FAST MULTI-GPU DEM SIMULATION FOR DATA-DRIVEN OPTIMIZATION IN GRANULAR DYNAMICS

by

Ruochun Zhang

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Mechanical Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2023

Date of final oral examination: 10/31/2023

The dissertation is approved by the following members of the Final Oral Committee:
Dan Negrut, Professor, Mechanical Engineering
Xiaoping Qian, Professor, Mechanical Engineering
Jinlong Wu, Assistant Professor, Mechanical Engineering
Lianyi Chen, Associate Professor, Mechanical Engineering
Pavana Prabhakar, Associate Professor, Civil and Environmental Engineering

To those who sail the unknown sky with their moral compass

ACKNOWLEDGMENTS

I would like to extend my heartfelt gratitude to my two advisors, Professor Xiaoping Qian and Dan Negrut, for their invaluable guidance. Their mentorship extended beyond academic research, playing a role in shaping many facets of my life's philosophy.

I am also deeply grateful to all my friends, both in and outside Madison. Even without specific mentions, you know who you are. We embarked on this academic journey together, each seeking respect, identity, and value in our respective pursuits. In a world fraught with mistreatments and adversity, where the relevance of our current research may eventually wane, what truly matters is the unwavering support and understanding we offer each other. In these times, we grow to become more sagacious and phlegmatic, and that personal growth, in facing the world, is what truly counts.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

FAST MULTI-GPU DEM SIMULATION FOR DATA-DRIVEN OPTIMIZATION IN GRANULAR DYNAMICS

Ruochun Zhang

Under the supervision of Professor Dan Negrut

At the University of Wisconsin-Madison

Upon carrying out a literature survey, the candidate identified that the Discrete Element Method (DEM) community was in need of a customizable and efficient solver to accommodate the growing specialization and data-driven trend in the research scene. In response to that, in this work, the candidate established Chrono DEM-Engine, a state-of-the-art, open-source, multi-GPU DEM package that advances beyond its predecessor, Chrono::GPU, by supporting complex element shapes and efficiently managing tens of millions of elements. The software has a unique feature that decouples kinematic and dynamic computations, leveraging asynchronous threading to optimize performance and integrates with the multi-physics simulation engine Chrono. The research showcases the capabilities of Chrono DEM-Engine with an in-house developed digital DEM simulant that mirrors the properties of the GRC-1 extraterrestrial simulant, capturing its behavior and validating it with real-world experimental data. Furthermore, the candidate undertakes a comprehensive, data-driven rover wheel design study, evaluating over four thousand designs and utilizing a Gaussian process model for performance prediction. The outcomes indicate that the optimized wheel design based on the trained model outperforms by a wide margin the benchmark MGRU3 design. The research also highlights the utility of Bayesian optimization in revealing novel design avenues. The GPU code for the simulator and all numerical experiments discussed are open-source and available on GitHub for reproducibility studies and unfettered use and distribution.

Dan Negrut

1 BACKGROUND AND MOTIVATION

1.1 Existing DEM solvers

The Discrete Element Method (DEM) is a numerical technique for predicting the mechanical behavior of granular materials [4]. In DEM, the motion of each individual particle is monitored, and interactions between particles are modeled in a fully detailed manner. Over time, DEM has evolved and is now a popular method for examining the dynamics of extensive granular systems [5], ranging from mixing [6], particulate flows [7], geomechanics events [8, 9, 10], to astrophysical scenarios [11]. Applications of DEM include modeling soil dynamics [12], tire-soil interactions [13], and rover movement on extraterrestrial surfaces [14].

Two main challenges make DEM simulations computationally expensive. Firstly, the small and often stiff elements necessitate the time integrator to adopt minimal time steps to ensure numerical stability. Secondly, collision detection processes are computationally demanding. To enhance computational speed, DEM has been enhanced using parallel computing with OpenMP [15] as seen in [16, 17]; MPI standard [18] for distributed memory clusters [19]; and combined MPI–OpenMP parallelism [20, 21, 22, 23, 24]. The Graphics Processing Unit (GPU) offers another architecture for parallel computations and has been incorporated into DEM, as in [25, 26, 27, 28]. Regardless of the computational platform, reported DEM studies typically involve between 10^3 and 10^5 elements [29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 28], which is considerably smaller than real-world scenarios. For instance, a cubic meter of sand contains around two billion particles [44].

A number of DEM solvers in the literature are surveyed in this section. LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [22] is a widely used open-source software package for molecular dynamics simulations, including DEM simulations. LAMMPS is written in C++ and is designed to run efficiently on parallel computing architectures using both MPI and OpenMP, making it suitable for simulating large-scale systems. LAMMPS provides a variety of built-in potentials for modeling interatomic and intermolecular interac-

tions, as well as the ability to define custom potentials. LAMMPS also supports a range of boundary conditions, including periodic, reflecting, and fixed boundaries. Another noteworthy design feature of LAMMPS is it uses a time integration scheme based on the Verlet algorithm. This algorithm is a second-order method that conserves energy and momentum, which are desirable properties in molecular dynamics. LAMMPS supports a range of contact models, including Hertz–Mindlin, linear-spring, cohesive and inter-particle bond models.

LAMMPS is well-known for its flexibility in modeling a wide range of materials. In [45], a notable research work using a coarse-grained model for simulating the behavior of DNA with LAMMPS is presented. The model is based on a bead-spring representation of DNA, where each nucleotide is represented by a single bead, and the bonds between nucleotides are represented by springs. The model includes electrostatic interactions between the DNA beads and interactions between the DNA and solvent molecules. Other noteworthy research works include simulating a pebble-bed reactor using the Monte Carlo code named RMC in conjunction with the DEM code LAMMPS to study the influence of randomness of mixed pebbles [46]; investigating the use of LAMMPS for simulating soft active matter [47].

These examples showcase the wide collection of problems supported by LAMMPS. Narrowing down the scope, LIGGGHTS (LAMMPS Improved for General Granular and Granular Heat Transfer Simulation) is a DEM package that is based on the LAMMPS code. Like LAMMPS, LIGGGHTS is optimized for parallel computing and leverages combined MPI–OpenMP parallelism. While LAMMPS is more versatile, LIGGGHTS focuses specifically on granular material simulations, offering features and capabilities tailored to that end, such as neighbor lists and domain decomposition. These added utilities come into play in granular flows, heat transfer in granular materials, and other DEM-specific concerns.

STAR-CCM+ [23] is a commercial Computational Fluid Dynamics (CFD) software package that includes a DEM solver for simulating the behavior of granular materials. The software also supports a range of contact models. One of the strengths of STAR-CCM+ is its ability to couple DEM simulations with fluid flow simulations, allowing for the simulation of complex

multiphase flows. The coupling between the DEM and fluid flow simulations is typically achieved through a two-way coupling algorithm that exchanges information between the two simulations at each time step. The software also includes models for turbulence, heat transfer, and chemically reactive flows, and incorporates design exploration and optimization tools, allowing engineers to not just simulate a given design, but also explore a variety of design possibilities.

A recognizable research study that used STAR-CCM+ for DEM simulations is reported in [48]. The study aimed to investigate the sand-retention mechanisms that occur at the opening of sand filters under various conditions, such as particle shape, size, and concentration. A coupled CFD–DEM model was used to predict the retention mechanisms under steady flow conditions of the well-bore, where CFD was used to model the fluid flow, and DEM was used to model the particle flow. The coarse grid unresolved and the smoothed unresolved (refined grid unresolved) coupling approaches implemented in STAR-CCM+ were used to transfer data between the fluid and solid phases and calculate the forces. Verification of the CFD–DEM model was then conducted by mesh sensitivity analysis. The growing trend in CFD–DEM coupling research underscores the community's heightened interest in integrating multi-physics into DEM simulations, likely driven by the rapid advancements in computational hardware.

Compared to the LAMMPS and STAR-CCM+, Chrono::GPU [49], an open-source DEM simulator developed originally as the granular dynamics support for Chrono [50], takes a different path in that it emphasizes efficiency. To maximize performance, Chrono::GPU operates on GPUs and exclusively supports monodisperse spherical DEM elements. Additionally, a custom data type scheme is used to reduce its memory footprint. A recent independent study [51] reveals that Chrono::GPU, while running on an RTX 2060 Mobile NVIDIA GPU card of a laptop, delivers performance that is two orders of magnitude faster than other well-regarded DEM packages operating on clusters with hundreds of CPU cores.

YADE (Yet Another Dynamic Engine) is yet another open-source DEM solver that is widely used for simulating the behavior of granular materials, powders, and other particulate

systems. It is written in C++ and Python, and is known for its ability to handle complex geometries and its extensibility through Python scripting. One research study that used YADE for DEM simulations is [52], which is about the modeling of the deformation of the particles under stress. The study uses a soft-particle approach, where the particles are modeled as a collection of smaller particles connected by springs. The authors made additional developments to the DEM model, so the volume of the element overlapping area is uniformly redistributed over the particle, the radius of each contact partner is increased, and in the end, the volume and mass are kept constant. Large deformations and complex element geometries are used in this study. Another recent study that used YADE for DEM simulations is reported in [53]. The study aimed to carry out a numerical simulation of the process of icing using the Euler–Lagrangian approach. YADE was used to calculate the motion of snow crystals, while the free open-source CFD package OpenFOAM was used in conjunction with YADE to simulate flow hydrodynamics. These examples show the ecosystem surrounding this open-source DEM solver, which proliferates due to the customizability of YADE that facilitated many multi-physics co-simulations, as well as the community that contributes to the software by adding new features, fixing bugs, and improving performance.

To circumvent extensive computation times, DEM packages often resort to simplistic element geometries to simplify collision detection. Predominantly, spheres of uniform size are chosen, significantly streamlining collision detection. The software development process benefits from this simplification, as complex shapes introduce complexity [54]. Yet, certain applications require more intricate geometries, necessitating the usage of nonspherical elements to ensure accurate system dynamics [55, 56, 57, 58, 59, 60, 61]. From the aforementioned packages, YADE can use superquadric shapes to represent particles. Superquadrics are a family of shapes that include ellipsoids, rectangles, and more. They can represent a range of shapes with varying degrees of roundness or sharpness. YADE also supports polyhedral-shaped particles. Another approach YADE employs is the use of the “multi-sphere method” [59], meaning grouping simpler particles (like spheres) together to form more complex shapes.

Likewise, LAMMPS supports this multi-sphere method, too. LAMMPS also supports ellipsoidal and spherical particles. STAR-CCM+, being an established commercial DEM solution, offers a library of predefined shapes (spheres, cylinders, tetrahedra, etc.), while retaining a general-purpose custom shape-support using triangulated surfaces. These custom shapes are treated as rigid bodies within the DEM framework. When these methods to address nonspherical elements are employed, the number of elements in simulations tends to reduce, since they are often proprietary, costly, and have limitations regarding large element counts.

Recognizing the characteristics, strengths, and limitations of the existing DEM solvers, the solution presented here, Chrono DEM-Engine [62], aims to strike a balance: (i) it accommodates a large number of discrete elements (into tens of millions); (ii) it employs a composition of multiple spheres to represent nontrivial geometries; (iii) it integrates a rapid collision detection method as per [63] and a novel asynchronized threads management algorithm to ensure a numerical performance ahead of state of the art; (iv) its API design leaves enough room and flexibility for easy integration in co-simulations (explained in Sec. 2.2 and 3.2.2), and gives users the freedom to define explicitly the physics they wish to simulate using a custom force model script (explained in Sec. 2.1.2.1).

1.2 Shape design of wheels of off-road vehicles

This section offers a literature review on the important aspects of wheel design for off-road vehicles, particularly extraterrestrial rovers. Section 1.2.1 delves into the significance of wheels' adaptability to diverse terrains and their energy optimization. Section 1.2.2 explores the emerging integration of Artificial Intelligence (AI) in the generative design process, providing a fresh perspective on design possibilities. Lastly, Sec. 1.2.3 emphasizes the interplay between advanced simulations and empirical testing in evaluating wheel performance. Together, these subsections present a comprehensive overview of the current state-of-the-art in off-road wheel design.

1.2.1 Terrain adaptability and energy efficiency

Terrain adaptability and energy efficiency are key parameters in the design of wheels for vehicles and, more particularly, for extraterrestrial rovers due to the varied and often challenging terrains these vehicles must navigate.

Planetary surfaces, like those of the Moon and Mars, can vary significantly over short distances [64]. The diverse nature of these surfaces emphasizes the need for rovers to have wheels that can adapt to the intended operation environments. The primary mission of rovers is to collect data, and vehicle immobilization due to challenging terrains can compromise this objective [65]. Thus, ensuring the rover's wheels can handle various terrains is essential to protect the substantial investment in space missions [66].

Rovers typically operate on constrained energy resources. Therefore, the design of their wheels must ensure that energy consumption is minimized, allowing the rover to cover more ground on the same energy reserves. This energy-efficient movement not only enables longer mission durations and a broader data collection scope but also reduces the wear and tear on both the wheels and the mobility system [67].

The understanding of vehicle mobility on deformable terrains can be traced back to seminal works such as Bekker's, which introduced essential parameters like sinkage and motion resistance, foundational for understanding wheel–terrain interactions [68]. Over time, various techniques were employed to measure terrain adaptability and energy efficiency. For instance, the sinkage test determines how deeply a wheel sinks into terrain, providing insights into wheel adaptability. The drawbar pull test measures the force needed to pull a wheel through a terrain, providing insights into energy efficiency [69]. Another crucial measure is the gradability test, which assesses a rover's ability to handle slopes and inclines [70]. While empirical tests have been the traditional approach, the rise of computational tools has led to an increase in simulated studies, further enhancing the understanding of wheel–terrain interactions [71].

1.2.2 AI-powered design studies

Generative design refers to an iterative design process that involves the generation of a large number of design variants to meet predefined objectives and constraints. Typically, traditional design methods involve a manual approach to developing designs, while generative design leverages computational algorithms to autonomously produce a multitude of options that meet or exceed desired performance metrics.

In recent years, AI has been integrated into the generative design process to better optimize solutions and discover innovative design possibilities. With AI's capability to learn from data, the generative design can transcend conventional human biases and limitations, thereby providing a broader design space exploration.

Various AI techniques have been explored. Neural networks, especially deep learning models, have shown promise in recognizing patterns and generating designs based on vast datasets.

Generative Adversarial Networks (GANs) stand out as a neural network architecture that has been extensively tapped into for generative design. Characterized by the duel between a generator and a discriminator, GANs progressively refine the design outputs generated by the former based on the evaluations of the latter. Originally devised for image generation, GANs have since found applications in myriad domains including design. A salient example is [72], wherein generative models were utilized for the conceptualization of aircraft shapes, subsequently validated using Finite Element Method (FEM) simulations. Another noteworthy study integrating GANs with topology optimization for vehicle wheel designs is [73], which provides insight into the confluence of generative models with traditional design optimization techniques. This trend even extends to novel fashion designs, which make use of AI's potential in generating creative outputs [74].

Evolutionary algorithms have also played a significant role in AI-driven generative design. They mimic the process of natural selection, iteratively refining designs based on performance metrics until an optimized solution is found. Bentley's work on the application of genetic

algorithms for design and optimization presents foundational insights into how these techniques can be effectively employed [75].

While the application of AI for rover wheel design has yet to experience broad proliferation, most existing investigations have often circumscribed their focus to relatively rudimentary setups—such as two-dimensional problems or assessments based on specific performance metrics. In the research endeavor that has led to this thesis, the emphasis is not placed on leveraging AI to directly generate wheel designs. Instead, the focus is on harnessing data-driven methodologies to comprehensively interpret an amassed database of wheel designs. Notwithstanding, the potential incorporation of generative models to autonomously yield additional designs represents a promising avenue for future exploration, potentially obviating the current necessity of manual design creation, fostering avenues yet to be chartered.

1.2.3 Performance gauging works

Performance evaluation is an essential facet of the wheel design process. The confluence of rigorous testing methodologies and comprehensive metrics provides unique insights into design efficacy and functional robustness for operation in extraterrestrial terrains.

Rover wheels, when deployed in extraterrestrial settings such as Mars or the Moon, face challenges in terms of wear and durability. The terrains of these celestial bodies are characterized by their rugged topography and the presence of fine particulates that can be highly abrasive. A recent, detailed analysis by NASA [76] explored the wear patterns observed on rover wheels during Mars missions, specifically those of the Curiosity rover. Their work elucidated the multifaceted causes behind wheel punctures, wear rates, and grousers' breakages. Such investigations highlight the indispensable role of durable materials, design resilience, and manufacturing techniques to ensure wheel longevity, thereby ensuring the rover's operational tenure in such unforgiving environments.

In the realm of wheel design and evaluation, simulations have consistently emerged as valuable tools. Leveraging advanced computational models, they replicate real-world scenarios

and terrains to gauge wheel performance under diverse conditions. DEM, previously discussed, is shown to capture the physics well especially with the multi-sphere method. One example is the research by Johnson et al. [14], which encapsulates the utilization of advanced simulations to analyze the mobility and interaction of rover wheels in granular media, replicating Martian regolith conditions using tri-sphere clumps, where the numerical simulations match well with the experimental data at high wheel slip ratios.

Further underpinning DEM's versatility in the domain, [77] demonstrates the use of three distinct multi-sphere representations, mirroring real-world grain types, and a deformable pneumatic tire simulated through finite element techniques, thus accentuating DEM's potential in simulating intricate off-road mobility scenarios. Similarly, the research presented in [78] leveraged the adaptability intrinsic to DEM simulations, examining parameters like element stiffness, damping, cohesion, and gravity. One of the significant observations presented in [78] was the amplification of soil movement, increased sinkage, and diminished drawbar pull upon gravity reduction.

However, as indispensable as simulations might appear, the imperativeness of empirical, real-world testing remains uncontested. Such tests, meticulously crafted to emulate envisioned terrains, not only corroborate simulation outcomes but also unveil unexpected nuances and hurdles. For instance, [79] has reported extensive tests on rover wheels operating in both simulated (coupled multi-body dynamic and DEM) and experimental basalt and silica test beds, in which the authors found the crushed basalt's small particle size combined with high interlock led to increased tractive performance compared to the simulation.

In retrospection, while the panorama of research concerning wheel design has been burgeoning, there are perceptible lacunae. On the simulation front, current endeavors predominantly incline towards smaller scales, or necessitate the utilization of prohibitively expensive and inaccessible hardware configurations, such as extensive multi-node clusters. Moreover, there is a palpable opacity surrounding software tools used, often circumventing accessibility and broader applicability. Extending this introspection, a substantial portion

of research, instead of fostering a comprehensive understanding, seems enmeshed in the granularities of singular design or test scenarios, akin to meticulous case studies. This myopic vantage obfuscates the broader, interconnected dynamics at play when multiple test scenarios or design paradigms intertwine. As a corollary, the sagacity that these studies offer to practical design initiatives remains constricted, emphasizing the need for more holistic, interconnected, and large-scale research ventures that we strive to provide in this thesis.

1.3 Dissertation organization

In the literature survey presented in this chapter, we discern a need within the DEM community: A requirement for an adaptable and efficient solver that can handle large-scale simulations. Chapter 2 delves into the numerical attributes of Chrono DEM-Engine, the response to this need, and elucidates the validation efforts for this solver by navigating a collection of DEM simulations. In Chap. 3, the focus is on the conception and validation of a digital representation for the GRC-1 lunar simulant. Chapter 4 elevates the insights from the preceding chapters, amalgamating their outcomes to drive a data-centric design optimization. Here, the objective is the enhancement of current rover wheel designs, a journey anchored by over four thousand randomly generated designs. Finally, Chap. 5 provides a summative overview of the research effort, highlighting two promising avenues for furthering this research.

2 CHRONO DEM-ENGINE

2.1 Implementation features

Chrono DEM-Engine was designed to represent one more step of the broad community effort that seeks to democratize the domain of DEM simulation. The simulator is open-source, it runs on commodity hardware, and it does so fast and at scale. Chrono DEM-Engine allows large-scale DEM simulations to be efficiently executed on not just clusters, but also desktops equipped with one or two graphic cards. Its open-source nature caters to the unique modeling requirements often found in academic research and exploration. This section introduces its key numerical features.

2.1.1 Multi-GPU solution and delayed contact detection

In DEM, the contact detection process is needed to identify the contact pairs in the simulation system before the force calculation step takes place. The contact detection and force calculation are typically done consecutively in each time step. DEM-Engine embraces a different strategy, which uses two distinct and parallel computational threads to update the active contacts set (done by the "kinematics thread") and the integration of the equations of motion (done by the "dynamics thread"), respectively. The dynamic thread is instructed to process each active contact at each time step to reassess the contact penetration δ_n and the ancillary information (see Eqs. 2.2 in Sec. 2.3.1), and only receives a new active contact update when the kinematic thread finishes producing it, or to wait for the update when it drifts too far ahead of the time stamp of the last update from the kinematic thread. Through this collaboration pattern, the two threads work concurrently and the cost of contact detection is nearly "hidden in the shadow". To avoid missing mutual contacts that might crop up between the moments the active contacts set is updated, for the task of updating the active contact set only, we artificially enlarge all contact geometries in the DEM system to preemptively detect potential contact pairs that might emerge in the near future.

It is worth noting that by adding this artificial margin to all contact geometries, the kinematic thread reports false positives. They will be identified by the dynamic thread when carrying out the force calculation, yet processing them costs extra time. The thickness of this added margin is determined by the simulation entities' velocity (which is bounded and known by the solver), the time step size which is typically small, and $n_{\max}$, the maximum time steps dynamic thread is allowed to advance without receiving an update from the kinematic thread. It usually assumes values of the order of tens of microns for millimeter-sized granular material. This is small compared to typical DEM element sizes. Overall, the overhead caused by the false-positive contacts does not offset the benefit of deferred updates of the active contacts set.

To further explain this synchronization pattern, we present Fig. 2.1. There, "**S**" means a time step that the dynamic thread executes, where the contact forces are calculated (see Sec. 2.3.1), and the system state is marched forward in time. A contact detection step that the kinematic thread executes is marked with "**CD**". Periodically, the kinematic thread finishes a contact detection step and sends the signal to the dynamic thread, allowing the dynamic thread to receive the contact array, "**CA**", from the kinematic thread. Then the dynamic thread will send a work order "**WO**" with the current simulation system state, for the kinematic thread to pick up and continue the next contact detection step. Before the next "**CA**" update is received, the dynamic thread will use this "**CA**" to execute the time steps.

Because the dynamic thread only receives updates from and sends work orders to the kinematic thread after a time step is finished, the kinematic thread could stay idle between contact detection runs. This is marked with "**W**" in Fig. 2.1. Having the kinematic thread wait occasionally is considered an ideal collaboration pattern since in this case, the dynamic thread runs continuously, therefore the system marches in time uninterruptedly. On the contrary, an unideal collaboration pattern is illustrated in Fig. 2.2. There, the dynamic thread occasionally waits for updates from the kinematic thread, reducing the overall efficiency of the solver. This happens when the dynamic thread advances the simulation beyond $n_{\max}$

time steps without receiving an update from the kinematic thread, and is therefore forced to idle. One could avoid this scenario by increasing $n_{\max}$. However, as discussed before this would consequently increase the thickness of the artificial margin added to contact geometries, leading to more undesirable false-positive contacts. Hence, $n_{\max}$ should be kept at the smallest value that does not cause the dynamic thread to wait. DEM-Engine will automatically use this principle and the execution timing history to adapt $n_{\max}$ to an appropriate value, and moderate itself so the collaboration pattern stays as depicted in Fig. 2.1.

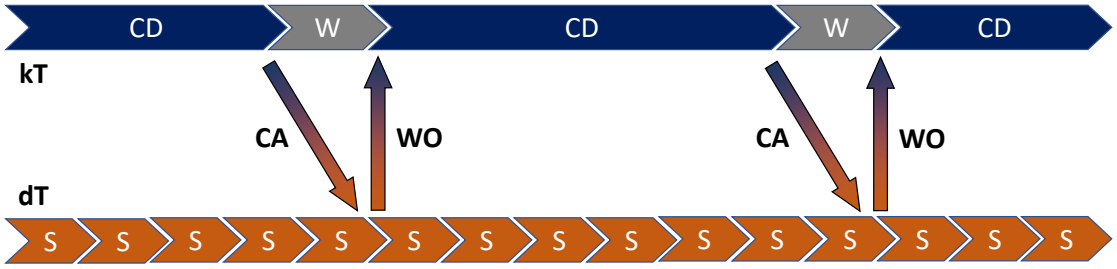


Figure 2.1: Ideal collaboration pattern, where the dynamic thread advances the physics continuously while the kinematic thread occasionally waits for a work order.

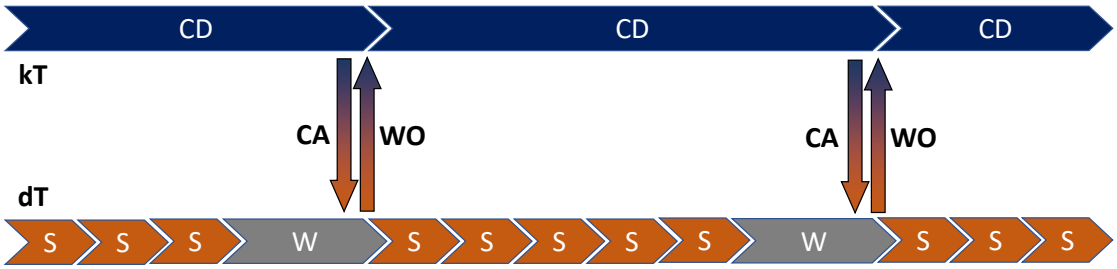


Figure 2.2: Unideal collaboration pattern, where the dynamic thread advances waits for the kinematic thread occasionally to generate the contact array. DEM-Engine will automatically avoid this scenario.

At the implementation level, DEM-Engine is currently optimized for using two GPUs, as each of the two threads is mapped to a GPU device respectively. The kinematic and dynamic thread collaboration pattern is summarized in Fig. 2.3. After being produced by the kinematic thread, the contact information is transferred to a buffer memory. Then the dynamic thread will be notified and copy the contact information to its working memory.

The dynamic thread carries out a similar routine when updating the kinematic thread with new element positions. Neither of them directly modifies the working memory of the other to avoid conflictive memory operations. Although logically there are two buffer memory pools in which each thread owns one, they are both allocated physically on the GPU mapped to the dynamic thread. This allows the dynamic thread to spend minimum time on copying from its buffer, speeding up the computation.

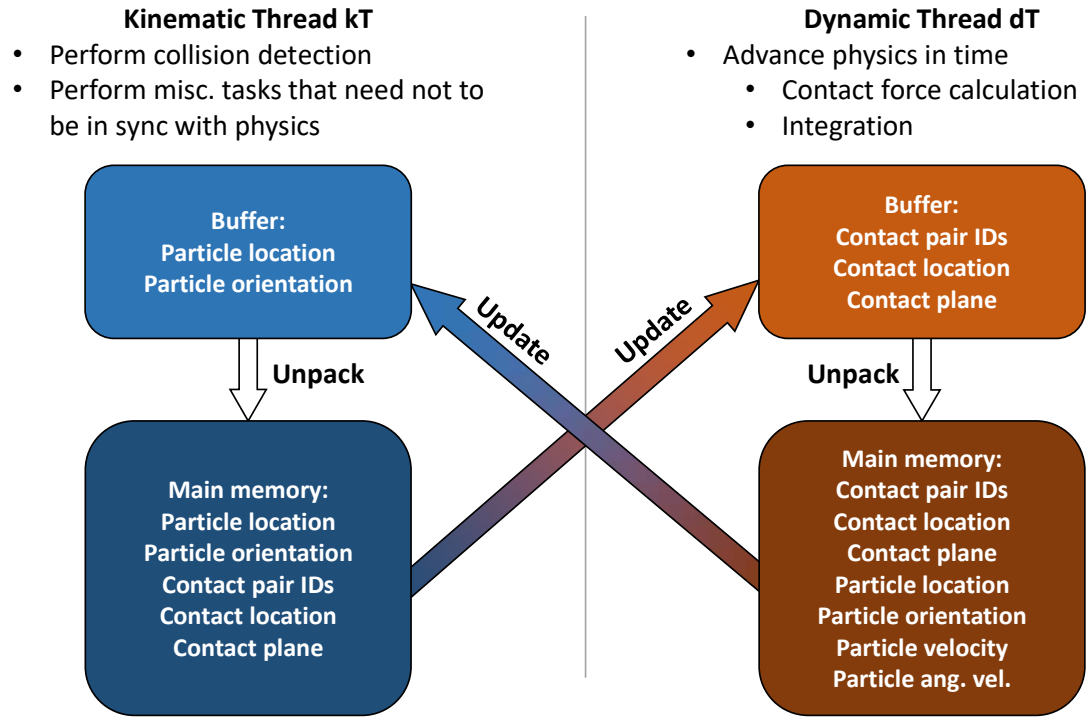


Figure 2.3: The collaboration pattern of the kinematic and dynamic thread. They each runs on a dedicated GPU.

2.1.2 Just-in-time CUDA kernel compilation

The CUDA kernels in DEM-Engine are compiled when the simulation starts being executed, rather than being statically compiled. This is done by leveraging the CUDA runtime compilation tool Jitify [80]. A number of benefits come with this design choice.

With Jitify, the solver can detect the capabilities of the GPU on which it is running and generate code specifically tailored for that device. For instance, if a program is designed to be used across a variety of architectures, just-in-time compilation ensures the utilization of the optimal instruction set for each device, ensuring the generated CUDA code is optimized for an end user's specific hardware and requirements. At the same time, since the compilation occurs at runtime, the code is not bound to a specific version of the CUDA toolkit. This characteristic can make applications more resilient against changes or updates in the CUDA environment.

It is worth mentioning that just-in-time compilation introduces an overhead. The first time a kernel is run, there is a delay due to its compilation. However, assuming the DEM simulations with DEM-Engine are generally large and invoke a time span of typically hours, this cost is negligible.

2.1.2.1 Custom force model

Since Jitify allows for dynamic code generation, we use it for implementing custom force models. The intricate and evolving nature of DEM simulations often requires a higher degree of adaptability to cater to the multifaceted modeling needs of its users, namely the expanding list of approaches in contact and cohesion force modeling [81, 82]. Rather than constraining the user to a predefined set of force models, DEM-Engine allows the custom force models to be supplied via a user-supplied C++ script, greatly increasing the solver's applicable use cases. A walk-through of a model implementation can be found in Sec. 2.3.2, which can serve as a starting point for the user to implement their own custom model.

2.1.2.2 Family tag

Jitify also allows for a low-cost implementation of prescribed motion. This is done through the family tag utility. Every simulation entity can be assigned an integer family tag between 0 and 255 (this is implemented through a `uint8_t`; though rarely needed, it can be changed

to a different data type such as uint16_t to expand the range), then the solver can be notified to apply prescribed motions to this family tag. This prescription information is just-in-time compiled as a part of the integration CUDA kernel, thus no branching overhead is introduced. The sample script in Sec. 2.2 showcases this functionality with the usage of the SetFamilyPrescribedAngVel method. On the other hand, if the use case calls for more fine-grain motion control, such as when the velocity of a simulation object is determined by some external process, then the "motion injection" approach detailed in Sec. 2.1.4 should be used.

As a side note, the family tags can also be used to mask contacts. The user is allowed to specify whether the solver should detect and resolve contacts between simulation entities in certain families. This is a utility used throughout the demos provided along with this package at [62].

2.1.3 Custom and mixed data type

In high-performance computing, the memory footprint plays a crucial role in determining the algorithm's efficiency and performance. As the complexities of our simulations grow, it becomes evident that relying solely on standard data types—such as double—might inadvertently lead to sub-optimal memory usage and consequently, potential performance bottlenecks. Given the hierarchical memory architecture in CUDA, from global to shared memory, the significance of ensuring that the memory bandwidth is utilized effectively and that latency is minimized becomes even more pronounced.

To this end, DEM-Engine introduces the utilization of custom and mixed data types. Unlike stock data types, which come with a predefined memory size, they offer more granular control over memory allocation. The general rule used in the development of DEM-Engine is that the data that reside in the global memory take a 4-byte or compressed data type. The examples are the state variables such as the quaternions of the elements. The temporary variables used in kernel functions that are essential in governing the physics, on the other

hand, use 8-byte data types, namely double. An example is the penetration depth between geometries in the Hertzian contact force calculation. As for representing a simulation entity's spatial location, a custom integer-based data type is used to effectively "compress" this information. The data type usage in DEM-Engine is summarized in Table 2.1. Since data type conversion is essentially a free operation and the main bottleneck in GPU-based physics simulations is the memory bandwidth limit, the design choice presented here enhances the performance without compromising the physics.

Table 2.1: Various data types in DEM-Engine and their memory location.

Data type	Variable	Memory type
uint64_t	Location, coarse grain	Global
uint16_t	Location, fine grain	Global
int32_t or float	Kinematics quantities, friction history etc.	Global
double	Penetration and kernel temp. variables	Register
float	Contact forces	Register
float	Particle shape information	Shared Memory

Furthermore, DEM-Engine has a level of encapsulation of the data types in use. Most data types are specified in a file named VariableTypes.h using typedef. If the user needs a different selection of data types, such as increasing the size of family tags from 1 byte to 2 bytes to allow for more varieties (see Sec. 2.1.2.2 for context), or reducing the size of spatial coordinates to allow for faster computation in a low-accuracy setting, they can modify the data types in VariableTypes.h then recompile to conveniently get the updated the executables.

2.1.4 Geometry hierarchy and tracker

DEM-Engine facilitates complex element geometries through a composition of multiple spheres, termed a "clump". This approach draws inspiration from [83]. A clump denotes a collection of potentially overlapping spheres that together depict a specific element shape. Some examples of these clumps are visually presented in Fig. 2.17 in Sec. 2.5. Throughout this thesis, the terms "element" and "clump" are used interchangeably to discuss DEM elements with complex

shapes. Beyond clumps, DEM-Engine supports integrating triangular meshes and analytical objects (such as rigid objects constructed from analytical planes or cylindrical surfaces) into the simulation framework. However, as a dedicated and performance-centric DEM package, DEM-Engine exclusively handles contacts between clumps and meshes, as well as between clumps and analytical objects. Should there be a requirement for contacts between meshes or between analytical objects, users can achieve this through co-simulation, as exemplified in Sec. 3.2.2.

An important aspect of DEM-Engine's utilization is understanding its geometry hierarchy, delineating the roles of the "owner" versus the "geometry". An owner constitutes a simulation entity endowed with mass properties, hence governed by physics. In DEM-Engine's current implementation, an owner can manifest as a clump, a mesh, or an analytical entity. Conversely, the term geometry elucidates the constituent parts of an owner. A geometric entity can be a sphere (within a clump), a triangular facet (within a mesh), or an analytical component (like a plane in a multi-component analytical object). Each geometric entity carries associated material attributes, granting users flexibility in designing discrete element systems with simulation entities that have spatially varying material properties.

Further, DEM-Engine provides users the control over diverse simulation entities via "tracker" objects. Users can associate trackers with any owner, facilitating real-time status inquiries such as position and velocity or enforcing state modifications, from setting coordinates to applying external loads. Beyond basic operations, trackers offer advanced features: Identifying clumps in contact with a tracked owner or, when monitoring a mesh, controlling its deformation. The DEMdemo_FlexibleMesh.cpp demo provides insights into the latter. A practical demonstration of tracker usage is encapsulated in Sec. 2.2.

2.1.5 Python wrapper

With the ascendancy of high-performance computing in modern research, there is a demand for tools that balance computational intensity with user accessibility. Responding to this,

DEM-Engine has been encapsulated within a Python interface, facilitated by the Pybind library. This interface provides researchers a conduit to harness CUDA's computational capabilities without the steep learning curve, blending robust simulations with Python's scriptable environment.

By leveraging Pybind, a library proficient at binding C++ code to Python, we established an efficient bridge for CUDA-based DEM simulations. This allows users, irrespective of their CUDA expertise, to tap into this package's features, all within Python's accessible library ecosystem and widely adopted science tools such as numpy and scikit-learn. The package has been made available on the Python Package Index (PyPI) and can be effortlessly installed using the familiar pip command. Simply executing pip install DEME ensures that all the computational capabilities and functionalities of our package are readily available in your Python environment, reducing the complexities often associated with software installations in high-performance computing scenarios.

An example script is given in Sec. 2.2.2, where it is compared against its C++ counterpart.

2.2 Sample script

In this section, we present the script responsible for the mixer timing analysis discussed in Sec. 2.5. For a deeper understanding of the physics underpinning this simulation and its implications on solver performance, the reader is directed to Sec. 2.5, while the focus here is to elaborate on the simulation's code implementation. We use this as an illustrative example to dissect the structure of a standard DEM-Engine script and highlight the customization options available to users. A visual representation of the simulation workflow is provided in Fig. 2.4.

Examples are provided in both C++ and Python. The scripts corresponding to all simulations addressed in this paper can be located in the DEM-Engine's demo directory [62].

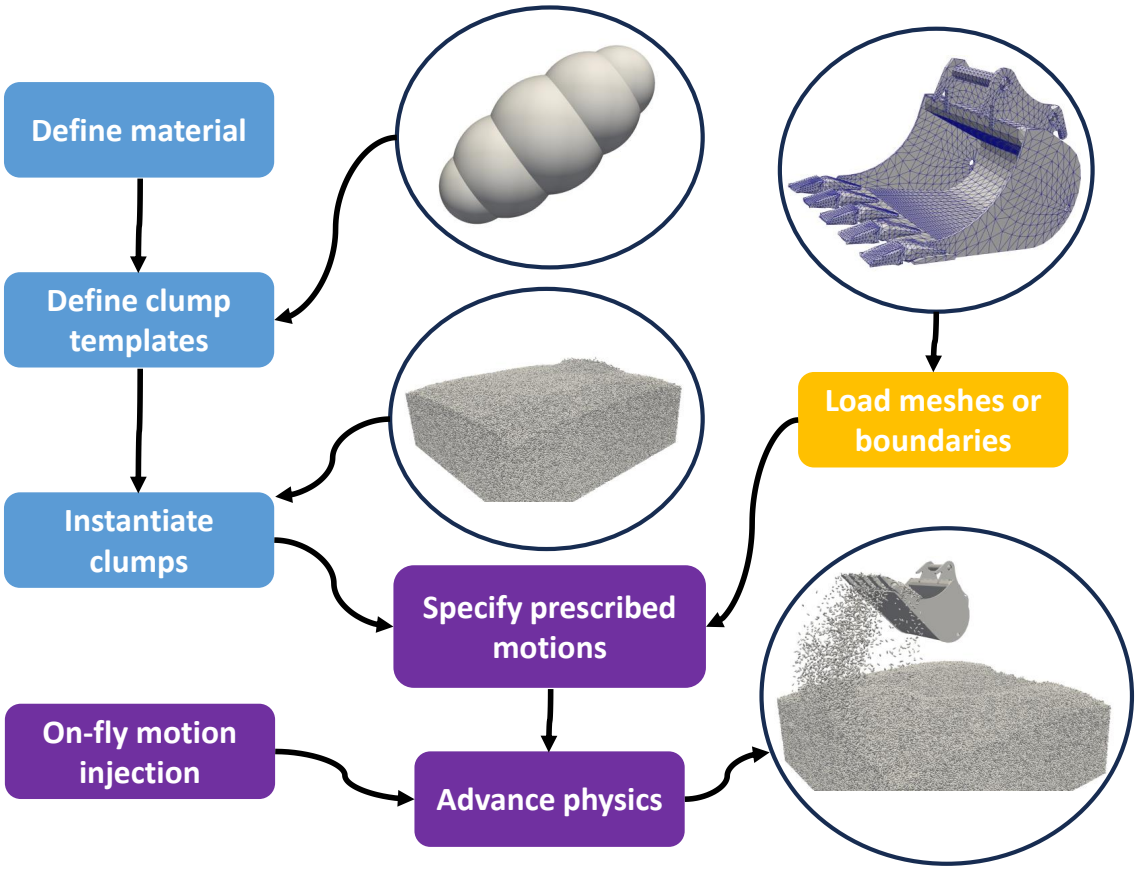


Figure 2.4: Typical workflow of running a DEM-Engine simulation.

2.2.1 C++ version

The user should first create the DEMSolver object. While the solver is equipped with default meta-parameters, users have the flexibility to modify its verbosity, output detail, and other settings.

```
DEMSolver DEMSim;
DEMSim.SetVerbosity("INFO");
DEMSim.SetOutputFormat("CSV");
DEMSim.SetOutputContent("ABSV");
DEMSim.SetMeshOutputFormat("VTK");
```

The following code snippet defines the material types for the mesh geometry and DEM elements. DEM-Engine will return a handle so this material can be used to define clump

templates. If a material property, such as the frictional coefficient μ, is defined between two materials, the method SetMaterialPropertyPair can be used to specify it.

```
auto mat_type_mixer = DEMSim.LoadMaterial({{"E", 1e8}, {"nu", 0.3}, {"CoR", 0.6}, {"mu",
    0.5}, {"Crr", 0.0}});
auto mat_type_granular = DEMSim.LoadMaterial({{"E", 1e8}, {"nu", 0.3}, {"CoR", 0.6}, {"mu",
    0.2}, {"Crr", 0.0}});
DEMSim.SetMaterialPropertyPair("mu", mat_type_mixer, mat_type_granular, 0.5);
```

The following snippet defines the analytical boundaries of the simulation domain.

```
const double world_size = 1;
DEMSim.InstructBoxDomainDimension(world_size, world_size, world_size);
DEMSim.InstructBoxDomainBoundingBC("all", mat_type_granular);
auto walls = DEMSim.AddExternalObject();
walls->AddCylinder(make_float3(0), make_float3(0, 0, 1), world_size / 2., mat_type_mixer, 0)
    ;
```

The following snippet shows the mixer mesh being loaded into the simulation. The stock mixer mesh is then scaled to fit the size of the simulation domain. The mixer is assigned the family code 10, then we use this family code to prescribe a constant angular velocity π rad/s to the mixer. A "tracker" object is created for the mixer so that we can extract information in real time for this simulation entity, or apply fine-grain motion control, while the simulation is running. In this example, we use it to set the initial location of the mixer to obtain the torque exerted by the DEM elements.

```
const float chamber_height = world_size / 3.;
auto mixer = DEMSim.AddWavefrontMeshObject((GET_DATA_PATH() / "mesh/internal_mixer.obj").
    string(), mat_type_mixer);
mixer->Scale(make_float3(world_size / 2, world_size / 2, chamber_height));
mixer->SetFamily(10);
DEMSim.SetFamilyPrescribedAngVel(10, "0", "0", "3.14159");
auto mixer_tracker = DEMSim.Track(mixer);
```

A clump template is created in the following snippet. Such a template contains information on the mass, shape and materials. There are stock clump shapes that the user can directly use to reproduce the examples we provide. The user can also easily scale or otherwise modify the template before using it to instantiate more DEM elements.

```
float granular_rad = 0.005;
float mass = 2.6e3 * 5.5886717;
float3 MOI = make_float3(2.928, 2.6029, 3.9908) * 2.6e3;
std::shared_ptr<DEMClumpTemplate> template_granular = DEMSim.LoadClumpType(mass, MOI,
    GetDEMEDataFile("clumps/3_clump.csv"), mat_type_granular);
template_granular->Scale(granular_rad);
```

When instantiating the DEM elements, the user has the option to leverage the sampler objects that come with the solver, as shown in the following snippet. A sampling region appropriate with respect to the simulation domain is defined, then the hexagonal close-packing sampler is used to create initial elements. These elements are duplicates of the clump template that has just been created.

```
const float fill_height = chamber_height;
const float chamber_bottom = -world_size / 2.;
const float fill_bottom = chamber_bottom + chamber_height;
HCPSampler sampler(3.f * granular_rad);
float3 fill_center = make_float3(0, 0, fill_bottom + fill_height / 2);
const float fill_radius = world_size / 2. - 2. * granular_rad;
auto input_xyz = sampler.SampleCylinderZ(fill_center, fill_radius, fill_height / 2);
DEMSim.AddClumps(template_granular, input_xyz);
```

An initialization call is needed to instruct the solver to set up data structures on the GPUs. Before that, crucial simulation specs should be inputted, such as the time step size and metrics that the solver should watch in identifying a diverged simulation, as shown in the following snippet.

```
float step_size = 5e-6;
DEMSim.SetInitTimeStep(step_size);
DEMSim.SetGravitationalAcceleration(make_float3(0, 0, -9.81));
DEMSim.SetErrorOutVelocity(20.);
DEMSim.SetForceCalcThreadsPerBlock(512);
DEMSim.Initialize();
```

Finally, the following code snippet shows the main simulation loop. The output directory is created, the simulation time length is indicated, and the mixer is translated to the correct initial position, before the main loop starts to iteratively make DoDynamics calls, advancing the simulation each time by a frame. The benefit of this design is that the user enjoys free

interfacing with the simulation data while it is running. For example, here we write the simulation status to a file, and inspect the torque that the mixer is experiencing, and output the execution stats from the kinematic and dynamic threads at the frequency of 20 times per simulation second. Another opportunity this design brings is the ease of co-simulation. A related example is in Sec. 3.2.2.

```
std::filesystem::path out_dir = current_path();
out_dir += "/DemoOutput_Mixer";
create_directory(out_dir);

float sim_end = 10.0;
unsigned int fps = 20;
float frame_time = 1.0 / fps;
unsigned int currframe = 0;

mixer_tracker->SetPos(make_float3(0, 0, chamber_bottom + chamber_height / 2.0));
for (float t = 0; t < sim_end; t += frame_time) {
    std::cout << "Frame: " << currframe << std::endl;
    char filename[200], meshfilename[200];
    sprintf(filename, "%s/DEMdemo_output_%04d.csv", out_dir.c_str(), currframe);
    sprintf(meshfilename, "%s/DEMdemo_mesh_%04d.vtk", out_dir.c_str(), currframe++);
    DEMSim.WriteSphereFile(std::string(filename));
    DEMSim.WriteMeshFile(std::string(meshfilename));

    float3 mixer_moi = mixer_tracker->MOI();
    float3 mixer_acc = mixer_tracker->ContactAngAccLocal();
    float3 mixer_torque = mixer_acc * mixer_moi;
    std::cout << "Contact torque on the mixer is " << mixer_torque.x << ", " << mixer_torque
        .y << ", " << mixer_torque.z << std::endl;

    DEMSim.DoDynamics(frame_time);
    DEMSim.ShowThreadCollaborationStats();
}
```

2.2.2 Python version

A Python version of the same mixer simulation is given in this section. It reflects the same workflow as the C++ version, including the material definition, template creation, clump

instantiation, mesh loading and motion control, initialization, and a main simulation loop. The names of the methods are not changed in the Python version, and certain data structures are simply converted to their Python counterparts, streamlining the learning experience of the users switching between these programming languages. For example, the C++ version uses a unordered_map to define the properties of a material, while the Python version uses a dictionary object; the C++ version takes a float3 at some places to specify a coordinate, while the Python version uses a list or a numpy array of three floats.

```
import DEME
import numpy as np
import os
import time
if __name__ == "__main__":
    out_dir = "DemoOutput_Mixer/"
    out_dir = os.path.join(os.getcwd(), out_dir)
    os.makedirs(out_dir, exist_ok=True)

    DEMSim = DEME.DEMSolver()
    DEMSim.SetVerbosity("STEP_METRIC")
    DEMSim.SetOutputFormat("CSV")
    DEMSim.SetOutputContent(["ABSV", "XYZ"])
    DEMSim.SetMeshOutputFormat("VTK")

    # E, nu, CoR, mu, Crr... Material properties
    mat_type_mixer = DEMSim.LoadMaterial(
        {"E": 1e8, "nu":  0.3, "CoR":  0.6, "mu":  0.5, "Crr": 0.0})
    mat_type_granular = DEMSim.LoadMaterial(
        {"E":  1e8, "nu":  0.3, "CoR":  0.8, "mu":  0.2, "Crr":  0.0})
    DEMSim.SetMaterialPropertyPair(
        "CoR", mat_type_mixer, mat_type_granular, 0.5)

    # Now define simulation world size and add the analytical boundary
    step_size = 5e-6
    world_size = 1
    chamber_height = world_size / 3.
    fill_height = chamber_height
    chamber_bottom = -world_size / 2.
    fill_bottom = chamber_bottom + chamber_height
    DEMSim.InstructBoxDomainDimension(world_size, world_size, world_size)
```

```
DEMSim.InstructBoxDomainBoundingBC("all", mat_type_granular)
walls = DEMSim.AddExternalObject()
walls.AddCylinder([0, 0, 0], [0, 0, 1], world_size / 2., mat_type_mixer, 0)

# Define the meshed mixer and its prescribed motion
mixer = DEMSim.AddWavefrontMeshObject(
    DEME.GetDEMEDataFile("mesh/internal_mixer.obj"), mat_type_mixer)
print(f"Total num of triangles: {mixer.GetNumTriangles()}")
mixer.Scale([world_size / 2, world_size / 2, chamber_height])
mixer.SetFamily(10)
DEMSim.SetFamilyPrescribedAngVel(10, "0", "0", "3.14159")
# Track the mixer
mixer_tracker = DEMSim.Track(mixer)

# Define the clump template used in the simulation
granular_rad = 0.005
mass = 2.6e3 * 5.5886717
MOI = np.array([2.928, 2.6029, 3.9908]) * 2.6e3
template_granular = DEMSim.LoadClumpType(mass, MOI.tolist(),
    DEME.GetDEMEDataFile("clumps/3_clump.csv"), mat_type_granular)
template_granular.Scale(granular_rad)
# Sampler uses hex close-packing
sampler = DEME.HCPSampler(3.0 * granular_rad)
fill_center = [0, 0, fill_bottom + fill_height / 2]
fill_radius = world_size / 2. - 2. * granular_rad
input_xyz = sampler.SampleCylinderZ(
    fill_center, fill_radius, fill_height / 2)
DEMSim.AddClumps(template_granular, input_xyz)
print(f"Total num of particles: {len(input_xyz)}")

DEMSim.SetInitTimeStep(step_size)
DEMSim.SetGravitationalAcceleration([0, 0, -9.81])
DEMSim.SetErrorOutVelocity(20.)
DEMSim.SetForceCalcThreadsPerBlock(512)
DEMSim.Initialize()

sim_end = 10.0
fps = 20
frame_time = 1.0 / fps

# Keep tab of the max velocity in the simulation
max_v_finder = DEMSim.CreateInspector("clump_max_absv")
```

```
print(f"Output at {fps} FPS")
currframe = 0

mixer_tracker.SetPos([0, 0, chamber_bottom + chamber_height / 2.0])

t = 0.
start = time.process_time()
while (t < sim_end):
    print(f"Frame: {currframe}", flush=True)
    filename = os.path.join(out_dir, f"DEMdemo_output_{currframe:04d}.csv")
    meshname = os.path.join(out_dir, f"DEMdemo_mesh_{currframe:04d}.vtk")
    DEMSim.WriteSphereFile(filename)
    DEMSim.WriteMeshFile(meshname)
    currframe += 1

    max_v = max_v_finder.GetValue()
    print(
        f"Max velocity of any point in simulation is {max_v}", flush=True)
    print(
        f"Solver's current update frequency (auto-adapted): {DEMSim.GetUpdateFreq()}",
            flush=True)
    print(
        f"Average contacts each sphere has: {DEMSim.GetAvgSphContacts()}", flush=True)

    mixer_moi = np.array(mixer_tracker.MOI())
    mixer_acc = np.array(mixer_tracker.ContactAngAccLocal())
    mixer_torque = np.cross(mixer_acc, mixer_moi)
    print(
        f"Contact torque on the mixer is {mixer_torque[0]}, {mixer_torque[1]}, {
            mixer_torque[2]}", flush=True)

    DEMSim.DoDynamics(frame_time)
    DEMSim.ShowThreadCollaborationStats()

    t += frame_time

elapsed_time = time.process_time() - start
print(f"{elapsed_time} seconds (wall time) to finish this simulation")
```

2.3 DEM model

In this section, we detail the implementation of geometry representations and the force models in DEM-Engine.

2.3.1 History-based Hertz–Mindlin model

In DEM-Engine, the default force model is grounded on the Hertzian contact model [84] and integrates the Mindlin friction model [85]. For a comprehensive analysis, readers may refer to [86]. For two bodies, namely i and j, when they are in contact, the normal force, $\boldsymbol{F}_n$, operates based on a spring–damper model. The tangential frictional force, $\boldsymbol{F}_t$, is computed considering material attributes and microscopic deformations, ensuring it adheres to the Coulomb limit via the friction coefficient μ. The mathematical representation is as follows:

$$\boldsymbol{F}_n = f(\bar{R}, \delta_n)(k_n \boldsymbol{u}_n - \gamma_n \bar{m} \boldsymbol{v}_n), \tag{2.1a}$$

$$\boldsymbol{F}_t = f(\bar{R}, \delta_n)(-k_t \boldsymbol{u}_t - \gamma_t \bar{m} \boldsymbol{v}_t), \quad \|\boldsymbol{F}_t\| \leq \mu \|\boldsymbol{F}_n\| , \tag{2.1b}$$

$$f(\bar{R}, \delta_n) = \sqrt{\bar{R}\delta_n}, \tag{2.1c}$$

$$\bar{R} = R_i R_j / (R_i + R_j), \tag{2.1d}$$

$$\bar{m} = m_i m_j / (m_i + m_j), \tag{2.1e}$$

where the constants k_n, k_t, γ_n, and γ_t are extrapolated from material characteristics, including Young's modulus E, the Poisson's ratio ν, and the restitution coefficient, CoR [87]. The terms $\bar{m}$ and $\bar{R}$ depict the effective mass and curvature radius for the specific contact. The foundational premise presumes that the geometries can undergo small penetration, δ_n, at the contact point. The normal penetration vector is $\boldsymbol{u}_n = \delta_n \boldsymbol{n}$. The relative speed, $\boldsymbol{v}_{rel} = \boldsymbol{v}_n + \boldsymbol{v}_t$, at the contact point is defined as:

$$\boldsymbol{v}_{rel} = \boldsymbol{v}_j + \boldsymbol{\omega}_j \times \boldsymbol{r}_j - \boldsymbol{v}_i - \boldsymbol{\omega}_i \times \boldsymbol{r}_i, \tag{2.2a}$$

$$\boldsymbol{v}_n = (\boldsymbol{v}_{rel} \cdot \boldsymbol{n})\, \boldsymbol{n}, \tag{2.2b}$$

$$\boldsymbol{v}_t = \boldsymbol{v}_{rel} - \boldsymbol{v}_n, \tag{2.2c}$$

where $\boldsymbol{v}_i$, $\boldsymbol{\omega}_i$ and $\boldsymbol{v}_j$, $\boldsymbol{\omega}_j$ denote the velocities at the mass centers and angular speeds of entities i and j. The position vectors, $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$, extend from the mass centers of bodies i and j to the shared contact point. The frictional force $\boldsymbol{F}_t$ varies based on the historical tangential micro-displacement $\boldsymbol{u}_t$, updated iteratively at each time interval throughout the interaction event based on $\boldsymbol{v}_t$. Let $\boldsymbol{u}_t'$ be the updated tangential micro-displacement, then

$$\boldsymbol{u}' = \boldsymbol{u}_t + h\boldsymbol{v}_t, \tag{2.2d}$$

$$\boldsymbol{u}_t' = \boldsymbol{u}' - (\boldsymbol{u}' \cdot \boldsymbol{n})\boldsymbol{n}, \tag{2.2e}$$

where h is the time step size. The strategy adopted to update $\boldsymbol{u}_t'$ is borrowed from [87]. After the update, we may need to clamp the updated tangential micro-displacement $\boldsymbol{u}_t'$ to get the final $\boldsymbol{u}_t$ for the next time step in order to satisfy the capping condition $\|\boldsymbol{F}_t\| \leq \mu \|\boldsymbol{F}_n\|$:

$$\boldsymbol{u}_t = \begin{cases} \boldsymbol{u}_t' & \text{if } \|\boldsymbol{F}_t\| \leq \mu \|\boldsymbol{F}_n\|, \\ \frac{\mu \|\boldsymbol{F}_n\|}{k_t} \frac{\boldsymbol{u}_t'}{\|\boldsymbol{u}_t'\|} & \text{otherwise.} \end{cases} \tag{2.2f}$$

The rolling resistance arises from an asymmetric normal stress profile at the contact patch [88]. In DEM-Engine's default force model, it is implemented as the torque $\boldsymbol{\tau}_r$. This torque is induced by a force that has the magnitude of the rolling resistance coefficient C_r times the normal force. The direction of this force is aligned with the rolling-contributed relative velocity at the contact point. This is summarized in the following equations:

$$\boldsymbol{F}_r = \frac{\boldsymbol{\omega}_j \times \boldsymbol{r}_j - \boldsymbol{\omega}_i \times \boldsymbol{r}_i}{\|\boldsymbol{\omega}_j \times \boldsymbol{r}_j - \boldsymbol{\omega}_i \times \boldsymbol{r}_i\|} C_r \boldsymbol{F}_t, \tag{2.2g}$$

$$\boldsymbol{\tau}_r = \boldsymbol{r}_i \times \boldsymbol{F}_r. \tag{2.2h}$$

As discussed in Sec. 2.1.4, a clump has mass properties associated, whereas its component spheres have material properties associated. This means $\boldsymbol{F}_n$ and $\boldsymbol{F}_t$ in Eq. 2.3a and 2.3b need to be derived from the contacts between component spheres. Then a reduction process is invoked to use these contact forces to update the element $\boldsymbol{v}_i$ and $\boldsymbol{\omega}_i$, based on each clump's m_i and I_i, as well as the location vector for the contact point, $\boldsymbol{r}_i$. This is visualized in Fig. 2.5, and the motion equations for entity i are therefore articulated as

$$m_i \frac{d\boldsymbol{v}_i}{dt} = m_i \boldsymbol{g} + \sum_{k=1}^{n_c} \boldsymbol{F}^k, \tag{2.3a}$$

$$I_i \frac{d\boldsymbol{\omega}_i}{dt} = \sum_{k=1}^{n_c} \left(\boldsymbol{r}^k \times \boldsymbol{F}^k + \boldsymbol{\tau}_r^k \right), \tag{2.3b}$$

where n_c is the number of contacts spheres that entity i has, and the the superscript k iterates through each contact. In these equations, $\boldsymbol{F}^k = \boldsymbol{F}_n^k + \boldsymbol{F}_t^k$ means the total force, containing both the normal and tangential components.

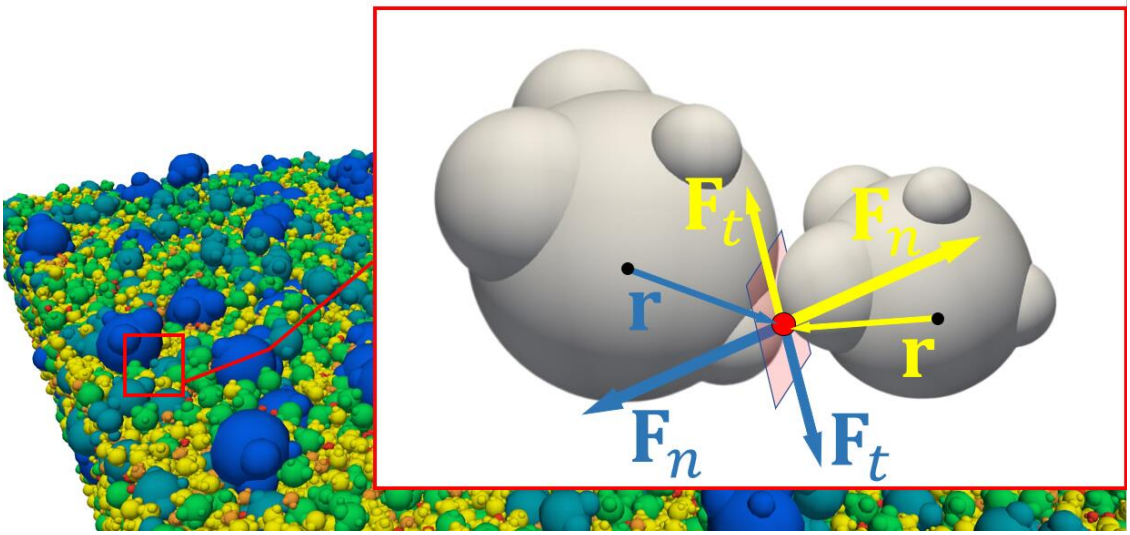


Figure 2.5: The normal and tangential contact forces between particles are calculated based on the penetration and displacement history of involved sphere components.

2.3.2 Custom model

To cater to diverse simulation needs, DEM-Engine supports custom force models through user-provided scripts. This section delves further into this functionality.

Users must supply the custom force model as a C++ script. This script undergoes just-in-time compilation at the onset of the simulation (as detailed in Sec. 2.1.2), replacing the default model. The "ingredients" of a custom force model are called user-referrable variables. A comprehensive list of these variables is provided in Table 2.2. For each contact pair, the solver automatically determines the values for these variables. Users can then harness these referrable variables to define the contact force derivation algorithm.

Central to scripting the force model is the modification of the user-referrable variable force, analogous to $\boldsymbol{F}^k$ in Eqs. 2.3a and 2.3b. This variable represents the force that geometry **A** experiences during contact in the global frame. The variable force takes the initial value of $(0, 0, 0)$. It is worth noting that the solver will auto-apply the corresponding reaction force to geometry **B**. In a similar vein, the user-referrable variable torque_only_force can be adjusted to store an action–reaction force pair that solely produces torque (without affecting the linear velocity of contact geometries, but only their angular momentum). This is congruent to $\boldsymbol{F}_r$ in Eq. 2.2g. In the default model, the implementation of rolling resistance hinges on this variable. As Eqs. 2.3a and 2.3b indicate, a subroutine executed by the solver in each iteration, will integrate the motions of simulation entities post the force calculation.

Note that the three "wildcard" type variables in Table 2.2 are the custom properties that the user is allowed to associate with contacts, owners (clump, mesh, or analytical object), and geometries (sphere, triangle facet, or analytical component), respectively. For the owner wildcards and geometry wildcards, the user can assign their values before or during the simulation, using trackers or family tags. These custom properties can then be used in the custom force model to derive force, or be modified so their values change during simulation according to a user-specified policy. The contact wildcards, on the other hand, work differently. If the user chooses to associate a wildcard to contacts, then the memory space associated with

Type	Name	Explanation
double3	contactPnt	Contact point coord in global
float3	B2A	Unit vector pointing from geometry **B** to geometry **A**
double	overlapDepth	The length of overlap
float	ts	Time step size
float	time	Current time in simulation
float3	locCPA, locCPB	Positions of the contact point in the contact geometries' frames
double3	AOwnerPos, BOwnerPos	Positions of both owners
double3	bodyAPos, bodyBPos	Positions of both contact geometries
float4	AOriQ, BOriQ	Quaternions of both owners
float	AOwnerMass, BOwnerMass	Masses of both owners
float3	AOwnerMOI, BOwnerMOI	Moment of inertia for both owners
float	ARadius, BRadius	Radius of curvature for both contact geometries at point of contact
uint8_t	bodyAMatType, bodyBMatType	Offset used to query the material properties for both contact geometries
uint8_t	AOwnerFamily, BOwnerFamily	Family number of both owners
float3	ALinVel, BLinVel	Linear velocity of both owners
float3	ARotVel, BRotVel	Angular velocity of both owners, in their local frames
unsigned int	AOwner, BOwner	Offset for both owners in system array
unsigned int	AGeo, BGeo	Offset for both contact geometries in system array
float	User-specified	Contact wildcards: Extra properties associated with contacts
float	User-specified	Owner wildcards: Extra properties associated with owners
float	User-specified	Geometry wildcards: Extra properties associated with geometries
float3	force	Accumulator for contact force
float3	torque_only_force	Accumulator for contact torque

Table 2.2: The user-referrable variables that can be used in composing the custom force model. All data types are the default data type. Some of the data types can be configured in VariableTypes.h upon compilation from the source to accommodate the user's specific needs, a concept introduced in Sec. 2.1.3.

a contact is allocated when this contact emerges, and deallocated when this contact vanishes. When it is allocated, it always takes the initial value 0. This is useful for recording some quantities that evolve during the lifespan of a contact. For example, as shown in Sec. 2.3.2.1, the default force model uses contact wildcards to record the contact history needed for the history-based Hertz–Mindlin model.

2.3.2.1 Default model implementation explained

We elaborate on the implementation of the default Hertz–Mindlin model in the remainder of this section, which can be found in the file FullHertzianForceModel.cu from the repository [62]. The code is an appropriate starting point for the user to implement their own force model, potentially adding to the existing physics.

The preliminary step, as presented in the ensuing code snippet, involves extracting material properties of the contact geometries. Material property arrays adopt naming conventions consistent with the user-defined property names in the LoadMaterial function call. Consequently, if the default force model is employed, Young's modulus (E), Poisson's ratio (nu), coefficient of restitution (CoR), friction coefficient (mu), and rolling resistance coefficient (Crr) must be specified in the LoadMaterial invocation. For users implementing a custom force model, it is imperative that material property names specified during the LoadMaterial function align with the array names in the force model file. For properties associated singularly with a material type (e.g., Young's modulus), one should utilize the offset variables bodyAMatType or bodyBMatType to retrieve the property pertinent to the contact material. Conversely, for properties defined between two materials (like the friction coefficient), both offset variables are employed concurrently to obtain the appropriate value for the contact, as elucidated in the subsequent code snippet.

```
// Material properties
float E_cnt, G_cnt, CoR_cnt, mu_cnt, Crr_cnt;
{
    // E and nu are associated with each material, so obtain them this way
    float E_A = E[bodyAMatType];
```

```
    float nu_A = nu[bodyAMatType];
    float E_B = E[bodyBMatType];
    float nu_B = nu[bodyBMatType];
    matProxy2ContactParam(E_cnt, G_cnt, E_A, nu_A, E_B, nu_B);
    // CoR, mu and Crr are pair-wise, so obtain them this way
    CoR_cnt = CoR[bodyAMatType][bodyBMatType];
    mu_cnt = mu[bodyAMatType][bodyBMatType];
    Crr_cnt = Crr[bodyAMatType][bodyBMatType];
}
```

In this implementation, because the force is set to be in the global frame, we do the calculation in the global frame. This requires us to compute the global angular velocity of the contact point on both contact geometries (albeit having the same location in space, the contact point on geometry **A** does not have the same velocity as that on geometry **B**, because of the intrinsic velocity that **A** and **B** have), since the user-referrable variables ARotVel and BRotVel only give their angular velocity in local frames. This section of the code does this task.

```
float3 rotVelCPA, rotVelCPB;
{
    // This is local rotational velocity (the portion of linear vel contributed by rotation)
    rotVelCPA = cross(ARotVel, locCPA);
    rotVelCPB = cross(BRotVel, locCPB);
    // This is mapping from local rotational velocity to global
    applyOriQToVector3(rotVelCPA.x, rotVelCPA.y, rotVelCPA.z, AOriQ.w, AOriQ.x, AOriQ.y,
        AOriQ.z);
    applyOriQToVector3(rotVelCPB.x, rotVelCPB.y, rotVelCPB.z, BOriQ.w, BOriQ.x, BOriQ.y,
        BOriQ.z);
}
```

Then the model calculates the normal force. Readers are referred to Sec. 2.3.1 to relate the implementation with the normal contact model. The material properties that are extracted previously, such as E_cnt, are used here to derive the force. One extra task carried out in this part is the update of the "wildcards" delta_tan_x, delta_tan_y, delta_tan_z and delta_time, which are used to record the friction history. The contact history is used in the friction and rolling resistance calculation. At the end of this snippet, the variable force is updated to

record the normal force.

```
// A few re-usable variables that might be needed for both the tangential and normal force
float mass_eff, sqrt_Rd, beta;
float3 vrel_tan;
float3 delta_tan = make_float3(delta_tan_x, delta_tan_y, delta_tan_z);

// Normal force calculation
{
    // The (total) relative linear velocity of A relative to B
    const float3 velB2A = (ALinVel + rotVelCPA) - (BLinVel + rotVelCPB);
    const float projection = dot(velB2A, B2A);
    vrel_tan = velB2A - projection * B2A;

    // Update contact history
    {
        delta_tan += ts * vrel_tan;
        const float disp_proj = dot(delta_tan, B2A);
        delta_tan -= disp_proj * B2A;
        delta_time += ts;
    }

    mass_eff = (AOwnerMass * BOwnerMass) / (AOwnerMass + BOwnerMass);
    sqrt_Rd = sqrt(overlapDepth * (ARadius * BRadius) / (ARadius + BRadius));
    const float Sn = 2. * E_cnt * sqrt_Rd;

    const float loge = (CoR_cnt < 1e-12) ? log(1e-12) : log(CoR_cnt);
    beta = loge / sqrt(loge * loge + deme::PI * deme::PI);

    const float k_n = 2. / 3. * Sn;
    const float gamma_n = 2. * sqrt(5. / 6.) * beta * sqrt(Sn * mass_eff);

    force += (k_n * overlapDepth + gamma_n * projection) * B2A;
}
```

In the following snippet, we calculate the rolling resistance. At the end of this snippet, the variable torque_only_force is updated to record the rolling resistance. Recall that this imaginary "force" contributes only the the contact torque, in agreement with the rolling resistance model in Eq. 2.2g.

```
if (Crr_cnt > 0.0) {
    bool should_add_rolling_resistance = true;
```

```
    {
        float R_eff = sqrtf((ARadius * BRadius) / (ARadius + BRadius));
        float kn_simple = 4. / 3. * E_cnt * sqrtf(R_eff);
        float gn_simple = -2.f * sqrtf(5. / 3. * mass_eff * E_cnt) * beta * powf(R_eff, 0.25
            f);

        float d_coeff = gn_simple / (2.f * sqrtf(kn_simple * mass_eff));

        if (d_coeff < 1.0) {
            float t_collision = deme::PI * sqrtf(mass_eff / (kn_simple * (1.f - d_coeff *
                d_coeff)));
            if (delta_time <= t_collision) {
                should_add_rolling_resistance = false;
            }
        }
    }
    if (should_add_rolling_resistance) {
        // Tangential velocity (only rolling contribution) of B relative to A, at contact
            point, in global
        float3 v_rot = rotVelCPB - rotVelCPA;
        // This v_rot is only used for identifying resistance direction
        float v_rot_mag = length(v_rot);
        if (v_rot_mag > 1e-12) {
            torque_only_force = (v_rot / v_rot_mag) * (Crr_cnt * length(force));
        }
    }
}
```

The following snippet implements the friction force. At the end of this snippet, the variable force is updated to record the friction force, and the contact history variables are updated (since we packed them together to be a float3 variable earlier for easier processing, we have to unpack it in the end for solver to pick up their changes in this round of force calculation), because of the potential clamping of tangential micro-displacement, see Eq. 2.2f.

```
if (mu_cnt > 0.0) {
    const float kt = 8. * G_cnt * sqrt_Rd;
    const float gt = -2. * sqrt(5. / 6.) * beta * sqrt(mass_eff * kt);
    float3 tangent_force = -kt * delta_tan - gt * vrel_tan;
    const float ft = length(tangent_force);
    if (ft > 1e-12) {
```

```
        // Reverse-engineer to get tangential displacement
        const float ft_max = length(force) * mu_cnt;
        if (ft > ft_max) {
            tangent_force = (ft_max / ft) * tangent_force;
            delta_tan = (tangent_force + gt * vrel_tan) / (-kt);
        }
    } else {
        tangent_force = make_float3(0, 0, 0);
    }
    force += tangent_force;
}

delta_tan_x = delta_tan.x;
delta_tan_y = delta_tan.y;
delta_tan_z = delta_tan.z;
```

The provided code snippets together constitute the complete Hertz–Mindlin contact force model implemented in DEM-Engine. For a practical example of a custom force model in application, users can refer to the DEMdemo_Electrostatic.cpp demo within the repository [62]. In this demo, elements are subjected not only to contact force but also to an electrostatic force.

2.4 Contact model validation and numerical experiments

As critical components in numerical simulations, the accuracy, robustness, and reliability of force models are paramount. The effective implementation within the DEM framework is not a trivial task, often subject to errors that may arise from computational approximations or algorithmic nuances. Therefore, rigorous validation tests are necessary to ensure that the model performs as expected and can reliably reproduce physical phenomena.

In this section, we introduce a series of numerical tests, from small-scale stacking tests to large-scale hopper flow rate tests, that are carefully designed to validate the implementation of DEM-Engine by comparing the simulations against the known physics. For the convenience of

Mode	Stationary	Sliding	Rolling	Sliding and rolling
Definition	$\omega = 0$, $v = 0$	$\omega = 0$, $v > 0$	$v = \omega r$	$\omega > 0$, $v > \omega r$

Table 2.3: The possible end status of the sphere in the rolling-on-incline test.

notation usage, for the rest of the thesis, variables have their scopes limited to the respective sections.

2.4.1 Sphere rolling on incline

We present a test borrowed from [86] where a sphere rolls up an incline. In this test, a sphere of radius $r = 0.2\,\text{m}$ and mass $5\,\text{kg}$ moves up on an incline with an initial velocity of $0.5\,\text{m/s}$, parallel with the incline and pointing up. In [86], the static friction coefficient μ_s and kinetic friction coefficient μ_k are allowed to have different values; however, in the default force model that we are validating, they assume the same value, and in this test $\mu_s = \mu_k = 0.25$. A test scene is illustrated in Fig. 2.6. The end status of the sphere can be one of the following modes depending on the incline angle α and rolling resistance C_r: Stationary; Sliding; Rolling; Sliding and rolling. These modes are defined by the final angular velocity ω and linear velocity v of the sphere, and are summarized in Table 2.3.

The outcome of this set of simulations is plotted in Fig. 2.7. It is shown in [86] that for the sphere to be stationary on the incline, $\alpha \leq \tan^{-1}(\frac{\mu_s}{\mu_k} C_r)$. For the sphere to roll down the incline without sliding, $\alpha \leq \tan^{-1}(3.5\mu_s - \frac{5}{2} C_r)$. These two conditions are plotted in Fig. 2.7 as the dashed and solid lines respectively, which evidently separate the stationary region, pure rolling region, and sliding–rolling mixed region as the theory suggests.

2.4.2 Sphere stacking

A set of three-sphere stacking tests were carried out to further validate the friction model implementation. This experiment is borrowed from [86, 49]. For each test, two identical spheres of mass $m_1 = 1\,\text{kg}$ and radius $R = 0.15\,\text{m}$ with a small gap d in between were settled

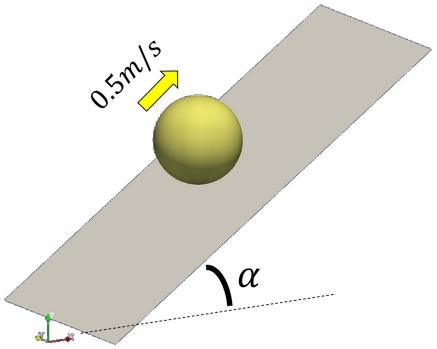


Figure 2.6: A rendering of the sphere moving up an incline.

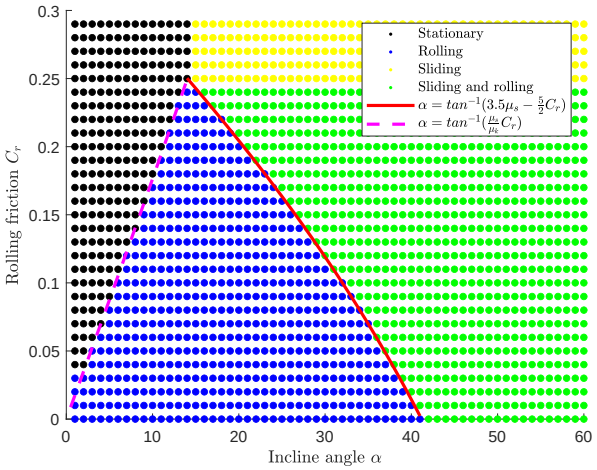


Figure 2.7: The end status of the sphere can be one of the following modes.

on a flat surface. A third sphere of the same radius R but a different mass was placed between and above the bottom spheres with zero initial velocity, as illustrated in Fig. 2.8. To minimize the influence of impact, the third sphere was initialized in contact with the bottom ones. Depending on m_1, the gap, and rolling resistance coefficient C_r, two scenarios can occur: The top sphere drops to the ground, or it moves down slightly but the structure eventually stabilizes with the bottom spheres supporting the top sphere. This is a type of physics that also comes into play on a larger scale in angle of repose experiments. For different selections of C_r, the mass of the top sphere was increased by 0.02 kg to find the critical mass m_2 for the pile to collapse, and the result is demonstrated in Fig. 2.9. The critical masses found for all initial gap sizes show exact matches with the outcome reported in [86], validating our force model implementation.

2.4.3 Ball impact test

We replicate a series of sphere-impact tests based on the experimental procedure detailed in [89]. A spherical projectile characterized by diameter D and density ρ_b was released from varying heights, h, onto a loosely packed pile of granular material, visualized in Fig. 2.10.

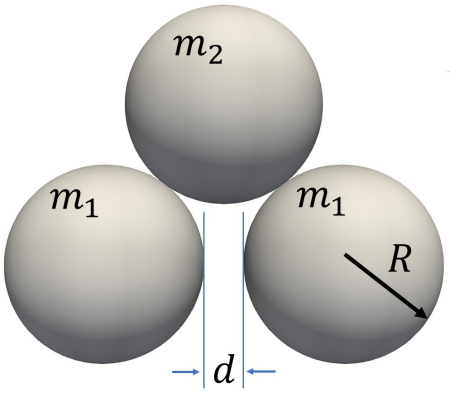


Figure 2.8: A rendering of the sphere-move-up-incline test.

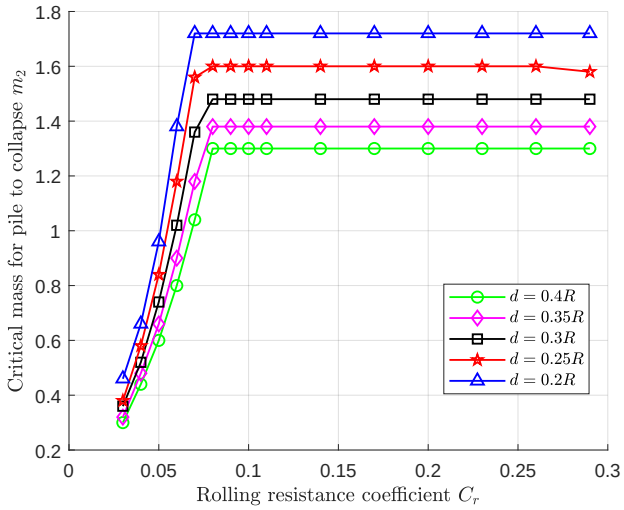


Figure 2.9: The end status of the sphere can be one of the following modes.

The resulting penetration depth d of this sphere was gauged and set against the empirical model derived from the experimental data in [89]

$$d = \frac{C}{\mu}\left(\frac{\rho_b}{\rho_g}\right)^{\frac{1}{2}} D^{\frac{2}{3}} H^{\frac{1}{3}}, \tag{2.4}$$

where ρ_g denotes the granular material's bulk density, and $H = h + d$ is the sum of penetration depth and drop height. In [89], the constant C is estimated from experiments to be $C = 0.14$.

Twelve numerical tests using DEM-Engine were executed aiming to reproduce the experiment in [89] as faithfully as possible. These tests incorporate combinations of projectile densities $\rho_b = 2.2$, 3.8, 7.8, 15 g/cm^3, resembling Teflon, ceramic, steel, and tungsten, respectively. The diameter of the spherical projectile is $D = 2.54$ cm. The release heights take values $h = 5$, 10, 20 cm. Each simulation uses eleven types of spherical elements with diameters evenly distributed in the range between 0.25 cm and 0.35 cm (inclusive), and each DEM element has an even chance of spawning as one of them. The grain material in use has density $\rho_{\text{grain}} = 2.5$ g/cm^3, resembling silica. This is to be differentiated from the bulk density of the granular bed, which is packed at $\rho_g = 1.46$ g/cm^3, with a sliding friction coefficient of

$\mu = 0.3$.

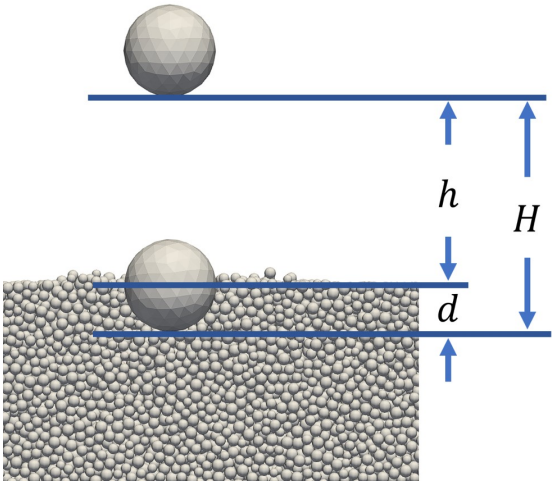


Figure 2.10: Diagram of the initial and final projectile positions.

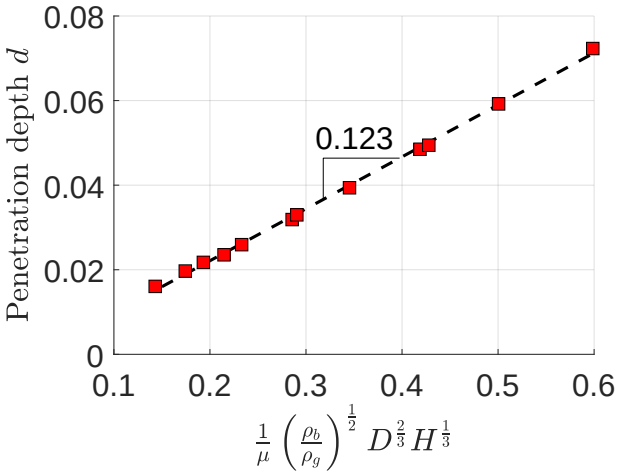


Figure 2.11: Penetration depth. Each red square represents a data point in the numerical test.

The correlation between depth d and the adjusted total release height H can be observed in Fig. 2.11. The line represents a linear regression of the numerical outcomes, showing a slope of 0.123, which confirms the experimentally established empirical model in Eq. 2.4. Comparable outcomes were also documented in [90] and [49], where both non-smooth and smooth contact dynamics approaches were leveraged for validating the same physics.

2.4.4 Flow sensitivity test

DEM-based numerical tools prove highly valuable in probing the dynamic behavior of granular flows, an area that leaves most of the challenging aspects of granular flow research untackled. This section investigates the flow characteristics of granular phases with heterogeneous properties, including variations in shape, density, and friction coefficient, using DEM-Engine.

2.4.4.1 Drum tests

First, we investigate the flowability of particle media that are comprised of four typologies: plastic spheres, plastic cylinders, wooden spheres, and wooden cylinders. The reference data

is presented in [1], in which experimental and numerical tests were performed on spherical and nonspherical particles. The experimental setup for the estimation of the angle of repose, a schematic of which is proposed in Figure 2.12, comprised a rotating drum with an inner diameter (D_d) of 0.19 m and a depth of 0.20 m (W_d), and was made of transparent acrylic. For this investigation, the considered physical test outcomes refer to the test performing the drum rotating angular velocity, $\dot{\theta}_d$, of 3.6 revolutions per minute (rpm).

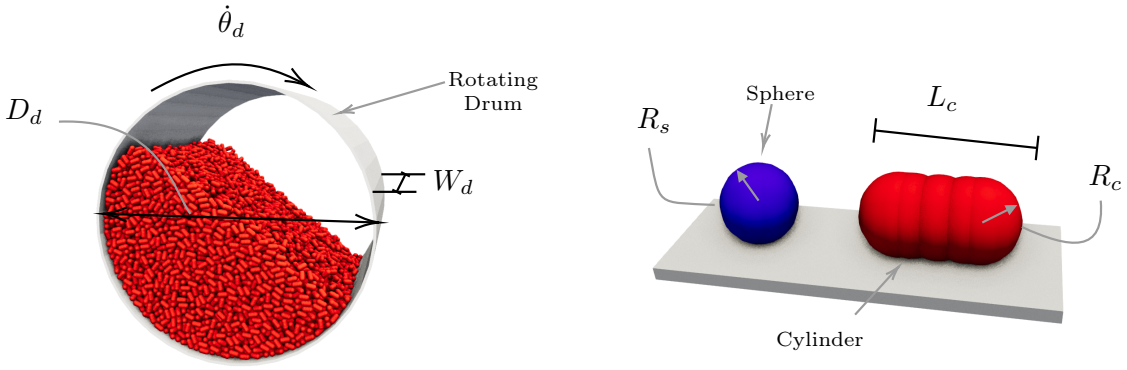


Figure 2.12: Schematic visualization of the drum rotating drum test.

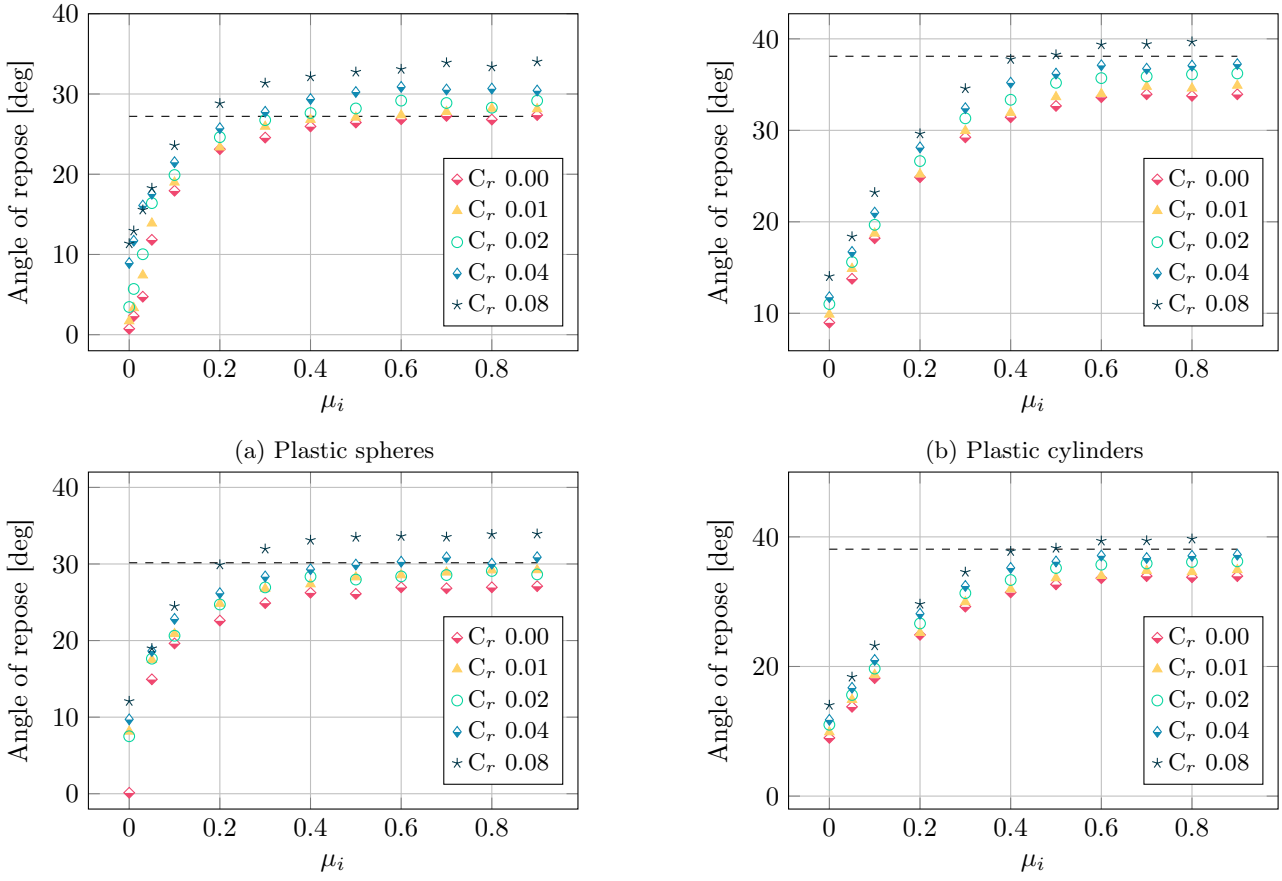
Table 2.4: Properties of four different particle setups used in this numerical investigation.

ID	Material	Shape	Radius [mm]	Length [mm]	Density [kg/m^3]	E [MPa]	ν [-]	CoR [-]
PS	Plastic	Sphere	3.0	-	1592	10.0	0.35	0.85
PC	Plastic	Cylinder	2.0	8.0	1128	10.0	0.35	0.85
WS	Wooden	Sphere	2.95	-	674	10.0	0.35	0.55
WC	Wooden	Cylinder	2.0	8.5	476	10.0	0.35	0.55

This test is also considered to assess the accuracy of DEM-Engine in simulating complex shapes represented by clumps. In the following, as shown in Fig. 2.12, the two shapes that characterize the tested particles comprise pure spheres with uniform radii, and five sphere clumps to mimic the geometric outer shape of cylinders.

Fig. 2.13 proposes the sensitivity of the attained numerical angle of repose when the rotating drum test is used. Each plot refers to a different material setup proposed in Table 2.4,

using a test matrix that spans over 13 values in [0.00, 0.80] for the definition of the inner friction (μ_i) and 5 in [0.00, 0.08] for the definition of the rolling friction (C_r). According to the shape of the particles, the material is initialized to fill half of the volume of drum; then, the drum initiates its rotation at a constant angular velocity of 3.6 rpm, and let run for two seconds, after which it is assumed the system achieves a steady state. The angle of repose, as reported in the charts, is computed as the mean value of thirty measurements taken at an interval during three seconds of simulation. Little deviation is observed throughout the post-processing phase, for all the cases.



(a) Plastic spheres

(b) Plastic cylinders

(c) Wooden spheres

(d) Wooden cylinders

Figure 2.13: Sensitivity of the angle of repose to the inner (μ_i) and rolling friction (C_r) for the four granular materials in Table 2.4. The dashed line in each chart reports the reference value for the corresponding experimental test [1].

Using the reference solution from the experimental results in [1], and reported in Fig. 2.13 as dashed black lines, the accuracy of the DEM in simulating granular media can be evaluated. The concordance is commendable, and Fig. 2.13 suggests some of the main features of granular materials when simulated using a DEM approach. First, it is easy to spot that when the internal friction assigned to the spheres tends to zero, the system response provides very small angles of repose, achieving a horizontal surface for no internal friction. Conversely, for cylinder particles without inner friction, the shape itself is capable of providing some bearing capacity to the system, as expected. Additionally, rolling resistance affects the angle of repose when μ_i is small, whereas the divide for cylinder particles with and without C_r is consistently lower.

2.4.4.2 Hopper tests

The following assesses the dynamic properties exhibited by a flow of DEM particles when simulated using the DEM-Engine. As reference solutions for this task, data regarding the mass discharge rate for both single and binary component systems are targeted, as made available in [91]. The physical testing was conducted using a flat-bottom hopper, as illustrated in Fig. 2.14.

This hopper possesses a total height of 0.40 m, a width of 0.20 m, and a depth of 0.04 m. An orifice of 0.055 m is symmetrically positioned on the lower surface, extending throughout its depth. For the experimental campaign, various particle configurations were investigated. However, for the purpose of this numerical validation, only four configurations, which precisely correspond to those outlined in Table 2.4, are considered. Specifically, Table 2.5 provides details on the hopper configuration for the tests presented in the subsequent sections. The parameters μ_i and C_r reported in the last two columns are set using the charts in Fig. 2.13. Note that the first two tests consist of single-component discharge tests, whereas the remaining, binary particle compositions are used.

In Fig. 2.15, the relative mass discharge is presented for test number 1 and 2, which

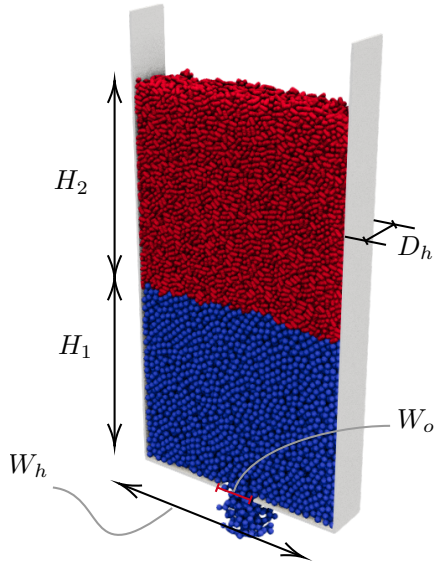


Figure 2.14: Schematic visualization of the flat-bottom hopper.

Table 2.5: Properties of four different particle combinations used in the hopper numerical investigation.

Test no.	Layer 1	Layer 2	H_1 [cm]	H_2 [cm]	μ_i [-]	C_r [-]
1	PS	-	36	-		10.0
2	WC	-	36	-		0.50
3	PS	PC	18	18	0.50	0.05
4	PC	PS	18	18	476	10.0

involve plastic sphere and cylinder particles, respectively. The chart depicts a comparison between the experimental and numerical time evolution of the system, showcasing the mass discharge relative to the total mass. For both tests, DEM-Engine demonstrates a fair level of accuracy in predicting the flow evolution. It exhibits an excellent match for the purely spherical material (PS), while consistently slightly overestimating for the cylinders (PC). This discrepancy, leaning towards a more *fluid* flow, can be attributed to the fact that the clumps of five spheres, used in place of cylindrical shapes, do not perfectly replicate the

behavior of the physically consistent cylinders.

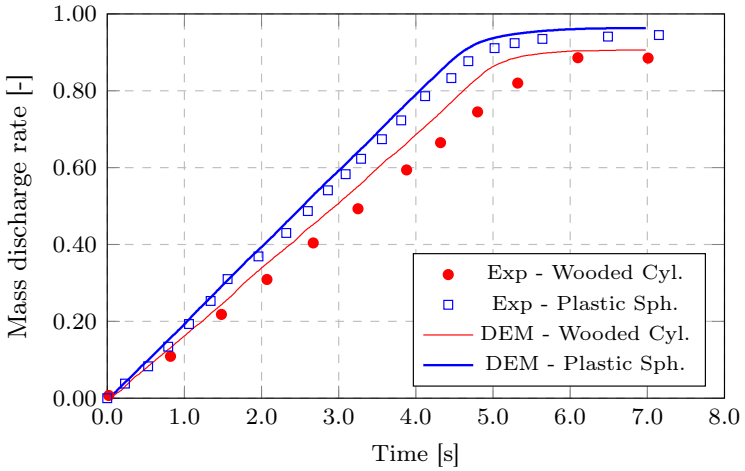


Figure 2.15: Experimental and numerical comparison of the mass discharge ratio for single component hoppers with plastic spheres (blue) and wooden cylinders (red).

In Fig. 2.16, a visual comparison is provided for the binary particle systems, test number 3 and 4 as outlined in Table 2.4. This comparison contrasts snapshots from both experimental and numerical perspectives, offering lateral views of the hopper at one-second intervals, starting from the initial configuration at Time = 0.00 s. The first and third rows respectively present data from [91], while the second and fourth rows showcase the results from DEM-Engine's simulation. The two timelines evolve in a remarkably similar fashion, highlighting that the numerical model accurately captures all the pertinent physical phenomena that unfold.

2.5 Performance

The scaling analysis in this section seeks to offer insights into the expected simulation performance of DEM-Engine. The chosen test scenario involves a bladed mixer interacting with granular material, where the mixer is modeled using a triangular mesh. Throughout the simulation, the mixer blades maintain a constant angular velocity of 2π rad/s. Initially, the elements are positioned within a cylindrical region with a radius of 0.5 m and a height of 1/3 m

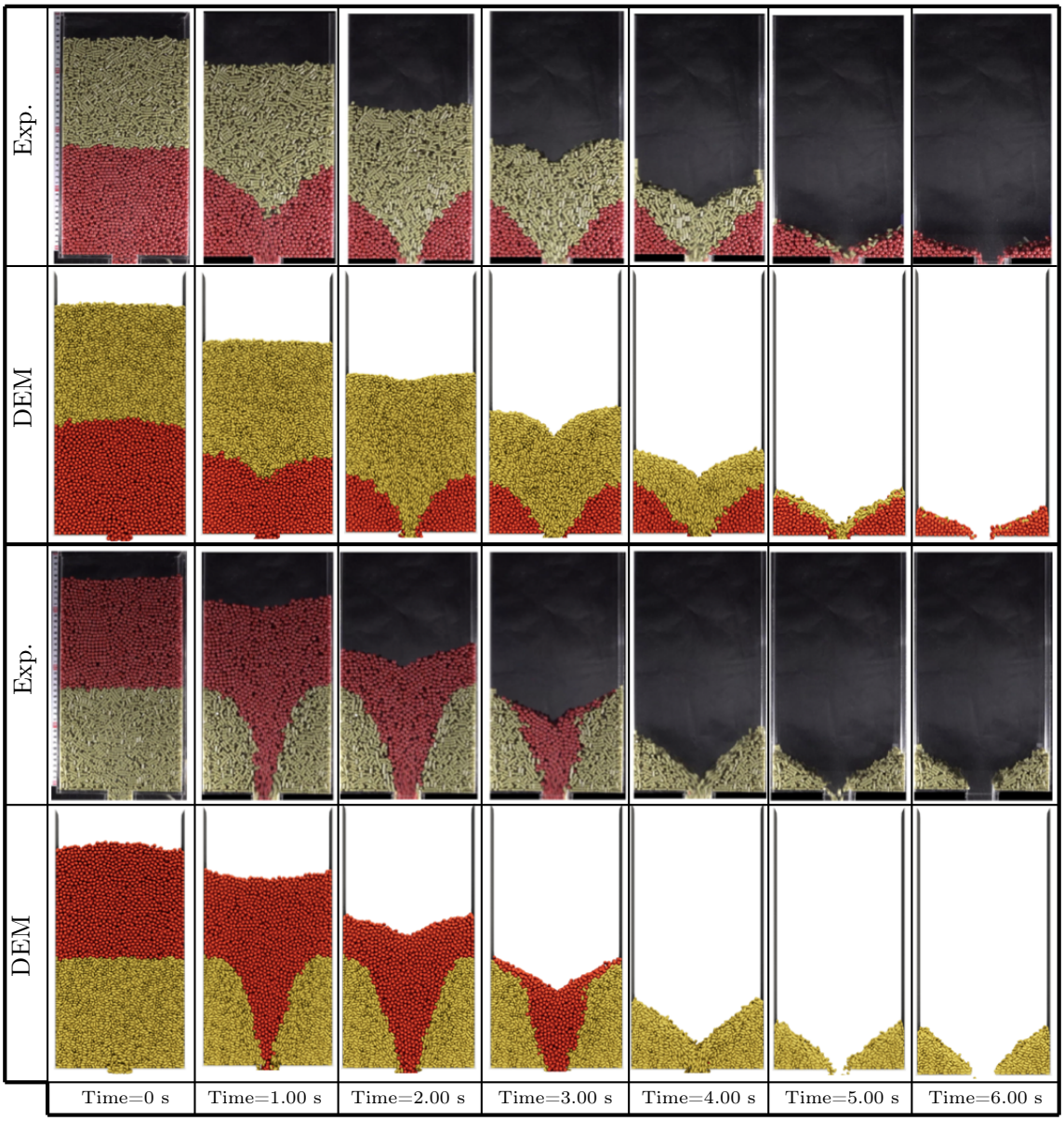


Figure 2.16: Experimental and numerical comparison of the discharging behavior of two different packing patterns of the Plastic Sphere and Cylinders.

above the mixer, and are subsequently released at the simulation's onset. The test's selection is due to its intensive particle–particle and particle–mesh interactions, demonstrated in Fig. 2.18. This puts the contact history preservation algorithm to the test, as contacts emerge and vanish in this highly dynamic problem. Essential material properties and simulation parameters can be found in Table 2.6.

In our analysis, we employ three clump types: individual spheres, three-sphere clumps, and six-sphere clumps, depicted in Fig. 2.17. Element sizes are adjusted to regulate the total element count. The mesh representing the mixer blades remains consistent across simulations, comprising 2892 triangular facets. Simulations are run until a pseudo-steady state is achieved at 1 s, post which the wall time required for 10^6 time steps is recorded. Fig. 2.19 displays the correlation between wall time and the total number of component spheres (distinct from the number of elements) via blue, green, and black markers. The simulations are performed on two NVIDIA Ampere A100 GPUs. On average, Chrono DEM-Engine takes 0.546, 0.313, and 0.264 hours to complete one million steps for every million component spheres in the simulations for the individual spheres, three-sphere clumps, and six-sphere clumps, respectively. The linear scaling persists to up to 150 million component spheres in our tests.

An identical simulation is also executed with Chrono::GPU (utilizing only one A100 as Chrono::GPU is limited to using a single GPU), and its scaling is represented with red markers. This juxtaposition is pertinent given a recent independent study's findings, which underscored that Chrono::GPU outperforms two other established DEM packages by two orders of magnitude [51]. Therein, for a 420 000-element pebble-packing simulation, Chrono::GPU running on a laptop GPU finished the simulation in an amount of time 261 times shorter than that required by LAMMPS, when the latter ran on 432 CPU cores of a cluster. For a 660 000-element pebble-packing simulation, Chrono::GPU executed 501 times faster than STAR-CCM+, which ran on 160 CPU cores. In both tests, Chrono::GPU ran on the RTX 2060 Mobile NVIDIA GPU card of a laptop. As indicated in Fig. 2.19, Chrono DEM-Engine demonstrates an additional twofold efficiency boost over Chrono::GPU in the

test case of spherical elements. Owing to its ability to handle complex DEM particle shapes, we deduce that Chrono DEM-Engine expands the modeling capacity of its antecedent without compromising per-GPU efficiency.

Table 2.6: Some material and simulation properties used in the mixer scaling analysis.

Density [kg/m^3]	E [Pa]	ν [-]	CoR [-]	Step size [s]
2.6×10^3	1×10^9	0.3	0.2	5×10^{-7}

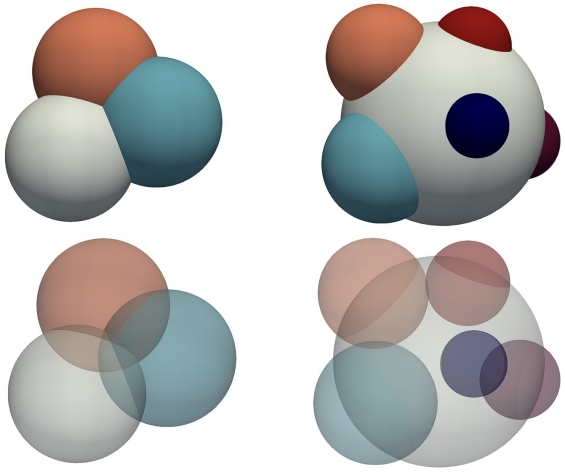


Figure 2.17: The element shapes for the three-sphere and six-sphere clumps.

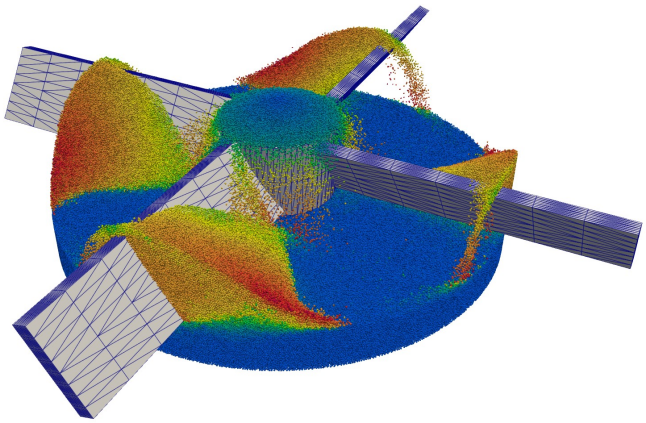


Figure 2.18: A rendering of the mixing process.

Figure 2.20 shows the itemized time consumption associated with the important steps in the kinematics and dynamics threads' work cycles in the largest six-sphere-clump mixer simulation presented in this scaling analysis. In this simulation, the amount of mutual contact data produced is relatively large, causing the kinematics thread to spend a large amount of time transferring it to the dynamics thread, reaching 26% of the former thread's total runtime. The dynamics thread spends minimal time on transferring data. This is done by design to enable the dynamics thread to almost exclusively focus on advancing the state of the system forward in time.

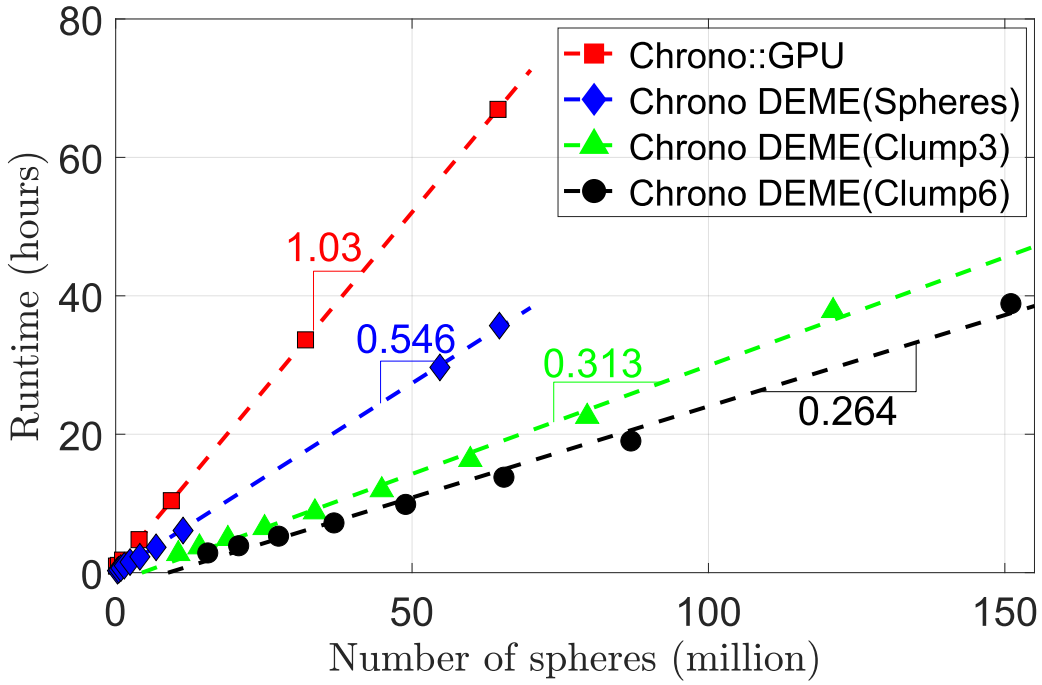


Figure 2.19: The scaling result of the mixer simulation using individual spheres, three-sphere clumps, and six-sphere clumps, on NVIDIA A100s. The wall time to finish simulating 10^6 steps is plotted against the number of component spheres in the simulation.

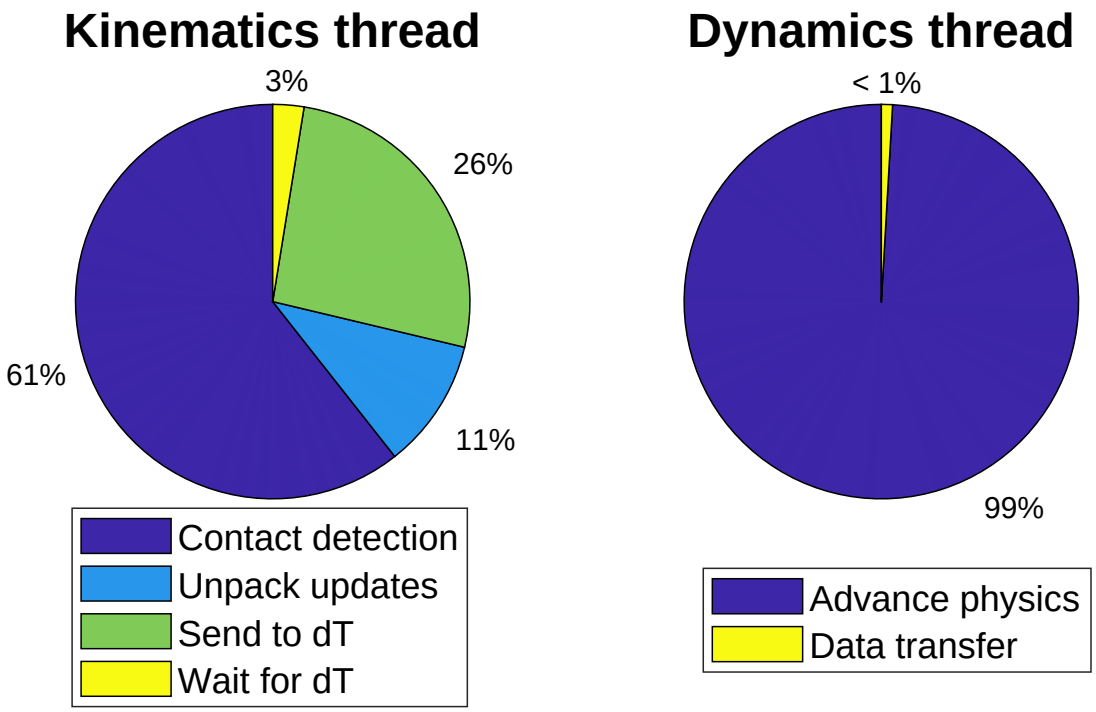


Figure 2.20: The runtime breakdown for the kinematics and dynamics threads, during the lifespan of the largest six-sphere-clump mixer simulation.

3 EXTRATERRESTRIAL SIMULANT DESIGN AND ROVER SIMULATION

In this section, owing to the modeling interests, the focus is shifted to a Chrono DEM-Engine-based simulation of a particular type of granular material. Specifically, we use a "Digital Simulant" (DS) inspired by GRC-1 [2], the latter developed as a lunar soil replica for Earth testing of rovers and similar implements. The GRC-1 simulant contains cohesionless and frictional particles that have complex shapes and span a range of sizes. The fact that some GRC-1 particle sizes are micron-scale posed insurmountable challenges to the simulator. Despite using a customized way of storing contact information that goes beyond what the IEEE double precision data type can offer in C/C++, see [62], working with the actual particle size proved intractable owing to the broad spectrum of sizes—from microns to millimeters. As a compromise, the DS design herein maintains the statistical size *distribution*, yet uniformly increased by a factor of 20 the actual particle sizes encountered in GRC-1. In other words, the DS size distribution is identical to GRC-1, but the DS sizes are shifted to larger values. The value 20 mentioned above has no particular meaning other than being the smallest value that we had to scale by so that the DEM simulation became acceptable, both in terms of results accuracy and time to completion. The simulation experiments of Sec. 3.2.1.1 confirm that this "scaling up" of the particle sizes does not significantly compromise the accuracy of the results obtained through simulation.

The DS consists of seven DEM element types, each with a specific size and percentage of the total weight, see Table 3.1. The size distribution is plotted in Fig. 3.1. Therein, although the size distribution is not continuous, it is shown with seven line segments, each corresponding to the weight percentage contributed by each element type. Figure 3.2 illustrates the DS grain shapes. Out of the seven distinct grain types, the two larger ones are made of six overlapping spheres and have a flat triangular shape. The five smaller types are each made of three component spheres. All of them have a 120° rotation symmetry. Having a triangular

aspect, the particle size reported in Table 3.1 is measured as the diameter of the bounding sphere. The radii of the elements' component spheres and the material properties used in the numerical tests throughout the paper are also summarized in Table 3.1.

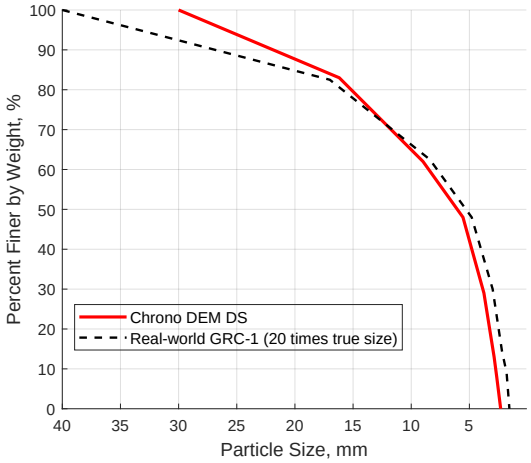


Figure 3.1: The size distribution of the DS, plotted against a scaled GRC-1 simulant size distribution.

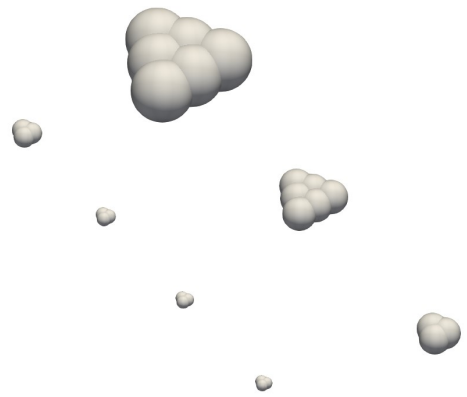
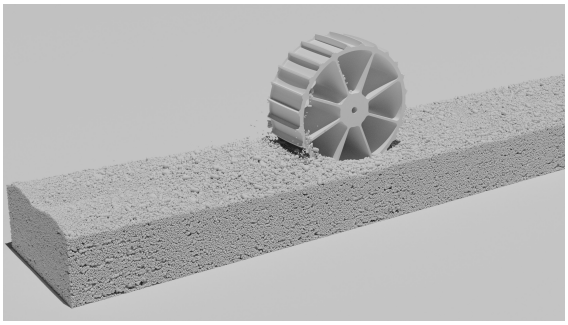


Figure 3.2: The seven clump shapes that show up in the DS.

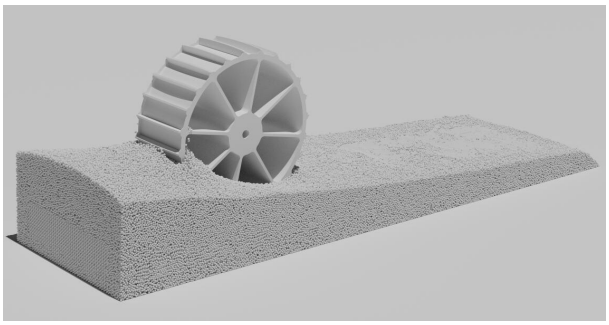
Type	1	2	3	4	5	6	7
Size [mm]	21	11.4	6.6	4.5	3	2.75	2.5
Component radius [mm]	3.6	1.95	1.81	1.24	0.82	0.75	0.7
%, by weight	17	21	14	19	16	5	8

Table 3.1: The weight distribution for the DS, percent-wise, by particle size. For all particle types, $E = 10^9\,\mathrm{N/m^2}$, $\nu = 0.3$, $\mu_s = 0.4$, and CoR $= 0.5$.

The nontrivial shape of the elements turned out to play an important role in the dynamics of the granular material. Compared to monodisperse granular material, which is commonly used in DEM simulations, the DS allowed for geometric locking, thus shaping the granular material's behavior, especially in shear-dominated experiments. Qualitatively, this observation is backed by the early results reported in Figs. 3.3a and 3.3b, wherein the same single-wheel slippage experiment was carried out once using the DS and then monodisperse granular material.



(a) The single-wheel test on a 25° incline, with DS.



(b) The single-wheel test on a 25° incline with spherical DEM elements. Snapshot suggests that monodisperse granular material support for a single wheel test is qualitatively different than what behavior observed for DS testing.

Figure 3.3: The renderings show that, on the same 25° incline, the single-wheel simulation on spherical DEM elements leads to much more terrain shearing compared to simulation on DS.

The simulations shown in the following sections consist of millions to tens of millions of DEM elements, a scale that is manageable on one or two modern GPU devices. If the true GRC-1 grain sizes were used, the number of elements would increase by close to four orders of magnitude, which places an insurmountable barrier to simulation owing to memory constraints, as well as arithmetic precision limits as some deformations δ_n become prohibitively small. Note that the smallest sphere component in a DS clump had a radius of approximately 700 µm.

3.1 Validation studies

The validation and refinement of a DEM representation is an arduous and lengthy process that spans an extended period. Herein, we report on three experiments that were used for the early validation of the DS design. The results in this section represent a confidence-building step before using the DEM-Engine to conduct additional studies for which there is limited or no experimental data available. The code used in Sec. 3.1 is available in a GitHub public repository [92] for unrestricted use, distribution, and reproducibility studies. The only caveat

is that the wheel model shared in the public repository is only an approximation of the actual wheel used in this study. The shared model was put together using data available in the public domain; the actual wheel model is not available for public access.

3.1.1 Sample preparation

This subsection describes the preparation strategy employed to produce a bed of granular material. For monodisperse spherical DEM elements, one can create the initial elements using a sampling algorithm, such as hexagonal close-packed, with a separation factor slightly larger than the diameter of each sphere. For the DS, due to the irregular grain shape and large variation in grain size, the separation factor needs to be larger than the size of the largest element to prevent overlap at the start of the simulation. This leads to a low space occupancy during the initial sampling process, as shown in Fig. 3.4a. Therefore, several batches of clumps need to be generated and settled to form a terrain sample of a certain depth, as depicted in Fig. 3.4b and 3.4c. The element size distribution shown in Fig. 3.1 is retained during the spawning of each DS batch. After ten batches of clumps settle, we obtain the resultant sample illustrated in Fig. 3.4d.

With this newly generated thin layer of clumps, we use a "copy-paste" process to accelerate the sample preparation. We duplicate several copies of the existing clumps, then add them back to the simulation, as shown in Fig. 3.4e. After settling, we create a virtual compressor object in the simulator and compress the sample to make the surface relatively even, see Fig. 3.4f. This terrain sample now is one meter by one meter in width and roughly 15 centimeters in height. We refer to it as a base DS patch in this paper. On two NVIDIA A100 GPUs, finishing the entire sample preparation process at time step size 10^{-6} s takes approximately 12 hours. There are 4 571 136 clumps, totaling 13 993 536 spheres that are present in the base DS patch.

The creation of this base patch is for the ease of scaling up the simulation. When a larger test bed is needed, several such patches can be instantiated, moved, and settled in the

simulator to create the test bed. Conversely, a subset of this patch can be extracted to create a smaller test bed. This saves the time that would otherwise be spent on settling the DEM terrain.

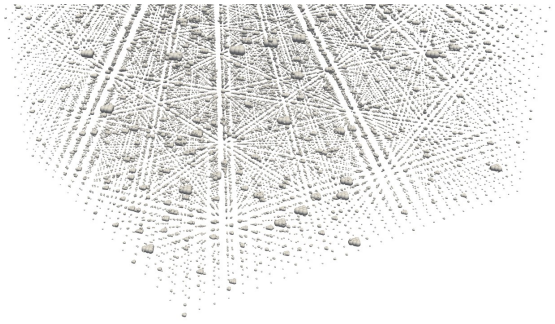
3.1.2 Repose angle validation

The angle of repose is expected to be equal to the residual internal friction angle [93]. The latter serves as a proxy for a material's shear strength that dictates a vehicle's climbing ability in off-road conditions, a topic discussed in Sec. 3.2. For the repose angle test, the initial sample is prepared by taking a cylindrical portion out of a DS patch, making three extra copies, and then translating everything into a funnel defined via a mesh, see Fig. 3.5a. The material flows through the funnel under gravity. The pile formed underneath can be seen in Fig. 3.5b. Figure 3.6 displays a 30° angle of repose, in line with results reported in [2] which is 29.8°. In total, 731 060 clumps participate in the simulation. On two NVIDIA A100 GPUs, running this 30 s simulation at time step size 10^{-6} s takes approximately 19 hours. There are 2 234 544 spheres that combine to make up the clumps in the material used for this test.

3.1.3 Cone penetration test

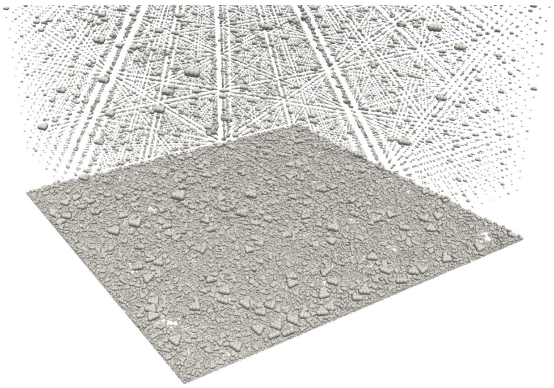
The cone penetration test is useful for gauging the soil's stiffness and density, as well as the stability of slopes and the potential for landslides [94]. Herein, we test in simulation the penetration experiments conducted with GRC-1 simulant reported in [2]. Same as in [2], the soil bin was selected to be 58.4 cm in diameter and 24 cm in depth, which is sufficient to minimize boundary effects [95].

Similar to the repose test, the initial sample is prepared by taking a cylindrical portion out of a DS patch, making two extra copies, and then translating everything into the cylindrical container and letting the material settle. Subsequently, a cone penetrates the sample with a constant velocity of 3 cm/s, see Fig. 3.7b. The cone, whose geometry is specified by a mesh, is shown in Fig. 3.7a. The cone has a base area of 323 mm^2 and an opening angle of 60°. The



(a) The clumps need to be generated in simulation with initial gaps determined by the biggest clump to avoid overlapping. This prevents a direct generation of a densely packed granular patch.

(b) The initial batch of clumps, after settling.



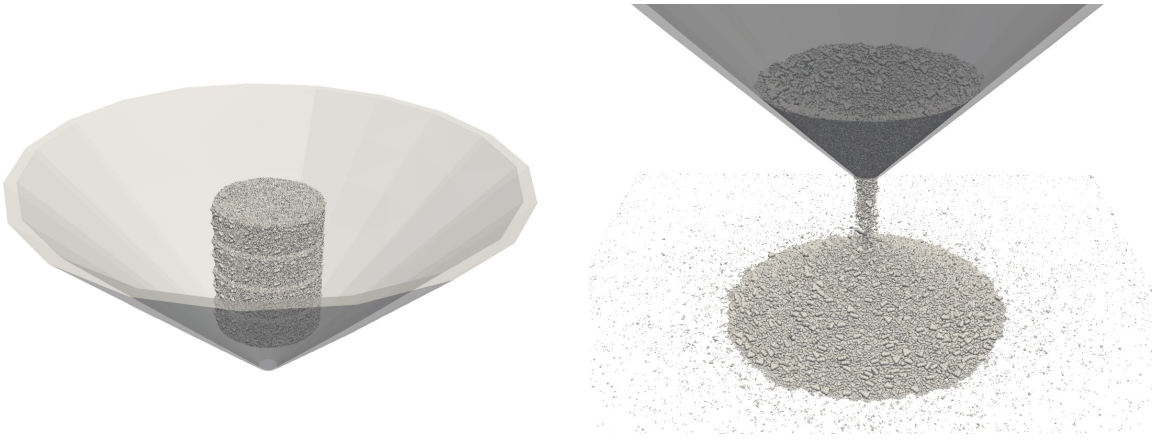
(c) The process is repeated and more batches of clumps are instantiated and let to be settled.

(d) The resultant granular patch after ten batches of clumps settle.

(e) To speed up the process, copies of the previously settled granular patch are instantiated into the simulation.

(f) The final settled and compressed base DS patch.

Figure 3.4: The rendering of DS preparation process.



(a) The initial sample is prepared using the pre-settled patch.

(b) A perspective view of the simulation.

Figure 3.5: The rendering of DS angle of repose test.

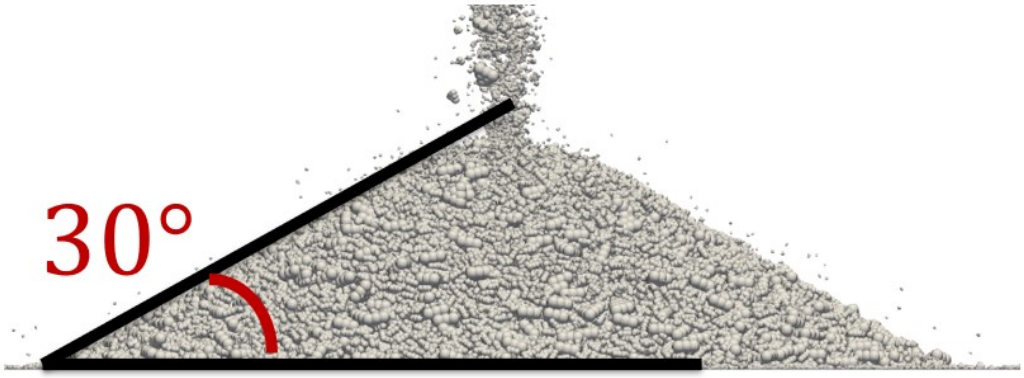
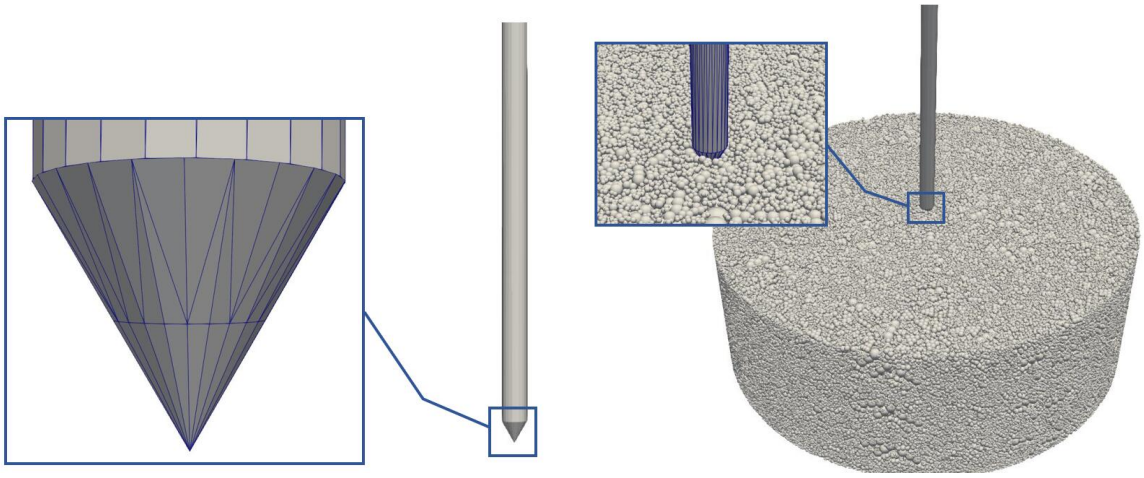


Figure 3.6: The settled DS forms a 30° pile.

contact force on the cone is measured and plotted against the penetration depth in Fig. 3.8. The different bulk density densities of DS in Fig. 3.8 are reached by compressing the material bed before the simulation starts.



(a) The cone mesh.

(b) A rendering of cone penetration test when the cone hits the sample.

Figure 3.7: The rendering of the cone penetration test using the DS.

As shown in Fig. 3.8, the DS reproduces a depth–pressure relationship in the cone penetration tests similar to the one noted in real-world experiments [2]. The simulator also reproduces the drastic increase of the pressure on the cone when the simulant is compressed to a higher bulk density, from $1.6\,\mathrm{g/mm^3}$ to $1.8\,\mathrm{g/mm^3}$. In all the subsequent numerical tests using the DS presented in the thesis, the DS operates at a bulk density around $1.6\,\mathrm{g/mm^3}$, which is close to what a naturally settled DS sample has.

In total, 773 097 clumps, which employ 2 362 698 spheres, participate in the simulation. On two NVIDIA A100 GPUs, running this 8 s simulation at time step size 10^{-6} s takes approximately 4 hours.

3.1.4 Single-wheel drawbar pull test

In [3], the authors used a test rig ARTEMIS to derive the force–slip relationship for a single Curiosity wheel operating on Mars soil simulant. We used the experimental setup described in their paper as a reference and demonstrated that our DS captures the terrain strength similar to existing simulants. Note that there is a slight discrepancy between the actual granular material used in [3] and the DS used in the simulations. This aspect will be revisited shortly.

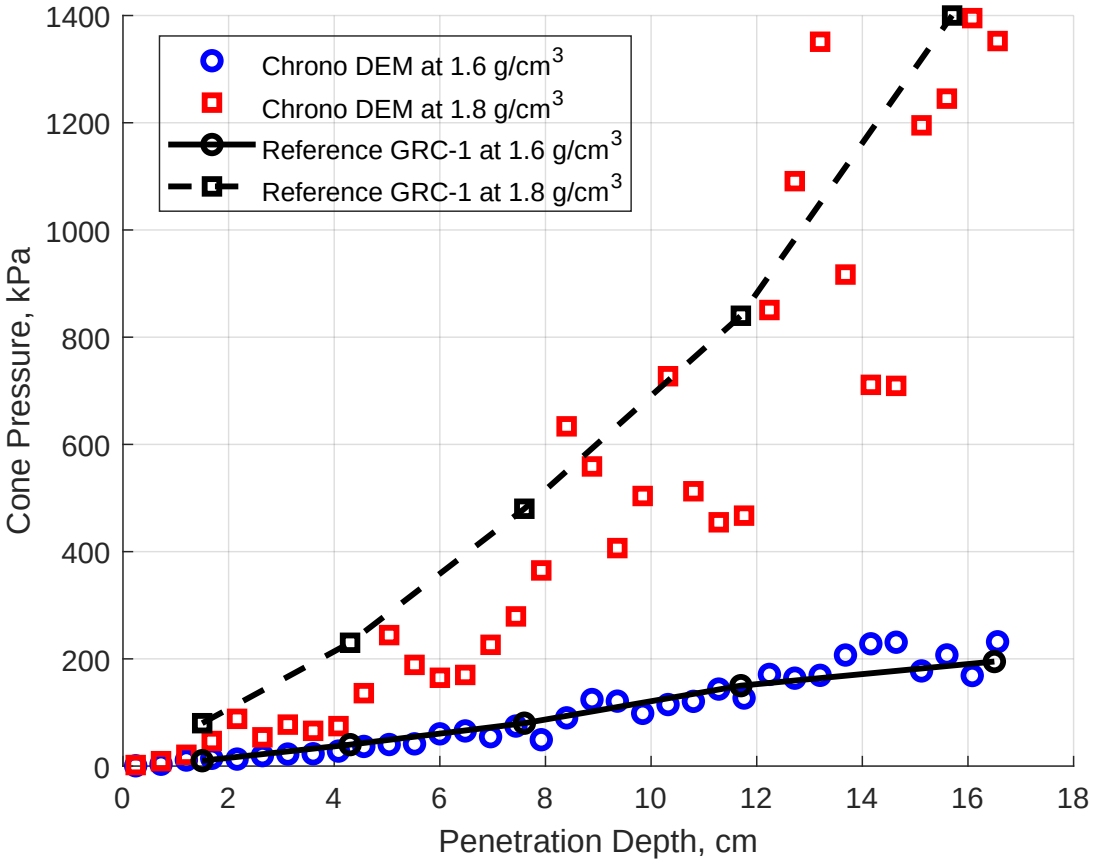


Figure 3.8: The DEM cone penetration test results with different bulk densities of the DS. They are plotted against the reference GRC-1 cone penetration experimental data. The experimental data are from GRC-1 simulant research paper [2].

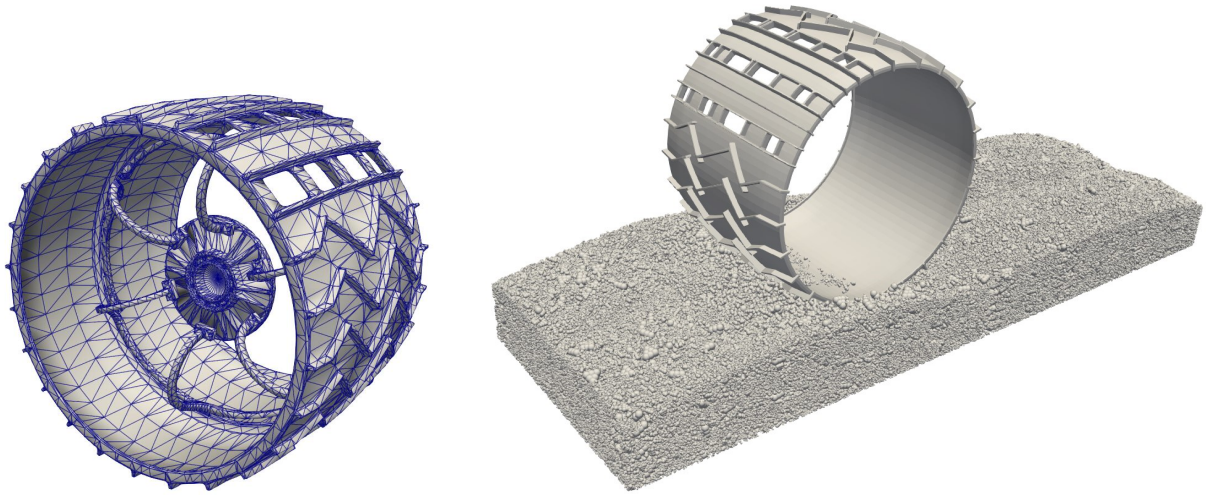
Reproducing drawbar pull tests in simulation involves firstly an abstraction of the test equipment. The terrain leveling and compacting process described in [96] has already been incorporated into the terrain preparation procedure discussed in Sec. 3.1.1. Additionally, enforcing a slip value does not require a mechanical system or attaching a hitch to the test vehicle as in the physical test; instead, specific linear and angular velocities are applied directly to the wheel.

In this test, we use Earth's gravitational acceleration $g = 9.81\,\mathrm{m/s^2}$. The Curiosity wheel is represented by the mesh shown in Fig. 3.9a. The grouser geometry is preserved exactly; the wheel has a radius of $r = 0.25\,\mathrm{m}$ and a weight of $750\,\mathrm{N}$. The mesh file can be obtained

from the Chrono repository [97], and readers are referred to [3] for comparison against the real Curiosity wheel. In this simulation, the wheel hub is removed to reduce the mesh size and save simulation time.

An overview of the simulation environment is illustrated in Fig. 3.9b. The granular terrain is prepared by making copies of the DS patch and concatenating these copies together so that a test bed is formed. A fixed angular velocity of $\omega = (\pi/12)\,\text{rad/s}$ is imposed on the wheel. The linear velocity v in the forward direction is enforced as $v = (1 - s)\omega r$, where s is the desired slip ratio.

The simulation results are presented in Fig. 3.10. The reference experimental data from [3] is shown with black markers in the same plot. Note that the simulations in [3] use a custom-made blend of two types of sand that is different from GRC-1, as we do not have access to the drawbar pull data of GRC-1. Therefore, this comparison is qualitative only. Nonetheless, our drawbar pull–slip curves exhibit notable similarities between our simulation and the reference experiment.



(a) Open-source mesh of the Curiosity wheel, publicly available in the Chrono repository [97].

(b) A perspective view of the wheel operating on the soil bed.

Figure 3.9: The mesh and the simulation scene of the drawbar pull test.

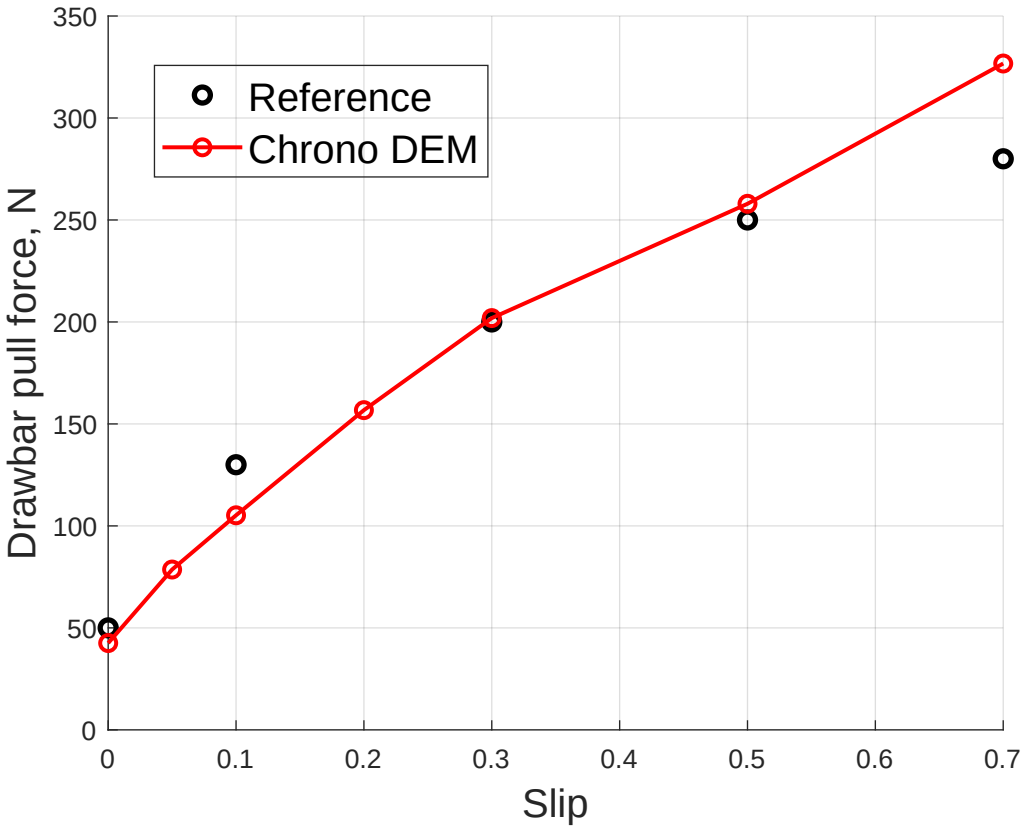


Figure 3.10: Curiosity wheel drawbar pull DEM simulation result with a vertical load of 750 N, compared against the reference ARTEMIS experimental data [3].

In total, 960 906 clumps employing 2 942 346 spheres participate in the simulation. On two NVIDIA A100 GPUs, running this 8 s simulation at time step size 10^{-6} s takes approximately 5 hours.

3.2 Further DEM studies related to rover mobility

This section focuses on single-wheel and full-rover incline tests. The limited amount of experimental data used for validation was produced by the Simulated Lunar Operations (SLOPE) laboratory at NASA's Glenn Research Center using GRC-1 and GRC-3 simulants. The wheel geometry used in the tests of this section is from NASA's Moon Gravitation

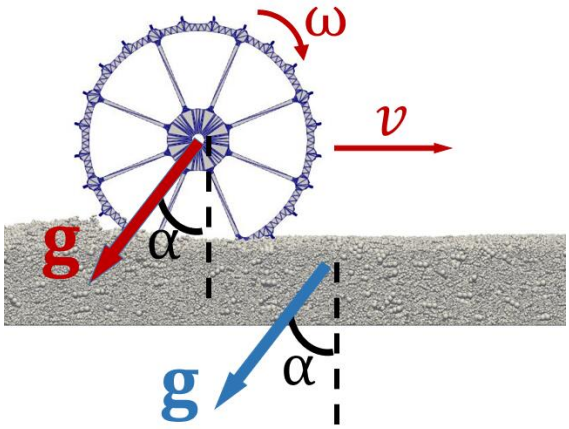
Representative Unit 3 (MGRU3).

3.2.1 Single-wheel slip test on incline

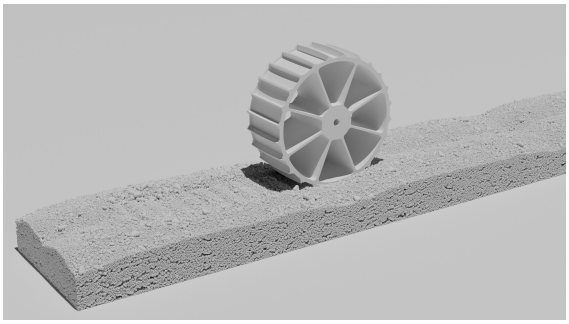
The inclines used in the simulation are created by altering the direction of gravity, see Fig. 3.11a. This approach to representing slopes requires minimum model adjustments when switching from one test scenario to another. Figure 3.11b is a screenshot of this test. The wheel in the test is subject to a fixed angular velocity ω and is free to move linearly. The steady-state average linear velocity over 6 seconds v is measured, and subsequently used to obtain the slip ratio $s = 1 - v/(\omega r)$. The wheel used has a mass of 5 kg but may have a different "effective" mass in these simulations. The "effective" mass is introduced to account for the chassis and suspension weight borne by each rover wheel. This modified mass is modeled as an extra gravity-aligned force applied to the center of the wheel. This force is dependent on the gravitational acceleration of the test scene. For example, if a wheel has an effective mass of 22 kg in a test that uses the Moon's gravity, that means an extra force of 17×1.62 N is applied to it. The granular terrain is prepared by making copies of the base DS patch and concatenating these copies together, similar to the previous drawbar pull test.

The ground-truth experimental data used for comparison is from the experiments done with MGRU3 climbing a "tilt bed" made with the GRC-1 simulant in the SLOPE lab. A picture of the test scene can be seen in Fig. 3.12, which is from a publicly available video of the test [98]. In this experiment, the wheels of MGRU3 were operating at $\omega' = 0.8$ rad/s under Earth's gravity $g = 9.81\,\mathrm{m/s^2}$. Note that the thickness of the bed of granular material has an influence on the experimental results, and the sufficient thickness to eliminate the boundary effect could depend on the vertical load. However, this secondary aspect, which might produce slightly different data in both experiments and simulations, falls outside the scope of this contribution.

Fig. 3.13a gives the slope–slip relationship for a wheel with an effective mass of 22 kg. Our first DEM experiment is set to match the experimental baseline test condition. The result



(a) A α-degree "incline" in DEM simulations is created by tilting the direction of the gravity.



(b) A rendering of single-wheel test scene.

Figure 3.11: The simulation setup for DEM single-wheel tests using DS.

Figure 3.12: MGRU3 climbing a "tilt bed" in NASA's SLOPE lab testing facility.

is shown with the red curve. Two additional single-wheel tests are subsequently conducted: the blue curve shows the result of a test with angular velocity $\omega = 1.96\,\mathrm{rad/s}$ under Earth's gravity; the green curve is associated with a test in with angular velocity $\omega' = 0.8\,\mathrm{rad/s}$ under moon gravity $g' = 1.62\,\mathrm{m/s^2}$. The similarity of the results indicates that the simulator captures the scaling law for locomotion in granular media [99, 100], which states that, if $g'/\omega'^2 = g/\omega^2$, the slip ratio at steady state is unchanged. At the same time, the fact that they are also close to the red curve suggests the slip ratio is relatively insensitive to the variation of the rotational speed, at least within the range we tested. These three test scenarios demonstrate that it is viable to reproduce the terrain response on the Moon by using the same simulant on Earth while keeping the rover *mass* constant. Using the 10° simulation as an example, we plot the linear velocity of the wheel along the incline direction against the 6-second time span in which we measure the average velocity v in Fig. 3.14. The linear velocity shows a well-defined mean value, meaning a steady state is reached during our measurement.

Next, we use another single-wheel test shown in Fig. 3.13b to demonstrate that matching the *ground pressure* under a wheel on the Moon and Earth is not an accurate approach for predicting the traction capability of a rover. The wheel in this test has an effective mass of 111 kg. If matching the ground pressure on Earth would reproduce the rover behavior on the Moon, then we would expect the blue curve in Fig. 3.13b to show a similar slip ratio at a given slope, compared to the red curve in Fig. 3.13a. However, this is not the case. To negotiate the same slope, a lot more slip is observed in this 111 kg test because the terrain elements also shear a lot more easily in the low-gravity condition. Note that the scaling law holds in this test condition too, as shown by the similarity between the two curves in this plot. The ground-truth experimental data used for comparison (black dashed line) are from ProtoInnovations' single-wheel experiments with the GRC-3 simulant, because the experimental data for 111 kg wheel operating on GRC-1 simulant is not available. Note that GRC-3 simulant has in general comparable properties to GRC-1, but contains

slit particles, and therefore this comparison is qualitative only. Those experiments were done using ProtoInnovations' in-house test bed, with the wheel operating at 1.96 rad/s under Earth's gravity $g = 9.81\,\mathrm{m/s^2}$.

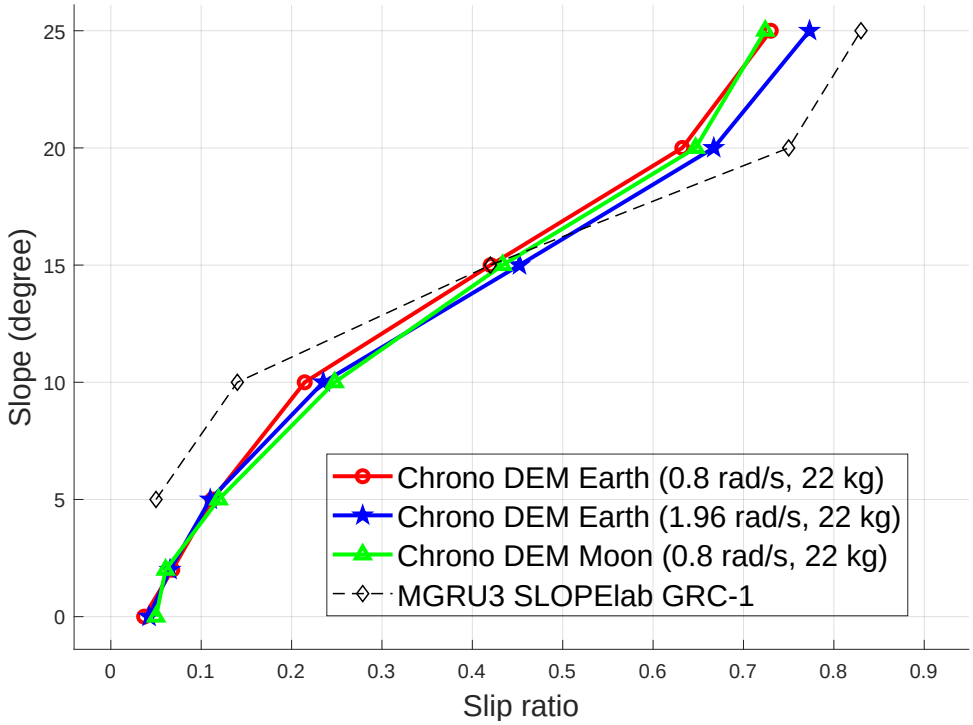
A number of 3 681 560 DEM elements employing 11 259 462 spheres are used in the simulations for which $\omega = 1.96\,\mathrm{rad/s}$. On two NVIDIA A100s, running these 8.4 s simulations at time step size 10^{-6} s took approximately 15 hours. For test cases where the angular velocity of the wheel is smaller, shorter test beds are used and the time cost is proportionally lower.

The simulation results obtained in this subsection are in agreement with the general scaling relation for locomotion in granular media [100] and they underline an important fact: to carry out single wheel tests on Earth that display slope vs. slip curves similar to the ones obtained on the Moon, one should keep the same mass of the wheel. The remaining single-wheel simulations carried out in this section employ the 22 kg wheel and look into the sensitivity of the slope vs. slip curves with respect to three factors: the size of the clumps that make up the terrain, value of the friction coefficient, and clump shape.

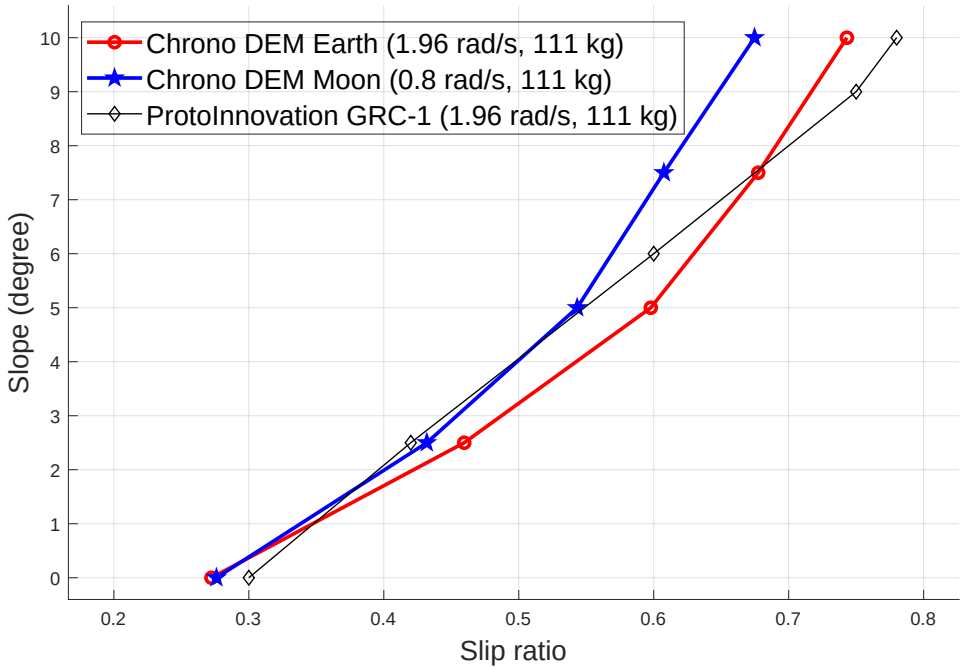
3.2.1.1 Sensitivity of slope vs. slip curve with respect to the clump size

We previously pointed out that the DS is 20 times larger than the GRC-1 simulant. In this section, we demonstrate that shrinking the clump size does not affect significantly the simulation accuracy, see Fig. 3.16.

We repeat the test where the 22 kg wheel operates at an angular velocity of 0.8 rad/s under Earth's gravity, with the terrain made of two new types of DEM elements. The blue curve shows the slip–slope relationship when each DS element's size is scaled by a 0.75 factor to obtain a different digital simulant. This reduces the apparent size of the smallest type of clumps from 2.5 mm to 1.875 mm. 2 846 244 clumps employing 8 707 887 are in the simulation and this 8.4 s simulation took approximately 11 hours. Note that this size change requires re-running the entire sample generation and test bed preparation process before carrying out the slope vs. slip test. Compared to the base red curve, we did not observe a significant



(a) The slip-on-incline test result with 22 kg wheel. The experimental data used for comparison (black line) are from Glenn Research Center's MGRU3 experiments with the GRC-1 simulant.



(b) Slip-on-incline test result with 111 kg wheel. The experimental data used for comparison (black line) are from ProtoInnovations' single-wheel experiments with the GRC-3 simulant.

Figure 3.13: The single-wheel slope vs. slip DEM test results with a variety of wheel effective mass, angular velocity, and gravitational pull.

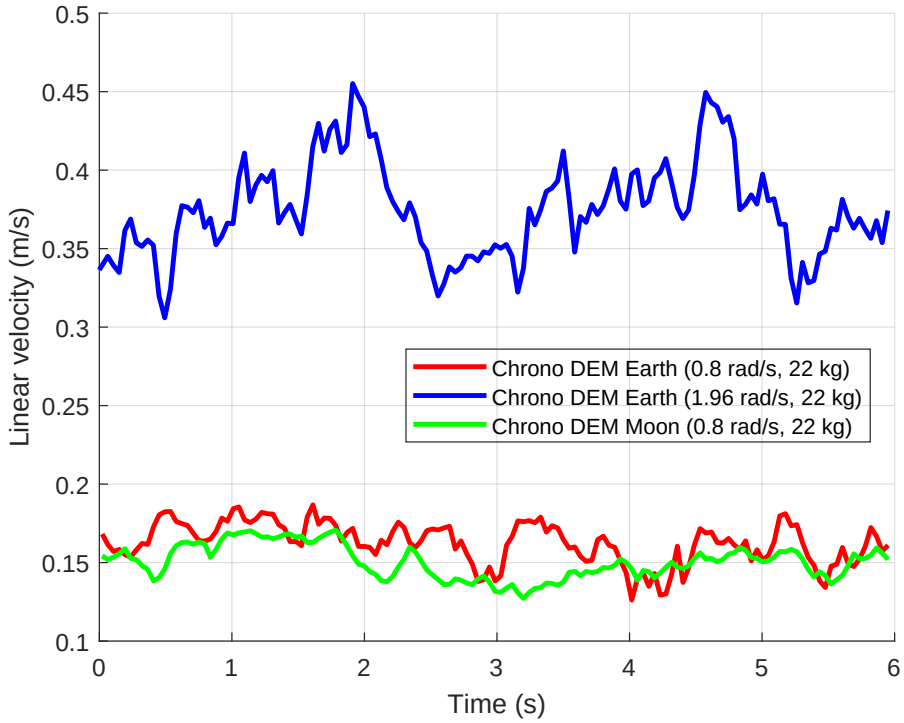


Figure 3.14: The linear velocity of the wheel along the incline direction plotted against the 6-second time span that we used to derive the slip ratio, at a 10° incline.

change in the slope vs. slip results. Conversely, the green curve shows slope vs. slip results when the DS element size is increased by a factor of 2, with 299 929 clumps (917 907 sphere components) participating in the simulation. A time of 1.5 hours is needed for this 8.4 s simulation. A rendering of this test is shown in Fig. 3.15. In simulations, larger clumps in this scenario reduced flowability with geometric locking, which led to the wheel slipping less, especially on steeper slopes.

In conclusion, although DS has a particle size that is on average 20 times larger than the particle size in GRC-1, reducing the DS element size does not negatively impact the simulation fidelity. However, if the DS element sizes are increased by a factor of 2 (i.e. simulation resolution is reduced), the accuracy is hindered due to the induced low flowability and geometric locking.

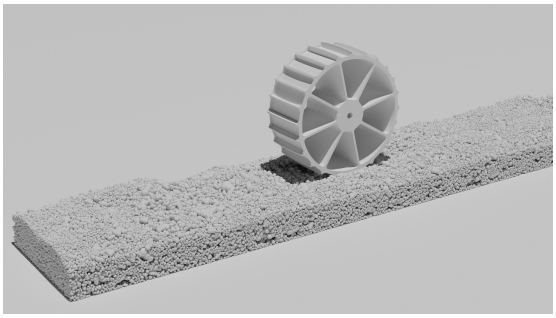


Figure 3.15: A rendering of the single-wheel test with enlarged terrain clumps.

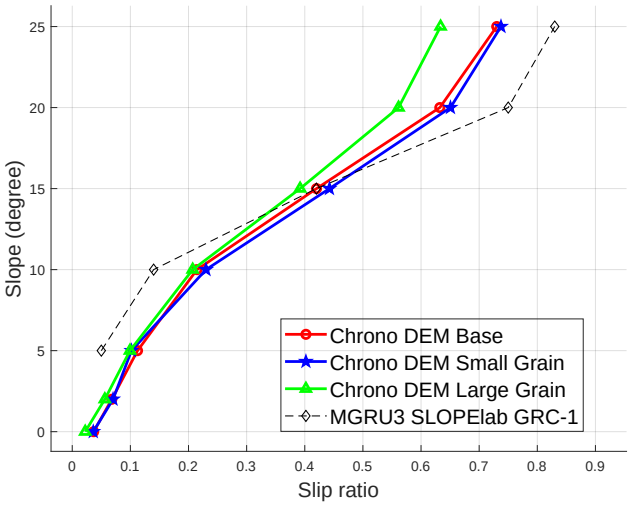


Figure 3.16: The single-wheel slope–slip test results with different selections of the clump sizes. The experimental data used for comparison (black line) are from Glenn Research Center's MGRU3 experiments with the GRC-1 simulant.

3.2.1.2 Sensitivity of slope vs. slip curve with respect to the friction coefficient

The choice of friction coefficient $\mu = 0.4$ is a typical value for silica–silica contact. However, it is well-known that the actual value of the friction coefficient can vary depending on the environment, e.g., humidity and preparation of the facets that come in contact [101, 102]. In this subsection, the focus is on investigating the sensitivity of the slope vs. slip curve with respect to the friction coefficient. To that end, we repeat the test where the 22 kg wheel operates at an angular velocity of 0.8 rad/s under Earth's gravity (red curve in Fig. 3.13a), with two different inter-element friction coefficient values. The blue curve in Fig. 3.17 shows results when $\mu = 0.2$—one can observe significantly more slip compared to the base case of $\mu = 0.4$. However, in the case where $\mu = 0.6$, the slip was comparable to the base case. This suggests that increasing μ helps prevent the wheel from slipping on the incline, up to a certain value, beyond which the influence of μ seems to diminish and other factors, e.g., particle locking, start playing an important role.

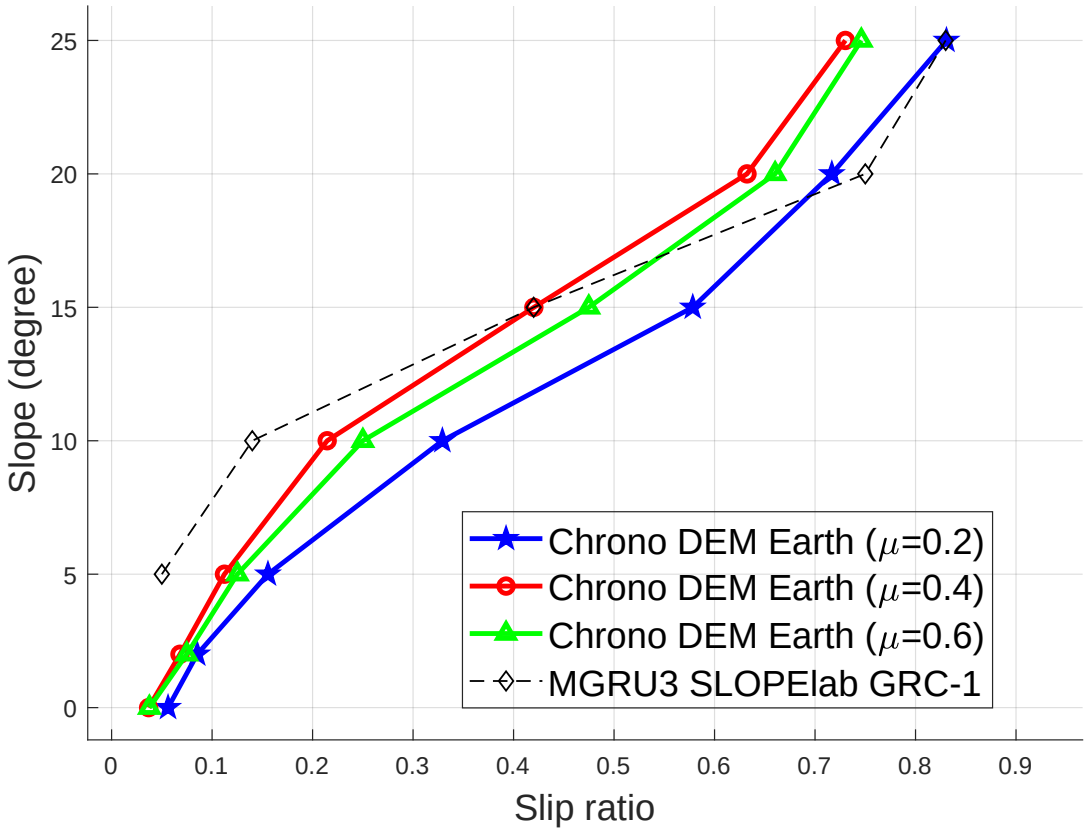


Figure 3.17: The single-wheel slope–slip test results with different selections of the friction coefficient. The experimental data used for comparison (black line) are from Glenn Research Center's MGRU3 experiments with the GRC-1 simulant.

3.2.1.3 Sensitivity of slope vs. slip curve with respect to the clump shape

The strength of the terrain in mobility tests is expected to change as the clump shapes move away from spheres. In this section, we report simulation results that quantify this change. One extreme is to have monodisperse granular material. To test this scenario, we repeat the single-wheel tests in Sec. 3.2.1 with the radius of the spheres being first 3 mm and then 6 mm. The wheel has a constant angular velocity of 0.8 rad/s and the gravitational acceleration is $g = 9.81\,\mathrm{m/s^2}$. The material properties of the granular material are as in Table 3.1, except that the friction coefficient μ is increased from 0.4 to 0.9. This change is motivated by difficulties making the spherical elements rest on a steep incline when using

$\mu = 0.4$. The latter friction coefficient, associated with dry silica, is acceptable for the DS tests since the elements' irregular shapes contribute to the locking of the grains.

The slope vs. slip relationship is summarized in Fig. 3.19. If the spherical elements are fine enough (3 mm in this test, the dashed magenta line), the sphere-based DEM appears to show a somewhat reasonable slip ratio in the flat terrain tests. A rendering is shown in Fig. 3.18. However, even for mobility tests on a gentle incline, the monodisperse terrain shears drastically more than the DS and the real-world GRC-1 simulant. The wheel is essentially spinning in place at a 15° incline. At a 25° incline, the elements immediately flow down the incline upon making contact with the wheel as previously shown in Fig. 3.3b.

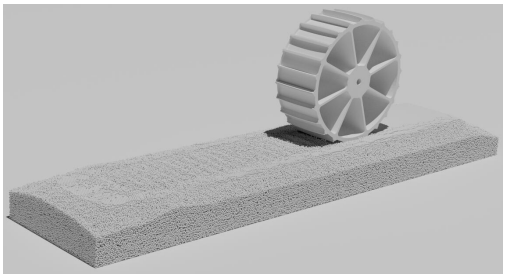


Figure 3.18: A rendering of the single-wheel test with the granular terrain represented by spherical elements, at 0° incline.

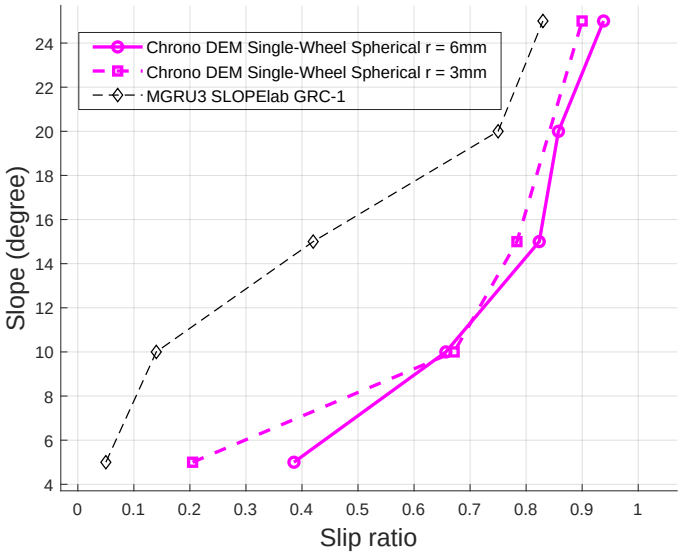


Figure 3.19: The slip-on-incline test result with *spherical terrain elements* of radius 6 mm and 3 mm. The experimental data used for comparison (black line) are from Glenn Research Center's MGRU3 experiments with the GRC-1 simulant. The results suggest that monodisperse elements cannot capture the physics noted in physical experiments.

We also present an opposite investigation where the aspect ratios of the clumps are enlarged by expanding the clumps without changing their thickness. This is accomplished by moving each component sphere slightly away from the geometric center of the clump while

maintaining the planar overall shape of the clump, see Fig. 3.20 for new shapes. Quantitatively, keeping other conditions unchanged, the seven clump templates' aspect ratio was increased by about 25%. After creating a new DEM terrain using the new clumps, the single-wheel test is repeated with a wheel angular velocity of 0.8 rad/s and gravitational pull of $g = 9.81\,\mathrm{m/s^2}$. We observe in Fig. 3.21 that the slip, shown with the blue curve, is reduced compared to the base red curve, and this effect is more pronounced on steeper slopes.

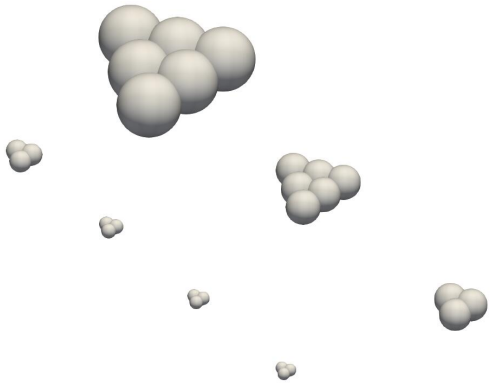


Figure 3.20: The clump shapes after enlarging their aspect ratio.

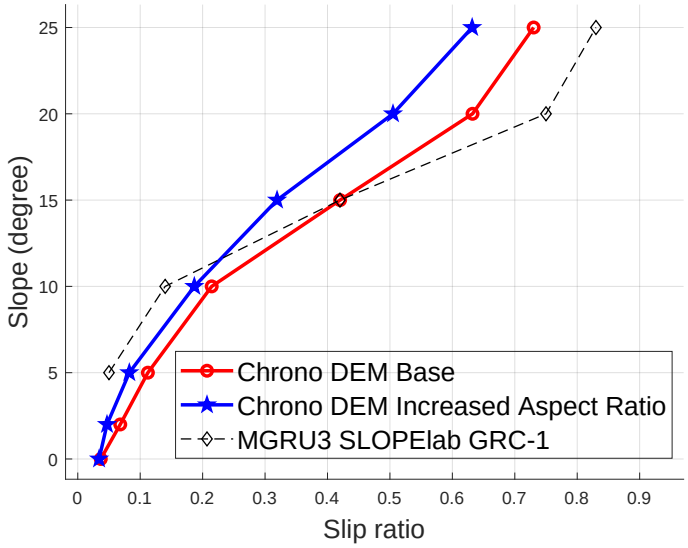


Figure 3.21: The slip-on-incline test result with the clumps that have larger aspect ratios, compared against the base clump shape. The experimental data used for comparison (black line) are from Glenn Research Center's MGRU3 experiments with the GRC-1 simulant.

Moving from spherical to clumped DEM elements has a significant impact on terrain strength in mobility simulations. The geometric locking effect that DS enables on incline tests cannot be replicated using monodisperse terrain. We note the existence of studies on reproducing the locking effect in monodisperse material with modified material properties such as the rolling resistance [103]. Its applicability in rover mobility simulations is unclear, but even if feasible, it necessitates extra calibration stages that require field test data for each

terrain configuration of interest. By using clumps, the DS does not require field test data but rather a good approximation of the elements' shapes.

Finally, note that dilating the clump at constant thickness also reduces the slope associated with a given slip, albeit not as significantly as noted for monodisperse material. The findings also suggest that if high fidelity is needed, one should pay attention to the shape of the elements. This can be nontrivial, particularly if the granular material does not have a known shape distribution, an aspect that can be addressed through a shape calibration process that falls outside the scope of this contribution.

3.2.2 Full-rover slip test on incline

This section discusses a slope vs. slip test carried out for a full rover. This calls for a co-simulation between a multi-body system (the rover) and a DEM system (the terrain). For the multi-body system, since the MGRU3 CAD model was inaccessible, we instead used a similar VIPER rover model publicly available in the latest Chrono distribution [50]. Shown in Fig. 3.23a, the rover is roughly 2.1 m in length, 1.5 m in width, and 1.4 m in height (excluding the antenna). Four wheels are connected to the rover body through revolute joints. The wheel geometry is from MGRU3, the same as that in Sec. 3.2.1. The rover moves around by prescribing all its four wheels a 0.8 rad/s angular velocity on inclines of 0, 5, 10, 15, 20, and 25°. A rendering of a test scene is provided in Fig. 3.23b.

3.2.2.1 Co-simulation

The co-simulation setup is shown in Figure 3.22. DEM-Engine handles the evolution of the granular terrain, while Chrono deals with the rover dynamics. The two simulators are bridged through the meshes that represent the wheels. DEM-Engine calculates the force exerted by the terrain on the wheel mesh. The force information is used when the Chrono numerical integrator propagates forward in time the evolution of the meshes. In turn, the new position of the wheels will become a set of updated boundary conditions for the granular

material. The rover's mobility is also influenced by forces that crop up in the chassis and suspension that are independent of the motion of the granular terrain. In the current Chrono co-simulation framework, we advance the multi-body rover system in Chrono by one time step for every ten DEM time steps.

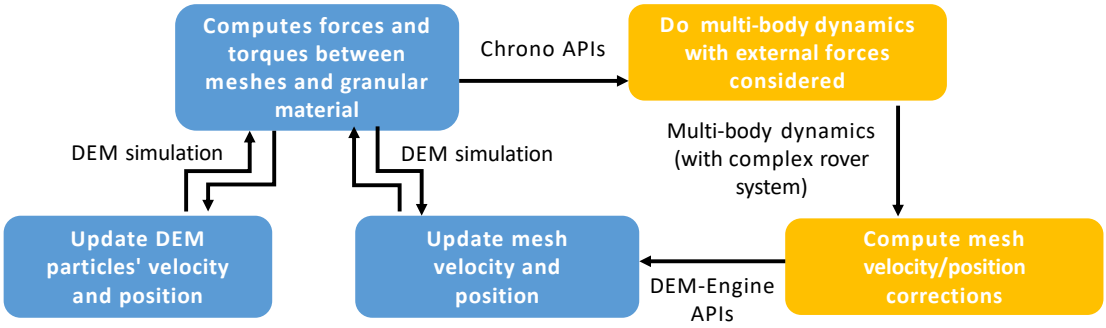
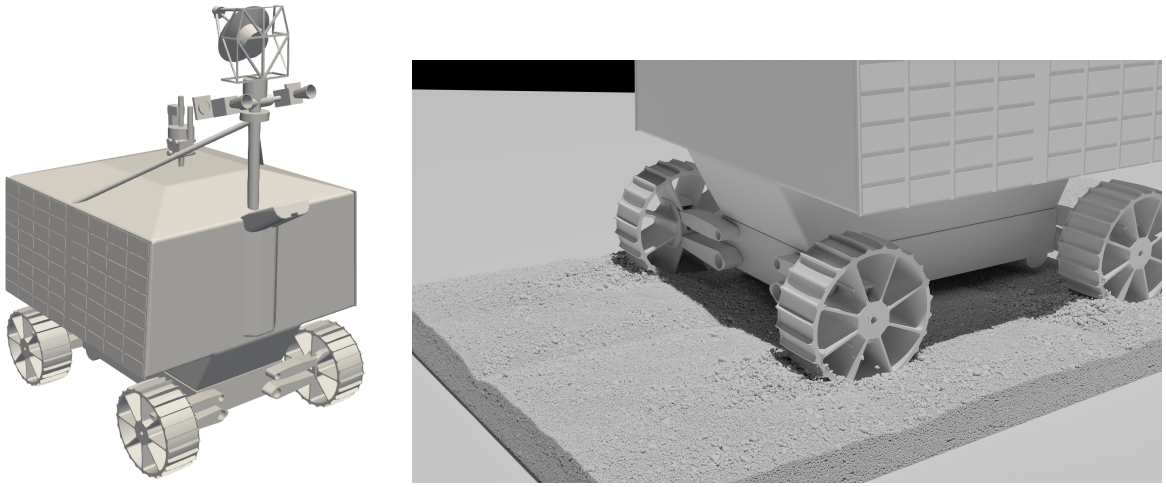


Figure 3.22: The co-simulation workflow between the multi-body system simulated by Chrono and the new DEM simulator.

Note that the full-rover slip data shown in Fig. 3.24 displays no notable difference compared to the single-wheel counterpart. This suggests that the more expeditious single-wheel tests are likely sufficient to gain insights into the rover's mobility attributes. The slip ratio increases relatively slowly with the slope angle in the interval between 0° and 10°. Past 10°, this rate of increase escalates, and the rover almost fails to climb on a 25° incline. Finally, the DS for this test uses 11 336 638 DEM elements that employ 34 691 952 component spheres. On two NVIDIA A100 GPUs, a 15 s long simulation requires approximately 109 hours of run time.



(a) A rendering of the VIPER rover used in the simulation.

(b) A rendering of the VIPER rover operating on a 20° incline.

Figure 3.23: The full-rover test using the DS.

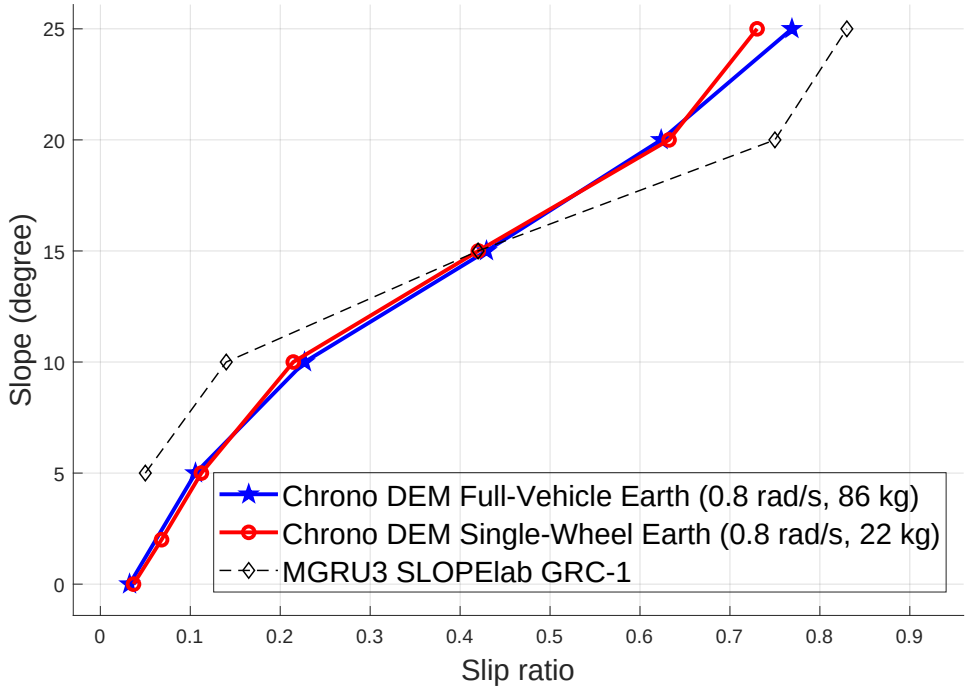


Figure 3.24: VIPER rover slip-on-incline test result, compared against the single-wheel test done in the same conditions. The experimental data used for comparison (black line) are from Glenn Research Center's MGRU3 experiments with the GRC-1 simulant.

3.2.2.2 Active box scheme

Using DEM-Engine's API, the user can implement a partially active simulation domain to reduce computational cost. The user can assign different family tags (introduced in subsubsection 2.1.2.2) to the elements inside and outside certain regions in the simulation domain to distinguish them. In this use case, we assign no prescribed motions to the elements inside the 1 m × 0.5 m boxes centered around each wheel, as shown in Figure 3.25. As such, these boxes are called active boxes. The DEM elements outside the active boxes are fixed in position, and do not participate in the contact detection, therefore remain dormant and contribute no computing cost.

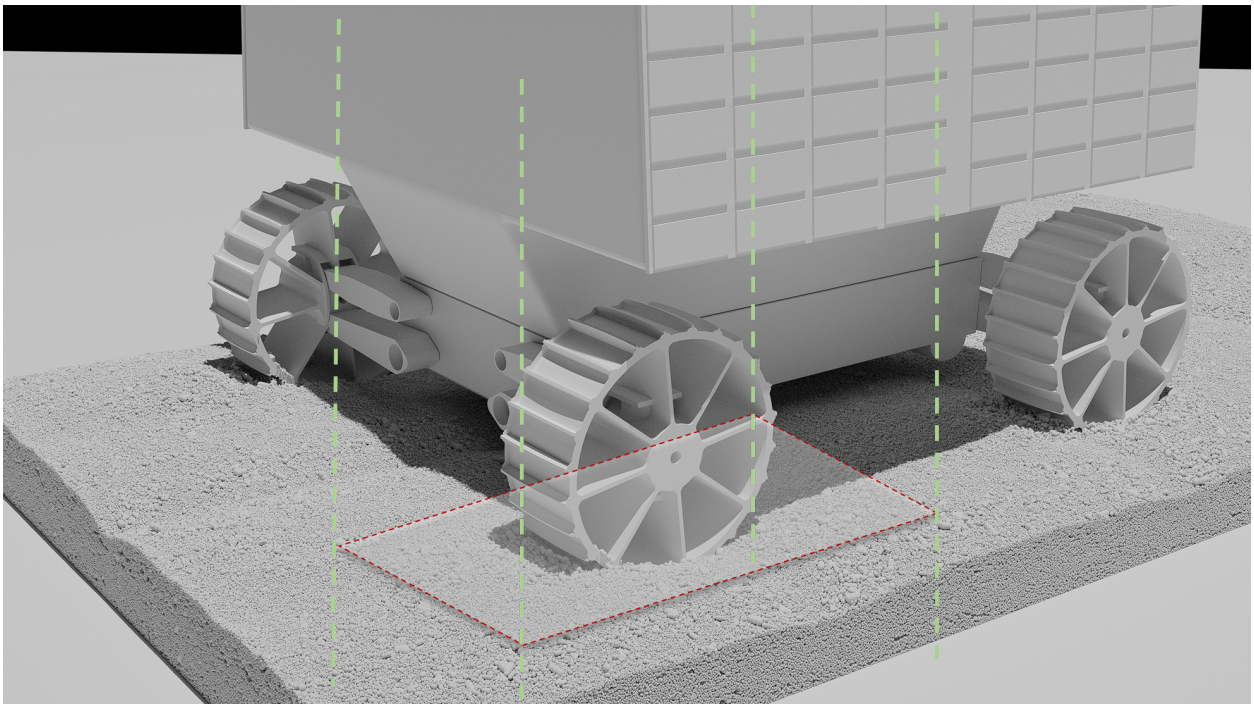


Figure 3.25: A rendering of the VIPER rover operating on a 20° incline. The active box is marked and only the elements in that region are subject to the simulation physics; the rest are fixed. There are in total four active boxes associated with four wheels, and this figure only shows one of them.

Note that the full-simulation data shown in Figure 3.26, again, displays no notable difference compared to the active box-based counterpart. The active box-based simulation

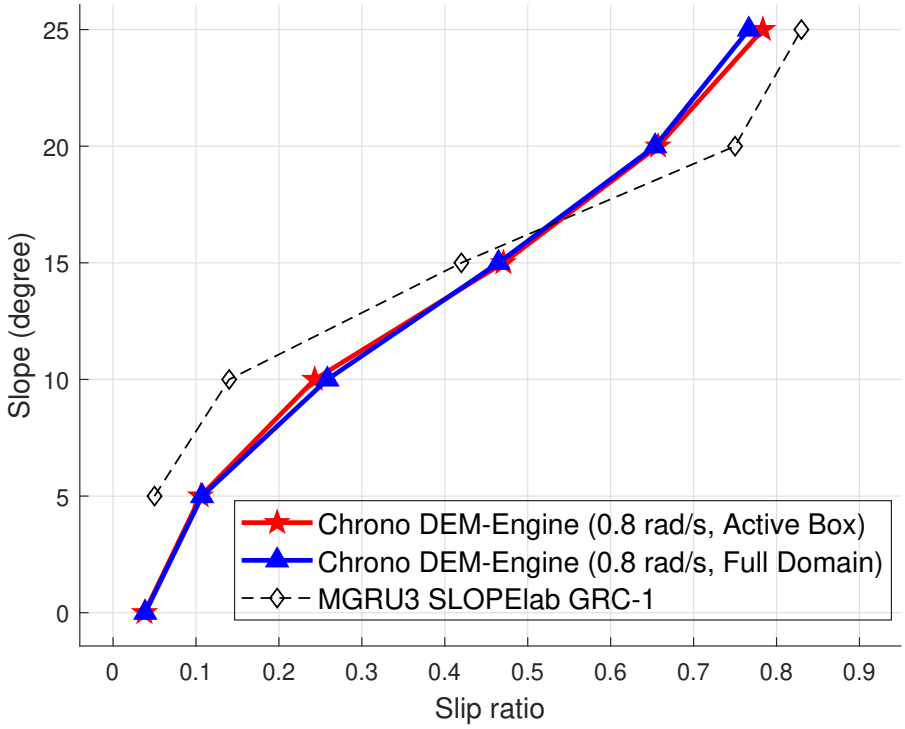


Figure 3.26: The comparison between the full-domain and active box-based VIPER rover slip test results.

reduces the runtime from 109 hours for the full-domain simulation to around 30 hours.

3.3 A simplified simulant representation

The DS proposed in this chapter offers a precise and expressive representation of GRC-1. However, its numerical demands are substantial. In light of our data-driven research in Chap. 4, which mandates the execution of numerous DEM simulations, we take inspiration from [14], adopting a tri-sphere clump as a streamlined representation of the simulant. Only one type of element is present in this reduced simulant representation, and the shape of the element remains consistent with the tri-sphere clump shown in Fig. 3.2. A parameter-sweep test is conducted to pinpoint an optimal apparent element size for the simulation.

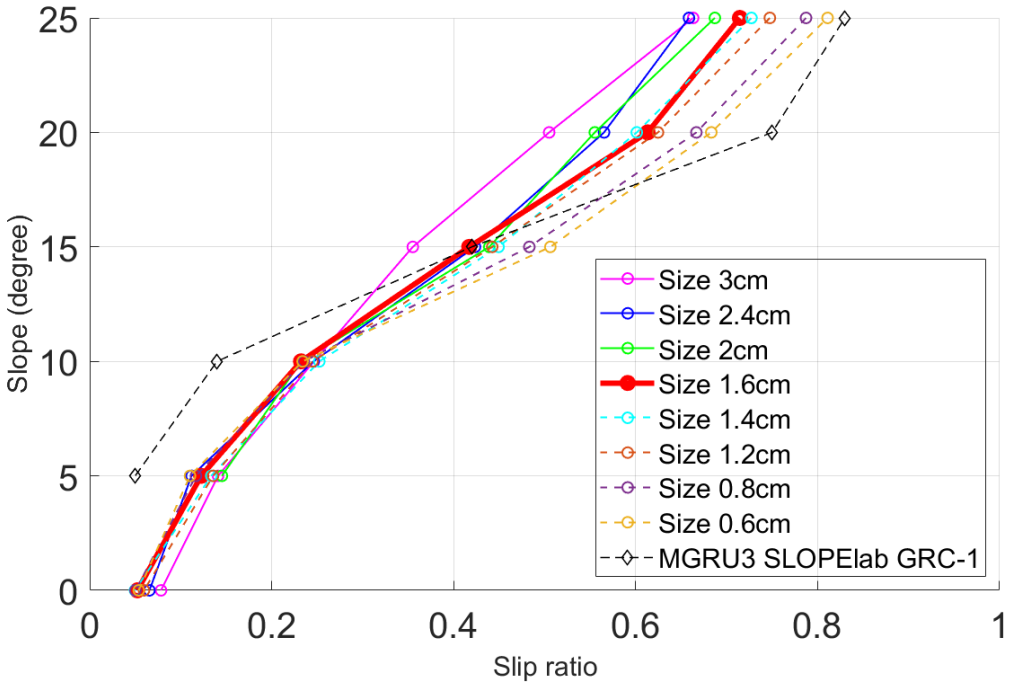


Figure 3.27: The performance of the tri-sphere clumps of various sizes in the incline slip tests.

Figure 3.27 illustrates the interplay between element size and the simulant's flowability. A smaller element size enhances flowability, diminishing the terrain's strength, which in turn predisposes the wheel to more slippage on inclines. Ultimately, an apparent element size of 1.6 cm (highlighted by the red curve) is chosen. This size strikes a judicious balance between faithfully emulating experimental slip data and economizing computational demands. A single-wheel test using this reduced DS is illustrated in Fig. 3.28.

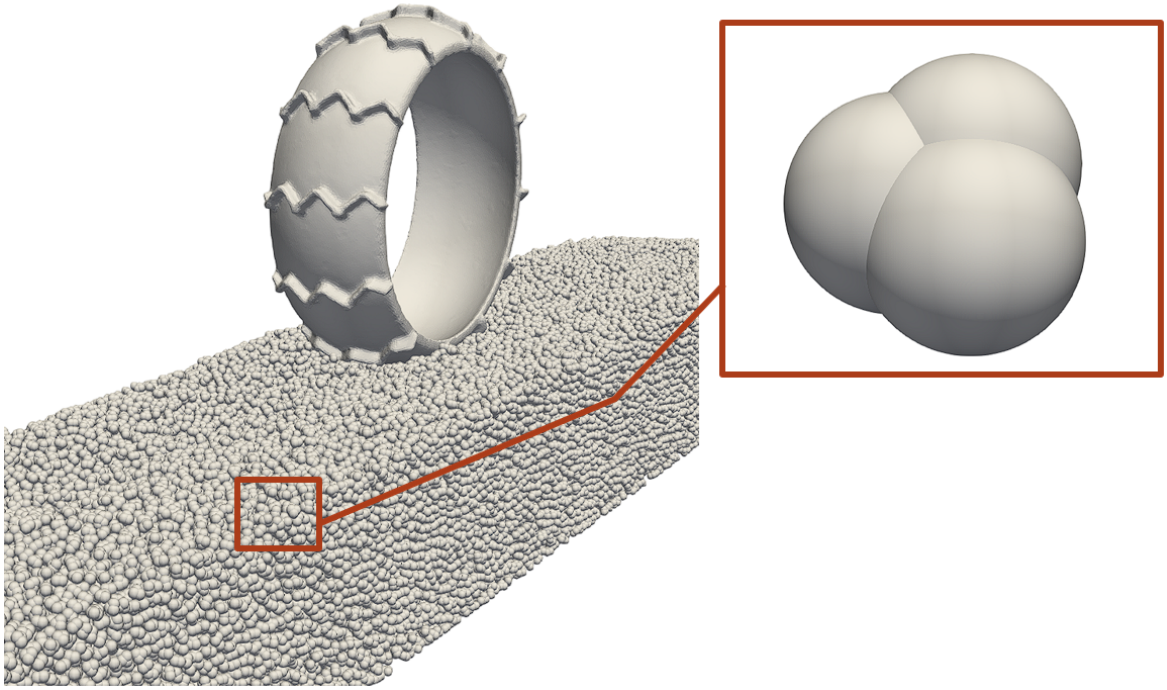


Figure 3.28: A rendering of a single-wheel test scene using the reduced simulant representation.

4 ROVER WHEEL OPTIMIZATION

Having outlined the capabilities of our state-of-the-art CUDA-based DEM package and regolith DS design, the focus pivots to an application that accentuates their potential. Leveraging the speed and accuracy of Chrono DEM-Engine, we embark on a data-driven research venture, which seeks to identify wheel designs that are good for rovers. To that end, we use the DEM simulator and run thousands of sims to generate an extensive dataset that we subsequently mine to identify good rover wheel configurations. We harness the Gaussian process model to decode patterns of each design parameter's influence on the performance. Subsequently, Bayesian optimization is brought to the forefront to iteratively refine and identify the optimized wheel designs even beyond the range of the dataset.

While many analytical models are effective in capturing many physical behaviors, they often rely on well-established first principles to do design optimization, like shape or topology gradients, which are commonly tied to problems where Partial Differential Equations (PDEs) govern the dynamics. DEM-based granular simulations, however, operate in a realm where such first principles are not straightforward. Rather than attempting to retrofit established principles to granular interactions, the approach in this thesis builds upon the raw insights obtained directly from empirical observations and simulations, distilling patterns and relationships from the datasets, even in the absence of well-defined guiding principles.

4.1 Database preparation

The foundation of any data-driven research lies in the quality and diversity of its dataset. For the investigation into rover wheel design optimization, it is paramount to construct a dataset that encompasses a broad spectrum of design variations, capturing the intricacies of wheel performance on granular terrains. This chapter elucidates the comprehensive process undertaken to generate this dataset.

To initiate the data generation phase, we need to parameterize the designs. We delineated

eight design parameters, as outlined in Table 4.1. Fundamentally, the wheel adopts a cylindrical form embellished with sinusoidal grousers. The grouser characteristics—size, form, and spacing—are dictated by parameters including amplitude a, wave number b, height h, thickness t, and the number of grousers n, as depicted in Fig. 4.1. The wheel's perimeter is orchestrated by a cubic Bézier curve. Its four control points are equidistant in the x-direction. In the y-direction, the central two points deviate from the outer ones by a magnitude of $c \times r$, enabling the parameter c to modulate the wheel perimeter's "convexity". A wheel surface bulging outward is termed as convex (with a positive c value), whereas an inward curve is labeled concave (with a negative c value). This distinction is further illustrated in Fig. 4.1.

Table 4.1: Eight design parameters of the rover wheel. See Fig. 4.1 for an illustrated explanation.

Parameter	Variable name	Range
Grouser wave amplitude	a	$[0\,\text{m}, 0.03\,\text{m}]$
Grouser wave number	b	$[1,\ 4]$
Grouser height	h	$[0.005\,\text{m}, 0.15\,\text{m}]$
Grouser thickness	t	$[0.01\,\text{m}, 0.025\,\text{m}]$
Number of grousers	n	$[8, 32]$
Control point deviation	c	$[-0.3, 0.3]$
Wheel radius	r	$[0.15\,\text{m}, 0.5\,\text{m}]$
Wheel width	w	$[0.1\,\text{m}, 0.5\,\text{m}]$

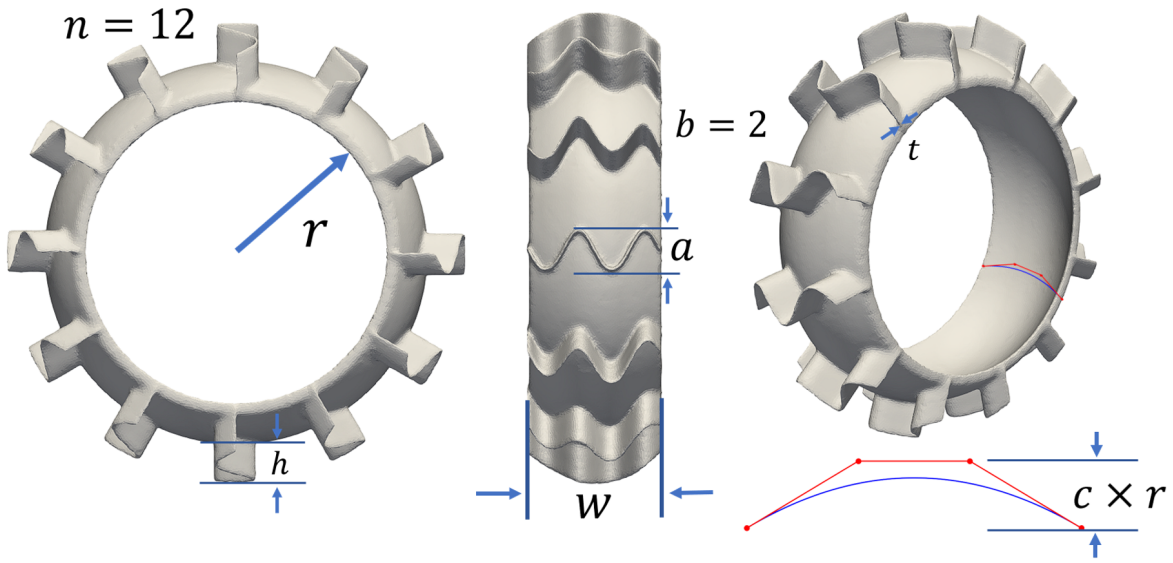


Figure 4.1: Eight design parameters of the rover wheel: Grouser wave amplitude a, grouser wave number b, grouser height h, grouser thickness t, number of grousers n, control point deviation c, wheel radius r, wheel width w.

To vary these parameters, we select a value for each parameter randomly within their ranges, ensuring a comprehensive exploration of the design space. Subsequently, a Python script is activated to craft a mesh-represented wheel based on these selections. Several outcomes of these wheel designs are showcased in Fig. 4.2. Note that for clarity, the designs within the figure are scaled to appear roughly equivalent in size. However, their actual sizes can vary considerably based on the radius and width values selected. Each of these mesh designs subsequently undergoes two DEM simulations to gauge their performance with granular terrains.

The first DEM simulation is an incline-climbing test, similar to the one in Sec. 3.2.1. This test aims to quantify the design's climbing ability. The inclines used in the simulation are created by altering the direction of gravity, whose magnitude is fixed at $g = 9.81\,\mathrm{m/s^2}$. The wheel in the test is subject to a fixed angular velocity $\omega = 0.8\,\mathrm{rad/s}$ and is free to move linearly. The wheel has an intrinsic mass of 5 kg, but is assigned a different effective mass (implemented as extra loads) in these simulations. This adjustment, detailed in Sec. 3.2.1,

compensates for the chassis and suspension weight borne by each rover wheel. The angle of incline α and effective mass m are the non-design parameters which are also selected randomly in each test, as specified in Table 4.2. Each simulation is run for 2 seconds, and then the measuring starts. The performance metrics being measured in this test are "time", "slip ratio", and "energy efficiency" in Table 4.3. Time T means the time taken for the wheel to climb 0.1 m in altitude. During this period, the slip ratio S_s is also measured, using the same definition $S_s = 1 - v/(\omega p)$, similar to Sec. 3.2.1, where v is the measured average linear velocity, and p is the wheel's overall profile. Here, the profile is defined as the sum of the wheel radius r, grouser height h, and the maximal outward deviation due to wheel convexity. For concave wheel designs, this deviation is considered zero since the inward curve does not add to the total size. The energy efficiency is defined as $E_e = (mg \times 0.1\,\mathrm{m})/W$, that is, the potential energy gain over the total work W. The work is measured by $W = \int_0^T \tau\omega dt$, where τ is the torque on the wheel measured during simulation. An example test scene is rendered in Fig. 4.3. The reduced DS simulant introduced in Sec. 3.3 filling a roughly $3\,\mathrm{m} \times 0.75\,\mathrm{m} \times 0.3\,\mathrm{m}$ (legnth×width×height) space, is used to represent the terrain in this test. 190 790 tri-sphere clumps (totaling 572 370 component spheres) participate in each simulation.

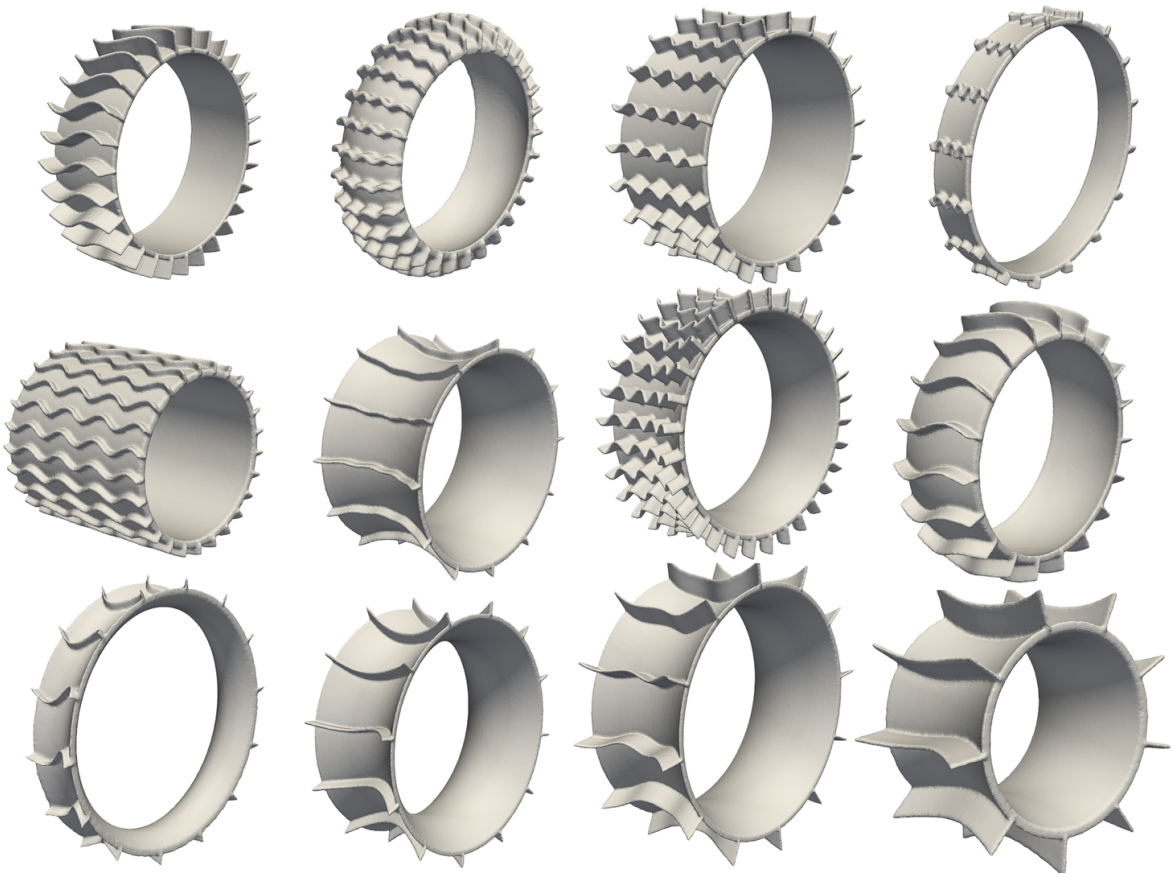


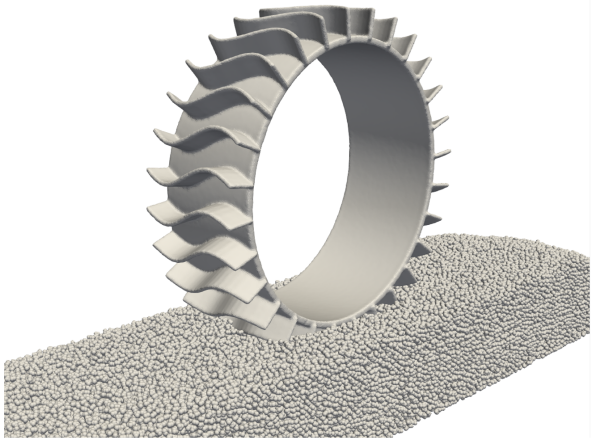
Figure 4.2: Samples of randomly generated wheels using the parameter range in Table 4.1.

Table 4.2: Non-design parameters in the climbing and steering tests.

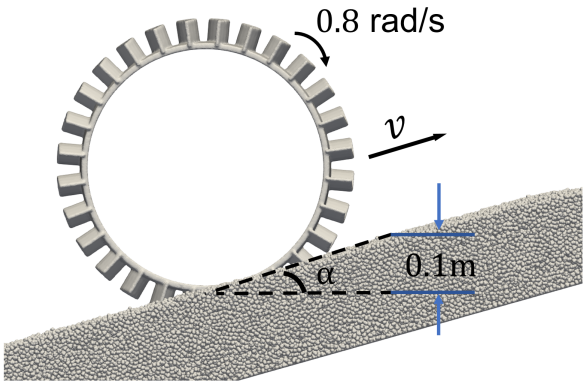
Parameter	Variable name	Range
Slope	α	$[5°, 15°]$
Mass	m	$[22\,\text{kg}, 132\,\text{kg}]$

Table 4.3: Performance metrics in the incline and steering tests.

Performance metric	Variable name
Time	T
Slip ratio	S_s
Energy efficiency	E_e
Steering ratio	S_t



(a) A perspective view of the climbing ability test.

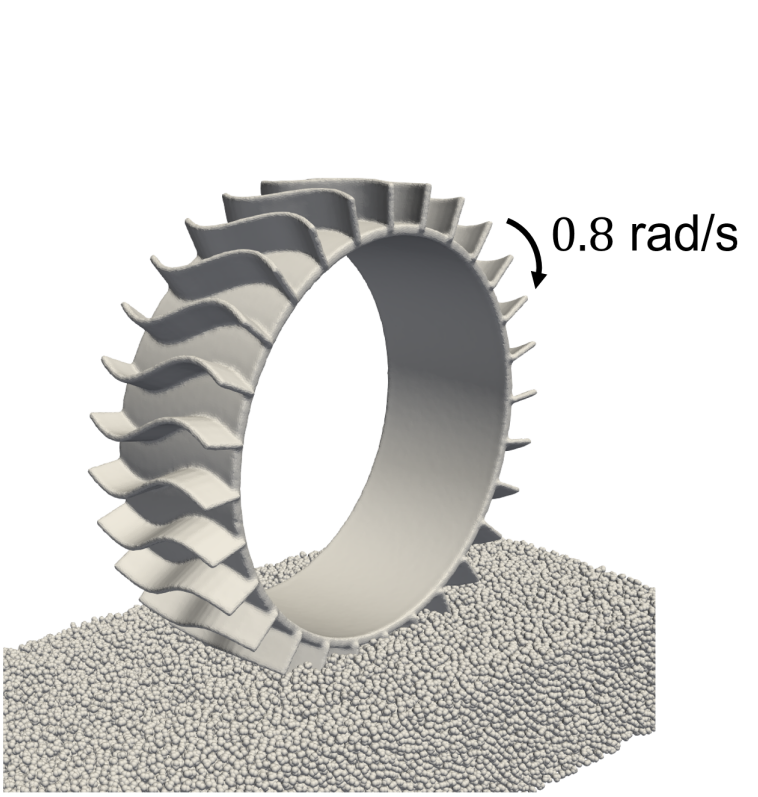


(b) The energy efficiency, slip ratio, and consumed time in climbing 0.1 m vertically is recorded for each wheel design.

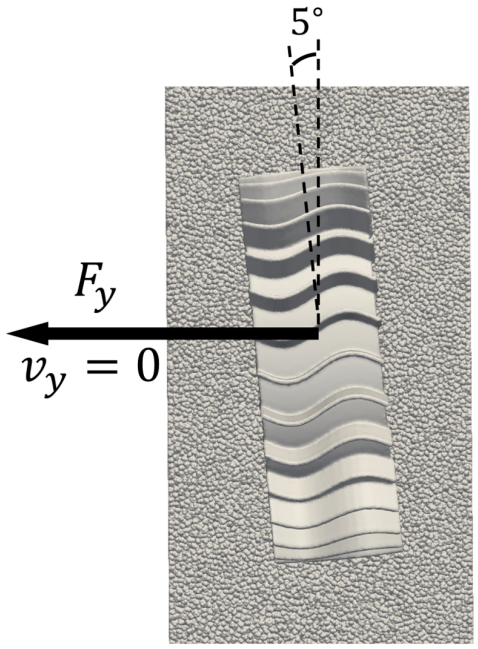
Figure 4.3: The renderings of the climbing ability test scenes.

The second DEM simulation is a steering ability test. Here, the gravitational pull's direction coincides with the z-axis, positioning the wheel on a level terrain. The wheel can freely traverse in the forward (x-direction) and vertical (z-direction) axes, yet its lateral movement (y-direction) is constrained. Notably, the wheel is not perfectly aligned with the forward direction but instead, its central plane deviates by a 5° angle. The wheel spins at a constant angular velocity $\omega = 0.8\,\text{rad/s}$, replicating a condition where the wheel pivots laterally slightly. We measure the metric named "steering ratio" (see Table 4.3), which is given by $S_t = F_y/(mg)$. It signifies the force experienced by the wheel in the y-direction relative to its overall load. A higher S_t indicates superior steering ability. An illustrative test scene can be observed in Fig. 4.4.

Using one A100 GPU, it requires roughly 30 minutes to finish one set of simulations, thus generating a data point.



(a) A perspective view of the steering ability test.



(b) The wheel's central plane deviates from the forward direction by 5°, and is forced to have zero linear velocity in the lateral direction. The lateral force experienced by the wheel is recorded.

Figure 4.4: The renderings of the steering ability test scenes.

4.2 Interpreting the data

In this research, 4209 data points were generated by varying the design and non-design parameters in Table 4.1 and 4.2, and recording the metrics stipulated in Table 4.3 based on DEM simulation results. In this section, we use Individual Conditional Expectation (ICE) plots to interpret the influence of the design parameters.

4.2.1 Role of Gaussian processes

ICE plots provide a tool to visualize how a model's predictions change for individual observations when varying a specific feature across its range. Gaussian Processes (GPs) are not inherently part of ICE plots, but are used as the predictive model in this research.

GPs are a flexible tool employed in machine learning and statistics for tasks such as regression, classification, and optimization. They offer a non-parametric approach, meaning they do not assume an explicit form for the function being modeled, but rather place a distribution over functions. Formally, a GP is defined as a collection of random variables, any finite number of which have a joint Gaussian distribution [104]. It can be entirely specified by its mean function $m(\mathbf{x})$ and covariance (or kernel) function $k(\mathbf{x}, \mathbf{x}')$

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \tag{4.1}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))], \tag{4.2}$$

where $f(\mathbf{x})$ represents the function values. The emphasis often lies on the covariance function, which determines the properties and behavior of functions drawn from the GP. The kernel function gauges the similarity between points in the input space. The choice of kernel function and its parameters is pivotal to modeling the measurement of similarity (or covariance) between two points. Popular kernel choices include the squared exponential (RBF), Matérn, and periodic kernels [105]. GPs are prized for their adaptability. By placing a distribution over functions, they allow for infinite-dimensional generalizations of multivariate Gaussian distributions [106]. Moreover, by giving both mean and variance predictions, GPs offer insights into the uncertainty associated with predictions. However, they can be computationally intensive, especially for large datasets, as they necessitate operations on large covariance matrices.

The application of GPs in this research is Gaussian Process Regression (GPR). In GPR, the aim is to infer a continuous function from observed data points; that is, to produce a surrogate response function. The non-parametric nature of GPs enables GPR to predict not just an output value for a given input but also the uncertainty (variance) associated with that prediction. The predictive mean and variance at an unobserved input are derived from the GP prior and the observed data. This results in a distribution over possible functions that fit the data, with the mean function representing the best estimate and the variance

providing a measure of confidence around this estimate. As such, GPR provides a probabilistic framework for regression, capturing both the expected behavior and the associated uncertainty of predictions. By the same token, integrating GPs with ICE offers insights into not just the model predictions but also their associated uncertainties.

The workflow of GPR is specified as follows. Given the training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ with N input–output pairs, the latent function values $f = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)]^T$ are described by the GP prior, $f \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$, where $\mathbf{K}$ is the $N \times N$ covariance matrix. The covariance between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is determined using the anisotropic squared exponential kernel in this research, which is given by

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\sum_{d=1}^{D} \frac{(x_{i,d} - x_{j,d})^2}{2l_d^2}\right), \tag{4.3}$$

where σ_f^2 is the signal variance, D is the dimensionality of the input space, and l_d represents the length scale associated with the d^{th} dimension. In the case of an anisotropic kernel, there are D different length scales, corresponding to the D dimensions of the input space. In this research, it is assumed that the observed outputs y_i are influenced by independent Gaussian noise, such that $y_i = f(\mathbf{x}_i) + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. The noise variance σ^2 is another hyperparameter in the model.

To train the GP model, the hyperparameters $\{l_d\}_{d=1}^{D}$, σ^2 and σ_f^2 are optimized to maximize the likelihood of the observed data given the model. The steps are given as pseudocode shown in Algorithm 1.

Algorithm 1 Iterative GP model update with hyperparameter optimization.

Input: Training dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$
Output: Trained GP model with optimized hyperparameters
Initialize hyperparameters $\{l_d\}_{d=1}^{D}$, σ_f^2, and σ^2
Define the likelihood function based on the GP model and the observed data
while not converged **do**
    for each data point $(\mathbf{x}_i, y_i)$ in $\mathcal{D}$ **do**
        Compute kernel value for $\mathbf{x}_i$ with all previous data points
        Update the covariance matrix with the new kernel values
    end for
    Optimize the likelihood with respect to the hyperparameters
    Update hyperparameters based on the optimization results
end while
return Trained GP model with optimized hyperparameters

The computational cost of GPR is intimately linked to the size of the dataset. Specifically, the formation of the $N \times N$ covariance matrix, for N data points, necessitates $O(N^2)$ computations by evaluating the covariance function for each pair of data points [104]. However, the most resource-intensive operation in GPR is the inversion of this covariance matrix, which possesses a computational complexity of $O(N^3)$ for dense matrices. Furthermore, storing the covariance matrix requires $O(N^2)$ memory. Due to this cubic growth in complexity, GPR can become prohibitive for large datasets, both in terms of computational time and memory. We note that to address these challenges, various approximation methods, such as Sparse Gaussian Processes, the Fully Independent Training Conditional (FITC) approximation, and techniques like Random Fourier Features and Nystrom methods, have been developed, enabling GPR to manage larger datasets but often at the cost of some accuracy [107]. However, the discussion of utilizing such methods is beyond the scope of this work.

4.2.2 Individual conditional expectations

ICE plots provide a tool to visualize how model predictions change for individual observations when varying a specific feature across its range while holding other features constant. The ICE plot for a specific observation can be thought of as a series of conditional expectations in

a regression setting [108]. Consider a model f that predicts an outcome y based on a vector of predictors $\mathbf{x}$ with p features. The ICE plot for the j^{th} feature, x_j, for a specific observation i, is represented as

$$\text{ICE}_j^{(i)}(x_j) = f(x_j, x_{-j}^{(i)}), \tag{4.4}$$

where x_j is the value of the j^{th} feature for which we want the prediction, $x_{-j}^{(i)}$ is the vector of all predictors other than the j^{th} feature for data point i, and f is the model's prediction function.

For clearer visualization and better interpretation, in this work, we use a variation called centered ICE (c-ICE) to visualize our data. The c-ICE for the j^{th} feature for a specific observation i is

$$\text{c-ICE}_j^{(i)}(x_j) = \text{ICE}_j^{(i)}(x_j) - \text{ICE}_j^{(i)}(x_j^*), \tag{4.5}$$

where $\text{ICE}_j^{(i)}(x_j^*)$ is the predicted value at the reference predictor value x_j^*. In this work, we follow the convention of selecting x_j^* to be the minimum value of each predictor in question. This centers the ICE lines around zero at the observed feature value, making it visually clearer to discern the impact of a feature's varying values relative to its observed value. This process is summarized with Algorithm 2. This pseudocode demonstrates the process of creating a c-ICE plot for each instance in the dataset. Each curve in the plot corresponds to an instance, showing how the prediction (centered around its original value) changes as the feature of interest varies.

Note that we also add the Partial Dependence Plots (PDPs) to our c-ICE plots with a red curve, so they serve as a tool to visualize the average effect of a predictor on the response, averaged over the distribution of the other predictors. Mathematically, for a predictor x_j, the PDP is defined as

$$\text{PDP}_j(x_j) = \frac{1}{N}\sum_{i=1}^{N} f(x_j, x_{-j}^{(i)}), \tag{4.6}$$

where $x_{-j}^{(i)}$ represents all predictors in data point i other than x_j, f denotes the model's prediction function, and N is the number of data points. In the context of c-ICE plots, the

Algorithm 2 c-ICE Plot generation using GP

Input: Trained GP model, dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, feature of interest x_{feature}
Output: c-ICE plot
for each instance $(\mathbf{x}_i, y_i)$ in $\mathcal{D}$ **do**
    for each value v of x_{feature} in its range **do**
        Set $\mathbf{x}_i[\text{feature}] = v$
        Predict $\hat{y}_i^v$ using the GP model with $\mathbf{x}_i$
    end for
    Determine $\hat{y}_i^*$ using the GP model with the reference value of x_{feature} for $\mathbf{x}_i$
    for each value v of x_{feature} **do**
        Compute centered prediction: $\Delta\hat{y}_i^v = \hat{y}_i^v - \hat{y}_i^*$
    end for
    Plot x_{feature} vs $\Delta\hat{y}_i^v$ for instance i
end for

PDP provides a summarized representation of the effect of the predictor x_j on the outcome. By juxtaposing the PDP curve against the c-ICE curves, it becomes feasible to compare the global average effect with the individual effects. If the majority of the c-ICE curves align closely with the PDP, it indicates that the influence of the predictor is generally consistent across different combinations of other predictors. Conversely, a notable divergence of c-ICE curves from the PDP may highlight interactions between the predictor of interest and other predictors in the model.

The c-ICE plots, illustrating the influence of the eight design parameters on performance, are presented in Fig. 4.5, 4.6, 4.7, and 4.8. Key insights from these figures are outlined below:

1. Metrics such as time (T), slip ratio (S_s), and energy efficiency (E_e) exhibit similar trends with respect to the design parameters. Specifically, greater values for grouser control point deviation (c), grouser height (h), wheel radius (r), and wheel width (w) simultaneously lead to decreased T, increased S_s, and increased E_e—indicating enhanced climbing performance. Conversely, an increase in grouser pattern wave amplitude (a), wave number (b), and grouser thickness (t) deteriorates this performance. Understandably, a lower slip ratio allows the wheel to attain the target height more swiftly while expending less energy interacting with the granular material. For convenience, we refer to this set of metrics as

the climbing ability.

2. Increasing r, h, and w generally improves performance, with their impact being the most significant. Hence, these parameters should ideally be maximized. However, this assertion should be contextualized within the boundaries of our study. Real-world extraterrestrial terrains might possess limited granular depth, affecting grouser efficacy. Additionally, taller grousers might pose structural challenges. The rover's motor must also furnish adequate power for maintaining wheel rotation as per our simulation settings. These realistic constraints are beyond the discussion of this work. It is noteworthy that a larger w seems to slightly diminish the steering ability S_t, possibly due to the reduced wheel sinkage.

3. Adding fine grouser features like a, b, and t only has subtle effects on wheel performance. In fact, as these features become more pronounced (e.g., with increasing a, b, and t), climbing capability tends to slightly diminish, advocating for a streamlined and slender rover grouser design. In [109], the authors also state that four different grouser shapes, parallel, slanted, V-shaped, and offset V-shaped were experimented and no notable difference is observed among them in terms of slip ratio, in line with our finding. However, we notice that greater a and b values seem to enhance the wheel's steering capacity, a point worth consideration.

4. The influence of the number of grousers n is more difficult to quantify, and appears to show different trends depending on other design parameters. However, on average, there seems to be a "sweet spot" at around $n = 10 \sim 20$, where both the climbing and steering ability are preferable.

5. Increasing wheel convexity (greater c) augments climbing performance, while increasing concavity diminishes it. Meanwhile, pronounced convexity might result in a more prominent wheel profile, a factor to be addressed in our subsequent optimization discussion. Interestingly, both high convexity and concavity tend to elevate the wheel's steering capability, compared to a wheel with a neutral perimeter.

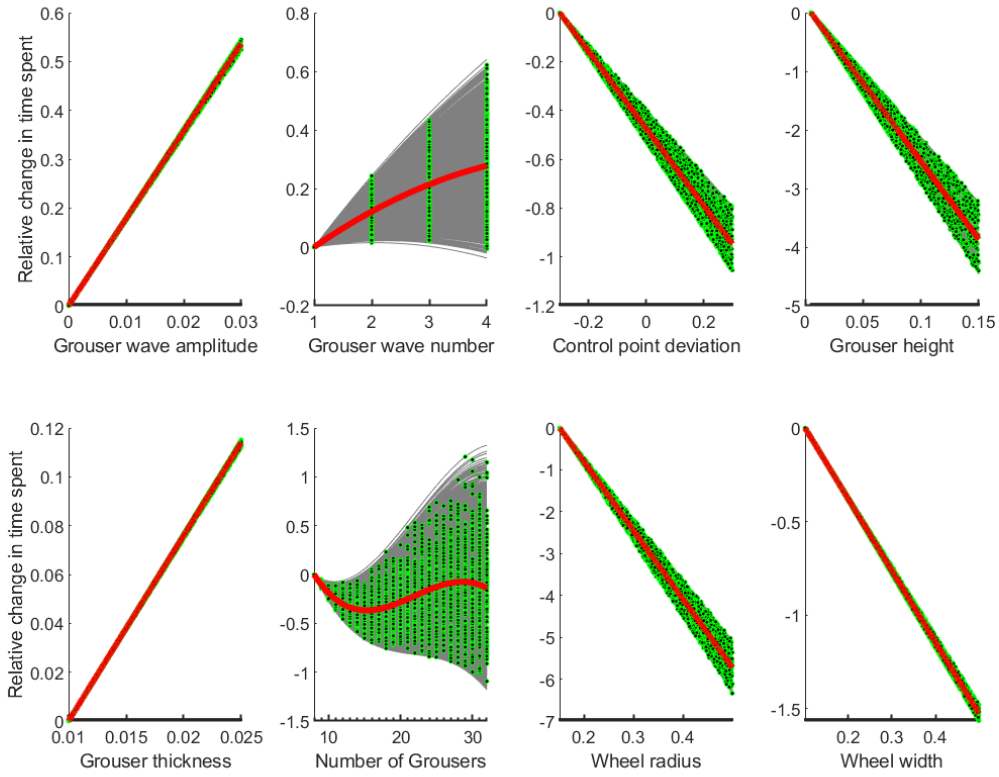


Figure 4.5: ICE plot of the wheel design parameters' influence on the time spent to climb 0.1 m vertically, with the PDP shown in red.

4.3 Gaussian processes-based optimization for improving MGRU3 wheel

With a foundational understanding of the effects of each design parameter, we now leverage this insight to formulate rover wheel designs that meet specific design objectives in this section. The basis for this optimization is the previously derived GP model. Essentially, the GP model serves as a mapping function that relates both design and non-design parameters to a suite of metrics, akin to a database. Due to that the numerical evaluation of the GP models is cheap, we can conveniently identify and "extract" feasible designs by assessing the function values and incorporating relevant scenario specifications and constraints as needed.

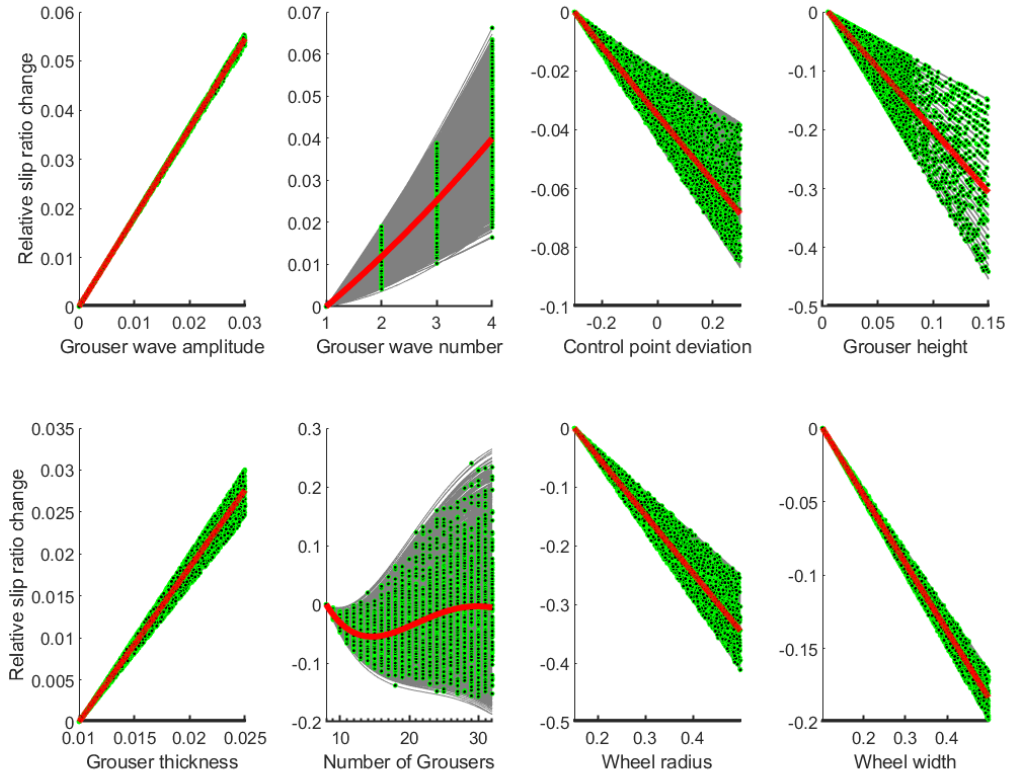


Figure 4.6: ICE plot of the wheel design parameters' influence on the slip ratio in the climbing tests, with the PDP shown in red.

In this section, the focus is on enhancing the MGRU3 wheel discussed in Sec. 3.2.1. The objective is to optimize the energy efficiency output, E_e, based on the outputs of the trained GP. As delineated in Sec. 4.2.2, time, slip ratio, and energy efficiency show similar capacity in quantifying the wheel's climbing proficiency. Hence, we exclusively choose E_e for optimization. We identify the maximum E_e by examining 2 000 000 randomly sampled points from the design space defined in Table 4.1, utilizing the GP model.

The optimization process integrates three constraints:

1. The wheel's overall profile p should not exceed 0.25 m. Recall that the profile is defined in Sec. 4.1 as the sum of the wheel radius r, grouser height h, and the maximal outward

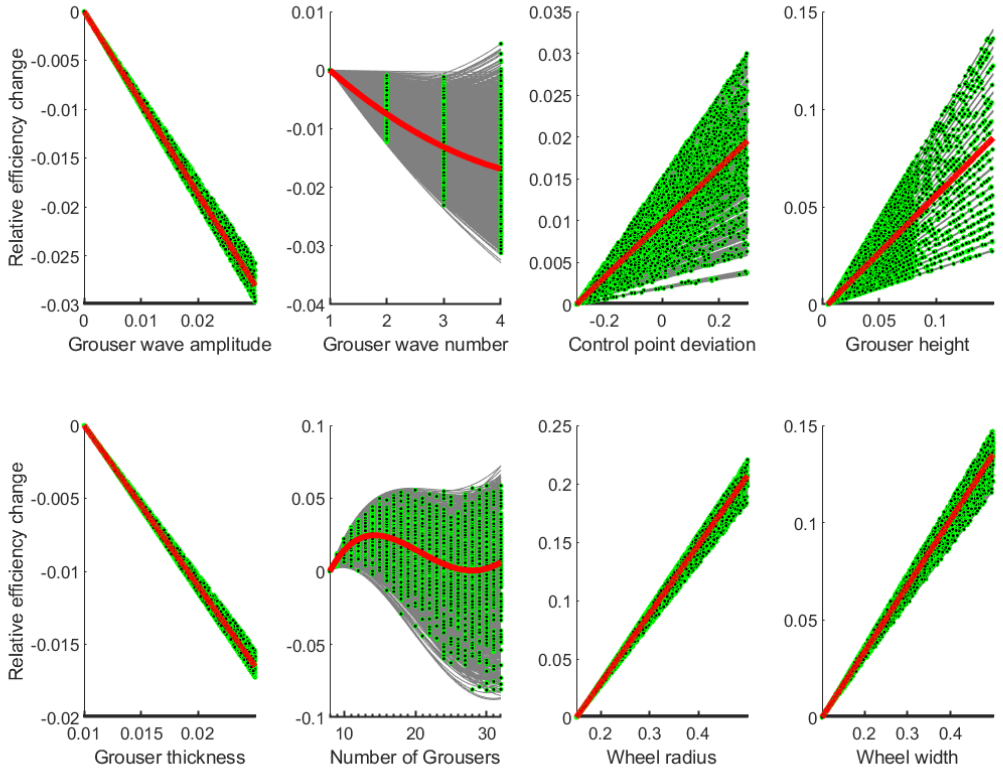


Figure 4.7: ICE plot of the wheel design parameters' influence on the energy efficiency in the climbing tests, with the PDP shown in red.

deviation due to wheel convexity. The profile constraint is well-justified since it plays a critical role when considering the transportation of the rover to its intended extraterrestrial destination. Rockets, the primary vehicle for such missions, operate under stringent spatial constraints due to their payload fairings. Additionally, a larger wheel size could also necessitate a larger landing apparatus or affect the rover's stability during landing. For the MGRU3 wheel, $p^* = 0.25\,\text{m}$.

2. The wheel's width w should not exceed 0.2 m. For the MGRU3 wheel, $w^* = 0.2\,\text{m}$. That is, our wheel design is not wider than the MGRU3.

3. The steering ratio S_t should remain above 0.13. Based on the steering DEM test presented

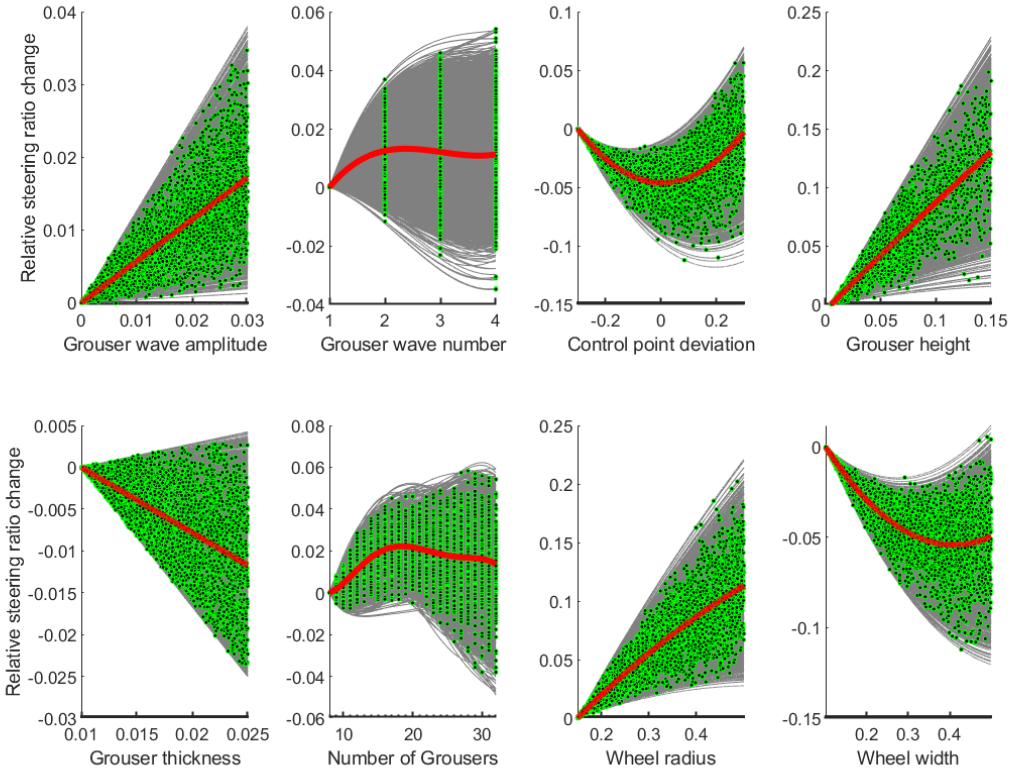


Figure 4.8: ICE plot of the wheel design parameters' influence on the steering ability, with the PDP shown in red.

in Sec. 4.1, the MGRU3 wheel registers $S_t^* \approx 0.12$ at a total mass of $m = 22\,\text{kg}$. This constraint guarantees that our optimized design retains a steering capability at least on par with the MGRU3, as detailed in Sec. 3.2.1.

All constraints are enforced by setting the evaluated E_e to 0 for non-compliant designs.

Nine optimization experiments are carried out, each adopting a unique set of non-design parameters. The outcomes are compiled in Table 4.4, labeled from tests 1 through 9. A visual representation of all derived designs is available in Fig. 4.9. The principal takeaways from these results include:

1. The optimization largely manifests as a balance between the wheel radius r and grouser

Table 4.4: Optimization results when wheel profile is constrained to be $p \leq 0.25\,\mathrm{m}$ and $w \leq 0.2\,\mathrm{m}$.

Non-design param.			Optimized design parameters and performance								
Test no.	α [°]	m [kg]	a [m]	b [-]	h [m]	t [m]	n [-]	c [-]	r [m]	w [m]	E_e [-]
1	5	22	0.0216	3	0.0874	0.0243	32	-0.2982	0.1601	0.1889	0.272
2	5	66	0.0066	4	0.0661	0.0149	12	-0.0226	0.1826	0.1853	0.16
3	5	110	0.0025	2	0.0898	0.0111	17	0.0436	0.1547	0.1909	0.118
4	10	22	0.0074	1	0.0369	0.0128	19	-0.0451	0.2107	0.1891	0.32
5	10	66	0.0015	4	0.0868	0.0118	13	0.0227	0.1527	0.1906	0.222
6	10	110	0.013	2	0.097	0.0108	13	-0.0605	0.1517	0.1927	0.154
7	15	22	0.0024	3	0.0764	0.0124	9	-0.0147	0.1648	0.19	0.346
8	15	66	0.0053	2	0.0992	0.0149	9	-0.0723	0.1502	0.1734	0.246
9	15	110	0.0201	2	0.0922	0.0131	10	0.0039	0.1548	0.1929	0.156

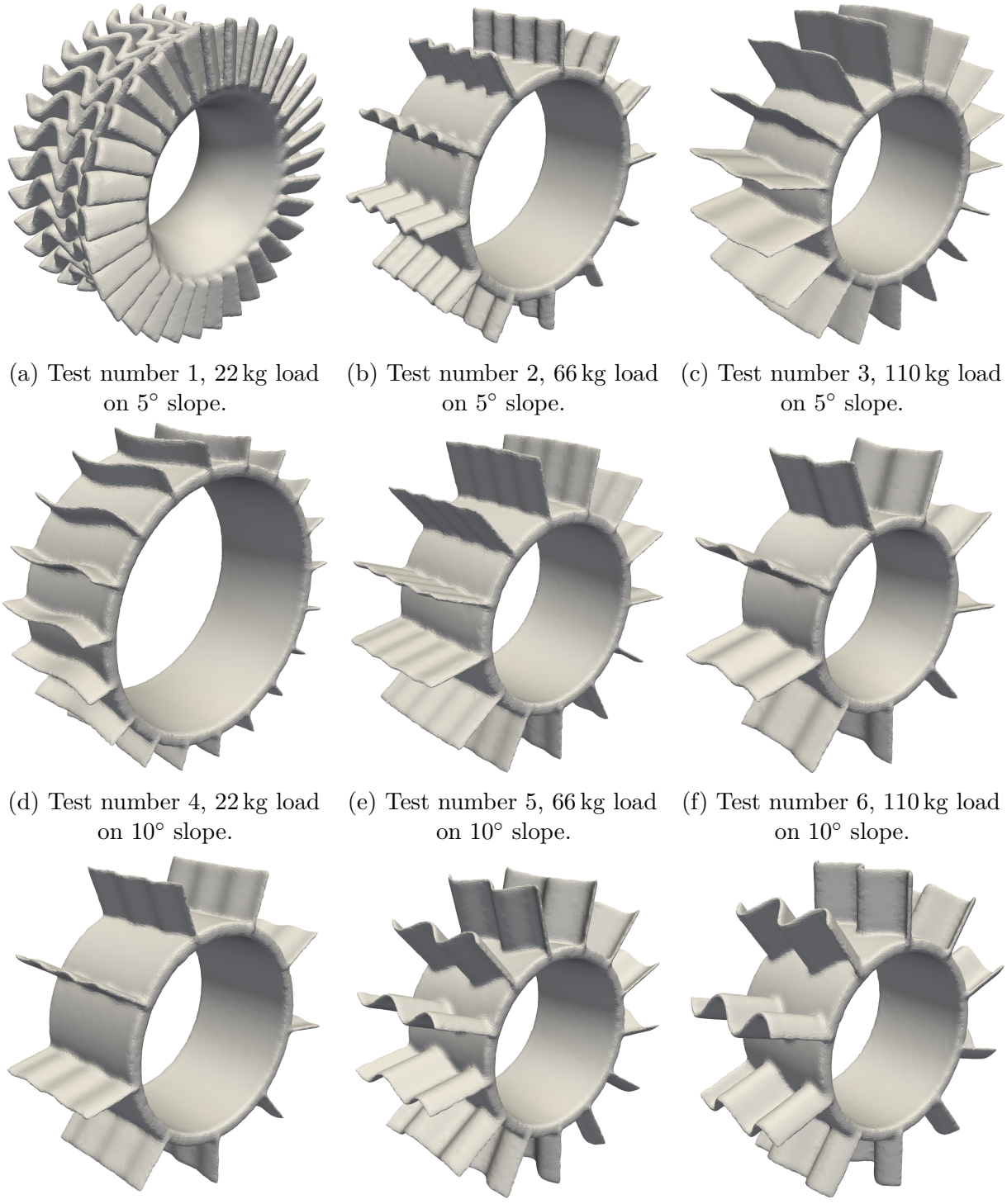
height h due to the constraints on the overall profile p. When the incline-climbing test poses more demanding conditions—marked by elevated effective mass m and steeper slope α—the optimizer often augments h at the expense of r. Conversely, under more lenient test conditions, the optimizer leans towards enlarging r at the cost of h. This implies that while wheels with a larger r offer superior climbing abilities, an adequate grouser height h is indispensable to preserve ground traction. The design optimization effectively navigates this delicate equilibrium.

2. The control point deviation c is generally minimal (in absolute value) in the optimized results. This is consistent with our ICE plot analysis, where negative values of c impair climbing efficiency. Although positive values of c can be beneficial, they inflate the profile p, and there are more effective means of enhancing the climbing ability. The wheel width w predominantly approaches its upper limit in our results. Other design parameters, such as the grouser wave number b and thickness t, exert minor influence, with their values appearing rather arbitrary across designs.

3. In certain instances, the optimizer opts to elevate the grouser wave amplitude a to make up for a shortfall in the steering ability metric S_t, even if it might slightly curtail the

climbing ability.

4. Notably, test number 1 represents an unintended design: The grousers are so densely packed that they coalesce. This outcome is logical, as the merged grousers effectively enlarge the wheel's radius while maintaining a textured surface for optimal traction and steering. Given that $\alpha = 5^\circ$ and $m = 22\,\mathrm{kg}$ signify a relatively lenient testing condition, this corresponds with the anticipated behavior of the optimizer, which leans towards favoring r over h.



(a) Test number 1, 22 kg load on 5° slope.

(b) Test number 2, 66 kg load on 5° slope.

(c) Test number 3, 110 kg load on 5° slope.

(d) Test number 4, 22 kg load on 10° slope.

(e) Test number 5, 66 kg load on 10° slope.

(f) Test number 6, 110 kg load on 10° slope.

(g) Test number 7, 22 kg load on 15° slope.

(h) Test number 8, 66 kg load on 15° slope.

(i) Test number 9, 110 kg load on 15° slope.

Figure 4.9: The optimized rover wheel designs, generated by querying the trained GP model.

4.3.1 Validating designs using DEM

The designs we have previously detailed are direct outcomes from the GP evaluation instead of rigorous numerical testing. Thus, we expose them to the same set of DEM simulations as described in Sec. 4.1, the basis for our original dataset generation. The comparison of the GP-predicted performance metrics against those derived from DEM simulations is tabulated in Table 4.5.

Table 4.5: Optimization results with their GP-predicted performance metrics validated with DEM simulations.

Test no.	Energy efficiency E_e			Steering ratio S_t		
	Predicted	Simulated	Discrepancy	Predicted	Simulated	Discrepancy
1	0.272	0.218	24%	0.147	0.147	0%
2	0.16	0.168	4.7%	0.132	0.125	5.6%
3	0.118	0.123	4.4%	0.132	0.124	6.4%
4	0.32	0.345	7.2%	0.133	0.118	13%
5	0.222	0.236	6.1%	0.138	0.137	0.27%
6	0.154	0.158	2.7%	0.138	0.127	8.7%
7	0.346	0.346	0%	0.144	0.13	10%
8	0.246	0.231	6.7%	0.142	0.135	5.5%
9	0.156	0.153	2.1%	0.132	0.131	0.78%

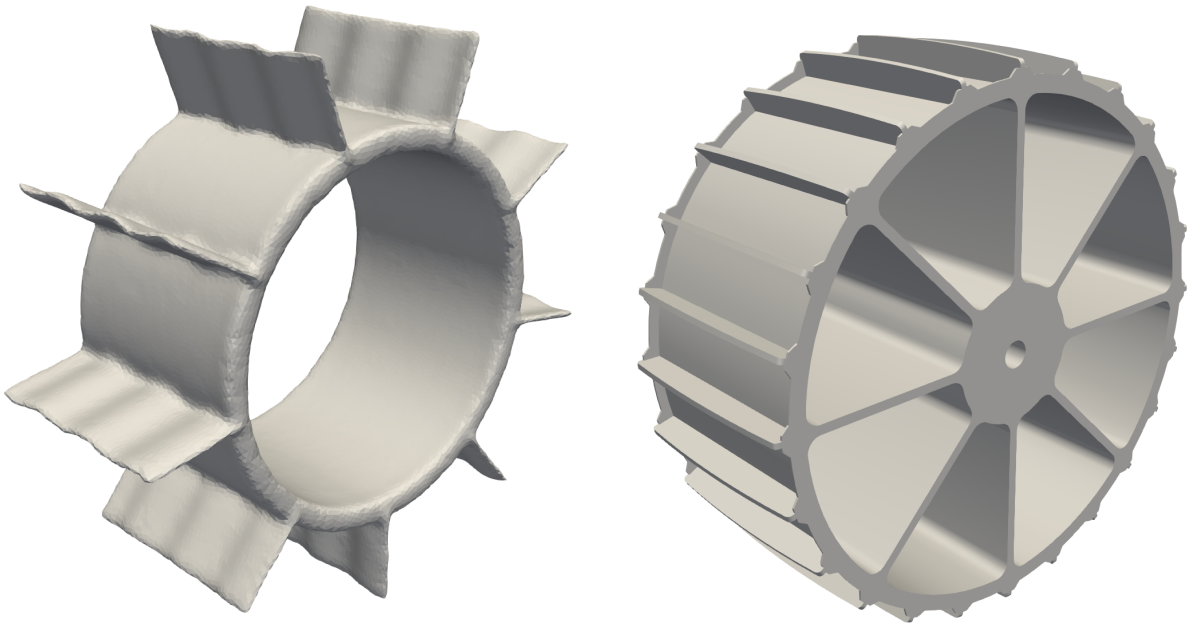
Most of the designs demonstrate congruence between the GP predictions and the DEM simulation outcomes, attesting to the robustness of our predictive model and, by extension, reinforcing confidence in our optimized designs. An exception is noted in test number 1. As previously highlighted, in this design the grousers emerged unexpectedly. We hypothesize that for such a corner case, our GP model might not be optimally trained to render precise performance predictions.

4.3.2 Sensitivity at optimum

We select the design corresponding to the test case with $\alpha = 15°$ and $m = 22\,\text{kg}$ (test number 7) for in-depth analysis, as it closely matches the test conditions of the physical experiments conducted on MGRU3. A side-by-side comparison of this chosen optimized design with the original MGRU3 wheel is depicted in Fig. 4.10.

To gauge the robustness of the optimal design displayed in Fig. 4.10a, we introduce perturbations to some of its parameters. Each modified design undergoes the set of DEM simulations in Sec. 4.1, allowing us to assess the impact of these perturbations. The altered design parameter values and their resulting performances are consolidated in Table 4.6.

The observations indicate that reallocating a mere one centimeter from the radius r to the grouser height h, and vice versa, leads to a slight decline in energy efficiency compared to the original optimal design. A similar degradation in performance is noted when either adding or subtracting two grousers. This underlines that our derived design for this test condition indeed sits at a local optimum.



(a) Optimized wheel at $\alpha = 15°$ and $m = 22\,\text{kg}$. (b) MGRU3 wheel.

Figure 4.10: Comparison between the optimized wheel design and the original MGRU3 wheel design. They have the same profile p and width w, yet the optimized design favors grouser height than radius.

Table 4.6: The climbing and steering ability of the designs that are slightly perturbed from the "base" case shown in Fig. 4.10a.

Test no.	Design parameters			Performance metrics	
	r	h	n	E_e	S_t
base	0.1648	0.0764	9	0.346	0.135
1	0.1548	0.0864	9	0.325	0.13
2	0.1748	0.0664	9	0.34	0.105
3	0.1648	0.0764	7	0.316	0.12
4	0.1648	0.0764	11	0.343	0.144

4.3.3 Full-rover performance comparison

To robustly evaluate the enhancements brought by our optimal design, we assess its performance employing the DS, the precise GRC-1 simulant representation used throughout

Chap. 3. The co-simulations, underpinned by the DS, conducted in this section align with the numerical specifications delineated in Sec. 3.2.2. These simulations feature the VIPER chassis paired with optimized wheels, as they ascend inclines ranging from 0° to 25°. An illustrative scene from the test is rendered in Fig. 4.11.

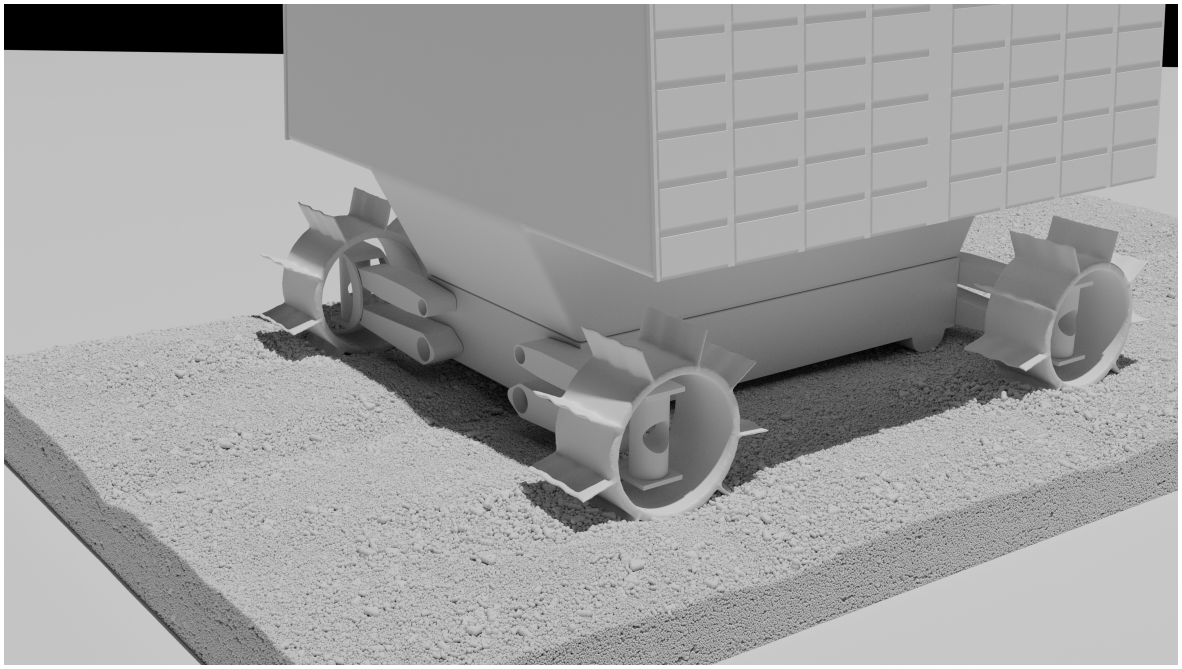


Figure 4.11: A rendering of the VIPER rover operating on a 20° incline, using the optimized wheel design.

A comparative analysis of the slip against slope data for both the optimized wheel and the original MGRU3 wheel is presented in Fig. 4.12. Concurrently, Fig. 4.13 offers a side-by-side comparison in terms of energy efficiency. Across both criteria, our optimized design clearly outperforms, with a noteworthy slip ratio reduction exceeding 0.2 on steeper inclines. However, it is pertinent to mention that on gentler slopes, the optimized wheel exhibits a slightly higher slip ratio. This can be attributed to the grousers' penetration into the terrain, leading to a reduced effective wheel radius contributing to linear velocity. This effect becomes more pronounced on flatter terrains.

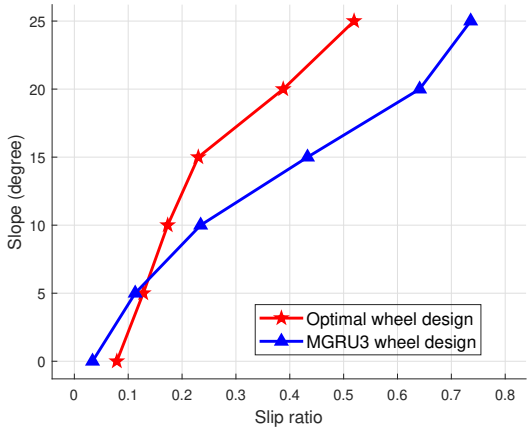


Figure 4.12: The comparison between the optimized wheel and the original MGRU3 wheel, in terms of the slip against slope data.

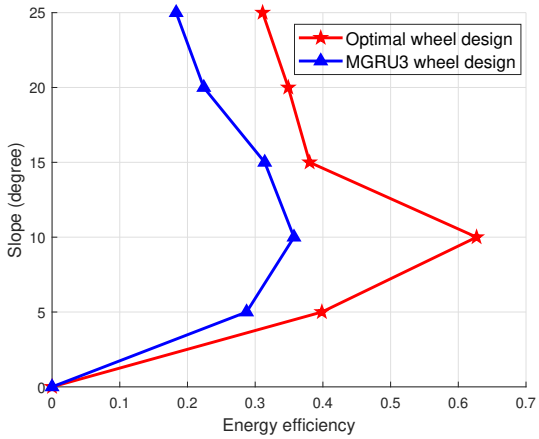


Figure 4.13: The comparison between the optimized wheel and the original MGRU3 wheel, in terms of energy efficiency.

Furthermore, for both slip and energy efficiency metrics, we demonstrate (as seen in Fig. 4.14 and Fig. 4.15) that the terrains modeled by both the DS and the reduced simulant representation yield nearly identical results. Such congruence provides confidence in the fidelity of our DEM simulations to accurately represent rover dynamics.

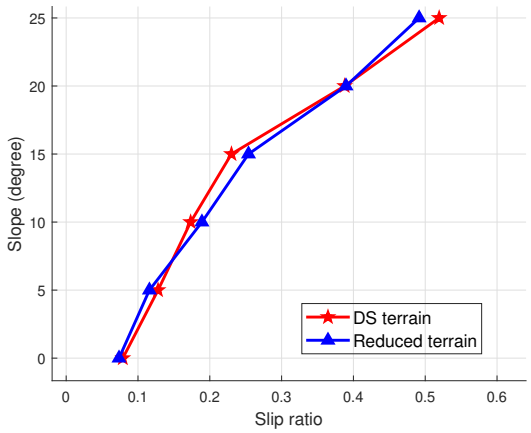


Figure 4.14: The comparison between the DS and the reduced simulant representation, in terms of the slip against slope results using the optimized wheel.

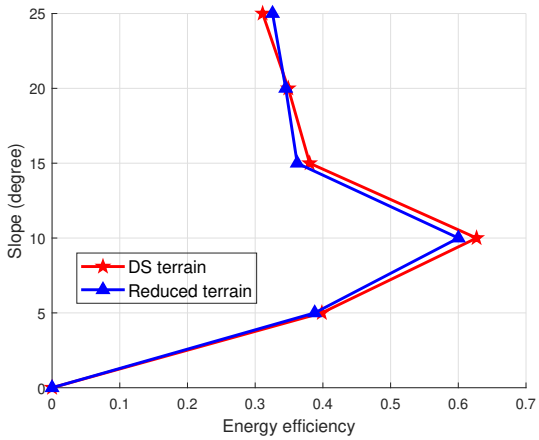


Figure 4.15: The comparison between the DS and the reduced simulant representation, in terms of the energy efficiency results using the optimized wheel.

4.4 Expanding design domain with Bayesian optimization

In the previous sections, we employ GP regression as an effective surrogate model, trained on an existing dataset to predict the objective function's response in the defined variable space. However, this question remains to be answered: What if the ambition of the optimization extends beyond the confinement of this predefined variable range? In the Bayesian optimization framework presented in this section, the objective function evaluation is divided into two parts: For sample points situated within the original dataset's variable range, a direct assessment through the GP surrogate model suffices; Conversely, for those falling outside the origial dataset, a recourse to a more computationally expensive DEM is used to procure the function value. By integrating these methods, our Bayesian optimization can expand the design space while capitalizing on the computational expediency of surrogate evaluations.

4.4.1 Bayesian optimization

Bayesian optimization is a model-based optimization strategy used to find the minimum (or maximum) of expensive and noisy objective functions [110]. This technique constructs a probabilistic model to approximate the unknown objective function and then utilizes this model to guide the search for optimal points. Specifically, Bayesian optimization typically employs a GP as a surrogate model due to the GP's capability to quantify the uncertainty in the function's estimates. The exploration–exploitation trade-off during optimization is managed by acquisition functions including Expected Improvement (EI) [111], which is used in this work. The acquisition function determines the next point to evaluate by weighing the predicted mean and uncertainty from the GP model. Bayesian optimization has demonstrated success in various domains, including hyperparameter tuning and experimental design, due to its efficiency and capability to handle noisy evaluations.

Given a Gaussian process surrogate model of our objective function, the EI at a point $\mathbf{x}$

quantifies the expected amount by which the function value $f(\mathbf{x})$ will surpass some benchmark value, typically the best value observed so far. Consider f_{best} as the best observed value of the objective function so far. In the context of Bayesian optimization with GPs, the posterior distribution of the objective function, given observed data, is also modeled as a GP. From this posterior GP, we can compute the mean $\mu(\mathbf{x})$ and the standard deviation $\sigma(\mathbf{x})$ for any input $\mathbf{x}$:

$$\mu(\mathbf{x}) \leftarrow \text{mean of the GP at } \mathbf{x}, \tag{4.7}$$

$$\sigma(\mathbf{x}) \leftarrow \text{standard deviation of the GP at } \mathbf{x}. \tag{4.8}$$

Given the current best observed objective value f_{best}, the EI at point $\mathbf{x}$ is:

$$EI(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f_{\text{best}})\Phi(Z) + \sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0, \\ 0 & \text{if } \sigma(\mathbf{x}) = 0, \end{cases} \tag{4.9}$$

where $Z = \frac{\mu(\mathbf{x}) - f_{\text{best}}}{\sigma(\mathbf{x})}$, Φ is the Cumulative Distribution Function (CDF) of the standard normal distribution, and ϕ is its Probability Density Function (PDF).

To summarize, the Bayesian optimization procedure using EI is provided with pseudocode in Algorithm 3.

Algorithm 3 Bayesian optimization with EI

1: **Input:** Initial observations $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
2: Train initial GP with dataset $\mathcal{D}$
3: **for** $t = 1, 2, \ldots, T$ **do**
4: Find $x_{\text{next}} = \arg\max_x \text{EI}(x)$
5: Evaluate objective function $y_{\text{next}} = f(x_{\text{next}})$
6: Update $\mathcal{D} = \mathcal{D} \cup \{(x_{\text{next}}, y_{\text{next}})\}$
7: Retrain GP with updated dataset $\mathcal{D}$
8: **end for**
9: **Output:** x^* with the smallest observed y

4.4.2 Optimization result

To evaluate the proposed two-part objective function methodology, we manufacture a test environment. In this setting, we remove data points with $r > 0.3\,\mathrm{m}$ from the original dataset and subsequently train a new GP that predicts the energy efficiency, E_e. The revamped GP model is based on 1744 data points, smaller than half of the original dataset. This setup mandates a direct DEM simulation for evaluating a design with $r > 0.3\,\mathrm{m}$ instead of simply querying the GP model. The function value $f(\mathbf{x})$ for this Bayesian optimization is expressed as

$$f(\mathbf{x}) = \begin{cases} \text{mean of the GP at } \mathbf{x}, & \text{if } r \leq 0.3\,\mathrm{m}, \\ \text{DEM simulation result at } \mathbf{x}, & \text{otherwise.} \end{cases} \tag{4.10}$$

The design space adheres to the criteria presented in Table 4.1. This setup simulates scenarios where online DEM simulations are integrated into the optimization process to widen the design space, when the original dataset falls short of covering the desired design scope.

The optimization constraints deviate slightly from those detailed in Sec. 4.3. Specifically, the overall profile must satisfy $p \leq 0.5\,\mathrm{m}$, the width is constrained by $w \leq 0.4\,\mathrm{m}$, and the steering ratio is $S_t \geq 0.13$. The climbing test in this optimization uses the effective mass $m = 110\,\mathrm{kg}$ and the slope $\alpha = 15°$. These conditions represent a target design that is two times larger than the original MGRU3 wheel, and carries a realistic load. They also aim to guide the optimization towards a design with $r > 0.3\,\mathrm{m}$, thereby assessing the optimizer's capability in integrating DEM simulations.

To implement the outlined Bayesian optimization strategy, we leverage the capabilities of skopt.gp_minimize, a well-established tool from the Scikit-optimize library [112]. EI is used as the acquisition function. We conduct 50 random samples to "warm start" the optimization, then carry out 150 additional Bayesian optimization steps to obtain the optimal design. The parameters for the optimized design are given in Table 4.7 as test number 1. The corresponding wheel is rendered in Fig. 4.16.

The baseline for comparison is, by contrast, obtained using the GP trained with the full

dataset, the same as the one used in Sec. 4.3. The parameters for the optimized design are provided in Table 4.7 as test number 2. The corresponding wheel is rendered in Fig. 4.17. Expectedly, the optimized designs shown in Fig. 4.16 and 4.17 are similar, underscoring the capability of Bayesian optimization to effectively explore the wheel design space not represented by the dataset. The design produced by Bayesian optimization has a smaller overall profile p and exhibits slightly inferior performance. We attribute this discrepancy to the penalty-based enforcement of constraints which deters the optimizer from investigating values of p nearing 0.5, since any trial exceeding $p > 0.5$ promptly sets the function value to zero.

A "hard" expansion of the dataset, namely generating more data points to fill the missing space, is often favored since it simplifies subsequent research inquiries about optimal design under varied use conditions. However, when the target use condition is predefined and constant, Bayesian optimization proves valuable for generating the optimum with a minimal set of resource-intensive evaluations, like DEM simulations for wheel performance. Notably, the data points generated during this optimization can also be incorporated into the existing dataset to expand it.

Table 4.7: Optimization results when wheel profile is constrained to be $p \leq 0.5\,\mathrm{m}$ and $w \leq 0.4\,\mathrm{m}$. Test number 1 shows the optimized design by Bayesian optimization using half the data points. Test number 2 shows the optimized design given by the full-data GP model. The resultant E_e is verified by DEM simulations.

Non-design param.			Optimized design parameters and performance								
Test no.	α [°]	m [kg]	a [m]	b [-]	h [m]	t [m]	n [-]	c [-]	r [m]	w [m]	E_e [-]
1	15	110	0	4	0.0855	0.01	14	0.0015	0.358	0.4	0.467
2	15	110	0.0001	2	0.0763	0.0138	18	-0.0309	0.42	0.3965	0.481

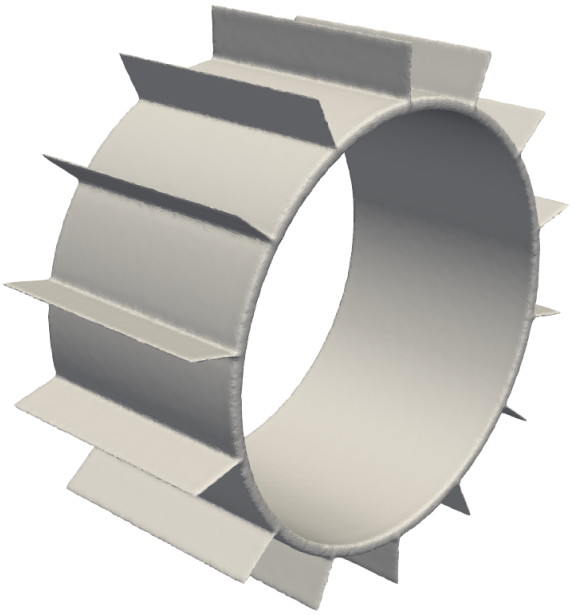


Figure 4.16: The optimized wheel design using Bayesian optimization, with the partial dataset.

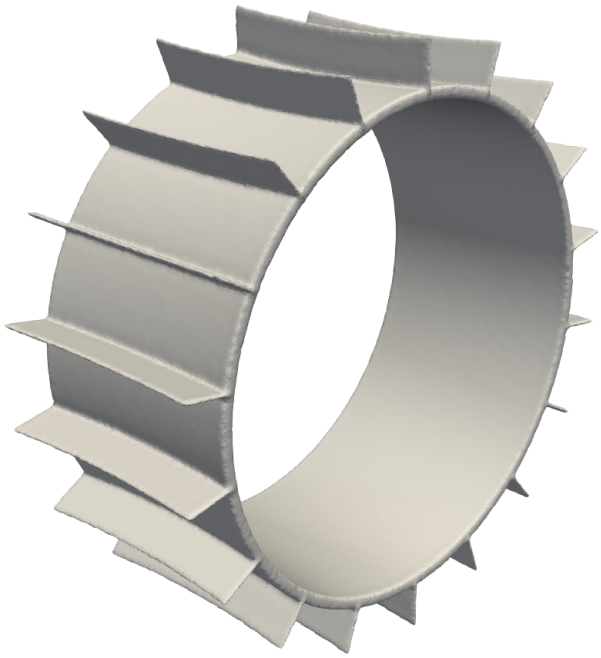


Figure 4.17: The optimized wheel design using GP model evaluations, trained with the full original dataset.

5 CONTRIBUTION AND FUTURE WORK

5.1 Summary of contributions

The hallmark achievement of this work is the introduction of Chrono DEM-Engine, a physics-based, open-source, multi-GPU DEM package with support for complex element shapes, serving as an upgrade from the existing Chrono::GPU simulator. This new DEM solver handles efficiently hundreds of millions of particles on two A100 GPUs, achieving a throughput of one million time steps for one million DEM elements within an hour. This DEM simulator offers a combination of efficiency and flexibility that is not seen in the existing tools the candidate surveyed. The software has been adopted by researchers from Northwestern University, Arizona State University, and users of the Chrono community.

The most salient numerical feature of Chrono DEM-Engine is its decoupling of kinematic processes (like contact detection) from dynamic computations (e.g., computing contact forces and performing numerical integrations) into two threads. Due to inherent differences in per-time computational demands, these threads operate asynchronously, exchanging contact state information only as necessary. Chrono DEM-Engine can support custom force models, enabled by just-in-time CUDA kernel compilation. It also offers support for complex grain geometries via clumped spheres and integrates with the multi-physics simulation engine Chrono, facilitating co-simulations with complex mechanical systems. While the Chrono::GPU predecessor of the new DEM solver was noted for running two orders of magnitude faster than two other established DEM packages, the scaling analysis in this thesis indicates that the new solver further amplifies this performance by a factor of 2×. The new solver also shows linear scalability for up to 150 million clump component spheres using two GPUs. Its implementation is validated in this work using fine-grain force model tests and macro-scale ball drop and hopper flow rate tests.

Chrono DEM-Engine is showcased in conjunction with a granular material representation research. A type of digital DEM simulant that closely resembles the GRC-1 extraterrestrial

simulant is modeled using elements of nontrivial shapes. Given that particle sizes in GRC-1 go down to the micron level, the digital simulant used herein qualitatively embraces the element size distribution of GRC-1 but increases all element sizes by a factor of 20. Despite this size augmentation, the digital simulant is shown to behave similarly to GRC-1 in several tests, i.e., angle of repose, cone penetration, drawbar pull, and single-wheel and full-rover incline slip vs. slope tests. The co-simulations between DEM-Engine and Chrono scale up to tens of millions of elements and finish in a reasonable amount of time on a single compute node.

With a numerically capable DEM simulator and physically representative extraterrestrial terrain modeling, a data-centric rover wheel design exploration is carried out. The primary objective is to identify wheel designs that exhibit superior climbing and steering attributes. While existing studies have often been restricted in scope, focusing on only a handful of design variations (for example, in a research study from NASA Jet Propulsion Laboratory [113], seven wheel designs were tested), the new DEM solver, owing to its speed of execution, allows us to cast a wider net. We evaluated over four thousand distinct designs, generated by parametrizing key wheel elements like radii and grouser heights. This exhaustive analysis endowed us with a rich dataset, facilitating the training of a Gaussian process model adept at predicting performance. Utilizing this model, we were able to:

1. Dissect the contribution of each design parameter, determining their influence using ICE plots; and

2. Find the balance between design parameters to produce optimized designs tailored for varying test scenarios.

In direct comparisons, the design proposed in this thesis surpasses the performance of the benchmark MGRU3 wheel in validation DEM simulations. Finally, we show that Bayesian optimization can be used in conjunction with an existing Gaussian process to explore and enrich the understanding of the uncharted design space.

5.2 Future research

The numerical capability of Chrono DEM-Engine, while commendable, does not diminish the inherent challenges associated with advanced simulation tools, primarily the steep learning curve. Users must navigate a multitude of provided APIs, each with its own functionalities and nuances. Understanding and overcoming the tool's limitations can also be challenging as they might necessitate time-intensive custom developments. This challenge amplifies in contemporary cross-disciplinary research where researchers must grapple with an expanding array of tools. Yet, with the advent of Large Language Models (LLMs) [114], a promising solution beckons. We propose utilizing APIs from LLM providers to develop assistant AIs capable of transforming user's natural language instructions into executable DEM-Engine scripts. Should this line of research come to fruition, the resulting product will remain open-source.

The design optimization endeavors presented reveal a limitation which is the manual, labor-intensive parameterization process. Limiting the design parameters to be the well-defined wheel features, such as the grousers, has the benefit of allowing for an intuitive interpretation of the features' influences, i.e. there is no black box. However, it also limits the diversity of the design space, and does not scale well into a broader type of design optimization scenarios. Drawing inspiration from existing AI-powered design explorations, an alternative data-centric parameterization strategy is proposed. This entails using a generative model fed by a random input number. Post training, this model should generate designs anchored in current understanding, with subsequent simulation validations enhancing a predictive model mapping the design's "hash" to its performance. This modus operandi has the potential of better scalability and adaptability, priming it for a broad array of simulation-driven design studies and fostering avenues yet to be chartered.

BIBLIOGRAPHY

[1] Xiaodong Cui, Jianjian Dai, Haotian Xu, and Xi Gao. SuperDEM simulation and experiment validation of nonspherical particles flows in a rotating drum. *Industrial & Engineering Chemistry Research*, 62(16):6525–6535, 2023.

[2] H.A. Oravec, X. Zeng, and V.M. Asnani. Design and characterization of GRC-1: A soil for lunar terramechanics testing in earth-ambient conditions. *Journal of Terramechanics*, 47(6):361–377, 2010.

[3] C Senatore, N Stein, F Zhou, K Bennett, R Arvidson, B Trease, R Lindemann, P Bellutta, M Heverly, and K Iagnemma. Modeling and validation of mobility characteristics of the mars science laboratory curiosity rover. In *Proc. Proceedings of the 12th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2014.

[4] Peter A Cundall and Otto DL Strack. A discrete numerical model for granular assemblies. *Geotechnique*, 29(1):47–65, 1979.

[5] Thorsten Pöschel and Thomas Schwager. *Computational Granular Dynamics: Models and Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2005.

[6] M Lemieux, G Léonard, J Doucet, L-A Leclaire, F Viens, J Chaouki, and F Bertrand. Large-scale numerical investigation of solids mixing in a v-blender using the discrete element method. *Powder Technology*, 181(2):205–216, 2008.

[7] K Apostolou and AN Hrymak. Discrete element simulation of liquid-particle flows. *Computers & Chemical Engineering*, 32(4-5):841–856, 2008.

[8] Chao-Lung Tang, Jyr-Ching Hu, Ming-Lang Lin, Jacques Angelier, Chia-Yu Lu, Yu-Chang Chan, and Hao-Tsu Chu. The Tsaoling landslide triggered by the Chi-Chi earthquake, Taiwan: insights from a discrete element simulation. *Engineering Geology*, 106(1-2):1–19, 2009.

[9] Diana Salciarini, Claudio Tamagnini, and Pietro Conversini. Discrete element modeling of debris-avalanche impact on earthfill barriers. *Physics and Chemistry of the Earth, Parts A/B/C*, 35(3-5):172–181, 2010.

[10] C. O'Sullivan. Particle-based Discrete Element Modeling: Geomechanics perspective. *Int. J. Geomech.*, 11(6):449–464, 2011.

[11] Paul Sánchez and Daniel J Scheeres. Simulating asteroid rubble piles with a self-gravitating soft-sphere distinct element method model. *The Astrophysical Journal*, 727(2):120, 2011.

[12] Frederik F Foldager, Lars J Munkholm, Ole Balling, Radu Serban, Dan Negrut, Richard J Heck, and Ole Green. Modeling soil aggregate fracture using the discrete element method. *Soil and Tillage Research*, 218:105295, 2022.

[13] A. M. Recuero, R. Serban, B. Peterson, H. Sugiyama, P. Jayakumar, and D. Negrut. A high-fidelity approach for vehicle mobility simulation: Nonlinear finite element tires operating on granular material. *Journal of Terramechanics*, 72:39 – 54, 2017.

[14] Jerome B Johnson, Anton V Kulchitsky, Paul Duvoy, Karl Iagnemma, Carmine Senatore, Raymond E Arvidson, and Jeffery Moore. Discrete element method simulations of Mars exploration rover wheel performance. *Journal of Terramechanics*, 62:31–40, 2015.

[15] OpenMP. Specification Standard 5.2. Available online at http://openmp.org/, 2021.

[16] Amit Amritkar, Surya Deb, and Danesh Tafti. Efficient parallel cfd-dem simulations using openmp. *Journal of Computational Physics*, 256:501–519, 2014.

[17] Margaret A Knuth, JB Johnson, MA Hopkins, RJ Sullivan, and JM Moore. Discrete element modeling of a mars exploration rover wheel in granular material. *Journal of Terramechanics*, 49(1):27–36, 2012.

[18] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard Version 3.0, 09 2012. Chapter author for Collective Communication, Process Topologies, and One Sided Communications.

[19] Beichuan Yan and Richard A Regueiro. A comprehensive study of mpi parallelism in three-dimensional discrete element method (dem) simulation of complex-shaped granular particles. *Computational Particle Mechanics*, 5(4):553–577, 2018.

[20] Abdoul Wahid Mainassara Checkaraou, Alban Rousset, Xavier Besseron, Sébastien Varrette, and Bernhard Peters. Hybrid mpi+ openmp implementation of extended discrete element method. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 450–457. IEEE, 2018.

[21] LIGGGHTS. Open source discrete element method particle simulation code. http://cfdem.dcs-computing.com/?q=OpenSourceDEM, 2013.

[22] LAMMPS. A molecular dynamics simulator. http://lammps.sandia.gov/, 2013.

[23] Simcenter STAR-CCM+ software website. https://plm.sw.siemens.com/en-US/simcenter/fluids-thermal-simulation/star-ccm/, 2023. Accessed: 2023-09-25.

[24] Radu Serban, Nicholas Olsen, and Dan Negrut. High performance computing framework for co-simulation of vehicle-terrain interaction. In *NDIA Ground Vehicle Systems Engineering and Technology Symposium*, 2017.

[25] Ji Xu, Huabiao Qi, Xiaojian Fang, Liqiang Lu, Wei Ge, Xiaowei Wang, Ming Xu, Feiguo Chen, Xianfeng He, and Jinghai Li. Quasi-real-time simulation of rotating drum using discrete element method with parallel gpu computing. *Particuology*, 9(4):446–450, 2011.

[26] Nicolin Govender, Daniel Wilke, and Schalk Kok. Blaze-DEMGPU: Modular high performance DEM framework for the GPU architecture. *SoftwareX*, 5:62–66, 2016.

[27] J. Gan, Z. Zhou, and A. Yu. A GPU-based DEM approach for modeling of particulate systems. *Powder Technology*, 301:1172–1182, 2016.

[28] Y He, T. Evans, A. Yu, and R. Yang. A GPU-based DEM for modeling large scale powder compaction with wide size distributions. *Powder Technology*, 333:219–228, 2018.

[29] Kazuyoshi Iwashita and Masanobu Oda. Rolling resistance at contacts in simulation of shear band development by DEM. *Journal of Engineering Mechanics*, 124(3):285–292, 1998.

[30] Alberto Di Renzo and Francesco Paolo Di Maio. Comparison of contact-force models for the simulation of collisions in DEM-based granular flow codes. *Chemical Engineering Science*, 59(3):525–541, 2004.

[31] F. da Cruz, S. Emam, M. Prochnow, J. N. Roux, and F. Chevoir. Rheophysics of dense granular materials: Discrete simulation of plane shear flows. *Physical Review E*, 72:021309, Aug 2005.

[32] C. H. Rycroft, G. S. Grest, J. W. Landry, and M. Z. Bazant. Analysis of granular flow in a pebble-bed nuclear reactor. *Physical Review E*, 74, 021306, 2006.

[33] H. Kruggel-Emden, M. Sturm, S. Wirtz, and V. Scherer. Selection of an appropriate time integration scheme for the discrete element method (dem). *Computers & Chemical Engineering*, 32(10):2263–2279, 2008.

[34] Tamer M Wasfy, Hatem M Wasfy, and Jeanne M Peters. Coupled multibody dynamics and discrete element modeling of vehicle mobility on cohesive granular terrains. In *ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V006T10A050–V006T10A050. American Society of Mechanical Engineers, 2014.

[35] S. Lommen. DEM speedup: Stiffness effects on behavior of bulk material. *Particuology*, pages 107–112, 2014.

[36] S. Utili, T. Zhao, and G.T. Houlsby. 3D DEM investigation of granular column collapse: Evaluation of debris motion and its destructive power. *Engineering Geology*, 186:3–16, 2015.

[37] M. Potticary, A. Zervos, and J. Harkness. An investigation into the effect of particle platyness on the strength of granular material using the discrete element method. In *IV International Conference on Particle-based Methods - Fundamentals and Applications*, 2015.

[38] M. Michael, F. Vogel, and B. Peters. DEM-FEM coupling simulations of the interactions between a tire tread and granular terrain. *Computer Methods in Applied mechanics and engineering*, 2015.

[39] M. Ciantia, M. Arroyo, J. Butlanska, and A. Gens. DEM modelling of cone penetration tests in a double-porosity crushable granular material. *Computers and Geotechnics*, 73:109–127, 2016.

[40] Z Zheng and M. Zang. Numerical simulations of the interactions between a pneumatic tire and granular sand by 3D DEM-FEM. In *7th International Conference on Discrete Element Methods*, pages 289–300, 2017.

[41] E. Parteli and T. Poschel. Particle-based simulation of powder application in additive manufacturing. *Powder Technology*, pages 96–102, 2016.

[42] R. Kivugo. *Tire-Soil Interaction for Off-Road Vehicle Applications*. Phd, Politecnico di Milano, 2017.

[43] F. Calvetti, C. Prisco, and E. Vairaktaris. DEM assessment of impact forces of dry granular masses on rigid barriers. *Acta Geotechnica*, 2016.

[44] Mikito Furuichi, Daisuke Nishiura, Osamu Kuwano, Arthur Bauville, Takane Hori, and Hide Sakaguchi. Arcuate stress state in accretionary prisms from real-scale numerical sandbox experiments. *Nature Scientific Reports - www.nature.com/scientificreports/*, 8, 12 2018.

[45] Oliver Henrich, Yair Augusto Gutierrez Fosado, Tine Curk, and Thomas Ouldridge. Coarse-grained simulation of dna using lammps. 02 2018.

[46] Ruihan Li, Zhaoyuan Liu, Zhiyuan Feng, Jingang Liang, and Li-Guo Zhang. High-fidelity MC-DEM modeling and uncertainty analysis of HTR-PM first criticality. *Frontiers in Energy Research*, 9, 01 2022.

[47] C. S. Dias. Molecular dynamics simulations of active matter using LAMMPS. 2021.

[48] Fatemeh Razavi, Alexandra Komrakova, and Carlos F. Lange. CFD—DEM simulation of sand-retention mechanisms in slurry flow. *Energies*, 14(13), 2021.

[49] Luning Fang, Ruochun Zhang, Colin Vanden Heuvel, Radu Serban, and Dan Negrut. Chrono::GPU: An open-source simulation package for granular dynamics using the discrete element method. *Processes*, 9(10), 2021.

[50] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut. Chrono: An open source multi-physics dynamics engine. In T. Kozubek, editor, *High Performance Computing in Science and Engineering – Lecture Notes in Computer Science*, pages 19–49. Springer International Publishing, 2016.

[51] David Reger, Elia Merzari, Paolo Balestra, Ryan Stewart, and Gerhard Strydom. Discrete element simulation of pebble bed reactors on graphics processing units. *Annals of Nuclear Energy*, 190:109896, 2023.

[52] Martin Haustein, Anton Gladkyy, and Rüdiger Schwarze. Discrete element modeling of deformable particles in YADE. *SoftwareX*, 6:118–123, 2017.

[53] Daria Romanova, Sergei Strijhak, and Matvey Kraposhin. Development of snowYadeFoam solver for snow particles simulation. In *2020 Ivannikov Ispras Open Conference (ISPRAS)*, pages 166–169, 2020.

[54] Christer Ericson. *Real Time Collision Detection*. Morgan Kaufmann, San Francisco, CA, 2005.

[55] JF Favier, MH Abbaspour-Fard, M Kremmer, and AO Raji. Shape representation of axi-symmetrical, non-spherical particles in discrete element simulation using multi-element model particles. *Engineering computations*, 1999.

[56] JE Hilton and PW Cleary. The influence of particle shape on flow modes in pneumatic conveying. *Chemical engineering science*, 66(3):231–240, 2011.

[57] Kiangi Kiangi, Alex Potapov, and Michael Moys. DEM validation of media shape effects on the load behaviour and power in a dry pilot mill. *Minerals Engineering*, 46:52–59, 2013.

[58] Bing Ren, Wenqi Zhong, Baosheng Jin, Yingjuan Shao, and Zhulin Yuan. Numerical simulation on the mixing behavior of corn-shaped particles in a spouted bed. *Powder technology*, 234:58–66, 2013.

[59] Wenqi Zhong, Aibing Yu, Xuejiao Liu, Zhenbo Tong, and Hao Zhang. DEM/CFD-DEM modelling of non-spherical particulate systems: theoretical developments and applications. *Powder technology*, 302:108–152, 2016.

[60] Reid Kawamoto, Edward Andò, Gioacchino Viggiani, and José E Andrade. All you need is shape: Predicting shear banding in sand with ls-dem. *Journal of the Mechanics and Physics of Solids*, 111:375–392, 2018.

[61] Eloïse Marteau and José E Andrade. An experimental study of the effect of particle shape on force transmission and mobilized strength of granular materials. *Journal of Applied Mechanics*, 88(11), 2021.

[62] Ruochun Zhang, Colin Vanden Heuvel, and Dan Negrut. DEM-Engine, a multi-gpu DEM solver with complex geometry support. https://github.com/projectchrono/DEM-Engine, 2022. Simulation-Based Engineering Laboratory, University of Wisconsin-Madison.

[63] Hammad Mazhar, Toby Heyn, and Dan Negrut. A scalable parallel method for large collision detection problems. *Multibody System Dynamics*, 26:37–55, 2011. 10.1007/s11044-011-9246-y.

[64] M.H. Carr and J.W. Head. Geologic history of mars. *Earth and Planetary Science Letters*, 294(3-4):185–203, 2010.

[65] S. Squyres. *Roving Mars: Spirit, Opportunity, and the Exploration of the Red Planet*. Hyperion Books, 2005.

[66] J. M. Logsdon. *Exploring the Unknown: Selected Documents in the History of the U.S. Civil Space Program*. NASA, 2010.

[67] R. Rieder et al. The new athena alpha particle x-ray spectrometer for the mars exploration rovers. *Journal of Geophysical Research: Planets*, 108(E12), 2004.

[68] M. G. Bekker. *Introduction to terrain-vehicle systems*. University of Michigan Press, Ann Arbor, MI, 1969.

[69] J. Y. Wong. *Theory of Ground Vehicles*. John Wiley & Sons, New York, 2001.

[70] H Shibly, Karl Iagnemma, and S Dubowsky. An equivalent soil mechanics formulation for rigid wheels in deformable terrain, with application to planetary exploration rovers. *Journal of Terramechanics*, 42(1):1–13, 2005.

[71] A. Konyukhov and K. Schweizerhof. *Introduction to computational contact mechanics: a geometrical approach*. Wiley, 2012.

[72] Dule Shu, James Cunningham, Gary Stump, Simon W. Miller, Michael A. Yukish, Timothy W. Simpson, and Conrad S. Tucker. 3D Design Using Generative Adversarial Networks and Physics-Based Validation. *Journal of Mechanical Design*, 142(7):071701, 11 2019.

[73] Sangeun Oh, Yongsu Jung, Seongsin Kim, Ikjin Lee, and Namwoo Kang. Deep generative design: Integration of topology optimization and generative models. *Journal of Mechanical Design*, 141(11), sep 2019.

[74] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint*, 2017.

[75] P.J. Bentley. *Evolutionary Design by Computers*. Morgan Kaufmann Publishers, 1999.

[76] NASA. Premature wear of the msl wheels. https://llis.nasa.gov/lesson/22401. Accessed: 2023-10-10.

[77] Xiaobing Guo, Zumei Zheng, Mengyan Zang, and Shunhua Chen. A multi-sphere de-fe method for traveling analysis of an off-road pneumatic tire on irregular gravel terrain. *Engineering Analysis with Boundary Elements*, 139:293–312, 2022.

[78] Eric Karpman, Jozsef Kövecses, Daniel Holz, and Krzysztof Skonieczny. Discrete element modelling for wheel-soil interaction and the analysis of the effect of gravity. *Journal of Terramechanics*, 91:139–153, 2020.

[79] Andrew Thoesen, Teresa McBryan, Darwin Mick, Marko Green, Justin Martia, and Hamid Marvi. Comparative performance of granular scaling laws for lightweight grouser wheels in sand and lunar simulant. *Powder Technology*, 373:336–346, 2020.

[80] Ben Barsdell and Kate Clark. A single-header c++ library for simplifying the use of cuda runtime compilation. https://github.com/NVIDIA/jitify. Accessed: 2023-08-24.

[81] Nathan Berry, Yonghao Zhang, and Sina Haeri. Contact models for the multi-sphere discrete element method. *Powder Technology*, 416:118209, 2023.

[82] Corné J. Coetzee and Otto C. Scheffler. Review: The calibration of dem parameters for the bulk modelling of cohesive materials. *Processes*, 11(1), 2023.

[83] Mathew Price, Vasile Murariu, and Garry Morrison. Sphere clump generation and trajectory comparison for real particles. *Proceedings of Discrete Element Modelling 2007*, 2007.

[84] H. Hertz. Ueber die verdunstung der flüssigkeiten, insbesondere des quecksilbers, im luftleeren raume. *Annalen der Physik*, 253(10):177–193, 1882.

[85] R. Mindlin and H. Deresiewicz. Elastic spheres in contact under varying oblique forces. *Journal of Applied Mechanics*, 20:327–344, 1953.

[86] Luning Fang and Dan Negrut. Producing 3D friction loads by tracking the motion of the contact point on bodies in mutual contact. *Computational Particle Mechanics*, 8:905–929, 2021.

[87] Jonathan Fleischmann, Radu Serban, Dan Negrut, and Paramsothy Jayakumar. On the importance of displacement history in soft-body contact models. *Journal of Computational and Nonlinear Dynamics*, 11(4):044502, 2016.

[88] Kenneth Langstreth Johnson. *Contact Mechanics*. Cambridge University Press, 1987.

[89] M. A. Ambroso, C. R. Santore, A. R. Abate, and D. J. Durian. Penetration depth for shallow impact cratering. *Physical Review E*, 71:051305, May 2005.

[90] T. Heyn. *On the Modeling, Simulation, and Visualization of Many-Body Dynamics Problems with Friction and Contact*. PhD thesis, Department of Mechanical Engineering, University of Wisconsin–Madison, http://sbel.wisc.edu/documents/TobyHeynThesis_PhDfinal.pdf, 2013.

[91] Bangxing Jian and Xi Gao. Investigation of spherical and non-spherical binary particles flow characteristics in a discharge hopper. *Advanced Powder Technology*, page 104011, 03 2023.

[92] Ruochun Zhang, Colin Vanden Heuvel, and Dan Negrut. DEM-Engine's grc-1-inspired digital simulant scripts. https://github.com/projectchrono/DEM-Engine/tree/Paper_Digital_Simulant, 2022. Simulation-Based Engineering Laboratory, University of Wisconsin-Madison.

[93] Hamzah M. Beakawi Al-Hashemi and Omar S. Baghabra Al-Amoudi. A review on the angle of repose of granular materials. *Powder Technology*, 330:397–417, 2018.

[94] Alois Steiner, Achim J. Kopf, Pierre Henry, Sylvia Stegmann, Ronan Apprioual, and Pascal Pelleau. Cone penetration testing to assess slope stability in the 1979 nice landslide area (ligurian margin, se france). *Marine Geology*, 369:162–181, 2015.

[95] HA Oravec, VM Asnani, and X Zeng. The development of a soil for lunar surface mobility testing in ambient conditions. In *Earth & Space 2008: Engineering, Science, Construction, and Operations in Challenging Environments*, pages 1–10. American Society of Civil Engineers, 2008.

[96] Colin Creager, Vivake Asnani, Heather Oravec, and Adam Woodward. Drawbar pull (dp) procedures for off-road vehicle testing. Technical Report GRC-E-DAA-TN31725, NASA, 2017.

[97] Project Chrono. Chrono: An open source framework for the physics-based simulation of dynamic systems. http://projectchrono.org, 2020. Accessed: 2020-03-03.

[98] Simulated Lunar Operations Laboratory. NASA's VIPER prototype motors through moon-like obstacle course. https://www.nasa.gov/feature/ames/nasas-viper-prototype-motors-through-moon-like-obstacle-course. Accessed: 2023-04-02.

[99] James Slonaker, D Carrington Motley, Carmine Senatore, Karl Iagnemma, and Ken Kamrin. Geometrically general scaling relations for locomotion on granular beds. *arXiv preprint arXiv:1604.02490*, 2016.

[100] James Slonaker, D. Carrington Motley, Qiong Zhang, Stephen Townsend, Carmine Senatore, Karl Iagnemma, and Ken Kamrin. General scaling relations for locomotion in granular media. *Phys. Rev. E*, 95:052901, May 2017.

[101] F. P. Bowden and D. Tabor. The friction and lubrication of solids. *American Journal of Physics*, 19(7):428–429, 1951.

[102] Keren Deng and Wen Hsiung Ko. A study of static friction between silicon and silicon compounds. *Journal of Micromechanics and Microengineering*, 2:14–20, 1992.

[103] Changhua Xie, Huaqing Ma, and Yongzhi Zhao. Investigation of modeling non-spherical particles by using spherical discrete element model with rolling friction. *Engineering Analysis with Boundary Elements*, 105:207–220, 2019.

[104] Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[105] M.L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer, Berlin, 1999.

[106] Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996.

[107] Joaquin Quiñonero Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *J. Mach. Learn. Res.*, 6:1939–1959, dec 2005.

[108] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015.

[109] Kenji Nagaoka, Kazumasa Sawada, and Kazuya Yoshida. Shape effects of wheel grousers on traction performance on sandy terrain. *Journal of Terramechanics*, 90, 08 2019.

[110] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[111] Jonas Mockus. *Bayesian approach to global optimization: theory and applications*, volume 37. Springer Science & Business Media, 2012.

[112] Tim Head, Manoj Kumar, Holger Nahrstaedt, Gilles Louppe, and Iaroslav Shcherbatyi. scikit-optimize/scikit-optimize: v0.9.0. Zenodo. 2021.

[113] Adriana Daca. *Predicting planetary rover mobility in reduced gravity using 1-g experiments*. PhD thesis, Concordia University, November 2022. Unpublished.

[114] OpenAI (2023), ChatGPT (Sep 25 version). https://chat.openai.com.