tag{4}$$

and it involves a square root operation. Furthermore, the number of particle pairs increases quadratically with the number of particles, exacerbating computational demands.

The efficiency of the simulation can be enhanced by partitioning the entire computational domain with a mesh and computing the distance d_{ij} solely for particles residing in the same or adjacent cells. The computation domain is resolved using $N_x \times N_y \times N_z$ cells each with dimensions $\Delta x \times \Delta y \times \Delta z$. An example mesh can be seen in Fig. A.15. The following equation uniquely assigns a particle to a cell in 3D:

$$c_{\mathrm{id}} = \left\lfloor \frac{x - x_{\min}}{\Delta x} \right\rfloor + N_x \left\lfloor \frac{y - y_{\min}}{\Delta y} \right\rfloor + N_x \cdot N_y \left\lfloor \frac{z - z_{\min}}{\Delta z} \right\rfloor, \tag{5}$$

where c_{id} represents the cell index, N_x and N_y denote the number of cells in the x and y directions, and $\lfloor \square \rfloor$ denotes the floor function, which returns the integer part of the number for positive numbers. Subsequently, based on c_{id}, it can be determined whether two particles are in the same or adjacent cells. This condition is met if

$$c_{\mathrm{id},i} = c_{\mathrm{id},j} + \left\{ \begin{matrix} 1 \\ 0 \\ -1 \end{matrix} \right\} \cdot 1 + \left\{ \begin{matrix} 1 \\ 0 \\ -1 \end{matrix} \right\} \cdot N_x + \left\{ \begin{matrix} 1 \\ 0 \\ -1 \end{matrix} \right\} \cdot N_x \cdot N_y, \tag{6}$$

where there are 27 combinations of numbers between brackets.

Once the mesh is established, the Cell-Linked List (CLL) algorithm can be employed, where for each cell, the particles currently occupying it are recorded (Cai et al., 2018). However, when using the GPU, this approach may suffer from thread divergence and consequently experience performance degradation. This occurs because each cell typically contains a different number of particles. Nonetheless, with a large number of particles, the number of potential contacts that need to be computed decreases by orders of magnitudes, thus making the simulation faster.

Implementing the CLL method on parallel architectures poses significant challenges. The primary potential problem is threads executing in parallel, meaning that multiple threads may attempt to access the same memory space simultaneously, leading to a phenomenon known as memory contention. In Listing 1, the loading of the cell-linked list in the global memory is illustrated. In the code, `globalmem.cellCount` is an array of size $N_C = N_x \cdot N_y \cdot N_z$, which records the number of particles currently present in each cell. Additionally, `globalmem.linkedCellList` contains the corresponding list of particle indices for each cell. The size of the array `globalmem.linkedCellList` is $N_C \cdot N_{P,C}$, where $N_{P,C}$ denotes the maximum number of particles in a cell. The thread that reaches row 4 first increments the value of the variable `globalmem.cellCount[cid]` in global memory. However, due to memory latency, for several cycles after the instruction, the old value remains in the global memory. During this time, if another thread reads from the same memory space, it will access the outdated previous value. Consequently, both threads might attempt to write to the same memory address in line 5, leading to previously written data being overwritten.

Listing 1: Race condition between threads during the cell-linked list contact search algorithm

```
1  int tid = ...; //thread index of the
       particle
2  ...
3  int cid = ...; //cell index, in which the
       particle is in
4  int cell_count=globalmem.cellCount[cid]++;//
       num.of particles in cell
5  globalmem.linkedCellList[cid*N + cell_count]
        = tid;
```

CUDA defines atomic expressions, which ensure that no other thread has access to the specified data when writing to global memory until the new data is stored in memory. The use of the `atomicInc()` function defined by CUDA in line 4 guarantees that the aforementioned situation is avoided. Finally, the equivalent radius R^* and equivalent mass m^* for each particle–particle contact need to be calculated, in accordance with the Hertz–Mindlin contact model (Pasthy et al., 2022).

For the particle–particle contact detection to perform optimally the cell size should be as small as possible, but not smaller than the largest particle diameter. The optimal cell size is usually $\Delta x = \Delta y = \Delta z = 2\max_i R_i$. The maximum number of particles within a cell ($N_{P,C}$) must be specified, because the memory space must be pre-allocated. Memory space limitations are not problematic even for small particles, as a $1\,\mathrm{m} \times 1\,\mathrm{m} \times 1\,\mathrm{m}$ domain with $2\max_i R_i = 0.01\,\mathrm{m}$ can contain millions of particles, and requires 1 million cells. If $N_{P,C} = 12$, then a total of 12 million indices must be stored, which only requires 45.8 MB of global memory.

2.5. Geometry-particle contact

The geometry encompasses the bounding walls and the tillage tools, all of which are defined as a geometry bordered by triangles in STL file format. STL files contain the vertices of the triangles and the surface normal vectors. In the GPUDEM framework, any STL file in ASCII format can be read and utilized. Presently, the geometry is stored in the local memory of the GPU during calculations. Employing local memory offers the advantage of faster data access. However, its size is restricted, allowing only geometries bordered by a few hundred triangles to be specified. Its maximum size is dependent on the GPU type and settings.

The advantage of utilizing STL files lies in the simplicity of detecting contacts between a sphere and a triangle. Let $\boldsymbol{p}$ denote the vector pointing to one vertex of the triangle, while $\boldsymbol{s}$ and $\boldsymbol{t}$ point to the other vertices of the triangle from $\boldsymbol{p}$, as depicted in Fig. 2. These vectors define the plane S, which can be expressed as follows:

$$S := \boldsymbol{p} + \boldsymbol{s}\sigma + \boldsymbol{t}\tau, \tag{7}$$

where τ and σ represent coordinates in the coordinate system defined by the vectors $\boldsymbol{s}$ and $\boldsymbol{t}$. For points lying within the triangle, the following inequalities hold:

$$\sigma \geq 0, \quad \tau \geq 0 \text{ and } \sigma + \tau \leq 1. \tag{8}$$

Let $\boldsymbol{n}$ represent the unit vector perpendicular to the plane S. By calculating the coordinates of the vector $\boldsymbol{u} - \boldsymbol{p}$ in the coordinate system $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{n})$, the coordinates σ, τ, and d can be obtained, as illustrated in Fig. 2. The transformation matrix between the coordinate systems ($\hat{\boldsymbol{i}}$, $\hat{\boldsymbol{j}}$, $\hat{\boldsymbol{k}}$) and ($\boldsymbol{s}$, $\boldsymbol{t}$, $\boldsymbol{n}$) can be expressed as follows:

$$\mathbf{T} = \begin{pmatrix} \boldsymbol{s}^t \\ \boldsymbol{t}^t \\ \boldsymbol{n}^t \end{pmatrix}. \tag{9}$$

The coordinates can then be calculated easily by performing the appropriate transformations,

$$\begin{pmatrix} \sigma \\ \tau \\ d \end{pmatrix} = \mathbf{T}^{-t}(\boldsymbol{u} - \boldsymbol{p}), \tag{10}$$

where $\mathbf{T}^{-t} = \left(\mathbf{T}^{-1}\right)^t$ represents the inverse transpose of matrix $\mathbf{T}$, and d denotes the distance of the center of the particle from the plane, given that the basis vector $\boldsymbol{n}$ has unit length. Moreover, the number of calculations and stored data can be further reduced, as

$$\mathbf{T}^{-t} = \begin{pmatrix} \tilde{\boldsymbol{s}}^t \\ \tilde{\boldsymbol{t}}^t \\ \tilde{\boldsymbol{n}}^t \end{pmatrix} \text{ and } \sigma = \tilde{\boldsymbol{s}} \cdot (\boldsymbol{u} - \boldsymbol{p}), \quad \tau = \tilde{\boldsymbol{t}} \cdot (\boldsymbol{u} - \boldsymbol{p}), \quad d = \boldsymbol{n} \cdot (\boldsymbol{u} - \boldsymbol{p}), \tag{11}$$

where the vectors $\tilde{\boldsymbol{s}}^t$, $\tilde{\boldsymbol{t}}^t$, and $\tilde{\boldsymbol{n}}^t$ represent the rows of the matrix $\mathbf{T}^{-t}$. It is important to note that the vector $\tilde{\boldsymbol{n}}$ is not needed in the above equation since distance d is simply the perpendicular projection of $\boldsymbol{u} - \boldsymbol{p}$ onto $\boldsymbol{n}$ and can therefore be calculated by a scalar product. From the computed coordinates, contact can be detected. A triangle and a sphere are in contact if the following two conditions are satisfied:

1. The center of the sphere is closer to the plane of the triangle S than its radius, i.e., $d < R$.
2. The projection of the center of the sphere on the plane S is inside the triangle, i.e., the inequalities in Eq. (8) are satisfied.

The algorithm described is highly efficient, requiring only four vectors to describe the coordinate transformation and the offset ($\tilde{\boldsymbol{s}}$, $\tilde{\boldsymbol{t}}$, $\boldsymbol{n}$, $\boldsymbol{p}$). Moreover, these vectors can be computed for each triangle at the outset of the simulation, provided that the given geometry remains only translated, as only the vector $\boldsymbol{p}$ is modified during translation. In the subsequent step, the particle–wall contact can be computed similarly to the particle–particle contact, with the distinction lying in the definition of the equivalent radius and mass:

$$R^* = R \quad \text{and} \quad m^* = m. \tag{12}$$

2.6. Force calculation

The force calculation follows the principles of the Hertz–Mindlin model (Pasthy et al., 2022). The normal and tangential overlap of particles (δ_n and $\boldsymbol{\delta}_t$) are determined as detailed in Golshan et al. (2023):

$$\delta_n = d - (R_i + R_j) \quad \text{and} \tag{13}$$

$$\boldsymbol{\delta}_t^{\{n+1\}} = \boldsymbol{\delta}_t^{\{n\}} + \Delta t \cdot \tilde{\boldsymbol{v}}_{i,j,\mathrm{t}}, \quad \boldsymbol{\delta}_t^{\{0\}} = \mathbf{0}, \tag{14}$$

where $\square^{\{n\}}$ denotes the state at the nth time step of contact, R_i represents the radius of the ith particle, Δt signifies the time step size, and $\tilde{\boldsymbol{v}}_{i,j,\mathrm{t}}$ denotes the relative tangential velocity between particles i and j.

In the following discussion, the contact between particles indexed i and j is addressed. To reduce operations within the Hertz–Mindlin model, the following quantity is introduced:

$$R_\delta = \sqrt{R^* \cdot \delta_n}. \tag{15}$$

where $R^* = (R_i^{-1} + R_j^{-1})^{-1}$ is the equivalent radius. The equivalent normal and shear stiffness can then be calculated as,

$$S_n = 2E^* \cdot R_\delta \quad \text{and} \tag{16}$$

$$S_t = 8G^* \cdot R_\delta. \tag{17}$$

where E^* is the equivalent Young modulus and G^* is the equivalent shear modulus. From the previously described quantities, the normal elastic force $\boldsymbol{F}_{ne}$, tangential elastic force $\boldsymbol{F}_{te}$, normal damping force $\boldsymbol{F}_{nd}$ and tangential damping force $\boldsymbol{F}_{td}$ can be calculated,

$$\boldsymbol{F}_{ne} = -\frac{4}{3}E^* R_\delta \delta_n \boldsymbol{r}_{i,j}, \tag{18}$$

$$\boldsymbol{F}_{te} = -\boldsymbol{\delta}_t \cdot S_t, \tag{19}$$

$$\boldsymbol{F}_{nd} = -2\sqrt{\frac{5}{6}} \cdot \beta \cdot \sqrt{S_n \cdot m^*} \cdot \tilde{\boldsymbol{v}}_{i,j,n} \quad \text{and} \tag{20}$$

$$\boldsymbol{F}_{td} = -2\sqrt{\frac{5}{6}} \cdot \beta \cdot \sqrt{S_t \cdot m^*} \cdot \tilde{\boldsymbol{v}}_{i,j,t}. \tag{21}$$

where m^* is the equivalent mass, $\boldsymbol{r}_{i,j}$ is the direction unit vector and $\tilde{\boldsymbol{v}}_{i,j,n}$ is the normal component of the relative velocity between particle i and j. Normal and tangential forces result from elasticity and damping, i.e.,

$$\boldsymbol{F}_n = \boldsymbol{F}_{ne} + \boldsymbol{F}_{nd} \quad \text{and} \tag{22}$$

$$\boldsymbol{F}_t = \boldsymbol{F}_{te} + \boldsymbol{F}_{td}. \tag{23}$$

The particles slip on each other if the following condition is not satisfied:

$$|\boldsymbol{F}_t| > \mu_0^* \cdot |\boldsymbol{F}_n|, \tag{24}$$

where μ_0^* is the equivalent coefficient of static friction. Then the tangential force is modified as follows:

$$\boldsymbol{F}_{t,\mathrm{new}} = \mu^* \cdot \boldsymbol{F}_t \cdot \frac{|\boldsymbol{F}_n|}{|\boldsymbol{F}_t|}, \tag{25}$$

where μ^* represents the equivalent sliding friction coefficient. Consequently, $|\boldsymbol{F}_{t,\mathrm{new}}| = \mu^* \cdot |\boldsymbol{F}_n|$, and the direction of $\boldsymbol{F}_{t,\mathrm{new}}$ aligns with the direction of $\boldsymbol{F}_t$. The total force acting on the particle due to a contact can now be calculated as:

$$\boldsymbol{F} = \boldsymbol{F}_n + \begin{cases} \boldsymbol{F}_t, & if \quad |\boldsymbol{F}_t| \leq \mu_0^* \cdot |\boldsymbol{F}_n| \\ \boldsymbol{F}_{t,\mathrm{new}}, & if \quad |\boldsymbol{F}_t| > \mu_0^* \cdot |\boldsymbol{F}_n| \end{cases}. \tag{26}$$

The tangential force induces a torque since its line of action does not intersect the center of the particle. The torque is given by the cross product:

$$\boldsymbol{M}_p = \boldsymbol{p}_{i,j} \times \boldsymbol{F}_t, \tag{27}$$

where $\boldsymbol{p}_{i,j}$ is the vector pointing from particle i to the contact point between particles i and j. Considering rolling friction introduces a torque in the direction of the angular velocity vector (Solutions, 2014),

$$\boldsymbol{M}_\mu = -\mu_r^* \cdot |\boldsymbol{F}_n| \cdot |\boldsymbol{p}_{i,j}| \cdot \frac{\boldsymbol{\omega}_i}{|\boldsymbol{\omega}_i|}. \tag{28}$$

The total torque exerted on a particle due to a contact is:

$$\boldsymbol{M} = \boldsymbol{M}_p + \boldsymbol{M}_\mu. \tag{29}$$

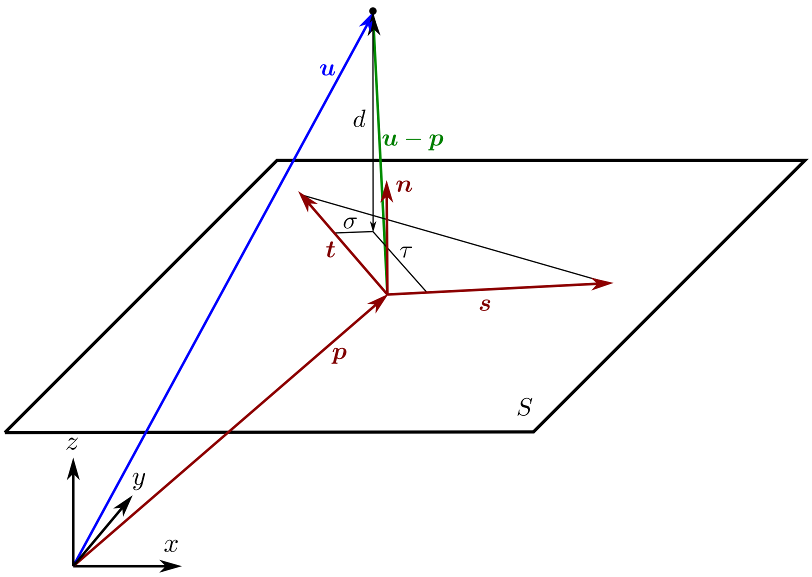


Fig. 2. The intersection of a triangle and a sphere. Plane S is defined by the vectors $\boldsymbol{s}$ and $\boldsymbol{t}$, which also characterize the triangle. Here, $\boldsymbol{p}$ represents the position vector of one vertex of the triangle, while $\boldsymbol{u}$ denotes the position vector of the center of the particle. The coordinates σ, τ, and d correspond to the coordinates in the local coordinate system ($\boldsymbol{s}$, $\boldsymbol{t}$, $\boldsymbol{n}$) of the triangle.

A particle may be in contact with several walls and other particles, hence the force and torque calculations outlined in Eqs. (26) and (29) must be executed for all contacts involving all of the particles. The number of contacts between the ith particle and the walls or other particles is denoted by $N_{C,i}$, where $N_{C,\max}$ represents the maximum number of contacts. The total force and torque acting on the ith particle is the cumulative sum of the forces and torques arising from all contacts:

$$\boldsymbol{F}_{i,\text{total}} = \sum_{j=0}^{N_{C,i}-1} \boldsymbol{F}_{i,j} \text{ and} \tag{30}$$

$$\boldsymbol{M}_{i,\text{total}} = \sum_{j=0}^{N_{C,i}-1} \boldsymbol{M}_{i,j}, \text{ where } N_{C,i} \le N_{C,\max}, \tag{31}$$

where $\boldsymbol{F}_{i,j}$ and $\boldsymbol{M}_{i,j}$ are the force and torque acting on the ith particle from the jth contact.

2.7. Temporal solution

To calculate a temporal solution the accelerations must be determined. Given that the particles are spherical,

$$\boldsymbol{a}_i^{\{n\}} = m_i^{-1} \cdot \boldsymbol{F}_{i,\text{total}}^{\{n\}} + \boldsymbol{g} \text{ and} \tag{32}$$

$$\boldsymbol{\beta}_i^{\{n\}} = \theta_i^{-1} \cdot \boldsymbol{M}_{i,\text{total}}^{\{n\}}, \tag{33}$$

where m_i denotes the mass, $\boldsymbol{g}$ the gravitational acceleration vector, θ_i the moment of inertia, $\boldsymbol{a}_i^{\{n\}}$ the acceleration and $\boldsymbol{\beta}_i^{(n)}$ the angular acceleration of the ith particle at time step n. These accelerations serve as the basis for computing the new velocity, position, and angular velocity at the given step size. Various time-step schemes can be chosen for this purpose.

Euler's method

The simplest choice is Euler's method, wherein the velocity $\boldsymbol{v}_i$, position $\boldsymbol{u}_i$, and angular velocity $\boldsymbol{\omega}_i$ are determined as follows:

$$\boldsymbol{v}_i^{\{n+1\}} = \boldsymbol{v}_i^{\{n\}} + \Delta t \cdot \boldsymbol{a}_i^{\{n\}}, \tag{34}$$

$$\boldsymbol{u}_i^{\{n+1\}} = \boldsymbol{u}_i^{\{n\}} + \Delta t \cdot \boldsymbol{v}_i^{\{n+1\}} \text{ and} \tag{35}$$

$$\boldsymbol{\omega}_i^{\{n+1\}} = \boldsymbol{\omega}_i^{\{n\}} + \Delta t \cdot \boldsymbol{\beta}_i^{\{n\}}, \tag{36}$$

where Δt is the step size. Higher order terms can be taken into account in the position calculation (Kruggel-Emden et al., 2008) and a more accurate estimate of the position can be obtained if

$$\boldsymbol{u}_i^{\{n+1\}} = \boldsymbol{u}_i^{\{n\}} + \Delta t \cdot \boldsymbol{v}_i^{\{n\}} + \frac{1}{2}\Delta t^2 \boldsymbol{a}_i^{\{n\}}. \tag{37}$$

Assuming that the acceleration is constant throughout a time step, Eq. (37) gives the exact position. For this reason in GPUDEM this time step scheme is called *Exact*. Of course, the acceleration is not constant during the time step, in reality the slightest displacement of a particle will change the forces and hence the acceleration.

Adams-method

The solver also incorporates the second-order Adams–Bashforth–Adams–Moulton time step method (Hairer et al., 1993; Butcher, 2016). This method offers the advantage of second-order accuracy but necessitates storing one additional time step. The essence of the method is the explicit computation of the velocity using the Adams–Bashforth method:

$$\boldsymbol{v}_i^{\{n+1\}} = \boldsymbol{v}_i^{\{n\}} + \Delta t \cdot \left(\frac{3}{2}\boldsymbol{a}_i^{\{n\}} - \frac{1}{2}\boldsymbol{a}_i^{\{n-1\}}\right) \text{ and} \tag{38}$$

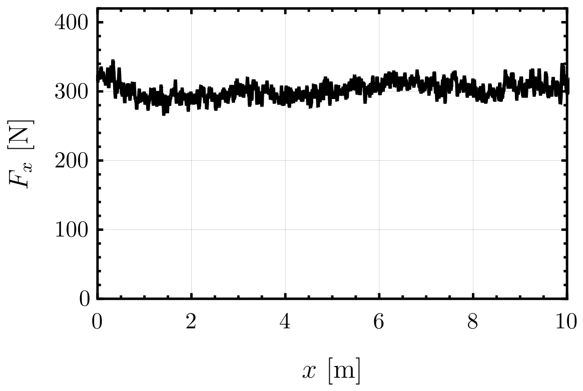
$$\boldsymbol{\omega}_i^{\{n+1\}} = \boldsymbol{\omega}_i^{\{n\}} + \Delta t \cdot \left(\frac{3}{2}\boldsymbol{\beta}_i^{\{n\}} - \frac{1}{2}\boldsymbol{\beta}_i^{\{n-1\}}\right). \tag{39}$$

Then, the implicit Adams–Moulton method is used to determine the position,

$$\boldsymbol{u}_i^{\{n+1\}} = \boldsymbol{u}_i^{\{n\}} + \frac{1}{2} \cdot \Delta t \cdot \left(\boldsymbol{v}_i^{\{n\}} + \boldsymbol{v}_i^{\{n+1\}}\right), \tag{40}$$

where $\boldsymbol{v}_i^{\{n+1\}}$ is already known from Eq. (38). The method requires more memory, as the acceleration and velocity values from the previous step are necessary. In the initial step ($n = 0$), in Eqs. (38) and (39), no information is available regarding the values of $\boldsymbol{a}_i^{\{-1\}}$ and $\boldsymbol{\beta}_i^{\{-1\}}$. Therefore, an Euler step is always utilized in the initial step, as described in Eqs. (34)–(36). Although higher-order Adams methods that offer the advantage of providing sufficiently accurate solutions with larger time steps exist, the nature of the problem restricts the potential increase in time step significantly. Contacts are continuously created and terminated during the motion of the particles, and a high temporal resolution of the collisions is essential. Hence, the implementation of higher-order numerical methods is not justified.

(a) The measurement setup and the sweep tool. The measurement data from the red tool on the left is used in this study.



(b) The draught force in the first 10 meters of the measurement in steady-state. The average draught force is $\overline{F}_x = 309.5$ N.

Fig. 3. The measurement setup and the recorded force. The sweep tool moved with a constant velocity $v = 0.706$ m/s, at a tillage depth of 150 mm.

2.8. Measurement details

Subsequent DEM simulations of draught force were compared to measurements conducted in the soil-bin facility at the Institute of Technology of the Hungarian University of Agriculture and Life Sciences in Gödöllő, Hungary. The details of the soil-bin measurements at the same location were discussed by Tamás (2018). This facility spans 50 meters in length and 1.95 meters in width (Tamás et al., 2013) and was filled with sandy soil having a moisture content of 3.96% on a dry mass basis. During the experiment, a sweep tool with the dimensions 313 mm × 289 mm (see Fig. A.17) moved at a velocity of $v = 0.706$ m/s with a standard deviation of $\sigma_v = 0.091$ m/s. The tillage depth was 150 mm. Strain gauges were placed on the tine of the sweep tool and calibrated to measure the draught force. The measurement setup is illustrated in Fig. 3a. The draught force of two sweep tools was measured independently at the same time, although, only the data for the red tool on the left in the picture were utilized in this research. A draught force of $F_x = 309.5$ N was recorded in the sandy soil, with a standard deviation of $\sigma_{F_x} = 14.5$ N. The results for the first 10 meters of displacement, in steady state, are illustrated in Fig. 3b.

3. Results

3.1. Validation based on comparison with commercial software

The aim of the validation process is to ensure that all of the methods in the program are implemented correctly, without any typos or logical errors. One validation approach involves comparing the program with other software packages that utilize similar methods. Altair EDEM® is one of the most widely used DEM software packages, implementing various contact models such as the Hertz–Mindlin model (Solutions, 2014). In a validation comparison, two gravitational deposition cases were considered: one using 509 particles and the other using 8519 particles. The objective of the deposition was to establish a suitable initial particle distribution for subsequent soil simulations. During gravitational deposition, particles initially placed in random locations within a given domain fall and stack on top of each other under gravity. Throughout deposition, the velocity of each particle is expected to converge to zero. In both EDEM and GPUDEM, the same initial particle distribution was employed.

The material parameters and time step settings are listed in column 1 of Table B.5. In both software packages, the velocity of all particles is expected to approach zero after 1 s. The success of deposition can be assessed by monitoring the total kinetic and potential energy of the particles. The total kinetic energy of the particle assembly is given by:

$$K = \frac{1}{2}\sum_{i=0}^{N_P-1}\left(m_i \cdot \boldsymbol{v}_i \cdot \boldsymbol{v}_i + \theta_i \cdot \boldsymbol{\omega}_i \cdot \boldsymbol{\omega}_i\right), \tag{41}$$

where $\boldsymbol{v}_i$ represents the velocity of the ith particle and $\boldsymbol{\omega}_i$ denotes its angular velocity. The potential energy of the particle assembly is calculated as:

$$P = \sum_{i=0}^{N_P-1} m_i \cdot g \cdot u_{z,i}, \tag{42}$$

given that the vector of gravitational acceleration is $\boldsymbol{g} = [0, 0, -g]^T$, and $u_{z,i}$ represents the z-coordinate of the ith particle. The potential and kinetic energy are illustrated in Fig. 4. It is evident that the potential and kinetic energy of the particle assembly are similar in both cases. The kinetic energy (continuous line) gradually converges to zero in all instances, although it never precisely reaches zero due to the finite time steps and rounding errors. The potential energy (dashed line) exhibits a slight disparity after a certain duration, probably due to differences in the implementation between GPUDEM and EDEM. Since the source code of EDEM is closed, the exact details of the implementation of the contact search and Hertz–Mindlin model remain unknown. Overall, Fig. 4 demonstrates that the deposition is successful, as the kinetic energy approaches zero and the potential energy converges to a constant value. A direct comparison of the runtimes is not possible as the EDEM simulations use CPU cores, but experience shows an order of magnitude increase when using GPUDEM.

3.2. Scaling for larger problems

Simulations were conducted with the same parameters but varying particle numbers N_P, employing both brute force and CLL contact search methods. In each case 2500 time steps were performed with $\Delta t = 1 \times 10^{-4}$ s. With the given time step settings, the simulations remained stable, that is, the kinetic energy converged to zero during deposition. The parameters are listed in Table B.5 in column 2. The runtimes (T_{run}) as a function of particle number N_P are depicted in Fig. 5 on a logarithmic plot. For a small particle number ($N_P < 250$) the Brute-Force approach proved to be efficient. However, for a larger particle size ($N_P > 10^4$) the runtime increases by almost threefold when doubling the particle number:

$$\frac{T_{\text{run}}(N_P = 32768)}{T_{\text{run}}(N_P = 16384)} = 2.84 \text{ and } \frac{T_{\text{run}}(N_P = 65536)}{T_{\text{run}}(N_P = 32768)} = 2.91, \tag{43}$$

thus, the scaling is not linear. This accelerated growth is attributed to the fact that the number of operations is proportional to N_P^2 during the contact search. However, the runtime growth does not quadruple with the doubling of the number of particles, since the GPU resources are not fully utilized even when employing 10^5 particles.

For the CLL contact search, the runtime initially decreases as the number of particles increases. The mesh in this case contains $N_C =$

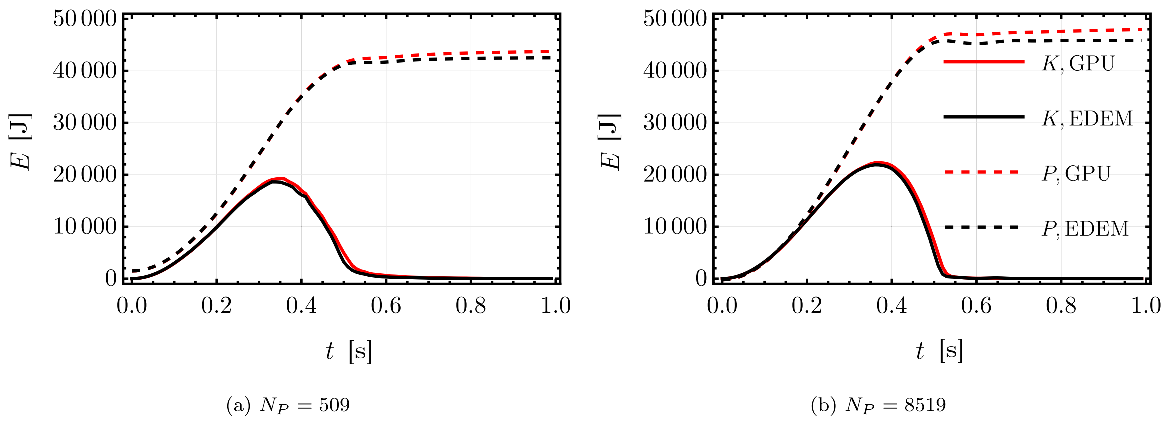


Fig. 4. The kinetic energy K and -1 times the potential energy $-P$ during deposition in GPUDEM and EDEM using a different number of particles (N_P).

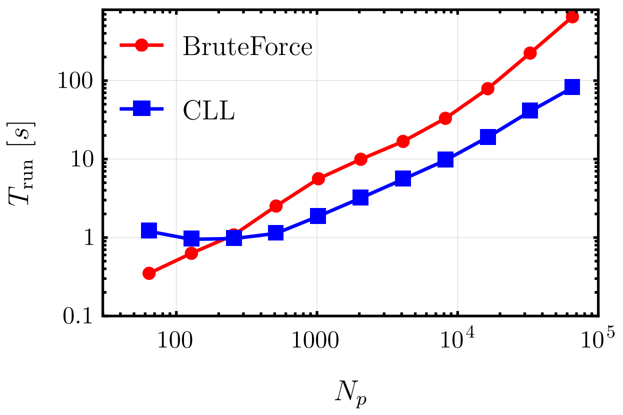


Fig. 5. Runtime of the deposition with different number of particles (N_P) and using different contact search algorithms (BruteForce and CLL).

$28^3 = 21952$ cells, and each cell can accommodate up to 8 particles. Since the cells need to be reinitialized before each time step, and the number of threads is low, the reinitialization is a slow process. This trend holds true until $N_P \approx 250$, as more threads are applied, leading to faster cell updates. As the number of particles increases further, the runtime also increases. It takes roughly twice as long to compute with twice as many particles. The advantage of the CLL method lies in its linear scaling, where the number of operations is proportional to N_P. In summary, the CLL contact search method is notably faster for calculations involving larger numbers of particles. The difference in runtime between the two methods becomes more pronounced as the number of particles increases, with CLL proving to be eight times faster for $N_P = 65536$ particles.

3.3. Time step size

Increasing the time step can reduce the runtime, but excessively large time steps may lead to instability. In DEM particle collisions have to occur gradually, which necessitates the use of sufficiently small time steps to maintain stability. To test the effect of step size, the length of a deposition simulation was set to $T = 2.5$ s, and the time step size was chosen within the range $\Delta t = 1 \times 10^{-5}$ s to 1×10^{-2} s. The number of steps (denoted as N_{step}) determines the total number of time steps to be executed in the simulation, and it is calculated as:

$$N_{\text{step}} = \frac{T}{\Delta t}. \tag{44}$$

It was observed that if the number of steps is sufficiently large, i.e., $N_{\text{step}} > 10000$, then doubling the number of time steps results in a doubling of the runtime. For a lower number of steps, this proportionality is not true. Detailed results can be found in Fig. 6(a). This discrepancy arises because the DEM calculation constitutes a decreasing portion of the total running time, while other operations such as writing outputs, initialization, and memory management become more significant. For instance, the particle positions and velocities are saved 50 times throughout the entire simulation, independent of N_{step}.

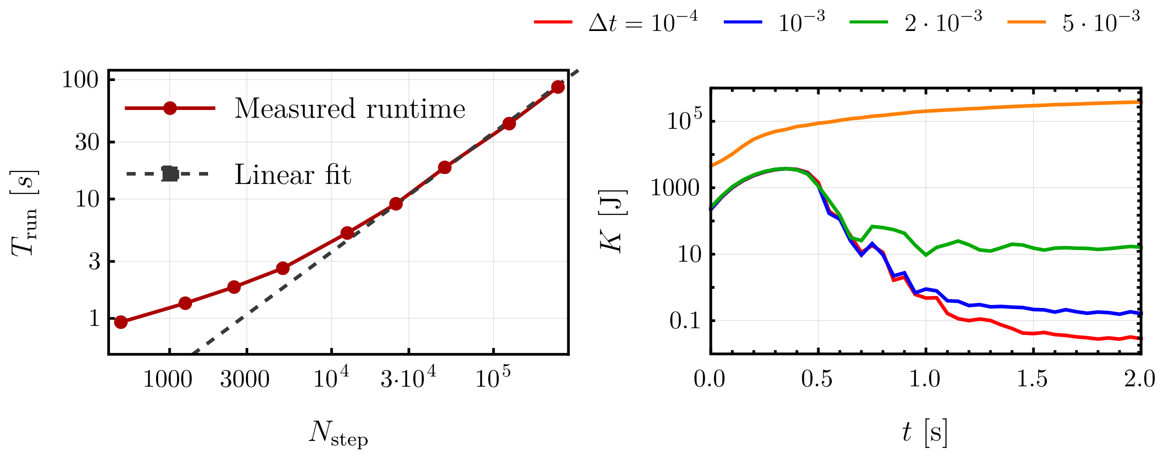
A time step that is too large ($\Delta t > 10^{-3}$ s) results in non-physical outcomes during settling, as illustrated in Fig. 6(b). It is evident from the figure that when a large time step is applied, the kinetic energy (K) of the particle assembly does not converge to zero. This phenomenon occurs because the large time step leads to the rapid development of significant overlaps during collisions. Determining the optimal time step size depends on the specific task, as it depends on factors such as the size, material parameters, and geometry of the particles involved.

All the implemented time step schemes were used with a time step of $\Delta t = 1 \times 10^{-3}$ s on the same case. There was no discernible difference in the runtime, as the computational requirement for time step calculation is negligible compared to the contact search and force calculation. Fig. A.16 illustrates the kinetic energy K during the simulations. When the first-order Euler method was applied, the kinetic energy of the particle assembly decreased to nearly 0.1 J. The kinetic energy decreased almost similarly when using the Exact method. Using the second-order Adams predictor–corrector method, an order of magnitude more kinetic energy remains. Although the order of the method is higher, this does not necessarily imply greater accuracy.

It is a well-established principle in numerical methods for differential equations that the order of convergence is guaranteed only in cases where the functions involved are suitably continuous (Bellen and Zennaro, 2013). However, during the onset of collisions between particles, the change in force is discontinuous. The normal elastic force is proportional to the normal overlap:

$$\boldsymbol{F}_{ne} \propto \delta_n^{\frac{3}{2}}, \tag{45}$$

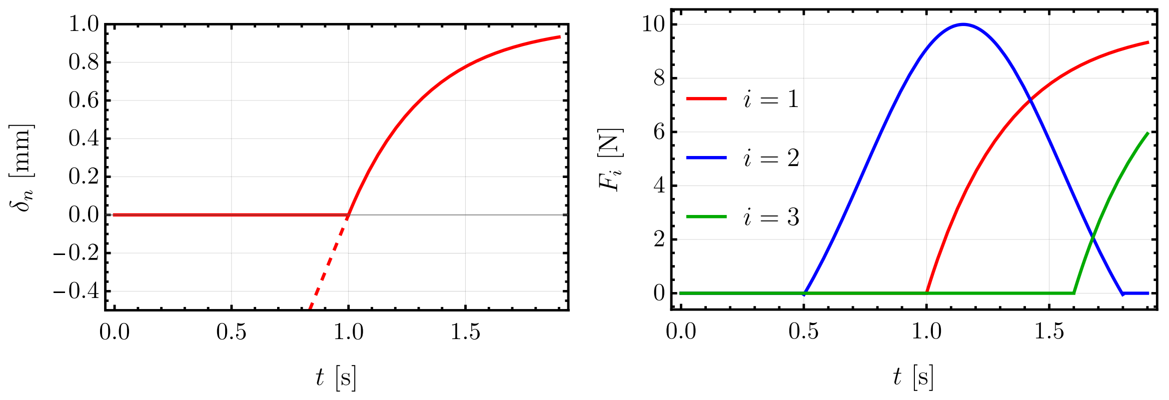
where $\delta_n = d - (R_s + R_i)$ is the normal overlap, and although it is seemingly continuous, it is only meaningful when $\delta_n > 0$. Fig. 7(a) illustrates the tangential overlap in a scenario in which two particles have been in contact since $t = 1$ s. It is evident that when $\delta_n < 0$, the function value is instead set to zero (indicating no collision), resulting in a discontinuity in the derivative at $t = 1$ s. If the numerical solution takes a time step from $t = 0.95$ s with a step size of $\Delta t = 0.1$ s, a discontinuity in the derivative of the function occurs during the time step. In such scenarios, convergence becomes first-order, meaning the error is proportional to Δt, as the continuity condition is not met (Wanner and Hairer, 1996). The ideal solution would involve the precise detection of all contact times and the numerical solution of the equation of motion on continuous sections only, with appropriate adjustments to the time step. As depicted in Fig. 7(b), connections



(a) The runtime as a function of the number of time steps N_{step}.

(b) The kinetic energy on a logarithmic diagram during the simulation using different time steps.

Fig. 6. The results of settling using different time step sizes.



(a) The change in δ_n during a collision. The dashed line depicts δ_n before collision, however only $\delta_n \geq$ is physically meaningful.

(b) Forces acting on a particle from multiple collisions. Discontinuities arise at the onset of collisions.

Fig. 7. Explanation of the discontinuity during particle contact.

between particles are constantly formed and broken, necessitating the identification of numerous discontinuous points to compute just one particle. Considering the large number of particles involved in DEM, accurate collision detection becomes impractical. Consequently, higher-order numerical methods should be avoided, as maintaining the order of convergence relies on precise collision detection, which is not possible when modeling a large number of particles. In the literature, the Euler method is commonly employed (Di Renzo and Di Maio, 2004; Melheim, 2005), often without explicit justification. Throughout the remainder of the paper, the first-order Euler method is utilized.

3.4. Profiling

Profiling aims to identify the bottleneck (the most resource-intensive part of the program) using data from low-level hardware counters that monitor memory and computational operations. The NVIDIA NSight Compute application was utilized to profile the software in the present research. Table 2 presents the primary profiling results at three different particle counts, revealing an almost linear increase in runtime. The CLL contact search procedure was employed, and only 100 time steps were executed. Analysis of Table 2 indicates that GPUDEM predominantly utilizes GPU memory over computational resources, a phenomenon often termed a *memory-limited* problem (Cheng et al., 2014). According to the profiling data, the L1 cache contains the requested data 12.5% of the time, while the L2 cache contains it 70.4% of the time.

Table 2
Results of profiling using the NVIDIA NSight Compute.

Number of particles	$2^{13} = 8192$	$2^{14} = 16384$	$2^{15} = 32768$
Runtime	490 ms	869 ms	1846 ms
Compute utilization	13.30%	22.03%	27.03%
Memory Bandwidth utilization	27.65%	40.92%	49.54%
Floating point operations	$241.0 \cdot 10^6$	$741.6 \cdot 10^6$	$2511.1 \cdot 10^6$
Instruction/cycle	0.22	0.32	0.36
L1 Cache hit rate	20.68%	15.90%	12.51%
L2 Cache hit rate	76.06%	77.28%	70.38%
No eligible warp	94.09%	91.46% %	90.76%
Warp stall: long scoreboard	15.79	25.88	57.21
Warp stall: barrier	6.55	7.23	8.95
Warp stall: wait	2.48	2.60	2.67
Warp stall: branch resolving	1.88	1.78	1.13

As Table 2 indicates, it is evident that both memory and computational utilization increase as the number of particles rises. This observation suggests that the runtime does not simply double when the particle count doubles. Further scrutiny of the data reveals that the number of floating-point operations nearly triples when the particle number doubles. This phenomenon is presumably due to the non-linear increase in connections between particles at the onset of the deposition simulation as the particle count increases.

The GPU's relatively low resource utilization stems from its low instruction/cycle count. Ideally, each warp should execute one operation per cycle throughout the program, resulting in an instruction/cycle ratio of 1. However, in the case of GPUDEM, the instruction/cycle count is 0.22 for all warps with a low particle count, increasing slightly to 0.36 as the particle count rises. The primary reason for this low count is the absence of eligible warps for execution. Even though computational capacity remains available, the initiation of new warps is hindered when no warp is ready for execution, accounting for over 90% of cycles in the code. The root cause of this issue can be found in warp stalls, particularly in connection with data retrieval from global memory (see row Warp stall: long scoreboard in Table 2). On average, a warp waits 15.79 cycles for data with a small particle count and 57.21 cycles in the case of the largest particle count. This delay arises because not all of the data can fit into the L1 and L2 caches, which leads to slower retrieval from global memory. The wait time for instruction results with fixed cycle numbers (e.g., floating-point operations) is minimal compared to waits induced by memory dependencies, as indicated in the "wait" row of Table 2.

Another aspect of profiling involves estimating the percentage of warp stalls and the contribution of various program segments. Over half of the warp stalls arise during the contact search phase, primarily due to data retrieval delays from global memory (see Table B.6 for more details). This finding underscores the significance of optimizing the contact search algorithm. Further enhancements to this algorithm could substantially reduce runtime and should be an important aim of further research.

4. Applications for soil-sweep tool interactions

4.1. Large scale simulations

Soil sweep tool tillage aims to loosen soil, enhancing its water absorption capacity and facilitating root penetration. The following examples showcase the utility of GPUDEM in addressing practical problems connected with tillage, notably in determining the forces exerted on the soil sweep tool during tillage. The DEM parameters of the simulated sandy soil and other simulation settings are summarized in Table B.5, column 3 (Tamás et al., 2013). The length and width of the domain was $l = 1\,\mathrm{m}$ and $w = 0.7\,\mathrm{m}$, respectively. The soil particles have an average size of $R = 4.8\,\mathrm{mm}$, with a maximal deviation of $0.8\,\mathrm{mm}$ and the particle sizes follow a uniform distribution. In this simulation, a layer of soil particles $140\,\mathrm{mm}$ deep is formed, and the sweep tool is positioned $100\,\mathrm{mm}$ deep within the soil.

The first step in the simulation involves preparing the soil through deposition. The deposition process used $N_P = 147\,456$ particles. During the simulation, it was observed that the particles accumulated more thickly at the corners of the domain, a phenomenon known as the wall effect. To eliminate this effect, all particles with a center above $140\,\mathrm{mm}$ were removed, which resulted in a completely smooth surface. After removing the top particles, $N_P = 146\,180$ particles remained. The second step involves adding the sweep tool geometry. The sweep tool model has been previously used in laboratory measurements, and a 3D scanned model was created for those studies. The model from the 3D scan contains $14\,304$ triangles, which is not applicable in the current implementation of GPUDEM, due to the limited size of GPU constant memory. Consequently, the size of the original model was reduced using Blender software. The new reduced model contains only 500 triangles. A comparison of the original and reduced models is shown in Fig. A.17.

Three different simulations were conducted at different tillage velocities. In the first simulation, the sweep tool moved at $v = 0.7\,\mathrm{m/s}$. The sweep tool leaves a V-shaped furrow in the soil, as shown in Fig. 8, with the profile of the furrow depicted in blue. Furthermore, the figure clearly shows that the upper red layer remained intact after the sweep tool had passed through the soil particles, with only the lower layers being mixed. Additional snapshots of the simulation are depicted in Fig. A.18. It was observed that the particles rise in front of the sweep tool and fall back behind it, rearranging the lower layers in the process, as shown in Fig. 8. The force acting on the sweep tool was also calculated by summing the force from each interaction between the particles and the sweep tool, a feature that was implemented in the program. Since the forces acting on the particles have already been calculated, creating a new global variable to summarize these forces allows the force acting on the tool to be determined.

The forces acting on the sweep tool in the x-, y-, and z-directions are depicted in Fig. 9(a). As the sweep tool moves in the x-direction the draught force is nearly $F_x \approx 20\,\mathrm{N}$. In the y-direction, despite the symmetry of the sweep tool, a small force oscillating around zero arises due to inhomogeneity. In the z-direction, a larger force is noticeable as the sweep tool is pushed downwards by the particles. Three intervals can be distinguished in the figure:

1. Initially ($x < 0.28\,\mathrm{m}$), the draught force increases as the sweep tool penetrates the soil.
2. In the stationary section ($0.28\,\mathrm{m} < x < 0.84\,\mathrm{m}$), the draught force is nearly constant, but fluctuations due to soil inhomogeneity are observed around a constant value.
3. Towards the end of the simulation ($x > 0.84\,\mathrm{m}$), as the sweep tool leaves the domain, the particles are compressed between the edge of the domain and the sweep tool, resulting in an unreal increase in the draught force.

In the second interval, the draught force fluctuates around a constant value, which is consistent with findings in the literature (Tamás and Bernon, 2021). However, the phenomena observed in the first and third intervals are not realistic and are attributed to the boundaries of the simulation domain. In the following simulations, the sweep tool velocity was increased. Fig. 9(b) illustrates the draught force as a function of the displacement of the sweep tool along the x-axis. The mean draught force as a function of tool velocity increases linearly, based on model fitting:

$$F_x = a + b \cdot v, \quad \text{where } a = 8.96\,\mathrm{N} \text{ and } b = 12.03\,\frac{\mathrm{N}}{\mathrm{m/s}} \quad (R^2 = 0.969) \tag{46}$$

where a and b are calibrated parameters and v is the tool velocity substituted with units m/s. The linear correlation between draught force and tool velocity is consistent with the literature (Tamás and Bernon, 2021; Tekeste et al., 2019; Chen et al., 2013; Zhang et al., 2023c; Pásthy and Tamás, 2023).

4.2. Parameter sensitivity studies

DEM simulations usually do not perfectly replicate real soil, as the soil particles used in DEM are much larger than the actual particles in empirical studies. However, by adjusting the soil parameters, it is feasible to find a configuration that reproduces the force measured in the experiments. In this section, the effects of soil particle density (ρ), micromechanical Young modulus (E), coefficient of restitution at impact (e), and sliding friction coefficient (μ) will be analyzed, aiming to determine the parameters that reproduce the measured force $F_x \approx 310\,\mathrm{N}$. Aforementioned parameters effect the apparent friction and cohesion of the soil. In the following simulations the number of particles and the domain size were reduced. Despite this simplification, the aim is to isolate a near steady-state phase. Deposition must be partially redone for each parameter combination. For instance, reducing E enables better compression of the particles while also altering the soil thickness. After deposition, the sweep tool can be moved through the domain. To ensure the same initial soil thickness for any parameter combination, the following steps are undertaken:

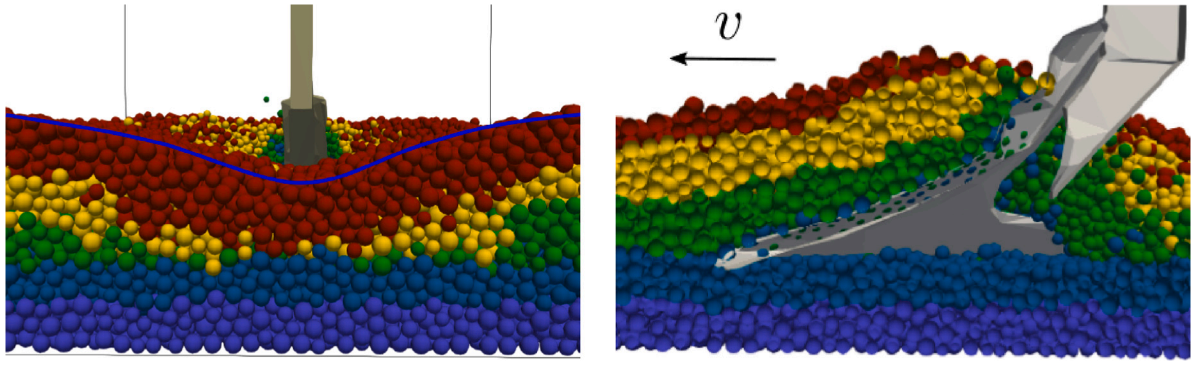
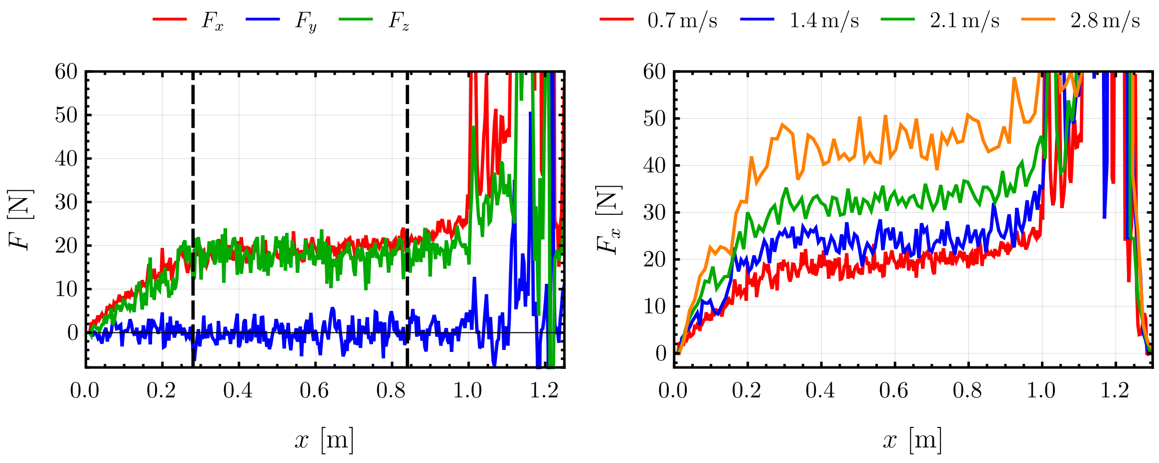


Fig. 8. The left figure shows the surface profile after tillage (surface depicted in blue). The right figure depicts the particles around the sweep tool during tillage, with v denoting the velocity in the given direction. The coloring of the particles are based on their initial heights.



(a) Forces acting on the sweep tool during soil loosening in the x-, y- and z-directions. The vertical dashed lines represent the bounds of stationary draught force section.

(b) Draught force as a function of sweep tool position along the x-axis for different tool velocities according to the legend above.

Fig. 9. Forces in the soil-sweep tool simulation involving $N_P = 146\,180$ spherical particles and domain length 1 m.

1. Initially, 38 912 particles of radius $R = 8.5$ mm are deposited in a domain of 0.7×0.6 m, resulting in a layer roughly 30–40 cm thick, depending on the parameters.
2. Particles with a center above 30 cm are removed to ensure a consistently 30 cm thick soil layer across all cases.
3. The final step involves moving the sweep tool through the domain at a velocity of $v = 0.7$ m/s and calculating the average draught force. The force is saveraged over the displacement range $0.28\,\mathrm{m} < x < 0.56\,\mathrm{m}$ to mitigate the influence of domain boundaries on the simulation results.

The steps outlined above were automated, resulting in a total runtime of approximately 20 s on an NVIDIA GeForce RTX 3060 Ti Graphics Card. The additional parameters can be found in column 4 of Table B.5.

4.2.1. *Effect of particle density*

First, the effect of particle density was investigated by varying it in the range $\rho = 500\,\mathrm{kg/m^3} \ldots 3000\,\mathrm{kg/m^3}$, with a total of 100 simulations made with even divisions. In the simulations, a linear increase in the draught force with increasing density was observed, as shown in Fig. 10(a). A straight line can be fitted to the simulation results in the following form:

$$F_x = b \cdot \rho, \quad \text{where} \quad b = 0.072\,\mathrm{N/(kg/m^3)} \quad (R^2 = 0.99969). \tag{47}$$

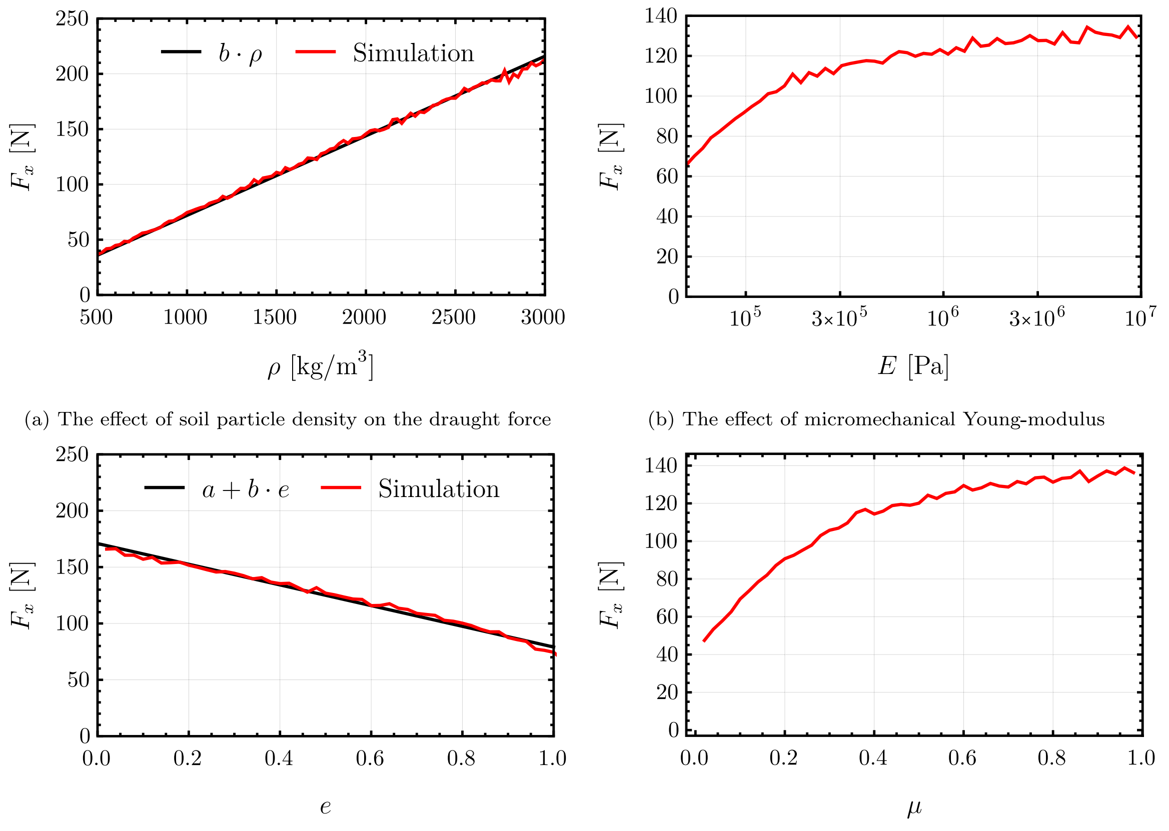
According to the literature (Ucgul et al., 2015), the density of the soil is linearly correlated with the required draught force, which was also confirmed by the GPUDEM simulation. The increase in density is expected to cause an increase in the draught force since the soil particles are heavier; therefore, a greater force must be exerted to move them.

4.2.2. *Effect of the micromechanical Young's modulus*

In the second analysis, the micromechanical Young's modulus was varied within the range of $E = 5 \times 10^4\,\mathrm{Pa} \ldots 1 \times 10^7\,\mathrm{Pa}$. A total of 100 simulations were conducted with a logarithmic scale. The results depicted in Fig. 10(b) show an increase in the draught force as the micromechanical Young modulus of particles increases. Interestingly, it can be observed that once the Young's modulus reaches a certain threshold ($E \approx 3 \times 10^6$ Pa), further increases in its value do not significantly impact the draught force. In a similar soil-tool study, Chen et al. (2013) found an increase in the draught force as a function of particle stiffness using the parallel bond contact model. The effect of the micromechanical Young modulus on the draught force was not investigated previously using the Hertz–Mindlin contact model.

4.2.3. *Effect of the coefficient of restitution between particles*

In the investigation of the effect of the coefficient of restitution, 100 simulations were conducted using the same procedure as before. The effect of this coefficient is illustrated in Fig. 10(c), revealing a linear decrease in the draught force along with an increase in the coefficient



(a) The effect of soil particle density on the draught force

(b) The effect of micromechanical Young-modulus

(c) The effect of the coefficient of restitution on the draught force

(d) The effect of the coefficient of sliding friction between particles on the draught force

Fig. 10. The draught force as a function of different micromechanical DEM parameters.

of restitution. This trend can be fitted to the simulation results with a straight line,

$$F_x = a + b \cdot e, \quad \text{where} \quad a = 170.9\,\text{N} \quad \text{and} \quad b = -91.8\,\text{N} \quad (R^2 = 0.99927). \tag{48}$$

If the coefficient of restitution is small, particles tend to stick together more, resulting in increased friction between the adhered particles. Consequently, more effort is required to move the sweep tool. As depicted in Fig. 11, a lower collision factor (e.g., $e = 0$) leads to the creation of a significantly more homogeneous compaction zone. This zone is thicker and contains more particles. Conversely, with a higher coefficient of restitution, the thickness of the soil layer above the sweep tool decreases, resulting in a smaller compaction zone. The soil in this reduced zone requires less force to move, as it contains fewer particles and has less mass. The effect of the coefficient of restitution on draught force has not been investigated in the literature previously.

4.2.4. Effect of the friction coefficient between particles

The pair of sliding (μ) and static (μ_0) friction coefficients were simultaneously adjusted, with the static friction coefficient set 10% higher than the sliding friction coefficient, i.e., $\mu_0 = 1.1\mu$. The results of the simulation are illustrated in Fig. 10(d), where an increase in draught force is observed with the rising friction factor. This increase can be explained without much difficulty: as friction increases, particles experience greater force while sliding against each other, necessitating a greater force to move them. Tekeste et al. (2019) established a similar relationship between friction and draught force, although, their results were only based on 4 data points.

4.3. Parameter calibration with an evolutionary algorithm

4.3.1. Differential evolution

Calibration involves adjusting the parameters to match the measurement results. Typically in DEM, this is carried out manually by running simulations iteratively and adjusting parameters based on the outcomes. However, this process is time-consuming and requires significant human effort. To streamline the process, an evolutionary algorithm was employed to facilitate automated parameter calibration. Differential evolution, known for its effectiveness in continuous optimization problems under specific constraints, was utilized (Feoktistov, 2006). In the differential evolution algorithm, the population size (NP) denotes the number of parameter combinations (agents). The initial population is randomly generated within the limits outlined in Table 3. Each $\mathbf{x}$ agent contains the following control variables (elements):

$$\mathbf{x} = [\rho \quad E \quad \nu \quad e \quad \mu \quad \mu_r]^t. \tag{49}$$

The goal is to calibrate the parameters to reproduce the measured draught force $F_{\text{meas.}} = 309.5\,\text{N}$ (see Section 2.8). The fitness function is accordingly

$$f(\mathbf{x}) = \left| F_x(\mathbf{x}) - F_{\text{meas.}} \right|, \tag{50}$$

where $F_x(\mathbf{x})$ is the draught force in the simulation as a function of the control variables. After generating the initial population and defining the fitness function, the evolution itself takes place:

1. Three unique agents, $\mathbf{a}, \mathbf{b}$ and $\mathbf{c}$, different from $\mathbf{x}$, are selected. Agent $\mathbf{y}$ is created from these,

$$\mathbf{y} = \mathbf{a} + F(\mathbf{b} - \mathbf{c}), \tag{51}$$

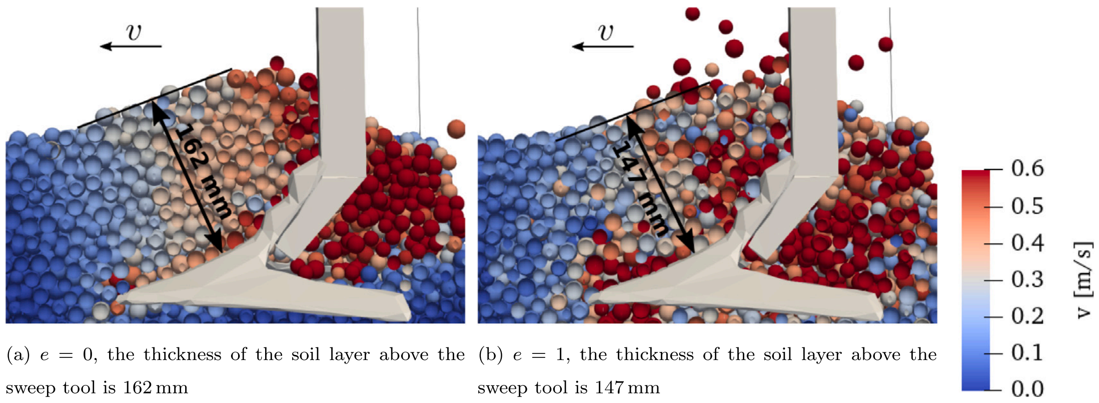

v
162 mm
v
147 mm
0.6
0.5
0.4
0.3
0.2
0.1
0.0
v [m/s]
(a) $e = 0$, the thickness of the soil layer above the sweep tool is 162 mm
(b) $e = 1$, the thickness of the soil layer above the sweep tool is 147 mm

Fig. 11. Velocity of particles around the sweep tool for coefficient of restitution $e = 0$ and $e = 1$. The coloring of the particles is proportional to the magnitude of the velocity vector according to the color scale. The thickness of the soil layer above the sweep tool is marked on the figures.

Table 3
Restrictions on the micromechanical DEM parameters (control variables) in the evolutionary algorithm.

Control Variable		Min	Max	Init. distribution
ρ	particle density	500 kg/m^3	2500 kg/m^3	linear
E	Young-modulus	5×10^4 Pa	1×10^7 Pa	logarithmic
ν	Poisson-number	0	0.5	linear
e	coeff. of restitution	0.05	0.95	linear
μ	sliding friction coeff.	0.05	0.70	linear
μ_r	rolling friction coeff.	0.01	0.30	linear

where F is the differential weighting.

2. The **y** and the **x** agents are crossed. This means that each element is replaced with probability CR in **x**, and one element is always randomly replaced. Thus, a new agent $\tilde{\mathbf{x}}$ is created.
3. The new agent ($\tilde{\mathbf{x}}$) is better than the old one (**x**) if the fitness value is smaller, i.e.

$$f(\tilde{\mathbf{x}}) \leq f(\mathbf{x}), \tag{52}$$

where f is the fitness. If Eq. (52) is satisfied, then the agent **x** is replaced by $\tilde{\mathbf{x}}$.
4. Points 1–3 are performed on each individual, then the next generation follows.

For parameter calibration, a population size of $NP = 60$, a differential weight of $F = 0.8$, and a crossover ratio of $CR = 0.5$ were chosen. A C++ implementation was developed for the differential evolution algorithm, which interfaces with the GPUDEM library for all of the function evaluations (draught force calculation). In each generation 60 function evaluations are performed, with the runtime for one generation ranging between 20 and 25 min on an NVIDIA GeForce RTX 3060 Ti GPU.

4.3.2. Results of the calibration

The optimization process ran for 45 generations over a period of 16 h, involving a total of 2 700 function evaluations. During this time, the settling and the sweep tool traversing the domain were performed 2 700 times, with more than 30 000 particles in each function evaluation. As the generations progressed, both the best individual and the average of the individuals approached the desired goal. Differential evolution demonstrated the ability to find optimal solutions after approximately 20 generations in this case. However, it is important to note that each individual represents a unique set of parameters, and the individuals do not converge to a single global optimum. Instead, they identify several local optima. These local optima were determined through clustering, using the built-in `FindClusters` function of Wolfram Mathematica using the K-Means algorithm.

After clustering, four larger groups were identified, as shown in Table 4, along with the number of agents in each group. The optimization process revealed several solutions, each characterized by a specific combination of parameters . For each parameter combination, the table includes the simulated draught force along with its standard deviation. Fig. 12 illustrates the simulated draught force for the four different parameter combinations, with the measured draught force indicated by a dashed black line. Overall, a good agreement was observed between the measurement and the simulation for the average force across all parameter combinations. However, the simulations exhibited larger fluctuations compared to the empirically measured results, which can be attributed to the discrete nature of the simulation, as the size of the spheres in the simulation exceeds that of real soil grains (Tamás and Bernon, 2021; Ucgul et al., 2014).

Fig. 13 illustrates the surface profile following tillage for the four different parameter combinations. In cases 1 and 3, the surface profile is rendered as a straight line, which is not realistic. Conversely, cases 2 and 4 depict a U-shaped furrow, consistent with findings in the literature (Ucgul et al., 2014). As can be seen in Table 4, parameter combinations 1, 3, and 2, 4 primarily differ in density and micromechanical Poisson number. However, increasing the micromechanical Poisson number to $\nu = 0.45$, or the density to $\rho = 2400\,\mathrm{kg/m^3}$, or both at the same time does not alter the surface profile in case 1; it remains flat in each instance. This suggests that the furrow profile is not influenced by a single parameter but results instead from the combination of several DEM parameters.

The calibration process could be further refined by considering the standard deviation as a quantity to be optimized, potentially leading to an even better match between the simulated and measured forces.

Table 4
The average values of the clusters containing the most agents, the average draught force and its standard deviation with the specified micromechanical DEM parameters.

Control Variable		#1	#2	#3	#4
ρ	particle density	2082 kg/m^3	2379 kg/m^3	1993 kg/m^3	2387 kg/m^3
E	Young-modulus	2.979×10^6 Pa	2.305×10^6 Pa	8.960×10^6 Pa	7.978×10^6 Pa
ν	Poisson-number	0.259	0.475	0.227	0.486
e	coeff. of restitution	0.063	0.258	0.104	0.113
μ	sliding friction coeff.	0.684	0.637	0.684	0.495
μ_r	rolling friction coeff.	0.229	0.277	0.282	0.264
agents in the cluster		5	4	9	10
$\overline{F}_x$	draught force	316.0 N	303.9 N	312.5 N	310.5 N
ΔF_x	standard dev. of F_x	25.6 N	26.2 N	25.5 N	25.8 N

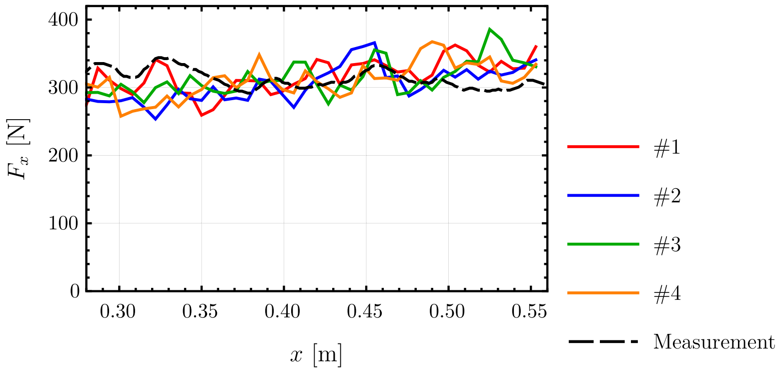


Fig. 12. The draught force during soil loosening with different parameters. The parameters can be seen in Table 4.

Another option for refinement would be to include more measurements with different tillage velocities or to include the furrow profile as an optimization parameter. Nonetheless, the present research demonstrates the feasibility of calibrating parameters and reproducing measurements using the in-house developed GPUDEM within a relatively short timeframe. Previous authors have not conducted such detailed calibration studies. In a similar study of the interaction between soil and a tool Tamás (2024) used a genetic algorithm to calibrate two soil parameters; only 85 simulations were run for that study and the duration of each was approximately 1 h. A DEM parameter calibration by Westbrink et al. (2021) run in 15 seconds/agent using the LIGGGHTS® open-source DEM package on multiple CPU cores, although it considered only 29 681 particles. Roessler and Katterfeld (2019) also calibrated three micromechanical DEM parameters using LIGGGHTS with 10 000 particles, but only 576 parameter combinations were tested, without the use of an optimization algorithm. Mohajeri et al. (2020) calibrated DEM simulations against ring shear tests; their simulation modeled 1 767 particles and ran for 11 min. In most cases reported in the literature, the evaluation of a single agent (i.e. simulation) takes hours (Lubbe et al., 2022; Nguyen, 2022); thus, employing an evolutionary algorithm in those cases takes several days or even weeks.

5. Conclusion

The paper introduced an in-house developed GPUDEM library. The main novelty of this paper is using GPUs to model the interaction between soil and a sweep tool at the required accuracy but with reduced runtime. GPUDEM is also applicable to a wide range of problems involving spherical elements, accommodating particles of varying sizes and materials. Beyond achieving a notable reduction in runtime, the following conclusions can be drawn:

- The main bottleneck of our GPU-based DEM software is the memory limitation during contact search. Despite that, runtime scales linearly with the number of particles if the domain decomposition based Cell-Linked List (CLL) contact search algorithm is used.
- All numerical time-step schemes exhibit first-order accuracy when the continuity condition is not fulfilled. Given that the forces undergo discontinuous changes during particle collisions, employing higher-order numerical integration methods becomes nonsensical, as their accuracy also drops to first-order, without the proper handling of the discontinuities.
- It was found that increasing particle density, and consequently particle weight, resulted in a linear increase in the draught force required to move the tool at a constant velocity. While increasing the friction coefficient and Young's modulus in the Hertz-Mindlin contact model also increased the draught force, the relationships observed in these cases were not linear. Conversely, increasing the coefficient of restitution was found to decrease the draught force linearly. The correlations were established based on 100 data points each, at a much higher resolution than previous studies. These results can be used to better understand the effect of micromechanical parameters in the Hertz–Mindlin model.
- Using differential evolution the micromechanical parameters of a soil were calibrated, resulting in the identification of several potential parameter combinations that can replicate the measurement results, but produce different surface profiles. Remarkably, the calibration process was completed within a single day, despite

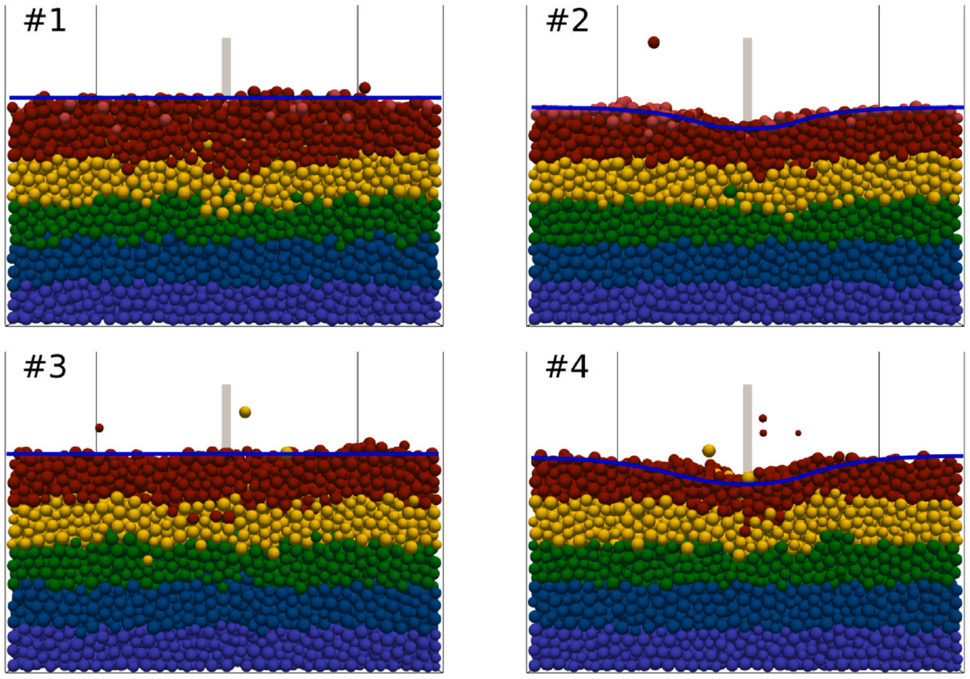


Fig. 13. The surface profile (blue line) after the sweep tool left the domain. The different subfigures depict the results using different parameter sets given in Table 4. The different colors show the layers based on the initial particle height.

2 700 instances of settling and sweep tool tillage being conducted, each involving 38 912 particles.

The exceptionally fast runtime of the GPUDEM software opens up opportunities to address additional real-world challenges. In the future, the integration of other contact models into GPUDEM is feasible. The DEM-FEM connection can also be seamlessly integrated, since the forces acting on the geometry triangles have already been calculated. With this information, deformation can be computed using an appropriate FEM program, allowing the simulation to progress with the updated geometry.

GPUDEM is particularly well-suited for mezo-sized DEM problems involving 10 000 to 100 000 particles, where its current design with a per-thread approach offers optimal efficiency. Using GPUDEM, soil parameters can be rapidly calibrated on a single GPU. The calibrated DEM model can then be employed to investigate the parameter dependence (e.g., tillage depths, velocity, geometrical parameters) of soil-tool interactions and optimize tool geometry. As our solver is focused on running efficiently on a single-GPU, it does not support distributing the workload across many GPUs, and the maximum number of particles is limited by the maximum number of threads that a GPU can handle. In the future, multi-GPU support can be added using domain-based decomposition to handle larger problems.

CRediT authorship contribution statement

Dániel Nagy: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Data curation, Conceptualization. **László Pásthy:** Writing – review & editing, Validation, Methodology, Conceptualization. **Kornél Tamás:** Writing – review & editing, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Data curation, 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.

Data availability

Data will be made available on request.

Acknowledgments

Supported by the ÚNKP-23-3-I-BME-51 New National Excellence Program of the Ministry for Culture and Innovation, Hungary from the source of the National Research, Development and Innovation Fund. This paper was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences, Hungary. The research reported in this paper is part of project no. TKP-6-6/PALY-2021, implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, Hungary, financed under the TKP2021-NVA funding scheme. The project supported by the Doctoral Excellence Fellowship Programme (DCEP) is funded by the National Research Development and Innovation Fund of the Ministry of Culture, Hungary and Innovation and the Budapest University of Technology and Economics, Hungary, under a grant agreement with the National Research, Development and Innovation Office. This research was supported by the ÚNKP, Hungary, funded by the National Research Development and Innovation Fund under grant number ÚNKP-23-5-BME-80. This paper was supported by the Hungarian Scientific Research Fund, Hungary (NKFIH FK-146067). This research was supported by the EKÖP, Hungary funded by the National Research Development and Innovation Fund under grant number EKÖP-24-3-BME-13.

Appendix A. Figures

See Figs. A.14–A.18.

Appendix B. Tables

See Tables B.5 and B.6.

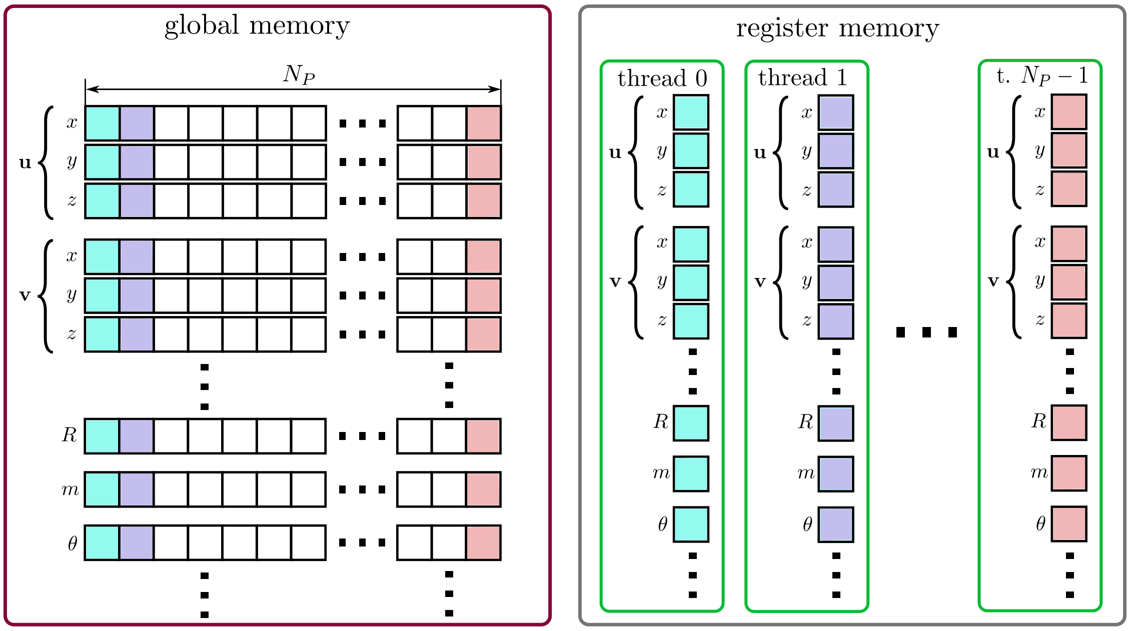


Fig. A.14. Storage of the particles in the global memory and registers. Cells with the same color contain the same particle data. Rows are contiguous memory areas of size N_P, where N_P is the number of particles. All data is stored in this way, including position ($\boldsymbol{u}$), velocity ($\boldsymbol{v}$), radius (R), mass (m) and moment of inertia (θ).

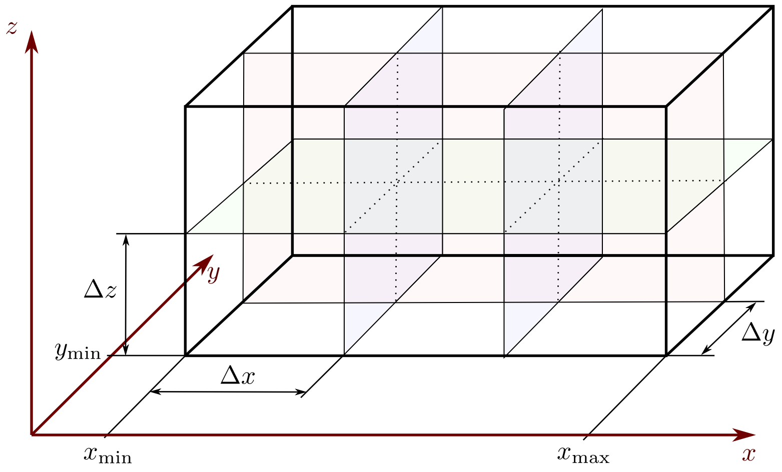


Fig. A.15. The underlying mesh in a block-based computational domain, where the size of the cells is $\Delta x \times \Delta y \times \Delta z$. The mesh extends in all directions between the specified minimum and maximum values (see $x_{\min}$, $x_{\max}\dots$).

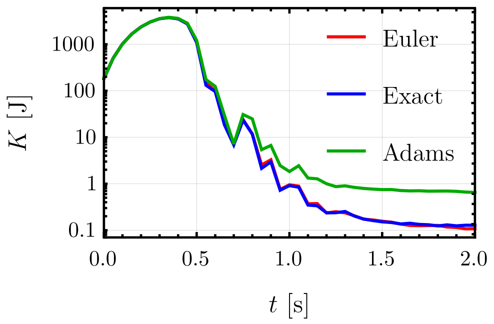


Fig. A.16. The change in kinetic energy in the same simulation with $\Delta t = 1 \times 10^{-3}$ s, but using different time stepping schemes as indicated in the legend on the right.

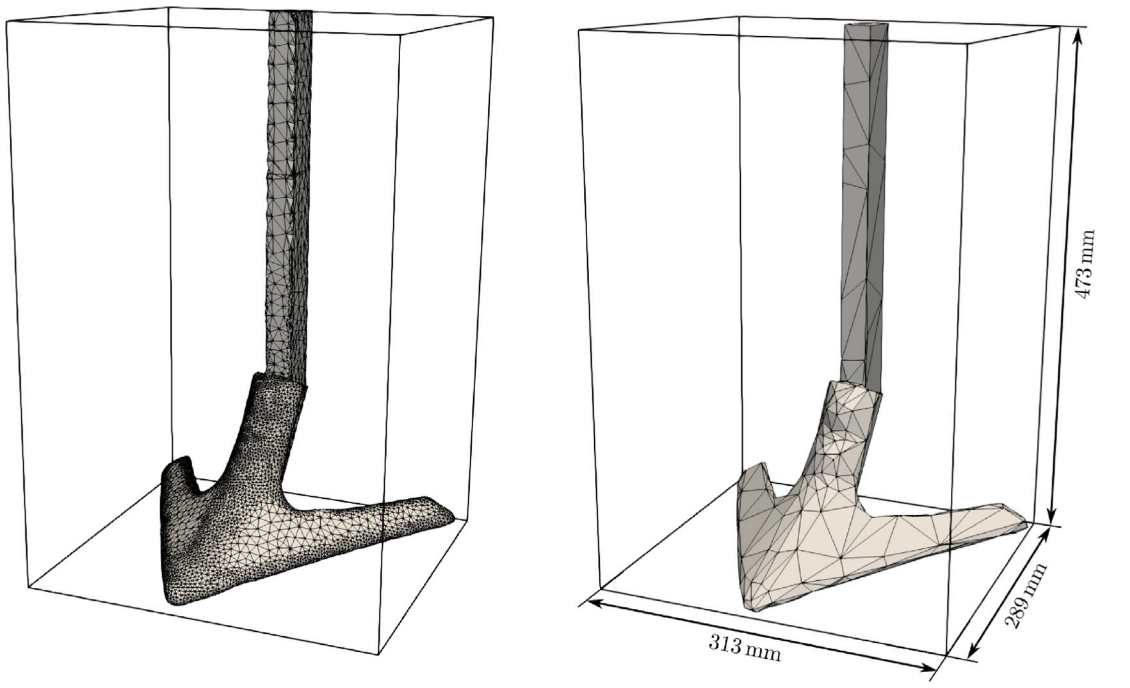


Fig. A.17. The 3D scanned sweep tool model with 14 304 triangles (left) and the simplified version for DEM simulations with 500 triangles (right)

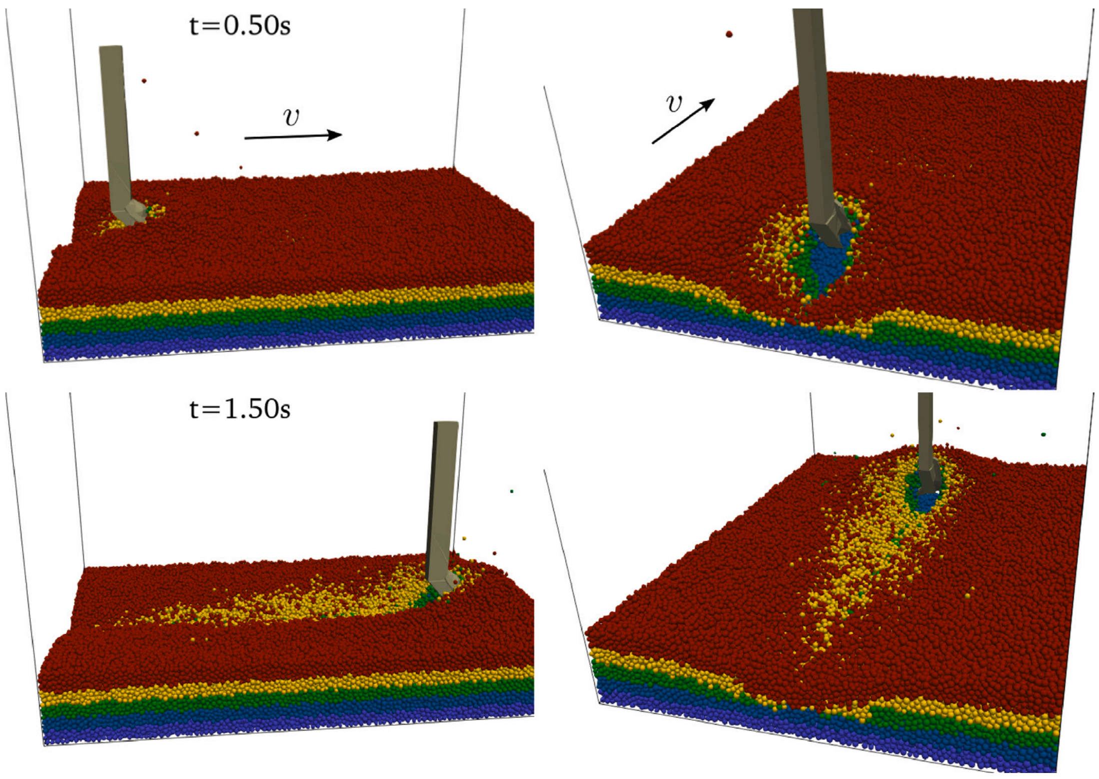


Fig. A.18. Sweep tool soil sweeping at a velocity of $v = 0.7\,\mathrm{m/s}$, viewed from two different angles. The domain size is $1\,\mathrm{m} \times 0.7\,\mathrm{m}$ and the simulation contains $N_P = 146\,180$ spherical particles. The different colors of the particles depict the equally thick initial layers.

Table B.5
Simulation settings.

Simulation	#1	#2	#3	#4
Dom. size l [m]	$3 \times 1 \times 3$	$2 \times 2 \times 2$	$1 \times 0.7 \times 2$	$0.7 \times 0.6 \times 0.7$
Mesh (cell number)	$25 \times 8 \times 25$	$28 \times 28 \times 28$	–	$32 \times 24 \times 32$
$N_{P,C}$	12	8	–	8
$N_{C,\max}$	12	12	12	12
time step	Exact	Exact	Euler	Euler
Δt [s]	1×10^{-4}	1×10^{-4}	5×10^{-5}	1×10^{-4}
Particle number N_P	509/8519	*varied*	147 456	38 912
Particle properties				
$\overline{R}$ [mm]	200/40	30	4.8	8.5
$\pm\Delta R$ [mm]	60/12	5	0.8	1.5
ρ [kg/m^3]	2500	1000	1850	1850
E [Pa]	2.5×10^8	2×10^5	2×10^6	2×10^6
G [Pa]	1×10^8	7.69×10^4	7.24×10^5	7.24×10^5
ν	0.25	0.3	0.38	0.38
e	0.5	0.1	0.5	0.5
μ	0.5	0.6	0.6	0.6
μ_0	0.5	0.7	0.7	0.7
μ_r	0.01	0.05	0.03	0.03

Table B.6
Warp stall rate and number of operations in the different parts of the program.

Part	Warp stall rate	Number of operations of total
Solver logics	2.33%	2.83%
Geometry handling	0.28%	1.86%
Contact search	57.64%	67.66%
Distance calculation	4.78%	7.74%
Force calculation	1.53%	3.74%
Acceleration calculation	0.08%	0.06%
time step	0.00%	0.22%
Synchronization	19.94%	11.43%

References

Abo-Elnor, M., Hamilton, R., Boyle, J., 2003. 3D dynamic analysis of soil–tool interaction using the finite element method. J. Terramech. 40 (1), 51–62. http://dx.doi.org/10.1016/j.jterra.2003.09.002.

Aikins, K.A., Ucgul, M., Barr, J.B., Awuah, E., Antille, D.L., Jensen, T.A., Desbiolles, J.M., 2023. Review of discrete element method simulations of soil tillage and furrow opening. Agric. 13 (3), 541. http://dx.doi.org/10.3390/agriculture13030541.

Asaf, Z., Rubinstein, D., Shmulevich, I., 2007. Determination of discrete element model parameters required for soil tillage. Soil Tillage Res. 92 (1–2), 227–242. http://dx.doi.org/10.1016/j.still.2006.03.006.

Bellen, A., Zennaro, M., 2013. Numerical Methods for Delay Differential Equations. Oxford Science Publications.

Bentaher, H., Ibrahmi, A., Hamza, E., Hbaieb, M., Kantchev, G., Maalej, A., Arnold, W., 2013. Finite element simulation of moldboard–soil interaction. Soil Tillage Res. 134, 11–16. http://dx.doi.org/10.1016/j.still.2013.07.002.

Binelo, M.O., de Lima, R.F., Khatchatourian, O.A., Stránskỳ, J., 2019. Modelling of the drag force of agricultural seeds applied to the discrete element method. Biosys. Eng. 178, 168–175. http://dx.doi.org/10.1016/j.biosystemseng.2018.11.013.

Busari, M.A., Kukal, S.S., Kaur, A., Bhatt, R., Dulazi, A.A., 2015. Conservation tillage impacts on soil, crop and the environment. Int. Soil Water Conserv. Res. 3 (2), 119–129. http://dx.doi.org/10.1016/j.iswcr.2015.05.002.

Butcher, J.C., 2016. Numerical Methods for Ordinary Differential Equations. John Wiley & Sons.

Cai, R., Xu, L., Zheng, J., Zhao, Y., 2018. Modified cell-linked list method using dynamic mesh for discrete element method. Powder Technol. 340, 321–330. http://dx.doi.org/10.1016/j.powtec.2018.09.034.

Chen, Y., Munkholm, L.J., Nyord, T., 2013. A discrete element model for soil–sweep interaction in three different soils. Soil Tillage Res. 126, 34–41. http://dx.doi.org/10.1016/j.still.2012.08.008.

Cheng, J., Grossman, M., McKercher, T., 2014. Professional CUDA C Programming. John Wiley & Sons.

Di Renzo, A., Di Maio, F.P., 2004. Comparison of contact-force models for the simulation of collisions in DEM-based granular flow codes. Chem. Eng. Sci. 59 (3), 525–541. http://dx.doi.org/10.1016/j.ces.2003.09.037.

Dosta, M., Skorych, V., 2020. MUSEN: An open-source framework for GPU-accelerated DEM simulations. SoftwareX 12, 100618. http://dx.doi.org/10.1016/j.softx.2020.100618.

Fang, L., Zhang, R., Vanden Heuvel, C., Serban, R., Negrut, D., 2021. Chrono:: GPU: An open-source simulation package for granular dynamics using the discrete element method. Processes 9 (10), 1813. http://dx.doi.org/10.3390/pr9101813.

Feoktistov, V., 2006. Differential Evolution. Springer.

Fielke, J.M., 1999. Finite element modelling of the interaction of the cutting edge of tillage implements with soil. J. Agricul. Eng. Res. 74 (1), 91–101. http://dx.doi.org/10.1006/jaer.1999.0440.

Gan, J., Zhou, Z., Yu, A., 2016. A GPU-based DEM approach for modelling of particulate systems. Powder Technol. 301, 1172–1182. http://dx.doi.org/10.1016/j.powtec.2016.07.072.

Godwin, R., O'dogherty, M., Saunders, C., Balafoutis, A., 2007. A force prediction model for mouldboard ploughs incorporating the effects of soil characteristic properties, plough geometric factors and ploughing speed. Biosys. Eng. 97 (1), 117–129. http://dx.doi.org/10.1016/j.biosystemseng.2007.02.001.

Golshan, S., Munch, P., Gassmöller, R., Kronbichler, M., Blais, B., 2023. Lethe-DEM: An open-source parallel discrete element solver with load balancing. Comput. Part. Mech. 10 (1), 77–96. http://dx.doi.org/10.1007/s40571-022-00478-6.

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

Govender, N., Wilke, D.N., Wu, C.-Y., Tuzun, U., Kureck, H., 2019. A numerical investigation into the effect of angular particle shape on blast furnace burden topography and percolation using a GPU solved discrete element model. Chem. Eng. Sci. 204, 9–26. http://dx.doi.org/10.1016/j.ces.2019.03.077.

Granlund, T., 2012. Instruction latencies and throughput for AMD and intel x86 processors. Technical report, KTH.

Gupta, P., Gupta, C., Pandey, K., 1989. An analytical model for predicting draft forces on convex-type wide cutting blades. Soil Tillage Res. 14 (2), 131–144. http://dx.doi.org/10.1016/0167-1987(89)90027-5.

Hairer, E., Nø rsett, S.P., Wanner, G., 1993. Solving Ordinary Differential Equations I. Nonstiff problems. Springer Series in Comput. Math..

Hilarov, V., Damaskinskaya, E., Gesin, I., 2023. The effect of materials structure on the features of fracture process in rocks: Discrete elements modeling and laboratory experiment. Izv. Phys. Solid Earth 59 (3), 477–485. http://dx.doi.org/10.1134/S1069351323030035.

Huimin, F., Changying, J., Ch, F.A., et al., 2016. Analysis of soil dynamic behavior during rotary tillage based on distinct element method. Nongye Jixie Xuebao/Trans. Chin. Soc. Agricult. Mach. 47 (3).

Karimi, K., Dickson, N.G., Hamze, F., 2010. A performance comparison of CUDA and opencl. http://dx.doi.org/10.48550/arXiv.1005.2581, arXiv preprint arXiv:1005.2581.

Kraus, S., Woitzik, C., Dosta, M., Düster, A., 2021. Simulation of granular materials with the discrete element method to investigate their suitability as crash-absorber in ship collisions. PAMM 21 (1), http://dx.doi.org/10.1002/pamm.202100036.

Kruggel-Emden, H., Sturm, M., Wirtz, S., Scherer, V., 2008. Selection of an appropriate time integration scheme for the discrete element method (DEM). Comput. Chem. Eng. 32 (10), 2263–2279. http://dx.doi.org/10.1016/j.compchemeng.2007.11.002.

Li, X., Luo, Z., Hao, Z., Zheng, E., Yao, H., Zhu, Y., Wang, X., 2024. Investigation on tillage resistance and soil disturbance in wet adhesive soil using discrete element method with three-layer soil-plough coupling model. Powder Technol. 119463. http://dx.doi.org/10.1016/j.powtec.2024.119463.

Liu, G.-Y., Xu, W.-J., Sun, Q.-C., Govender, N., 2020. Study on the particle breakage of ballast based on a GPU accelerated discrete element method. Geosci. Front. 11 (2), 461–471. http://dx.doi.org/10.1016/j.gsf.2019.06.006.

Lubbe, R., Xu, W.-J., Zhou, Q., Cheng, H., 2022. Bayesian calibration of GPU–based DEM meso-mechanics part II: Calibration of the granular meso-structure. Powder Technol. 407, 117666. http://dx.doi.org/10.1016/j.powtec.2022.117666.

Melheim, J.A., 2005. Cluster integration method in Lagrangian particle dynamics. Comput. Phys. Comm. 171 (3), 155–161. http://dx.doi.org/10.1016/j.cpc.2005.05.003.

Milkevych, V., Munkholm, L.J., Chen, Y., Nyord, T., 2018. Modelling approach for soil displacement in tillage using discrete element method. Soil Tillage Res. 183, 60–71. http://dx.doi.org/10.1016/j.still.2018.05.017.

Mohajeri, M.J., Do, H.Q., Schott, D.L., 2020. DEM calibration of cohesive material in the ring shear test by applying a genetic algorithm framework. Adv. Powder Technol. 31 (5), 1838–1850. http://dx.doi.org/10.1016/j.apt.2020.02.019.

Munshi, A., 2009. The opencl specification. In: 2009 IEEE Hot Chips 21 Symposium. HCS, IEEE, pp. 1–314. http://dx.doi.org/10.1109/HOTCHIPS.2009.7478342.

Nguyen, Q.H., 2022. Machine learning in the calibration process of discrete particle model. University of Twente.

Odey, S.O., 2016. Design steps of narrow tillage tools for draught reduction and increased soil disruption–a review. Agricul. Eng. Int.: CIGR J. 18 (1), 91–102.

Onwualu, A., Watts, K., 1998. Draught and vertical forces obtained from dynamic soil cutting by plane tillage tools. Soil Tillage Res. 48 (4), 239–253. http://dx.doi.org/10.1016/S0167-1987(98)00127-5.

Pasthy, L., Graeff, J., Tamás, K., 2022. Development of a 2D discrete element software with labview for contact model improvement and educational purposes.. In: ECMS. pp. 203–209.

Pásthy, L., Tamás, K., 2023. Talaj-eke-szármaradvány egymásra hatás diszkrételemes szimulációjának paraméter érzékenységi vizsgálata=Parameter Sensitivity Analysis of the Soil-Plough-Stem Interaction with Discrete Element Method.

Pratibha, G., Srinivas, I., Rao, K., Raju, B., Shanker, A.K., Jha, A., Kumar, M.U., Rao, K.S., Reddy, K.S., 2019. Identification of environment friendly tillage implement as a strategy for energy efficiency and mitigation of climate change in semiarid rainfed agro ecosystems. J. Clean. Prod. 214, 524–535. http://dx.doi.org/10.1016/j.jclepro.2018.12.251.

Qi, L., Chen, Y., Sadek, M., 2019. Simulations of soil flow properties using the discrete element method (DEM). Comput. Electron. Agric. 157, 254–260. http://dx.doi.org/10.1016/j.compag.2018.12.052.

Rádics, J.P., Jóri, I.J., 2010. Development of 3E tillage system and machinery to challenge climate change impacts. Periodica Polytech. Mech. Eng. 54 (1), 49–56. http://dx.doi.org/10.3311/pp.me.2010-1.08.

Roessler, T., Katterfeld, A., 2019. DEM parameter calibration of cohesive bulk materials using a simple angle of repose test. Particuology (ISSN: 1674-2001) 45, 105–115. http://dx.doi.org/10.1016/j.partic.2018.08.005.

Saunders, C., Godwin, R., O'Dogherty, M., 2000. Prediction of soil forces acting on mouldboard ploughs. In: Fourth Int. Conf. on Soil Dyn., Adelaide.

Shen, J., 2017. Soil-machine interactions: a finite element perspective. http://dx.doi.org/10.1201/9780203739228, Routledge.

Solutions, D., 2014. EDEM 2.6 Theory Reference Guide. Edinburgh, United Kingdom.

Tamás, K., 2018. The role of bond and damping in the discrete element model of soil-sweep interaction. Biosys. Eng. 169, 57–70. http://dx.doi.org/10.1016/j.biosystemseng.2018.02.001.

Tamás, K., 2024. Modelling the interaction of soil with a passively-vibrating sweep using the discrete element method. Biosys. Eng. 245, 199–222. http://dx.doi.org/10.1016/j.biosystemseng.2024.06.006.

Tamás, K., Bernon, L., 2021. Role of particle shape and plant roots in the discrete element model of soil–sweep interaction. Biosys. Eng. 211, 77–96. http://dx.doi.org/10.1016/j.biosystemseng.2021.09.001.

Tamás, K., Jóri, I.J., Mouazen, A.M., 2013. Modelling soil–sweep interaction with discrete element method. Soil Tillage Res. 134, 223–231. http://dx.doi.org/10.1016/j.still.2013.09.001.

Tekeste, M.Z., Balvanz, L.R., Hatfield, J.L., Ghorbani, S., 2019. Discrete element modeling of cultivator sweep-to-soil interaction: Worn and hardened edges effects on soil-tool forces and soil flow. J. Terramech. 82, 1–11. http://dx.doi.org/10.1016/j.jterra.2018.11.001.

Ucgul, M., Fielke, J.M., Saunders, C., 2014. 3D DEM tillage simulation: Validation of a hysteretic spring (plastic) contact model for a sweep tool operating in a cohesionless soil. Soil Tillage Res. 144, 220–227. http://dx.doi.org/10.1016/j.still.2013.10.003.

Ucgul, M., Fielke, J.M., Saunders, C., 2015. Three-dimensional discrete element modelling (DEM) of tillage: Accounting for soil cohesion and adhesion. Biosys. Eng. 129, 298–306. http://dx.doi.org/10.1016/j.biosystemseng.2014.11.006s.

Ucgul, M., Saunders, C., Fielke, J.M., 2017. Discrete element modelling of tillage forces and soil movement of a one-third scale mouldboard plough. Biosys. Eng. 155, 44–54. http://dx.doi.org/10.1016/j.biosystemseng.2016.12.002.

Ucgul, M., Saunders, C., Li, P., Lee, S.-H., Desbiolles, J.M., 2018. Analyzing the mixing performance of a rotary spader using digital image processing and discrete element modelling (DEM). Comput. Electron. Agric. 151, 1–10. http://dx.doi.org/10.1016/j.compag.2018.05.028.

Wanner, G., Hairer, E., 1996. Solving Ordinary Differential Equations II, vol. 375. Springer Berlin Heidelberg New York.

Westbrink, F., Elbel, A., Schwung, A., Ding, S.X., 2021. Optimization of DEM parameters using multi-objective reinforcement learning. Powder Technol. 379, 602–616. http://dx.doi.org/10.1016/j.powtec.2020.10.067.

Yang, P., Zang, M., Zeng, H., Guo, X., 2020. The interactions between an off-road tire and granular terrain: GPU-based DEM-fem simulation and experimental validation. Int. J. Mech. Sci. 179, 105634. http://dx.doi.org/10.1016/j.ijmecsci.2020.105634.

Zhang, Y., Hou, S., Di, S., Liu, Z., Xu, Y., 2023a. DEM–SPH coupling method for landslide surge based on a GPU parallel acceleration technique. Comput. Geotech. 164, 105821. http://dx.doi.org/10.1016/j.compgeo.2023.105821.

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

Zhang, W., Wu, Z., Peng, C., Li, S., Dong, Y., Yuan, W., 2023b. Modelling large-scale landslide using a GPU-accelerated 3D MPM with an efficient terrain contact algorithm. Comput. Geotech. 158, 105411. http://dx.doi.org/10.1016/j.compgeo.2023.105411.

Zhang, C., Xu, J., Zheng, Z., Wang, W., Liu, L., Chen, L., 2023c. Three-dimensional DEM tillage simulation: Validation of a suitable contact model for a sweep tool operating in cohesion and adhesion soil. J. Terramech. 108, 59–67. http://dx.doi.org/10.1016/j.jterra.2023.05.003.

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