The 5th Joint International Conference on Multibody System Dynamics
June 24 – 28, 2018, Lisboa, Portugal

partsival – Collision-based Particle and many-body Simulations on GPUs for Planetary Exploration Systems

Roy Lichtenheldt[1], Simon Kerler[1,2], Andreas Angerer[3] and Wolfgang Reif[2]

[1]*Institute of System Dynamics and Control, German Aerospace Center (DLR), roy.lichtenheldt@dlr.de, contact@simonkerler.de*
[2]*Institute for Software & Systems Engineering, University of Augsburg, reif@informatik.uni-augsburg.de*
[3]*XITASO GmbH, andreas.angerer@xitaso.com*

ABSTRACT — *Nowadays simulations are crucial for timely development and cost reduction. However, large scale particle simulations are still time-consuming, as most modern simulation frameworks are CPU (Central Processing Unit) bound and thus limited in terms of parallelization. Massive parallelism would require the use of clusters or supercomputers. In this paper the GPU (Graphics Processing Unit) based massively parallel simulation framework partsival is introduced for Discrete Element Method (DEM). It targets realistic penalty based contacts on workstation computers to enable simulation and design engineers, to simulate complex particulate systems. Therefore partsival features special algorithms for implicit time integration as well as boundary conditions to further speed up the simulation. The article will show the scaling and performance compared to modern parallel CPU code, as well as the validation of the results. Another focus is on the framework architecture and usability. Finally real applications from the usage in aerospace engineering at DLR will be given.*

1 Introduction

Developing new or optimizing existing locomotion systems for planetary exploration faces two major challenges: First, environmental conditions like reduced gravity are hard to mimic in laboratories. Second, physical prototyping can be time-consuming and cost-intensive. To address both, virtual prototypes are simulated to verify new designs and concepts, e.g. for rover wheels, before first physical prototypes are constructed. Often a method to model soil is required to correctly simulate tool interactions. A common way is the Discrete Element Method (DEM) which discretizes soil volumes into particles. The software currently used for DEM computations at DLR is a CPU-bound, multi-threaded engine. Due to its high degree of parallelism, the DEM profits from being executed on graphics processing units (GPUs). In order to make absolute statements and predictions, high accuracy is required and therefore the penalty based contact formulation is

preferred, as it will be explained in section 2.1. However, most of the DEM implementations for GPUs employ constraint based, hard contact formulations [1], e.g. Project Chrono [2]. Therefore, this paper presents partsival – a collision-based particle simulation framework targeting graphics cards [1].

To allow for integration into the existing visualization software DLR SimVis [3], which is based on OpenSceneGraph (OSG) [4], the library is provided as a so called NodeKit, a plugin, extending OSG. All relevant computations for numerical integration, inter-particle and particle-mesh contacts are executed on the GPU using OpenGL as rendering and compute framework to ensure vendor independency, to integrate natively into OSG and to avoid context switches and synchronization.

In this article, first the state of the art is shown and it is stated why partsival is not simply yet another engine. Second, partsival's architecture is covered, followed by an explanation of the underlying physics and numerics. The correctness of the engine and its contact models is verified by comparing simulation results of partsival to those of reference implementations in existing DEM frameworks. Furthermore, the performance gain to CPU bound code is evaluated with more detailed profiling results highlighting computational bottlenecks. Finally, simulations from engineering and science are given proving partsival's applicability to real-world problem domains.

2 State of the Art

This section presents the state of the art for DEM simulations. At first, the Discrete Element Method is introduced and existing frameworks are evaluated. Afterwards, the basic concepts of GPU computing are covered which are required for the understandig of the rest of this article.

2.1 Discrete Element Method

The DEM represents granular matter as discrete particles. Each particle corresponds to a volume of grains rather than a single grain due to computational complexity [5]. Commonly, spherical particles are used, allowing for efficient contact detection and force calculations, but more complex geometries are possible as well [6].

Within a simulation, a spherical particle i can be represented by a set of (time dependent) state variables including its radius r^i, mass m^i, inertia $\vec{I}^i$, position $\vec{x}(t)^i$, velocity $\dot{\vec{x}}(t)^i$, rotation unit quaternion $\underline{q}(t)^i$ and angular velocity $\vec{\omega}(t)^i$. Note that some contact models require additional attributes, e.g. initial contact points between particles [7]. From these variables, particle dynamics can be computed by numerically integrating NEWTON and EULER equations:

$$m^i \ddot{\vec{x}}^i = \sum_{j=0}^{n} \vec{F}_{\mathrm{C}}^{ij} \tag{1}$$

$$\vec{I}^i \cdot \dot{\vec{\omega}}^i + \vec{\omega}^i \times \vec{I}^i \cdot \vec{\omega}^i = \sum_{j=0}^{n} \vec{T}_{\mathrm{C}}^{ij} \tag{2}$$

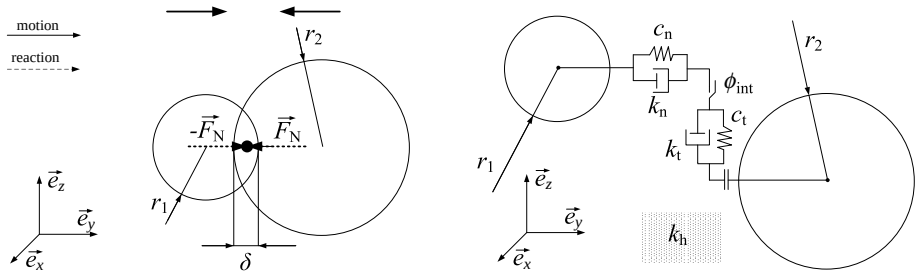

Fig. 1: Overlapping contact between particles [8] (left) and the underlying penalty based particle contact model in normal and tangential direction (right) [5].

This requires to evaluate force $\vec{F}_C^{ij}$ and torque $\vec{T}_C^{ij}$ which are provided by contact models and the correspondent contact detection algorithms. Depending on the use case and simulated material, different contact models can be applied to calculate forces and torques exerted on a particle [7]. One possibility are hard particles which employ contact constraints, with a constraint solver resolving violations in each iteration of the simulation loop. As shown in [9] this approach has improved computational efficiency for low numbers of iterations. However, if sufficient accuracy for absolute prediction of forces and torques is required, constraint contact models tend to perform similar to penalty based models or worse. These considerations from [9] are well in line with our experience.

Thus in partsival, particles are considered soft bodies which can be deformed based on their stiffness. Thereby particles are modeled as rigid spheres and deformation is represented by overlapping contacting particles. Fig. 1 depicts this concept as KELVIN Elements. Normal contact forces can be calculated using HERTZIAN contact laws whereas COULOMB friction determines tangential forces. From the latter one, torques are obtained necessary to solve Equation 2.

Besides particle-particle contacts, most simulations require to compute tool interactions with granular matter. A common way of representing objects in computer graphics is by means of triangulated surface meshes which reduces particle-mesh contacts to sphere-triangle interactions with simpler intersection tests [10].

As mentioned above, DEM systems are numerically integrated over time which is due to the non-linearity and discontinuity of underlying ordinary differential equations [11]. Implicit time integration methods like the predictor-corrector integrator NEWMARK-β [12] are preferred over explicit ones [10] since they provide improved system stability and allow for larger time step sizes. The main disadvantage of implicit integrators compared to explicit schemes is that each corrector iteration requires the evaluation of contact detection and contact models, which is an issue for GPU based frameworks as will be explained later. This problem may be overcome using the force-predictive implicit integration scheme from [13].

2.2 DEM Simulation Frameworks

There are numerous DEM simulation frameworks available with five of them being evaluated here: LIGGGHTS [14], YADE [15], EDEM [16], Project Chrono [2] and Pasimodo [10]. Almost all of them provide predefined contact models, (explicit) integration schemes and extension points to add missing functionality, e.g. implicit integrators. Pasimodo and YADE are CPU bound by using the

message passing interface (MPI) and OpenMP for parallelization. EDEM, LIGGGHTS and Project Chrono provide GPU acceleration through OpenCL or Nvidia CUDA with disadvantages covered in the next section. Project Chrono, however, supports GPU computing only for hard contacts. EDEM is designed to be used with its own GUI and its proprietary nature prevents being natively integrated into OSG applications. An additional requirement imposed on partsival is to select floating point precision on simulation startup. This is not supported by these frameworks since precision is either fixed at compile time of the engine or of the simulation or cannot be changed at all. Regarding software architecture, these frameworks feature a large number of components and abstraction layers. The target user group at DLR however are engineers rather than software developers and therefore complex class hierarchies are more hindering than helpful. Additionally some frameworks do not provide plugin interfaces, which requires access and manual alterations to the library source code.

In consequence of the above discussion, it has been decided to create a custom GPU based, lightweight DEM framework tailored to the requirements imposed by the simulations at DLR.

2.3 GPU Computing

Today's graphics hardware features a hierarchical architecture tailored to GPU computing [17, 18]. For this introduction, Nvidia's Compute Unified Device Architecture (CUDA) is used as a reference. Furthermore, note that only compute relevant components are covered and rendering related ones are omitted.

At top level, a Nvidia GPU is divided into graphics processing clusters (GPCs), each containing a set of streaming multiprocessors (SM). Each SM contains a large number of CUDA cores, capable of integer and floating point arithmetic, load/store units for data access and special function units for e.g. square root operations. The memory model follows a similar hierarchy: Each CUDA core has its dedicated set of private registers. They also have access to the shared memory of their SM which allows for communication of threads executed on the same multiprocessor. On the highest level, L2 caches and memory controllers allow to access the graphics card's main memory.

The programming paradigms of available computing frameworks are tightly coupled to the GPU architecture. Code executed on the GPU is programmed in compute shaders[1] which define the control flow of a single thread. When dispatched to the graphics card, multiple instances of this thread are created called work items. Each work item belongs to a local work group which are further subsumed into one global work group. Work items are scheduled in groups of 32[2] called warps executing code in lockstep. Therefore, local work group size should be a multiple of the warp size to increase occupancy. The optimal size however depends on the GPU and should be tweaked to maximize throughput.

All work items within a work group execute the same instruction on different data, shifting the focus from "what should each thread do" to "on which data should each thread operate" [1]. Although very similar to single instruction multiple data (SIMD), this concept is called single

[1] Note that this article uses the nomenclature of OpenGL. Concepts however are also valid for other frameworks.
[2] 64 in case of AMD GPUs.

instruction multiple thread (SIMT) to underline the multi-threaded execution of work items. Since all work items have to execute the same instruction, branching control flow should be avoided because each code path has to be run sequentially. When accessing shared or global memory, additional synchronization has to be applied in code to avoid write after write (WAW) or read after write (RAW) errors between work items.

From the available frameworks capable of GPU computing – Microsoft Direct Compute [19], OpenCL [20], Nvidia CUDA [21], Vulkan [22] and OpenGL [23] – the last one has been chosen for partsival due to several reasons. At first, it is available for all operating systems and grants independency from graphics cards vendors. In contrast to the relatively new Vulkan specification, it is well established with a stable API and plenty of documentation. Since on-line visualization is a major goal of the framework, interoperability between render and compute framework have to be considered as well. By using OpenGL for both, communication overhead and context switches are avoided since render shaders can directly access results from compute shaders. Furthermore, existing components of the OSG OpenGL abstraction layer can be reused allowing to integrate GPU computing natively into OSG programs.

3 Architecture

This section covers the software architecture of the simulation framework as well as the graphical user interface (GUI) and available programming models.

3.1 Simulation Framework

The general software architecture of partsival is depicted in Fig. 2. To add DEM capabilities to an OSG program, an instance of `ParticleSystem` is added to the scene graph. This main class is used to configure a simulation by defining attributes and behavior of particles. It also controls the simulation loop and rendering. At the moment, the number of particles within a system is fixed and cannot be changed dynamically at runtime.

Particle attributes are defined by providing an identifying name and the data type, e.g. scalar integers or vectors. The particle system internally creates a shader storage buffer object (SSBO) which allocates memory on the GPU. Its size is defined by the number of particles times the size of the data type. The buffer is returned allowing to initialize its content and therefore the attributes for each particle. A special method to create "dynamic" floating point vectors is provided which uses either single or double precision depending on the float precision hint set to the particle system. Since the OpenGL shading language (GLSL) is type safe, shaders have to know the chosen precisions as well. This is accomplished by using partsival's types (e.g. `psv_float`) in GPU code. When a shader is created, its source is loaded via the `ShaderPreprocessor` which resolves them to their native equivalents, e.g. `double` and `float` for double and single precision respectively. On newer graphics hardware also half precision is available, but omitted in partsival for numerical stability reasons.

Setting up volumes of particles and their attributes is a recurring step when building simulations. The preferred way to take over this task is to implement reusable `ParticlePackages` which can be applied to a `ParticleSystem`. partsival ships with packages suitable for the built-in contact models arranging particles in a face centered cubic (FCC) structure.

The simulation loop is defined by attaching `SimulationSteps` to the `ParticleSystem`. Each `SimulationStep` provides a single, reusable functionality, e.g. a contact detection model or an integrator. `SimulationSteps` have one or more `Runnables`, because multiple operations might be required, e.g. contact detection followed by force calculation. Since all computations are offloaded to the GPU, a runnable contains a compute shader and provides a simple interface for its execution, e.g. by controling the size of the global work group (cf. section 2.3). A specialized `IntervalRunnable` allows to call a shader only every n-th time step. On execution, a `SimulationStep` binds required attribute buffers to the OpenGL context to make them accessible for the compute shaders of its runnables. Their unique attribute names are used to determine the variables to which the buffers are bound. To avoid data races, each `Runnable` employs a memory barrier to synchronize writes with subsequent compute or render shaders. `SimulationSteps` and `Runnables` are invoked in the order they have been added to their respective parent object.

When the simulation loop has completed, the particle system is rendered. To do so, the user has to provide render shaders as well as the geometry of a single particle in form of a mesh. partsival ships with pre-defined renderers and geometry for spherical particles to get users started quickly. Note that at the moment, the geometry is used for visualization only and has no effect on contact detection and force evaluation. Instancing is used to efficiently display all particles with a single OpenGL draw call avoiding massive CPU overhead [23]. Furthermore, render shaders have direct access to the SSBOs holding particle attributes computed previously. Therefore, no context switch or data copying impacts performance.

For post-processing of simulation results, it is necessary to persist the system state. partsival provides an interface to define particle attribute serializers which can be added at arbitrary points

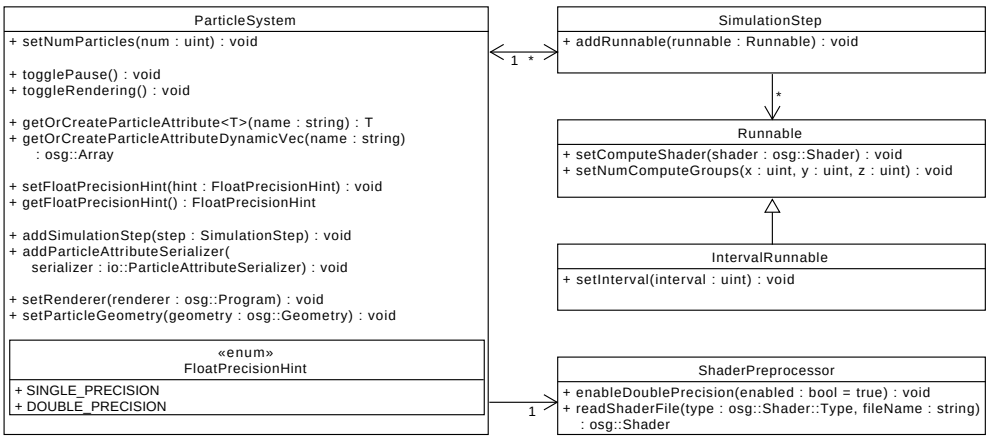

Fig. 2: The main classes of partsival's software architecture.

of the simulation loop. To limit output file sizes and impact on performance (since data has to be copied from GPU to CPU), an interval can be specified to dump the state only every n-th time step. Two built-in serializers are provided, one printing to the standard output device and one writing to file in H5Part format [24] which allows to load and visualize results in ParaView [25].

3.2 User Interface & Programming Models

Besides high performance and accuracy, partsival has to be easy to use to efficiently support engineers' day-to-day development flow. Therefore another main focus has been usability and user experience resulting in two major paths that are closely entwined: A graphical user interface (GUI) and a highly abstracted C++ layer for interfacing to the main library of partsival.

The first allows to specify a model and simulation by simply calling functions and creating instances of objects – both concepts well known from common numerical tools in CAE and numerics. The API is documented using doxygen and example models provide insight to all major functions and features. These "programmed" models allow for high degrees of automation, variation and even optimization needed in modern development. In order to allow to create simpler models without any programming knowledge at all, the GUI partsivalQT (Fig. 3) features the main functionality, too. However, its main focus is not direct modeling but creating a more shallow learning curve towards programming models via C++. Therefore it provides an auto code generation feature, that records every GUI action and writes the correspondent code in a ready to compile and ready to use model. Thus the user is able to understand which GUI-actions refer to the code that has to be used to automate this part of the model. So far this concept has proven to be a great success.

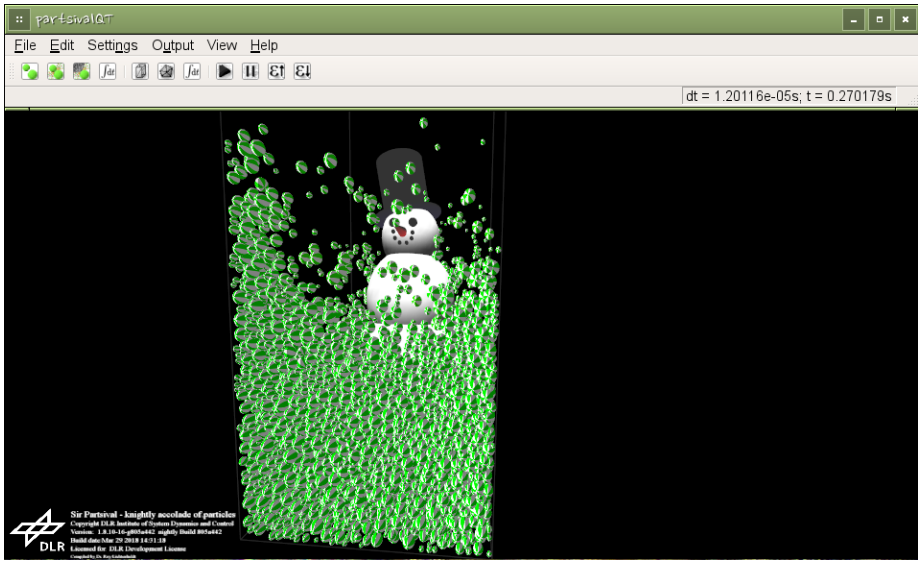

Fig. 3: partsivalQT - the graphical interface to partsival. The example shows a dynamic snowman dropping into particles and has been fully implemented using the GUI.

Besides mentioned interfaces, partsival features a full python wrapper "pysival" to allow for scripted modeling. For the input of parameters in both C++ and python a simple json parser is used and also features a routine for automatic JSON input file creation in external variational analysis and optimizations.

4 Physics, Modeling and Solver

In this section the physics modeled in partsival as well as the boundary conditions and numerical algorithms will be explained. partsival features several ideas from recent research well ahead of the state of the art. These special features of the simulator will be highlighted.

4.1 Contact Detection

As partsival is developed mainly for dense particle packings, where neighborhoods persist over several time steps, the neighboorhood search is implemented using a broad and a near phase. The first one has been designed to create a VERLET list [26]. The VERLET list has been chosen as a memory saving and yet still efficient algorithm. Building the list is implemented as a separate `IntervalRunnable` since the algorithm is of complexity $\mathcal{O}(n^2)$. It is planned to include several more broad phase algorithms to improve performance for certain simulation cases by lowering complexity to $\mathcal{O}(n \cdot \log(n))$. So far, contact detection is improved as every particle searches for neighbors with higher IDs only. Thus contacts may later be calculated per contact pair rather than per particle. However, this requires special routines to ensure that memory conflicts and race conditions due to lockstep execution on the GPU are avoided. The near phase is implemented in the contact models themselves which allows for overlap based forces and attractive forces without alteration of the framework. The VERLET radius has to be chosen carefully, to reduce branching control flow in the consecutive step: If only one particle of a work group detects contact in near phase, all non contacting particles in this work group have to wait for completion of the full contact model evaluation, too.

4.2 Contact Models and Boundary Conditions

The contact models included in partsival are currently mainly focused on the simulation of sandy and cohesive soils. The main model is therefore a combination of HERTZian contact in normal direction and regularized stick-slip friction in tangential direction. The normal force $\vec{F}_{\mathrm{N}}^{ij}$ between particles i and j for one time step evaluates to [5]:

$$\vec{F}_{\mathrm{N}}^{ij} = \left(\frac{2E}{3(1-\nu^2)} \sqrt{r_{\mathrm{C}}^{ij} |\vec{\delta}^{ij}|^3} \right) \vec{n}_c^i + k_{\mathrm{Nmin}}^{ij} \dot{\vec{\delta}}^{ij} \tag{3}$$

$$k_{\mathrm{Nmin}}^{ij} = D \cdot \frac{4}{3} \cdot \sqrt{\frac{\rho_{\mathrm{P}} E \pi \cdot \min(r^{i,j})^4}{1-\nu^2}}; \quad \forall D \in [0.1, 0.3] \tag{4}$$

with $E, \nu, \vec{\delta}^{ij}, r_{\mathrm{C}}^{ij}$ and $\vec{n}_c^i$ being the YOUNG's modulus, POISSON ratio, inter-particle overlap, the contact radius and the contact normal vector respectively. The damping factor k_{Nmin}^{ij} is evaluated for the smaller of the two particles using LEHR's damping fraction D, particles density ρ_{P}, particle radius r^i and the maximum assumed overlap δ_0. To calculate a physical damping value, the user has to enter LEHR's damping fraction only. The frictional force $\vec{F}_{c\mathrm{T}}^{ij}$ evaluates to [8]:

$$\begin{aligned}
\vec{F}_{c\mathrm{T}}^{ij} &= c_{\mathrm{T}}^{ij} \cdot \vec{\delta}_{\mathrm{T}}^{ij} \cdot \mathrm{sign}\left(\vec{\delta}_{\mathrm{T}}^{ij} \cdot \dot{\vec{\delta}}_{\mathrm{T}}^{ij}\right) + k_{\mathrm{T}}^{ij} \cdot \dot{\vec{\delta}}_{\mathrm{T}}^{ij}; \quad \vec{F}_{\mathrm{hT}}^{ij} = |\vec{F}_{\mathrm{N}}^{ij}| \cdot \tan(\phi_{\mathrm{h}}) \\
\vec{F}_{\mathrm{T}}^{ij} &= \begin{cases} \vec{F}_{c\mathrm{T}}^{ij} & \forall\, |\vec{F}_{c\mathrm{T}}^{ij}| \leq \vec{F}_{\mathrm{hT}}^{ij} \;\wedge\; |\dot{\vec{\delta}}_{\mathrm{T}}^{ij}| \leq v_{\mathrm{Tmin}}^{ij} \\ |\vec{F}_{\mathrm{N}}^{ij}| \cdot \tan(\phi_{\mathrm{g}}) \cdot (\dot{\vec{\delta}}_{\mathrm{T}}^{ij})_0 & \forall\, |\vec{F}_{c\mathrm{T}}^{ij}| > \vec{F}_{\mathrm{hT}}^{ij} \;\vee\; |\dot{\vec{\delta}}_{\mathrm{T}}^{ij}| > v_{\mathrm{Tmin}}^{ij} \end{cases}
\end{aligned} \tag{5}$$

with $c_{\mathrm{T}}^{ij}, \vec{\delta}_{\mathrm{T}}^{ij}, k_{\mathrm{T}}^{ij}$ and $\phi_{\mathrm{g,h}}$ are the tangential stiffness, deformation of the tangential KELVIN element, tangential damping and the angle of slipping and sticking friction respectively. The regularization velocity v_{Tmin}^{ij} is used to allow for stable sticking. For rolling resistance, the simple rolling friction model developed in [7] is used to provide comparability to other frameworks. For more complex analysis, partsival also features the tilting contact model proposed in [5, 27]. The parameters of the contact models are determined from real material values using the method proposed in [8]. This procedure is mostly integrated into partsival to ease the parametrization for the user. The main particle-particle contact is implemented using one `IntervalRunnable` and two `Runnables`. The first one generates the VERLET-List followed by a preparation step, copying the contacts from the previous time step. Computation is concluded by the near-phase contact detection and contact force calculation within the last `Runnable`.

Additional contact models include:

- cohesion forces,
- persistent forces to clump particles,
- and fully user defined plugin contact models.

Thereby the plugin models are written as simplified compute shaders – the user only needs to pass the equations needed, without caring about variable availability. Hence all the interfacing overhead is taken away from the user who only has to define the equations to calculate the forces/torques and thus accelerations respectively. Additional force models may be implemented as `Runnable` or `IntervalRunnable` dependent on the users choice.

partsival provides state of the art smooth and rough boundary conditions for DEM (see [28]). In order to speed up simulations, symmetry conditions [28] as well as rheonomic and dynamic boundary conditions are implemented. The latter two are rough boundaries moving in $\mathbb{R}^3$. The difference between these two is that rheonomic conditions move at predefined velocity profiles whereas dynamic boundaries move exactly according to the tool's path. Thereby the tool is allowed to move with 6 DOF and full dynamics. Using this class of moving boundaries, a set of boundary particles is fixed at 0 DOF around the influence region. Particles outside of this region are deactivated and thus

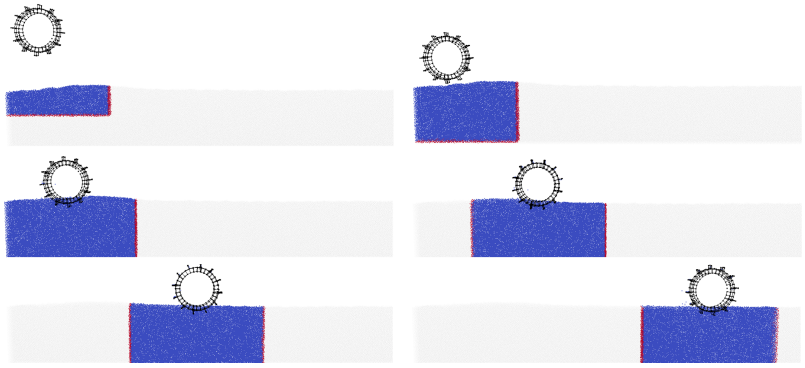

Fig. 4: Example of dynamic boundaries in a 3D wheel simulation. Blue particles are full 6 DOF, while red particles are locked at 0 DOF and translucent grey particles are excluded from simulation. For details on the algorithm see [5, 28].

not part of the calculations, but are retained in GPU memory to avoid GPU-CPU communication. The different regions of particles are determined by membership functions shown in detail in [28]. Fig. 4 shows the procedure and different particle regions. For recent GPUs, dynamic boundaries yield enormous performance gains if the influence region includes at least 20000 particles to maximize the GPU utilization. Thus these boundaries are especially worthy for multi- and large-scale simulations.

4.3 Time Integration

As it has been mentioned in section 2.1, implicit integration schemes are favourable in DEM simulations. However they do not feature the ease of implementation of explicit schemes. Implicit schemes are widely implemented as predictor corrector algorithms. Especially for GPU computation, the renewed force calculation in the corrector loop causes problems. In partsival, renewed contact detection and force calculation would not be possible without either sacrificing performance or extensibility. Thus the force-predictive LICHTENHELDT-jolt integration scheme proposed in [13] is adopted. However, experience showed that the multi-step BEEMAN scheme known from molecular dynamics [29] of order 4 yields better results than the previously used NEMARK-β scheme. Thus the LICHTENHELDT-jolt-BEEMAN scheme predictor yields:

$$\ddot{\vec{x}}_0^{jl}(t+\Delta t) = \ddot{\vec{x}}(t) + \int_t^{(t+\Delta t)} \left.\frac{\mathrm{d}^3\vec{x}}{\mathrm{d}t^3}\right|_{(t-\Delta t)}^{t} \mathrm{d}t \tag{6}$$

$$\ddot{\vec{x}}_0^{r}(t+\Delta t) = \left(\ddot{\vec{x}}_0^{jl}(t+\Delta t) + \Delta t \cdot \mathrm{sign}\left(\dot{\vec{x}}_0^{jl}(t+\Delta t) \circ \dot{\vec{x}}_m(t)\right) \circ \int_{(t+\Delta t)}^{(t+2\Delta t)} \left.\frac{\mathrm{d}\ddot{\vec{x}}_0}{\mathrm{d}t}\right|_{t}^{(t+\Delta t)} \mathrm{d}t\right) \tag{7}$$

$$\ddot{\vec{x}}_0^{c}(t+\Delta t) = (1-\alpha)\cdot\ddot{\vec{x}}_0^{jl}(t+\Delta t) + \alpha\cdot\ddot{\vec{x}}_0^{r}(t+\Delta t); \quad \forall\,\alpha\in[0,1] \tag{8}$$

$$\dot{\vec{x}}_0^{c}(t+\Delta t) = \dot{\vec{x}}_m(t) + \kappa^{-1}\cdot\Delta t\left(7\cdot\ddot{\vec{x}}_0^{c}(t+\Delta t) + 6\cdot\ddot{\vec{x}}_m(t) - \ddot{\vec{x}}_m(t-\Delta t)\right); \kappa = 24 \tag{9}$$

$$\vec{x}_0^{c}(t+\Delta t) = \vec{x}_m(t) + \Delta t\cdot\dot{\vec{x}}_m(t) + \kappa^{-1}\cdot\Delta t^2\left(3\cdot\ddot{\vec{x}}_0^{c}(t+\Delta t) + 10\cdot\ddot{\vec{x}}_m(t) - \ddot{\vec{x}}_m(t-\Delta t)\right); \kappa = 24 \tag{10}$$

with α being the weight between left and right handed accelerations. Using this implicit prediction, the semi-implicit corrector loop is then calculated as follows:

$$\ddot{\vec{x}}_n^c(t+\Delta t) = \ddot{\vec{x}}_m(t) + \xi \int_{\vec{x}_m(t)}^{\vec{x}_{n-1}(t+\Delta t)} \left(\nabla \circ \ddot{\vec{x}} \Big|_{\vec{x}_m(t-\Delta t)}^{\vec{x}_m(t)} \right) \mathrm{d}\vec{x}; \quad \forall \xi \in [0,1] \tag{11}$$

$$\vec{x}_n(t+\Delta t) = \vec{x}_m(t) + \Delta t \cdot \dot{\vec{x}}_m(t) + \kappa^{-1} \cdot \Delta t^2 \left(3 \cdot \ddot{\vec{x}}_n^c(t+\Delta t) + 10 \cdot \ddot{\vec{x}}_m(t) - \ddot{\vec{x}}_m(t-\Delta t) \right); \kappa = 24 \tag{12}$$

with ξ being the weight for the force-law based part of the acceleration. As the velocity is not required for iteration, it is just calculated once outside of the loop in order to improve performance:

$$\dot{\vec{x}}_m(t+\Delta t) = \dot{\vec{x}}_m(t) + \kappa^{-1} \cdot \Delta t \left(10 \cdot \ddot{\vec{x}}_m^c(t+\Delta t) + 16 \cdot \ddot{\vec{x}}_m(t) - 2 \cdot \ddot{\vec{x}}_m(t-\Delta t) \right); \kappa = 24 \tag{13}$$

The corrector is stopped after a fixed number of iterations according to [30]. As per [13], the force prediction error can be evaluated after the time step has been completed with no cost, as the contacts have to be evaluated anyway. This error is then used to control the variable time step size. Rotational values are integrated accordingly. The time integration is implemented using two `Runnables`: The integration scheme itself and the evaluation of the force error for step size control. The control of the time step size is then carried out on the CPU.

4.4 Multi Body Interaction

"External" rigid bodies are modeled as meshed surfaces in partsival to simplify the complex contact situation of arbitrary shapes to the well defined contact between spheres and triangles.

The interaction is implemented as five `Runnables`/`IntervalRunnables`. At first, an axis aligned bounding box (AABB) is computed for each triangle which serves as input to the second step, a broad phase contact detection similar to the VERLET list of particle-particle contacts. Both are implemented in terms of `IntervalRunnables` to improve performance. Afterwards, the copy contacts `Runnable` (as described for particle-particle contacts) is invoked followed by the actual contact model including the near phase of contact detection. In order to calculate the resulting loads on external bodies, the forces and torques acting on each triangle are summed up at the end [10]. This is carried out in a parallel reduction operation to facilitate the full potential of parallel processing on the GPU. The summed loads are then transferred to the CPU as two compact $\mathbb{R}^3$ vectors.

As the movement of single external bodies has to be possible in partsival in order to perform stand alone single wheel simulations and such, it features a dedicated integrator for external bodies. For simplicity and performance the VERLET scheme [26] is used. Single degrees of freedom may be locked and/or the body may be moved rheonomically along user defined functions instead of evaluating its dynamics.

As complex mechanisms commonly employ special forces and constraint based joints, partsival does not aim to cover those. Instead a TCP-IP based co-simulation interface allows to couple it to other simulators directly or via a moderator setting an interfacing time step. This co-simulation, the contact models and boundary conditions implemented in partsival allow for simulations similar to the complex analysis on the CPU shown in [28, 31].

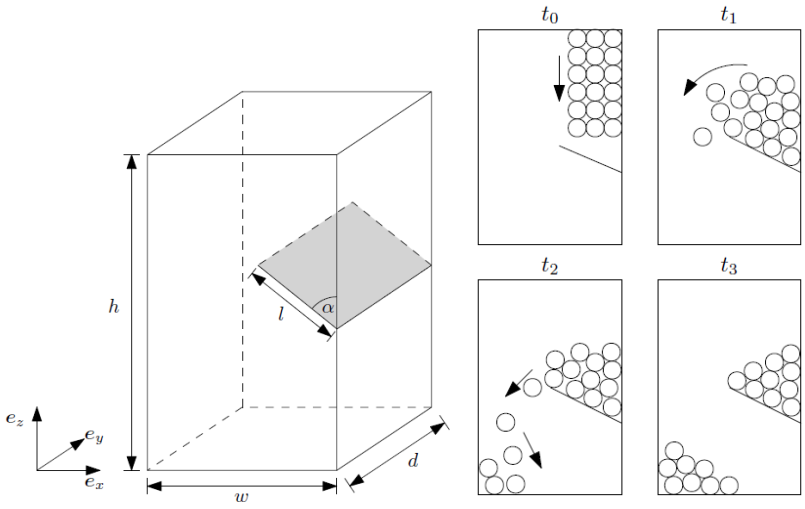

Fig. 5: Model for the benchmark: 3D view of the container (left) and principle effects during the piling process (right).

5 Benchmark and Validation

In this section, partsival is compared to well verified commercial CPU-bound DEM code in regard to performance and accuracy.

5.1 Benchmark Model

In order to allow for a fair comparison, a simulation scenario which contains dense as well as free flow states of granular matter is used. This ensures that the performance gain of partsival is not only due to the possibly longer intervals in between the renewed contact detection cycles. All contact models, boundary conditions and even the time integrator are identically implemented in both frameworks, so that performance increases are solely due to the code and hardware platform.

The scenario is a piling process in a closed container with an inclined plane attached to a wall (Fig. 5). At the beginning, particles are falling in a loose package (t_0) onto the plane where a first pile emerges (t_1). When the angle of repose is exceeded, particles are flowing down (t_2) and finally form another pile on the ground (t_3). With this scenario almost all states of granular material are covered: Free falling non-contacting particles, dense stationary packages and particulate flow. In order to assess the performance, the particle count is increased by simply using higher numbers of particle layers in the width d as shown in Fig. 5 on the left.

5.2 Validation

A first validation was carried out with a “bouncing two spheres” model to examine the energy conversation and numerical stability of the framework. In this model one dynamic sphere is falling and bouncing on top of another, fixed sphere. This check yielded great accordance between the

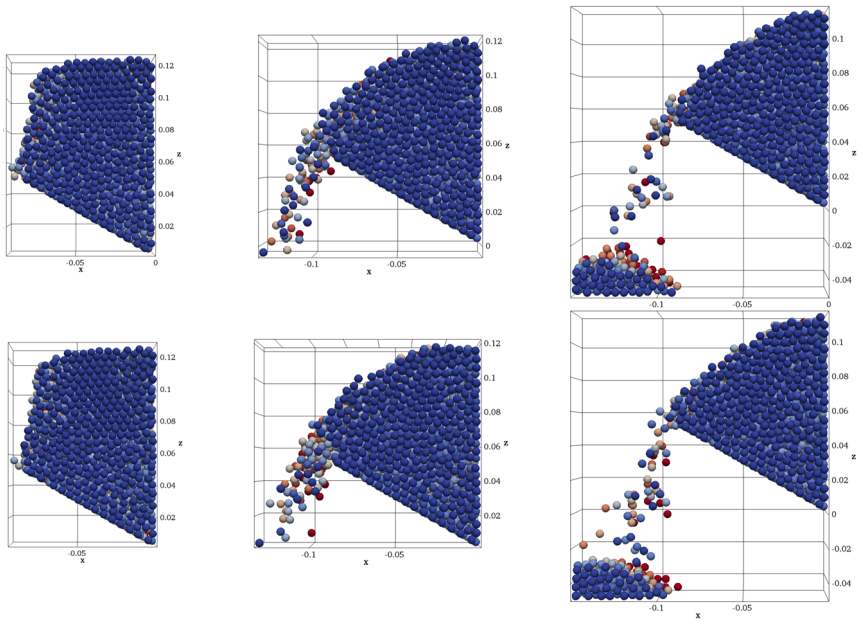

Fig. 6: States t_1 and t_2 ($t \in \{0.1\,\mathrm{s}, 0.18\,\mathrm{s}, 0.4\,\mathrm{s}\}$) of the accuracy benchmark between CPU-bound DEM (bottom) and partsival (top) both in double floating point precision.

partsival simulations, analytical calculations and CPU-code experiments. Quantitative comparisons are given in the following paragraphs.

Using the previously described piling benchmark model, the comparison of both simulators in Fig. 6 yields almost perfect resemblance of the CPU-reference simulation. As it can be seen, throughout the whole process both provide almost identical results. In order to allow for a direct comparison of the final state and thus the stationary angle of repose, Fig. 7 shows an overlay of both simulation states. Thereby the CPU-code is shown as translucent green particles, while partsival's particles are shown in colored opaque style. The angles are in good accordance: partsival yields an average angle of repose of 28.5° and the CPU-code yields 27.6° and thus partsival is within 3.3% of error. With the error range lower than 5%, partsival shows sufficiently high accuracy, as the errors are mostly resulting from CPU multiprocessing and implementation details.

It is noteworthy that the CPU code has some level of indeterministic behavior explained by the fact that distributing particles on different CPU cores will impose differences on microscopic level, making predictions for single particles indeterministic, while the macroscopic result stays deterministic. This results from the operating system and other processes influencing the scheduling, load control and instruction queue on the CPU (cf. [15]). Thus single particles may even end up in different positions in subsequent simulation runs. The models currently contained in partsival do not feature this behavior thus simulation runs have deterministic results.

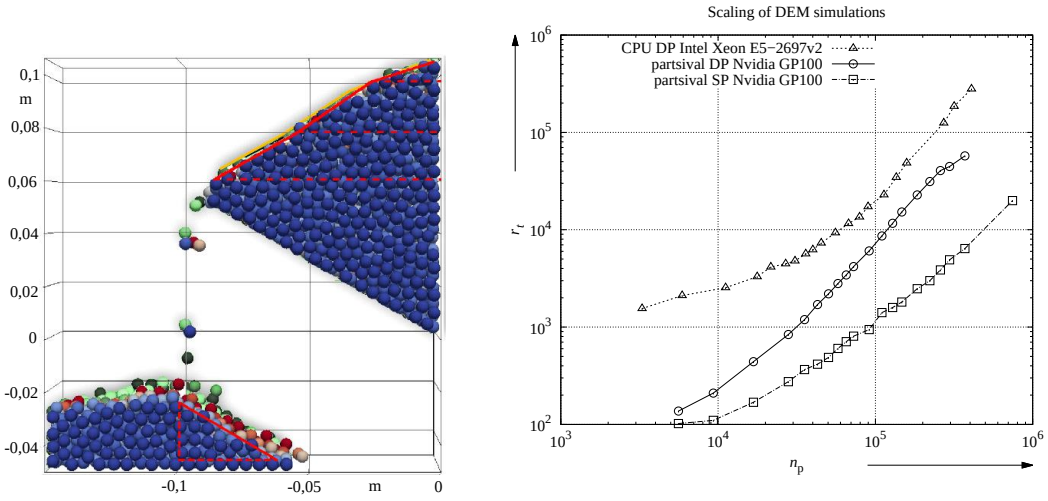

Fig. 7: Left: Final state $t_3 = 3\,\mathrm{s}$ of the accuracy benchmark between CPU-bound DEM (translucent green particles) and partsival (opaque colored particles) both in double floating point precision. Right: Performance benchmark between CPU-bound DEM (uppermost dashed line) in double floating point precision and partsival in both double and single floating point precision. The graph shows real time factor r_t (smaller is better) over number of particles n_p.

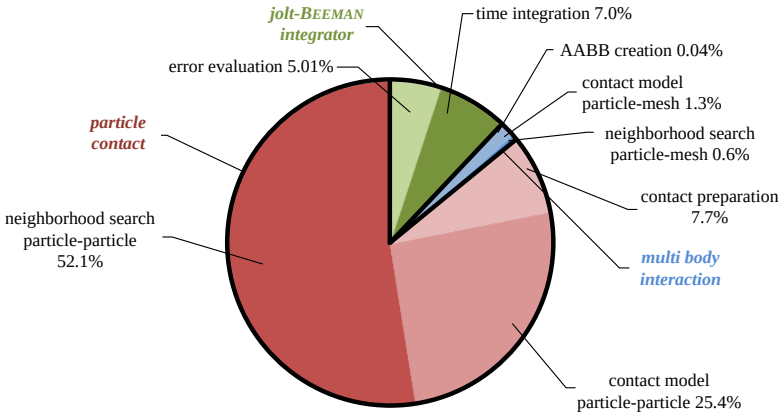

Fig. 8: Profiling results for partsival's built-in `SimulationSteps` (bold-framed areas) and their `Runnables` (shaded colors).

5.3 Performance Benchmark

The particle count for this benchmark is increased from starting with low numbers of a few thousand particles up to almost one million of particles. The test had to be stopped at this number, as the CPU-code would not have finished within reasonable time on given resources, while partsival was successfully tested to deliver results even for several million particles. As CPU-bound code does not scale well using low numbers of particles and large numbers of CPU cores, the number of cores is increased together with the particle count. Thereby up to a maximum of 48 threads the optimal number of 2000 particles per thread is kept. Constant work group sizes for all the simulations are used, optimized for the targeted GPU (cf. section 2.3), the Nvidia top level card

Quadro GP100. This card was chosen, as it delivers similar performance in single and double precision and features 16 GB of main memory. The GPU-setup is operated on a workstation using a Intel Xeon E5-1620v3 CPU with 32 GB of main memory. In single precision the GP100 performs similar to the cheaper GeForce GTX1080 and is out performed by the GTX1080Ti. Both of these gaming grade cards cannot hold up in double precision computations and are thus omitted for this benchmark. The CPU used for the benchmark was chosen to be in the same price segment and thus yielded in a dual Intel Xeon E5-2697v2 setup with 128 GB of main memory.

As it can be seen in the logarithmic plot in Fig. 7, partsival outperforms the CPU-based code starting from the lowest numbers of particles on and keeps the lead even in the highest numbers. Starting from 50000 particles the performance difference is decreased, which is due to the fact that the VERLET list algorithm in partsival is still $\mathcal{O}(n^2)$, while the CPU code features a complexity of $\mathcal{O}(n \cdot \log(n))$. However, it was not possible to find an intersection of both graphs – partsival keeps the lead. Thus even though there is further potential for speeding up the performance, partsival is even better than expected. In single floating point precision mode, partsival is getting even faster and performs very well on cheap consumer grade GPUs. While evaluation in test simulations showed that single precision is not suited for absolute predictions single precision simulations may be used for relative statements like variational analysis or optimization runs. Hence for big simulation campaigns cheaper hardware might be sufficient, while further speeding up computation.

While the comparison needed to be stopped before reaching 1×10^6 particles, partsival has been tested up to 2×10^6 particles. In this range it was still possible to run shorter simulations in reasonable time. The bottleneck of the calculation is the more and more sequential calculation because the limits for fully parallel operations are exceeded by far.

In terms of GPU memory usage, partsival consumes constant 32.2 MiB plus 3.9 KiB per particle for single precision as well as constant 41.3 MiB plus 7.1 KiB per particle for double precision in an ideally linear scaling, as no additional overhead is imposed. For single/double precision, this results in 2.0/1.1 million particles on a GTX1080, 2.8/1.5 million on a GTX1080Ti and 4.1/2.2 million on a GP100. With single step integrators this number could even be increased as the particle inventory could be shrinked.

Analyzing the time spent per `SimulationStep` yields the results shown in Fig. 8. It is clearly visible that the contact detection and contact model is the actual bottleneck of the computation. Thus this performance potential will be freed in future versions by implementing more efficient contact detection, especially for dense granular phases.

6 Applications

In this section simulations solving real-world problems in engineering and science are shown to stress partsival's applicability in a variety of domains.

6.1 Scout Rover & Conventional Rover Wheels

partsival has been used to develop a novel wheel concept for both extremely soft and hard terrain featuring obstacles not traversable for current exploration rovers. A screenshot of this study is shown in Fig. 9, displaying a so called rimless wheel which allows to swim through the softest regolith and also aims for milli-g locomotion. It also enables resonant operation on hard ground

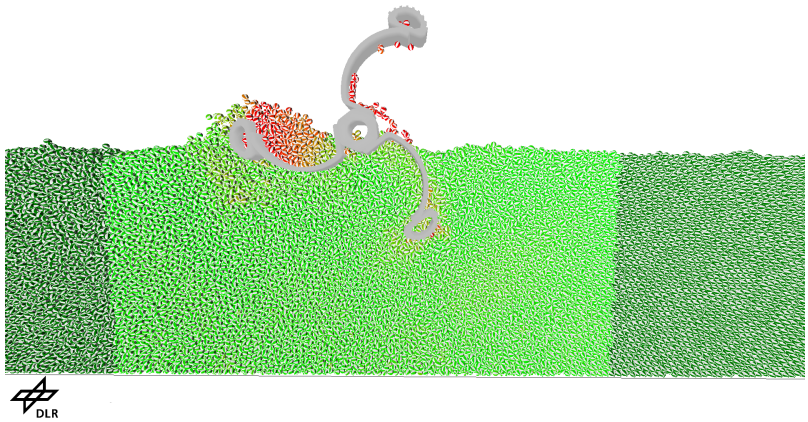

Fig. 9: Simulation of a scout rover wheel with almost 300000 particles for 3 m, dark particles denote particles inactivated by the dynamic boundaries.

to move on unstructured terrain in an energy efficient manner [32]. With this example the first long range DEM-simulation in DLR was conducted, made possible by partsival. Fig. 10 shows this large scale simulation of the single wheel traveling 6 m. In this simulation approx. 750000 particles are used to model the regolith. One full run of the simulation took roughly two hours and twenty minutes of real time. A similar simulation utilizing commonly used CPU code would have lasted several days.

Apart from the rimless wheel, studies on conventional rover wheels can be conducted as well (cf. Fig. 4). Thereby resulting, partsival will be used for the first extensive wheel optimization campaign for planetary rovers.

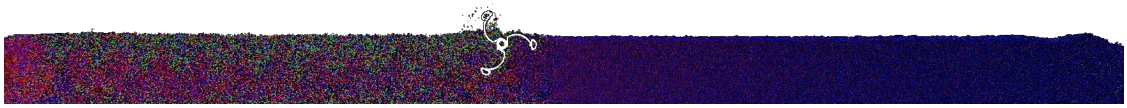

Fig. 10: Large scale simulation of 6 m distance with a scout rover wheel with approx. 750000 particles.

6.2 Automated Soil Preparation

The need for automated soil preparation arises mainly from laboratory tests for planetary exploration rovers and the requirement for reproducible soil conditions. In order to find suitable trajectories and tools for automated soil preparation, simulations are conducted. Results allow to program

industrial robots, like the DLR TROLL facility [33], to prepare a soil bin in less than a minute, while the state of the art manual preparation lasts hours and provides results of strongly varying quality. The robot, however, creates the required level of reproducibility and allows for faster and cheaper test campaigns. As a technology transfer, these techniques may also be transferred to agriculture or powder processing and enhance crop yields or reduce production cost.

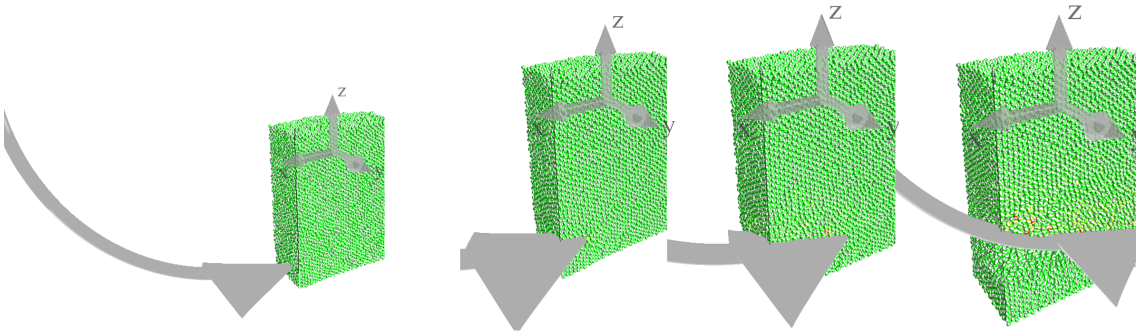

Fig. 11: Tillage tool driven through the soil in order to loosen up the soil volume. Image credit: Christian Mack, DLR-SR.

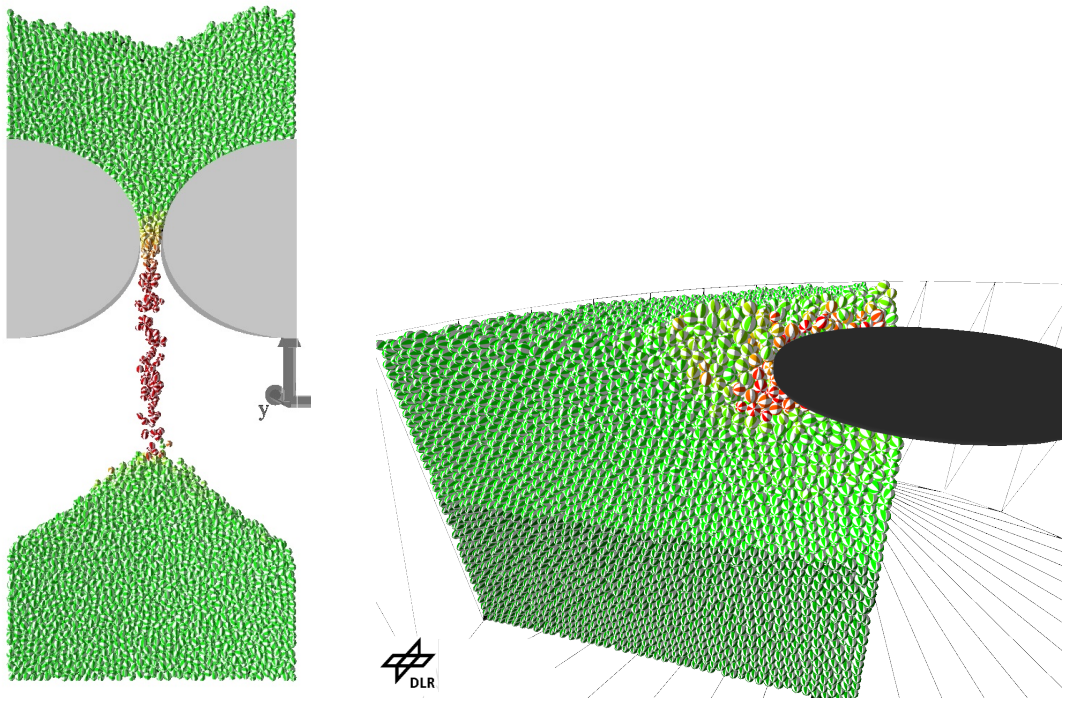

Fig. 12: Left: Sandwatch-like piling experiment to assess the behaviour of granular material in different gravitational environments [34]. Image credit: Shoko Ono, DLR-SR. Right: Bevameter simulation using partsival with symmetry conditions to allow a quarter simulation of the bucket. Image credit: Shoko Ono, DLR-SR.

6.3 Piling

This application (Fig. 12, left) has been built to provide a narrow particulate flow to measure both static and dynamic angle of repose in granular material. This allows for simple and fast numerical experiments to first analyze the detailed influence of contact model parameters on the shear strength and second the influence of different gravities [34]. From this simulation, scaling laws for the development of planetary exploration gear, most prominently planetary rovers, may be derived.

6.4 Bevameter

The bevameter developed by BEKKER [35] is a common tool to measure soil parameters and conditions in terramechanics. In simulation it is used to validate new DEM soil models as well as to assess influences of environmental conditions like gravity on shear strength. The model is implemented to be a quarter model of the cylindrical bin as shown on the right in Fig. 12. Therefore the symmetry features of partsival are used. In addition, dynamic boundaries have been applied to keep a constant depth of active particles below the indentation plate.

7 Conclusion and Future Work

This article presented partsival, a novel, high performance GPU-computing framework for accurate particle simulations. partsival features contact physics, boundaries and time integration beyond the current state of the art of other frameworks. With the piling simulation, a benchmark application has been implemented to validate and measure the performance gain of partsival compared to a state-of-the art CPU framework. By combining GPU computing and a soft contact approach, it is possible to reach high accuracy while at the same time speeding simulations up by factors of ≈ 10 at the current state. Given the rapid-model prototyping by allowing to switch floating point precision after compile time, model development in DEM can speed up significantly.

Given the profiling results, future versions will incorporate more efficient contact detection algorithms to further improve the performance. The interface concept focuses on highly efficient automation of simulation processes to lower the human interaction with simulation campaigns. At the same time, partsival provides faster learning success to simulation engineers with the software by means of auto-generated code by the easy to use GUI.

Future work will focus on both performance and additional features. The performance improvements over common CPU-code allow for more complex grains. Thus clump particles will be added to allow for more accurate simulations. Additionally, the user interface is another point of interest, in order to further improve the user experience for simulation engineers. With this ongoing simplification of usage and abstraction of complex simulation structures, more and more applications at DLR will be simulated via partsival.

Acknowledgements

Special thanks go to Shoko Ono and Christian Mack who used partsival for their thesis and contributed real-world applications from science and engineering shown in section 6, thus also delivering new requirements for the feature development of partsival.

References

[1] S. Kerler, "Collision-based Particle Simulations on GPUs for Planetary Exploration Systems," Master's thesis, University of Augsburg, 2017.

[2] "Project::Chrono." http://projectchrono.org/, accessed: 2016-10-12.

[3] T. Bellmann, "Interactive simulations and advanced visualization with modelica," in *Modelica Conference*, 7th Modelica Conference, Linköping University Electronic Press, 2009.

[4] R. Wang and X. Qian, *OpenSceneGraph 3.0: Beginner's Guide*. Packt Publishing Ltd, 2010.

[5] R. Lichtenheldt, *Lokomotorische Interaktion Planetarer Explorationssysteme mit weichen Sandböden - Modellbildung und Simulation*. PhD thesis, Ilmenau University of Technology, ISBN 978-3-8439-2704-8, 2016.

[6] F. Radjai and F. Dubois, eds., *Discrete-element Modeling of Granular Materials*. Wiley, 2011.

[7] M. Obermayr, *Prediction of Load Data for Construction Equipment using the Discrete Element Method*. PhD thesis, Universität Stuttgart, 2013.

[8] R. Lichtenheldt, "A novel systematic method to estimate the contact parameters of particles in discrete element simulations of soil," in *4th International Conference on Particle-based Methods - Particles 2015, pp 430-441, ISBN:978-84-944244-7-2, Barcelona*, 2015.

[9] J. Kleinert, M. Obermayr, and M. Balzer, *Modeling of large scale granular systems using the discrete element method and the non-smooth contact dynamics method: a comparison*. 2013.

[10] F. Fleissner, *Parallel Object Oriented Simulation with Lagrangian Particle Methods*. PhD thesis, Universität Stuttgart, 2010.

[11] M. Paz and W. Leigh, *Structural Dynamics*. Kluwer Academic Publishers, 2004.

[12] N. Newmark, "A method of computation for structural dynamics," *Journal of the Engineering Mechanics Division*, pp. 67–94, 1959.

[13] R. Lichtenheldt, "A stable, implicit time integration scheme for discrete element method and contact problems in dynamics," in *Particle-based Methods V: Fundamentals and Application, CIMNE, Barcelona, ISBN: 978-84-946909-7-6*, 2017.

[14] C. Kloss, C. Goniva, A. Hager, S. Amberger, and S. Pirker, "Models, algorithms and validation for opensource DEM and CFD-DEM," *Progress in Computational Fluid Dynamics, an International Journal*, vol. 12, no. 2-3, pp. 140–152, 2012.

[15] V. Šmilauer and B. Chareyre, *DEM formulation*, 2015. http://yade-dem.org/doc/.

[16] "EDEM." http://www.edemsimulation.com/, accessed: 2016-10-11.

[17] Nvidia, " NVIDIA GF100 ," tech. rep., 2010.

[18] AMD, "AMD GRAPHICS CORES NEXT (GCN) ARCHITECTURE," tech. rep., 2012.

[19] J. Zink, M. Pettineo, and J. Hoxley, *Practical Rendering and Computation with Direct3D 11*. A K Peters/CRC Press, 2011.

[20] The Khronos Group Inc., "The OpenCL Specification," tech. rep., 2015.

[21] Nvidia, "Cuda C Programming Guide," tech. rep., 2016.

[22] The Khronos Group Inc., "Vulkan 1.0.31 - A Specification (with all registered Vulkan extensions)," tech. rep., 2016.

[23] M. Segal and K. Akeley, "The OpenGL Graphics System: A Specification. Version 4.3 (Core Profile)," tech. rep., The Khronos Group Inc., 2012.

[24] "H5Part." online. http://vis.lbl.gov/Research/H5Part/, accessed: 2017-02-11.

[25] "ParaView." online. http://www.paraview.org/, accessed: 2017-02-11.

[26] L. Verlet, "Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules," *Phys. Rev.*, vol. 1591 98, 1967.

[27] R. Lichtenheldt and B. Schäfer, "Planetary Rover Locomotion on soft granular Soils - Efficient Adaption of the rolling Behaviour of nonspherical Grains for Discrete Element Simulations," in *3rd International Conference on Particle-Based Methods, S. 807-818, ISBN 978-84-941531-8-1, Stuttgart*, 2013.

[28] R. Lichtenheldt, "Covering shock waves on Mars induced by InSight's HP3 -mole - efficient co-simulation using dem and multi-domain dynamics," in *VII International Conference on Computational Methods for Coupled Problems in Science and Engineering COUPLED PROBLEMS 2017, ISBN: 978-84-943928-3-2, Artes Graficas Torres S.L.*, 2017.

[29] D. Beeman, "Some multistep methods for use in molecular dynamics calculations," *Journal of Computational Physics*, vol. 20, pp. 130–139, 1976.

[30] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, p.943, third edition ed., 2007.

[31] R. Lichtenheldt, B. Schäfer, and O. Krömer, "Hammering beneath the surface of Mars - Modeling and simulation of the impact-driven locomotion of the HP3-Mole by coupling enhanced multi-body dynamics and discrete element method," in *Shaping the future by engineering: 58th Ilmenau Scientific Colloquium IWK, URN (Paper): urn:nbn:de:gbv:ilm1-2014iwk-155:2 Technische Universität Ilmenau, 08 - 12 September 2014*, 2014.

[32] L. Stubbig, R. Lichtenheldt, F. Becker, and K. Zimmermann, "Model-based development of a compliant locomotion system for a small scout rover," in *59th International Scientific Colloquium, Technische Universität Ilmenau, urn:nbn:de:gbv:ilm1-2017iwk-012:0, September 11 - 15*, 2017.

[33] F. Buse, T. Bellmann, R. Lichtenheldt, and R. Krenn, "The DLR Terramechanics Robotics Locomotion Lab," in *ISAIRAS*, 2018.

[34] S. Ono, R. Lichtenheldt, and K. Yoshida, "Parametric influnces on the behaviour of planetary regolith using dem simulations," in *iSARIAS 2018, Madrid, Spain*, 2018.

[35] M. G. Bekker, *Introduction to Terrain - Vehicle Systems*. University of Michigan Press, Ann Arbor, 1969.