DEVELOPMENT OF A 3D DEM SIMULATION SOFTWARE FOR COUPLED SIMULATIONS

József Balázs Gräff[1], László Pásthy[2], Kornél Tamás[3]

[1]Bachelor's student, Budapest University of Technology and Economics,
e-mail: graff.jozsef@gszk.bme.hu
[2]Department of Machine and Product Design, Budapest University of Technology and Economics,
Faculty of Mechanical Engineering, e-mail: pasthy.laszlo@gt3.bme.hu
[3]Department of Machine and Product Design, Budapest University of Technology and Economics,
Faculty of Mechanical Engineering, e-mail: tamas.kornel@gt3.bme.hu

ABSTRACT

This study is based on the development and methodology of building 3 dimensional discrete-, and finite element (DEM-FEM) simulation engines, as well as the potential way of improving usability and simulation speed. The program demonstrated here, BMEDEM3D, natively supports discrete element, finite element, and mass spring simulations. Currently no simulation software supports all these methods without the need for coupling several different programs to achieve these results. BMEDEM3D contains a powerful Graphical User Interface, which lets the user configure everything without needing to understand programming.

INTRODUCTION

Depending on the type, and physical properties of the materials being simulated, there are several ways of creating a simulation, each with their benefits and drawbacks. In BMEDEM3D multiple methods are used, meaning that there is at least one viable option when simulating solid materials (Pásthy & Tamás, 2023). The discrete element method (DEM), for example, is designed for the simulation of granular materials like soil (Cundall & Strack, 1979). It splits the material body into small, discrete particles, that can interact with both other particles of the original material and with external bodies. The finite element method (FEM) on the other hand, is designed for continuum materials that, while able to deform, still stay in a similar position relative to other parts of the body (Zienkiewicz et al., 2005). FEM bodies are split into a finite number of different elements that connect to each other through a set of nodes (Nikishkov, 2004). The various forces that act on these nodes are calculated, and the calculated results are then interpolated to the entire body. The connection between these nodes is predetermined and unchangeable, therefore deformations are a part of the model, but tearing is not. Which is why the final simulation method used by us, the mass-spring model (MS) was implemented. In this model, the body is split into several points of given mass, which are connected to others using a spring-dampener system (Provot, 1995). If a critical pressure is surpassed, the connection may tear. Mass spring models are often used in the simulation of textiles, as well as elastic, organic tissues (Pieczywek & Zdunek, 2017).

While these simulation models are implemented by various software programs, they usually have disadvantages. The biggest, and most important one is that most programs only implement one of the above simulation models, meaning that the simulation of interactions between vastly different materials are difficult to impossible, requiring several different programs to be coupled with each other.

This isn't the only flaw that simulation programs have, as the two categories into which simulation software is usually split, all have major setbacks, these being open source and commercial programs. Free Open-Source Software (FOSS), for example, lets the user do whatever they want, meaning that the program can be tailored to specific needs, features can be freely added and merged with the original program, all while being freely available to everyone. The problem is that FOSS software tends to be difficult to use, especially in the case of DEM simulation programs, which tends to both lack a user interface, and which expect the user to understand programming to configure the simulation. For example, Yet Another Dynamic Engine (Yade), a popular FOSS DEM simulation program requires users to create the simulation using the Python programming language (Kozicki & Donzé, 2009). Commercial software, on the other hand, is great at usability, with everything from configuring to analysing the simulation being part of one, large and powerful Graphical User Interface (GUI), and which also have a highly optimized solver that can be run on server farms. Due to their nature as commercial products, however, they are locked down, with extensibility being limited or non-existent. With BMEDEM3D, the goal is to create a software, which fixes these issues, having both an easy-to-use interface, and a multi-model, highly performant simulation engine.

MATERIAL AND METHODS

There are several accepted models for the different types of simulation methods. In this chapter, the contact models used in BMEDEM3D are explained.

Communications of the ECMS, Volume 38, Issue 1,
Proceedings, ©ECMS Daniel Grzonka, Natalia Rylko,
Grazyna Suchacka, Vladimir Mityushev (Editors) 2024
ISBN: 978-3-937436-84-5/978-3-937436-83-8(CD) ISSN 2522-2414

The Discrete Element Method and the Hertz-Mindlin model

In the case of DEM simulations, the Hertz-Mindlin contact model is used. In this model, every particle of the material is made up of non-deformable spheres with the following properties (Pasthy et al., 2022) seen in Table 1.

Table 1: Properties of Particles in the Hertz-Mindlin model

Name	Notation	Unit
Density	ρ	kgm^{-3}
Radius	R	m
Velocity	v	ms^{-1}
Angular Velocity	ω	$rads^{-1}$
Position	r	m
Rotation	φ	rad
Young's modulus	E	Pa
Poisson's ratio	ν	-
Damping factor	β	-
Sliding friction coefficient	μ	-
Rolling friction coefficient	μ_0	-

Every interaction can be split into 2 different parts: the normal and tangential force from a parallel spring-dampener system. Some global constants also need to be defined, as seen in
Table 2.

Table 2: Global Constants in the Simulation System

Name	Notation	Unit
Gravity	g	ms^{-2}
Timestep	Δt	s

At the beginning of the simulation, some particle-specific constants are calculated Table 3.

Table 3: List of Precalculated, Particle-specific Constants

Name	Notation	Unit	Equation
Mass	m	kg	$\frac{3}{4}\cdot\pi\cdot R^3\cdot\rho$
Shear modulus	G	Pa	$\frac{E}{2(1+\nu)}$

Afterwards, some so-called equivalent parameters are calculated between two interacting particles, denoted using the notation of the value and a star, Equations (1-4).

$$m^* = (m_1^{-1} + m_2^{-1})^{-1} \quad (1)$$

$$R^* = (R_1^{-1} + R_2^{-1})^{-1} \quad (2)$$

$$E^* = \left(\frac{1-\nu_1^2}{E_1} + \frac{1-\nu_2^2}{E_2}\right)^{-1} \quad (3)$$

$$G^* = \left(\frac{1-\nu_1}{G_1} + \frac{1-\nu_2}{G_2}\right)^{-1} \quad (4)$$

The steps above can be calculated even before the start of the simulation, and stored in memory, while every equation below is step specific. Each simulation step can be split into several parts: contact detection, determining the various properties of the contact, the calculation of the normal and tangential forces, as well as the application of motion onto the particles using Newton's laws and some approximation functions.

Using Equations (5), (6) and (7), the normal force for the particle interaction can be defined.

$$F_{ne} = \frac{4}{3}\cdot E^*\cdot\sqrt{R^*}\cdot\delta_n^{\frac{3}{2}} \quad (5)$$

$$S_n = 2\cdot E^*\cdot\sqrt{R^*\cdot\delta_n}$$

$$F_{nd} = -2\cdot\sqrt{\frac{5}{6}\cdot S_n\cdot m^*}\cdot\beta\cdot v_{nrel} \quad (6)$$

$$F_n = F_{ns} + F_{nd} \quad (7)$$

Where δ_n [m] is the normal overlap between particles.
v_{nrel} [ms^{-1}] is the relative normal velocity.
S_n [Nm^{-1}] is the normal stiffness.
F_{ne} [N] is the normal elastic force.
F_{nd} [N] is the normal dampening force.
F_n [N] is the sum of all normal forces.

The calculation of the tangential force follows a similar procedure, however, the equation used can depend on the state of friction between particles, as seen in Equation (8).

$$S_t = 8\cdot G^*\cdot\sqrt{R^*\cdot\delta_t}$$

$$F_{te} = -S_t\cdot\delta_t$$

$$F_{td} = -2\cdot\sqrt{\frac{5}{6}\cdot S_t\cdot m^*}\cdot\beta\cdot v_{trel}$$

$$F_t = \begin{cases} F_{te} + F_{td} & \text{if } |F_{te}+F_{td}| < \mu_0\cdot F_n \\ F_n\cdot\mu & \text{if } |F_{te}+F_{td}| \geq \mu_0\cdot F_n \end{cases} \quad (8)$$

Where δ_t [m] is the tangential overlap between particles.
v_{trel} [ms^{-1}] is the tangential relative velocity.
S_t [Nm^{-1}] is the tangential stiffness.
F_{te} [N] is the tangential elastic force.
F_{td} [N] is the tangential dampening force.
F_t [N] is the tangential force.

The first three steps of the calculations are like that of the normal forces, however, the last step Equation (8), might be entirely different depending on the state of friction between the two objects. Because of this, the equation is conditional, where – depending on a threshold – either the elastic and dampening forces are summed like with the normal force, or it is the result of multiplying the sliding coefficient of friction with the normal force.

Applying Motion to the Particle

After every possible interaction has been evaluated, the last step is using the forces previously calculated to determine the new motion state of each particle. Firstly, the sum of all torques and forces are applied to the particle according to Equation (9) and (10).

$$F = \sum_{i=1}^{n} F_{ni} + F_{ti} \quad (9)$$

$$M = \sum_{i=1}^{n} r_{reli} \times F_i \quad (10)$$

Where $n\ [-]$ is the number of interactions of a particle.

F [N] is the sum of all the forces applied to a particle.

M [Nm] is the sum of all torques applied to a particle.

Then using Newton's laws, the acceleration and angular acceleration of the sphere particle can be determined as defined in Equations (11) and (13).

$$a = \frac{F}{m} + g \quad (11)$$

$$\theta = \frac{2}{5} \cdot m \cdot R^2 \quad (12)$$

$$\varepsilon = \frac{1}{\theta} \cdot M \quad (13)$$

Where $a\ [ms^{-2}]$ is the acceleration of the particle.

$\theta\ [kgm^2]$ is the moment of inertia of the particle.

$\varepsilon\ [rads^{-2}]$ is the angular acceleration of the particle.

Using these, and the first order Euler formula, the positional-, and angular velocities of the particle can be calculated according to Equations (14) and (15).

$$v_t = v_{t-1} + a_{t-1} \cdot \Delta t \quad (14)$$

$$\omega_t = \omega_{t-1} + \varepsilon_{t-1} \cdot \Delta t \quad (15)$$

Using the same method, the new position and rotation of the particle can be determined, as seen in Equation (16) and (17).

$$r_t = r_{t-1} + v_{t-1} \cdot \Delta t \quad (16)$$

$$\varphi_t = \varphi_{t-1} + \omega_{t-1} \cdot \Delta t \quad (17)$$

The Finite Element Method

The Finite Element Method consists of three main parts: discretisation of the given material body, pre-calculations to speed up simulation time, and as the simulation steps themselves.

The first is to mesh the given body into the finite elements, that connect to each other (Bathe, 2008). Both external and internal loads act on these nodes, and the effects of these are then finally reapplied to the original body. The data of the nodes made in the first step aren't stored individually, rather they are combined into vectors and matrices as seen in Table 4, because all FEM calculations in the other two steps can be represented as matrix operations.

Table 4: The properties of a FEM object

Name	Notation	Unit
Displacement Vector	U	m
Velocity Vector	$\dot{U}$	ms^{-1}
Acceleration Vector	$\ddot{U}$	ms^{-2}
Mass Matrix	M	kg
Stiffness Matrix	K	Nm^{-1}
Displacement to Degrees of Freedom	D_{d2dof}	-
Damping factor (stiffness)	α	s
Damping factor (mass)	β	s^{-1}

D_{d2dof} is a transformation matrix, which allows for the transformation of the external, DEM-based forces to the web of nodes used in FEM simulations.

The second part of FEM simulations is the calculation of constants, whose values do not change during the simulation, to minimize unnecessary computations (Pásthy et al., 2024).

$$C = \alpha \cdot K + \beta \cdot M \quad (18)$$

$$A_1 = \frac{1}{\Delta t^2} \cdot M + \frac{1}{2 \cdot \Delta t} \cdot C + \frac{1}{3} \cdot K \quad (19)$$

$$A_3 = \frac{2}{\Delta t^2} \cdot M - \frac{1}{3} \cdot K \quad (20)$$

$$A_4 = \frac{1}{\Delta t^2} \cdot M + \frac{1}{2 \cdot \Delta t} \cdot C - \frac{1}{3} \cdot K \quad (21)$$

Where $C\ [Nsm^{-1}]$ is the Rayleigh dampening factor.

$A_1\ [Nm^{-1}], A_2\ [N], A_3 [Nm^{-1}], A_4\ [Nm^{-1}]$ are unnamed constant matrices.

The last parts of a FEM simulation are the simulation steps themselves, where the program solves the linear transient basic equation of the given body.

$$F_{n+1} = D_{d2dof} \cdot F_{facet} \quad (22)$$

$$A_2 = \frac{1}{3} \cdot (F_{n-1} + F_n + F_{n+1}) \quad (23)$$

$$U_{n+1} = A_1^{-1} \cdot \left(A_2 + (A_3 \cdot U_n) + (A_4 \cdot U_{n-1})\right) \quad (24)$$

$$U_k = U_{k-1} + (D_{dof2d} \cdot U_n) - (D_{dof2d} \cdot U_{n-1}) \quad (25)$$

Where F_{facet} [N] is the forces of all the nodes of the FEM object arranged into one vector.

U_n [m] is the relative displacement vector of the FEM nodes.

U_k [m] is the vector containing the displacement of the keypoints of the surface.

$D_{dof2d} = D_{d2dof}^T$.

RESULTS

The demonstration program – going by the name "BMEDEM3D" – incorporates the methods and models as depicted in the text, as well as many other systems to make visualizing, and analysing data simpler. BMEDEM3D is made of two, entirely separate programs. The first part, the one which the user sees, is the GUI, as seen in Figure 1, written in LabVIEW with the purpose of making setting up simulations easier, and allowing people with a limited knowledge of programming to be able to use the software. The other part, the solver, is an entirely separate program, made in C++, which handles the simulation and calculations. They communicate by exporting data into files, which the other one can read and understand.

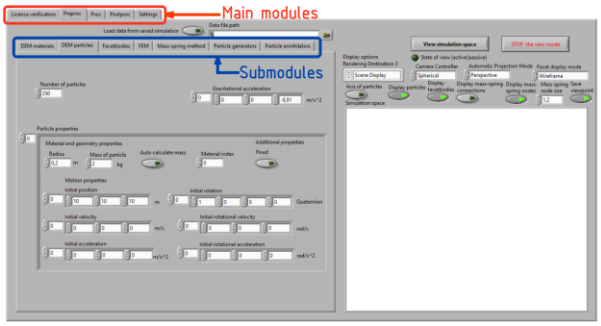


Figure 1: The BMEDEM3D Graphical User Interface

At the start of every simulation, the GUI exports every piece of data provided to it by the user into *.bd3* configuration files, and then using a hidden terminal command, it launches the solver, which then reads these files, and – according to the files' contents – it also enables several "subsystems", which handle optional parts of the simulation. After this is done, the actual simulation phase begins, during which the program exports the particle-, and facet positions after a certain number of simulation steps. These files are then read by the GUI to display the status of the simulation. After all steps are completed, the solver shuts down, and all optional tracking files are exported. The files created during one simulation can always be loaded, even without relaunching the solver. A flowchart of these steps can be seen in Figure 2.

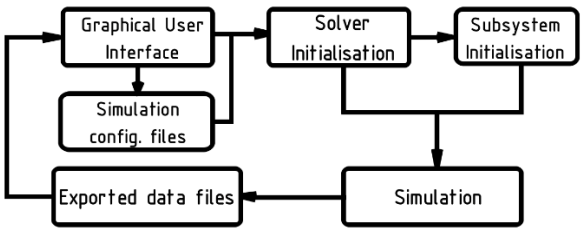


Figure 2: The Simulation Process in BMEDEM3D

An advantage of this method of completely separating the solver program from the GUI is that if the need arises that one must be replaced or altered in a major way, the other part stays intact, and an issue in one of them can be isolated easier based on how the other half responds. There are plans to make an advantage of this architecture, as the GUI program might be replaced in the future.

The BMEDEM3D Graphical User Interface

The BMEDEM3D GUI is split into 3 main modules, which are responsible for the three phases of every simulation: Preprocessing, Processing, and Postprocessing. In this case, preprocessing stands for all the mechanical input data provided to the solver, processing handles the solver's runtime settings, and visualises the current state of the simulation, while postprocessing lets the user see the finished simulation from the export data, as well as handling data evaluations.

The Preprocessor

The preprocessor is where the simulation itself is set up. It includes tabs, where each aspect of the DEM, FEM, and MS system is configured, including disabling unused methods altogether. As the name suggests, BMEDEM3D is primarily a DEM software, so the FEM and mass spring parts of the simulator can be turned off, if they're not needed. If they are, the preprocessor lets us import 3D objects as STL files (Zhang & Shi, 2014), which then can be configured to adhere to one of these models, stay fixed and act like a floor or wall, or even non-colliding for the purpose of creating an interval in space. If no STL files are available, and only standard fixed boxes are needed, those can be created from the Facetbodies menu as well, as seen in Figure 3. These boxes can have another purpose besides collisions because they can also be set to be the volume in which a particle generator creates DEM particles, or a particle annihilator that destroys them.

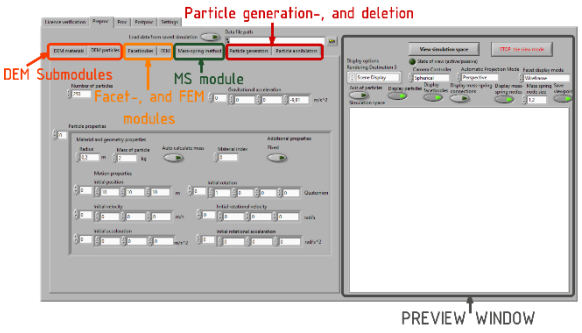


Figure 3: The Preprocessor of BMEDEM3D

The Processor

Whereas the preprocessor is designed for handling the mechanical objects being simulated, the processor's purpose is to manage the solver, and to visualise an already running simulation, like in Figure 4. It is where settings like the timestep, export time intervals and

multithreading are located. There are currently three ways of running a simulation: single-threaded (slow but very stable), OpenMP-multithreaded (fast but unstable) and a custom, thread pool solution (slightly slower but more stable, although unfinished). The processor is also where the logging option can be turned on, which – if enabled – exports all time and solver configuration data into a CSV file upon the simulation's end.

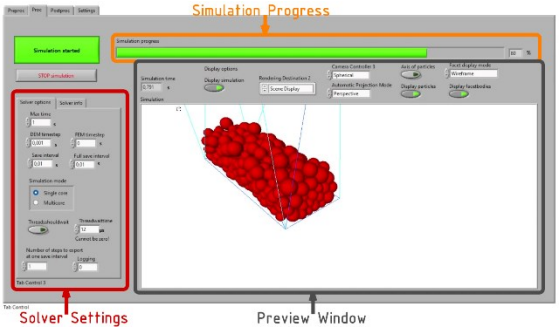


Figure 4: The Processor Mid-Simulation

The Postprocessor

The postprocessor is meant for when a simulation has already been run. It allows users to view the simulation space and visualise particle properties. It can also be used to make charts about specific properties, as shown in Figure 5.

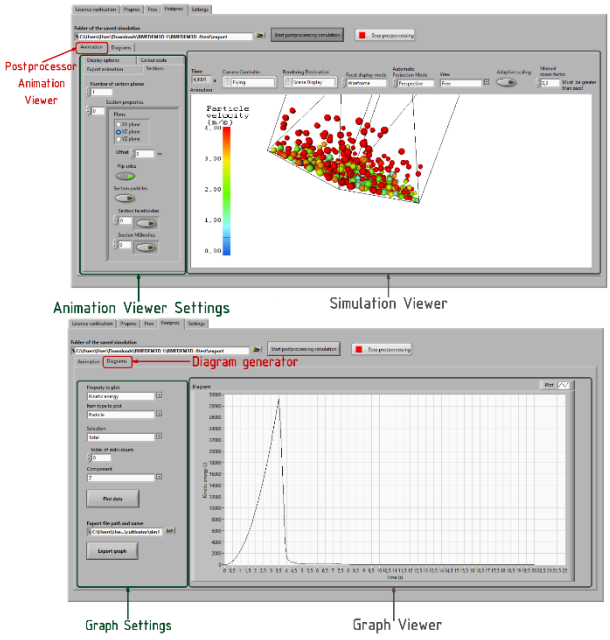


Figure 5: The Postprocessor of BMEDEM3D

The BMEDEM3D Solver

The solver consists of the main core, which handles the DEM simulation, as well as many optional subsystems, which can be enabled or disabled.
These subsystems are the following:

Facetbodies: It can import 3D surfaces as bodies to use in other systems. Can be box based, STL file based or manually defined.
Particle generators: Can create particles with given properties in a set 3D box defined in the above subsystem.
Particle annihilators: Can destroy particles that either enter or leave a set 3D box.
FEM simulator: The FEM simulations are toggleable. They rely on a 3D STL-based facetbody, as well as precalculated matrices, exported from Ansys Mechanical APDL (Pásthy et al., 2024)
Mass spring simulator: Just like FEM, this system is toggleable. It relies on 3D facetbodies, but every piece of data can be defined in the GUI without needing to use any other piece of software.
Logging: exports basic data (time and simulation conditions) in the form of a CSV file when the simulation ends. If the solver is run by itself in the terminal, it also displays information there while the simulation is running.
Trackers: lets the user track certain properties a specific particle, or FEM node. All data gathered by the trackers is exported in one, large CSV file when the simulation finishes. It is not implemented into the GUI yet.

Optimizing the Solver

Making certain parts of the solver toggleable is only one of the choices made in the design of the program to decrease the time needed to finish a simulation.
One of these choices was the development of a custom, built-in mathematical library for handling calculations. Originally, these were handled by the Eigen library (Mouton, 2009), which – while powerful – was not specialised and extensible enough for the program to run efficiently. The new solution was designed to make full use of the C++ compiler, generating custom code for specific use cases, to run faster while having minimal overhead. For example, the implemented matrix class determines during compile time to use a faster, smaller matrix that stores its data on the stack, or – when that's not possible – put its data on the heap, while still being considered part of the original class, making interfacing between types automatic.
Another optimisation is changing the collision detection between DEM particles. The so-called brute force method involves checking every possible particle combination to see if they collide. With higher numbers of particles, this would include large numbers of useless calculations, and for every N particles, N^2 operations must be performed. To decrease the number of unnecessary operations, a modified version of the Cell Linked List (CLL) model was implemented (Mattson & Rice, 1999). In this model, the simulation space is split into a grid of cubic cells. Every simulation step, particles are assigned to a cell based on their coordinates using spatial hashing (Eitz & Lixu, 2007). Collisions can be checked individually for every cell, meaning that particles far away from each other will never be checked for collisions. An advantage of this is that it requires far

less calculations for higher particle counts, but if there are fewer particles, the overhead can decrease performance.
Using these optimizations, as well as many other, smaller ones, the required time for simulations has dropped by several orders of magnitude, although it's not at the same level as other DEM software yet, as shown in Figure 6.

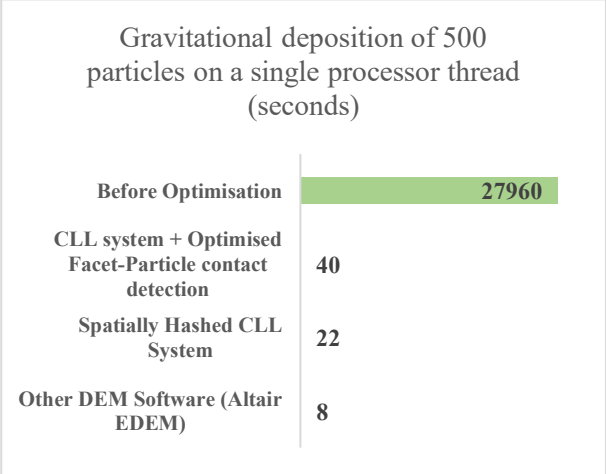


Figure 6: Benchmarking the Simulation Time in BMEDEM3D, and Comparing it to existing DEM Software

Another way of decreasing the time required for simulations to run is to parallelize them. Modern computer processors are equipped with several cores, meaning that tasks can be split into chunks, with each core doing different parts of these tasks. BMEDEM3D has two different implementations of this: OpenMP as well as a custom thread pooling system. OpenMP is a popular extension of the C/C++ and Fortran programming languages, which allows for macros to be used which makes the compiler automatically parallelise the specified functions or loops that it is applied to. It is already used in simulation software, for example, Yade (Yet Another Dynamic Engine), a popular FOSS DEM program is built on the OpenMP framework (Vaclav Smilauer et al., 2021). It is relatively easy to use, and very powerful, however, its reliance on the C++ preprocessor to generate code independently from the programmer means that debugging bugs and data races can be difficult. This is the reason why an alternative implementation also exists, one made specifically for BMEDEM3D. It uses so-called thread pools and splits up tasks into equal segments. For example, during DEM simulations, the particle array is split into several sectors, each sector being simulated by a different thread. To avoid data races, a technique called double buffering is used. Double buffering means that there are two copies of every array: one being read from, and one being written to. Because reading unchanging data does not cause races, and every value in the write buffer only being changed by the one thread it is assigned to, data races do not occur. This implementation is more stable, and easier to debug than the OpenMP solution, however, both the thread pool and double buffering have a significant amount of overhead both in memory-, and processor usage.

EXAMPLE SIMULATION

As an example of the capabilities of BMEDEM3D, a combined DEM-MS simulation will be demonstrated. During this, a mass-spring sheet, constrained at its four corners has 500 DEM particles fall on it. The parameters of these particles can be seen in Table 5.

Table 5: The DEM Particles' Parameters in the Example Simulation

Radius	$[0{,}15; 0{,}24]$ m (Linear Distribution)
Density	$2500\ \text{kgm}^{-3}$
Young's modulus	$2 \cdot 10^7$ Pa
Poisson's ratio	0,3
Sliding Friction Coefficient	0,2
Rolling Friction Coefficient	0,3
Coefficient of Restitution	0,5

The sheet is an STL file of a rectangular cuboid, whose body was subdivided to increase the number of facets. Its most significant properties are defined in Table 6.

Table 6: The Properties of the Mass Spring Sheet

Size	$1 \times 1 \times 0{,}2$ m
Mass	500 kg
Distance between nodes	0,1 m
Stiffness	$2 \cdot 10^4\ \text{Nm}^{-1}$
Damping	$1000\ \text{Nsm}^{-1}$

The simulation involves randomly creating 500 of the DEM particles in a $1 \times 1 \times 2\ \text{m}^3$ area, 2 m above the sheet. The sheet's four corners are fixed by bounding spheres with a radius of 0,1 m. All objects begin with an initial velocity of zero. The simulation has a timestep of $10^{-4}\ s$ and the time span of the simulation is 20 seconds. The simulation was run under the following conditions:

Processor: Intel Core i7 12700H (2,8 GHz)
Available RAM: 32 GB
Simulation Mode: Single threaded

Results of the DEM-MS simulation

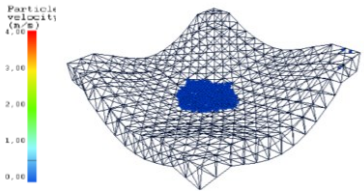


Figure 7: Postprocessor View of the Example Simulation

The simulation was stable, with particles either having bounced off the sheet, or entered a quasi-stable state on it, as seen in Figure 7.. The main concern, however, was its memory usage shown in Table 7, which suggested that the solver, specifically the mass spring subsystem has a memory leak.

Table 7: Computational Results of the DEM-MS Simulation

Time elapsed	8 minutes 50 seconds
Maximum RAM usage	11,5 GB
CPU Frequency Interval	$[2{,}8 - 3{,}6]$GHz

CONCLUSIONS

In this paper, the discrete element, finite element, and mass spring methods of simulating solid materials was introduced. The goal of creating a software which natively supports all three methods, and lets the user easily configure them using a powerful Graphical User Interface was successful, with BMEDEM3D achieving them. The software is in its early pre-alpha stages, with plans in the future including further optimizations to the solver, as well as a new GUI made in a 3D game engine for better graphics and performance for displaying simulations.

ACKNOWLEDGEMENTS

This paper was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. 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, financed under the TKP2021-NVA funding scheme. This research was supported by the ÚNKP, 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 (NKFIH FK-146067). 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 and Innovation and the Budapest University of Technology and Economics, under a grant agreement with the National Research, Development and Innovation Office. The publication of the work reported herein has been supported by ETDB at BME.

REFERENCES

Bathe, K.-J. (2008). Finite Element Method. In *Wiley Encyclopedia of Computer Science and Engineering* (pp. 1–12). John Wiley & Sons, Ltd. https://doi.org/10.1002/9780470050118.ecse159

Cundall, P. A., & Strack, O. D. L. (1979). A discrete numerical model for granular assemblies. *Géotechnique*, *29*(1), 47–65.

Eitz, M., & Lixu, G. (2007). Hierarchical spatial hashing for real-time collision detection. *IEEE International Conference on Shape Modeling and Applications 2007 (SMI'07)*, 61–70.

Kozicki, J., & Donzé, F. V. (2009). YADE-OPEN DEM: An open-source software using a discrete element method to simulate granular material. *Engineering Computations*, *26*(7), 786–805. https://doi.org/10.1108/02644400910985170

Mattson, W., & Rice, B. M. (1999). Near-neighbor calculations using a modified cell-linked list method. *Computer Physics Communications*, *119*(2–3), 135–148.

Mouton, C. (2009). *Linear Algebra Libraries* (p. 35)

Nikishkov, G. P. (2004). Introduction to the finite element method. *University of Aizu*, 1–70.

Pásthy, L., Farkas, Z. J., Haba, T., & Tamás, K. (2024). Development of a multi-way coupled discrete-finite element method simulation procedure for modelling soil-passive vibration tool interaction. *Computers and Electronics in Agriculture*, *216*, 108459.

Pasthy, L., Graeff, J., & Tamás, K. (2022). Development Of A 2D Discrete Element Software With LabVIEW For Contact Model Improvement And Educational Purposes. *ECMS*, 203–209.

Pásthy, L., & Tamás, K. (2023). *Modeling the Soil-Tool-Root or-Stem Interaction with Coupled Discrete Element and Mass-Spring Methods*.

Pieczywek, P. M., & Zdunek, A. (2017). Compression simulations of plant tissue in 3D using a mass-spring system approach and discrete element method. *Soft Matter*, *13*(40), 7318–7331.

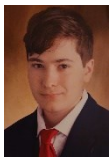
Provot, X. (1995). Deformation constraints in a mass-spring model to describe rigid cloth behaviour. *Graphics Interface*, 147–147.

Vaclav Smilauer, Angelidakis, V., Catalano, E., Caulk, R., Chareyre, B., Chèvremont, W., Dorofeenko, S., Duriez, J., Dyck, N., Elias, J., Er, B., Eulitz, A., Gladky, A., Guo, N., Jakob, C., Francois Kneib, Kozicki, J., Marzougui, D., Maurin, R., … Yuan, C. (2021). *Yade documentation*. https://doi.org/10.5281/ZENODO.5705394

Zhang, Y., & Shi, X. (2014). Research on the three-dimensional displaying of STL ASCII and binary file. *Advanced Materials Research*, *940*, 433–436.

Zienkiewicz, O. C., Taylor, R. L., & Zhu, J. Z. (2005). *The finite element method: Its basis and fundamentals*. Elsevier.

AUTHOR BIOGRAPHIES



JÓZSEF BALÁZS GRÄFF is a BSc mechatronics engineering student at the Budapest University of Technology and Economics. His professional field is low level and high-performance computing in the C and C++ programming languages. His e-mail address is graff.jozsef@gszk.bme.hu

LÁSZLÓ PÁSTHY is a PHD student of mechanical engineering at the Budapest University of Technology and Economics where he received his BSc and MSc degrees. His field of research is coupled discrete element method (DEM) and finite element method (FEM) simulations. His e-mail address is pasthy.laszlo@gt3.bme.hu and his personal webpage is https://gt3.bme.hu/oktatoi_oldal.php?lepes=4&oid=190

KORNÉL TAMÁS is an assistant professor at the Budapest University of Technology and Economics where he received his MSc and PHD degrees. His field of research is the modelling of granular materials using the discrete element method (DEM). His e-mail address is tamas.kornel@gt3.bme.hu and his personal webpage is https://gt3.bme.hu/oktatoi_oldal.php?lepes=4&oid=162