Robotics: Science and Systems 2023
Daegu, Republic of Korea, July 10-July 14, 2023

GranularGym: High Performance Simulation for Robotic Tasks with Granular Materials

David Millard*, Daniel Pastor†, Joseph Bowkett†, Paul Backes† and Gaurav S. Sukhatme*
*University of Southern California, Los Angeles, USA. Email: dmillard@usc.edu
†Jet Propulsion Lab, Pasadena, USA

***Abstract*—Granular materials are of critical interest to many robotic tasks in planetary science, construction, and manufacturing. However, the dynamics of granular materials are complex and often computationally very expensive to simulate. We propose a set of methodologies and a system for the fast simulation of granular materials on Graphics Processing Units (GPUs), and show that this simulation is fast enough for basic training with Reinforcement Learning algorithms, which currently require many dynamics samples to achieve acceptable performance. Our method models granular material dynamics using implicit timestepping methods for multibody rigid contacts, as well as algorithmic techniques for efficient parallel collision detection between pairs of particles and between particle and arbitrarily shaped rigid bodies, and programming techniques for minimizing warp divergence on Single-Instruction, Multiple-Thread (SIMT) chip architectures. We showcase our simulation system on several environments targeted toward robotic tasks, and release our simulator as an open-source tool.**

I. Introduction

Robots are expanding their operational domains from structured environments, like factory floors, into the unstructured world of homes [31], the outdoors [11], and other planets in our solar system [10]. Successful operation in any environment requires a robot to predict the effect of its actions on the world, so that it may select an action that best achieves its goal.

The dynamics of the objects in a robot's environment may be quite complex. In addition to rigid objects, robots may encounter deformable [2] or sliceable [15] objects including cloth [26] or ropes [19]. Here, our focus is on robots interacting with large quantities of granular material like rock, sand, or loose soil, and which are given tasks that require prediction of the bulk state of such materials, such as material transport or shaping.

While machine learning is a powerful and generalizable tool for robot prediction and perception [12], it remains necessary to use large models and a large number of samples to achieve accurate performance. Physical modeling, which we refer to in this paper as simulation, is a well-tested and interpretable method for dynamical system prediction. Robust and efficient simulation provides robotics engineers with accurate, safe, and fast environments to test algorithms, train data-driven learning agents, and serve as a predictive model in robotic control algorithms.

To these ends, we have developed `GranularGym`, a faster-than-realtime simulation engine for the mechanics of granular materials with tens of thousands of particles and at interactive speeds for hundreds of thousands of particles, running on a single commodity GPU. Additionally, we contribute a set of environments that we utilize as benchmarks for algorithms intending to solve granular material manipulation tasks. We document the equations of motion for the simulated dynamics, describe the algorithms and data structures used for high-performance simulation on GPUs, analyze the performance of our engine and how it scales with various parameters, and present several benchmark environments and associated scaling of performance, and show that our engine is performant enough to train state-of-the-art reinforcement learning algorithms to achieve high rewards.

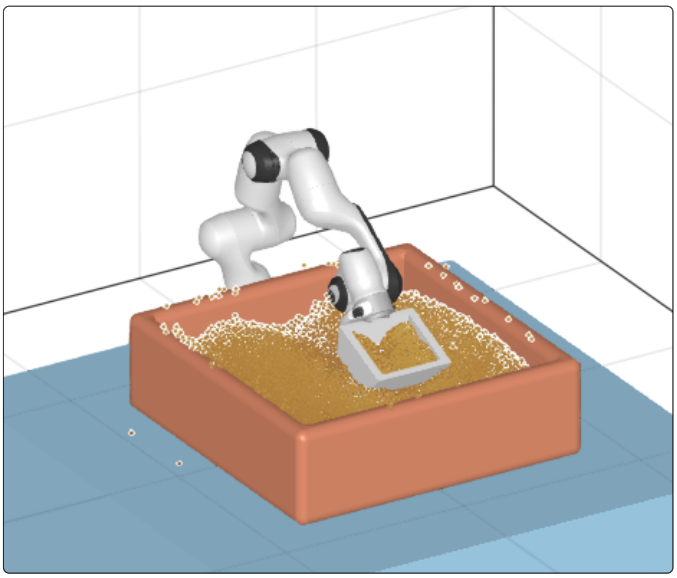

Fig. 1. Simulation of a Franka Emika Panda robot with a rigidly attached scoop attachment, interacting with a bed of 50000 particles, running at realtime on a single NVIDIA GeForce RTX 3080 Ti with a simulated timestep of 5×10^{-4}. Our simulation engine approximates granular material state and dynamics as a system of rigidly interacting spherical particles, which may interact with kinematically driven rigid bodies of arbitrary geometry.

Our approach uses a simplified model of granular interaction based on rigid body interparticle contact, which may not capture the full, vast space of rheological phenomena found in nature. Nevertheless, we believe that fast, approximate simulation in complex domains is a powerful tool for the development of robotic autonomy, particularly in closed-loop robotic systems where modeling errors may be accounted for and corrected based on sensor observations.

Algorithm 1: Projected Jacobi Algorithm

$\Delta v \leftarrow 0$;
foreach iteration from 1 to n **do**
  foreach contact pair (i, j, ψ) **do**
    $b \leftarrow J_{i,j}^T(v_i^{(k)} - \gamma v_j^{(k)} + \Delta t F_{ext,i} + \Delta v_i)$;
    $b[1] \leftarrow \max(b[1] + \frac{\alpha\psi}{\Delta t}, 0)$;
    `// Project onto Coulomb friction cone.`
    if $||b[2:3]||_2 > \mu \cdot (b[1])$ **then**
      $b[2:3] \leftarrow \mu \cdot (b[1]) \frac{b[2:3]}{||b[2:3]||_2}$;
    end if
    $\Delta v_i \leftarrow \Delta v_i + J_{i,j} b$;
  end foreach
end foreach

The code for `GranularGym` is released under an open-source license and is written in Julia [5], with a Python interface available, and targets multithreaded CPU and GPU architectures that support the NVIDIA Compute Unified Device Architecture (CUDA) or Apple Metal frameworks, and is easily portable to other parallel compute platforms.

II. Methods

We simulate rigid, dry-frictional contact of n_p spherical particles and n_b rigid bodies of arbitrary geometry interacting in a scene. The particle states and velocities are described by matrices $x, v \in \mathbb{R}^{3\times n}$. Each rigid body's state is given by a six Degree of Freedom (DoF) transform ${}_0X^i \in SE(3)$ from the world frame 0 into the frame of body i. Rigid body i also has velocity $v_i \in se(3)$, where $se(3)$ is the Lie algebra of the Special Euclidean Lie group $SE(3)$. Rigid bodies in our engine are fully driven by time and may exert forces on granular particles but do not accumulate a reaction force.

In the following subsections, we describe our algorithm for computing contact impulses across large systems of particles, efficient parallel data structures for broadphase collision detection and non-convex rigid body collision detection, and branched execution considerations for efficient programming in SIMT GPU environments.

A. Implicit Contact Impulses with Projected Jacobi

We compute particle impulses with rigid dry frictional contact using an implicit timestepping approach common in rigid body simulation by constructing a Nonlinear Complementarity Problem (NCP) to solve for the systemic contact impulse. A thorough treatment and comparison of implicit timestepping methods is given in Horak and Trinkle [16]. We use a parallelized variant of the serialized Projected Gauss-Seidel (PGS) method to solve this NCP, which we call the Projected Jacobi Algorithm (PJA), listed in Algorithm 1. For completeness, in this section, we detail the computation of the contact impulse NCP and then describe the PJA solver and its implementation on a CUDA GPU.

During collision impulse calculation, it is useful to establish a *contact frame*, shown in Figure 2. A contact frame is a local coordinate system with the origin at the point of contact and the e_1-axis oriented along the contact normal. As a result, the contact normal force is axis-aligned with e_1, and frictional forces exist only in the e_2e_3 plane. For contact between particles i and j, the *contact Jacobian* $J_{i,j} \in R^{3\times 3}$ is the transformation into the associated contact frame. We assemble these into a large, sparse contact Jacobian for the entire system, $J \in \mathbb{R}^{n_c \times n_p}$, where n_c is the number of contacts at the current timestep. The contact impulses Δv is then computed from the solution to Equation (1), where M is the mass-inertia matrix, $v^{(k)}$ is the system velocity at timestep k, $c^{(k)}$ is the post-contact relative velocity in the contact frame at timestep k, F_{ext} is the summary of any external forces applied to the system, such as gravity, and Δt is the timestep.

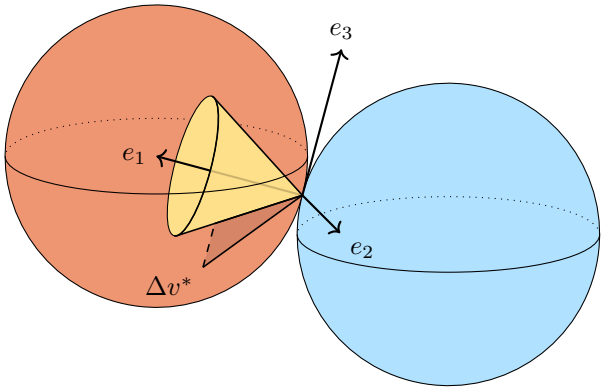

Fig. 2. Graphical representation of a contact frame between two objects (red and blue). The contact normal is aligned with the frame's e_1-axis, while the e_2 and e_3 axes are an arbitrary basis for the frictional force plane. A cone representing the bounds of dry Coulomb friction is shown in yellow. Δv^* shows a candidate impulse, and the dashed line shows its projection onto the friction cone via the solution of the contact NCP described in Section II-A.

$$c^{(k+1)} = \underbrace{JM^{-1}J^T}_{A}\Delta v + \underbrace{J(v^{(k)} + M^{-1}\Delta t F_{ext})}_{b} \quad (1)$$

To enforce the non-penetration of rigid body contact, where objects may either exert a contact force on one another, or accelerate away from one another, but not both, we add the complementarity constraint in Equation (2) to Equation (1) on the normal directions

$$0 \leq \Delta v \cdot e_1 \quad \perp \quad (A\Delta v + b) \cdot e_1 \geq 0 \quad (2)$$

Furthermore, to enforce a Coulomb approximation of dry friction, we include the nonlinear constraint in Equation (3) between the tangent and normal impulses

$$||\text{proj}_{e_2e_3}\Delta v||_2 \leq \mu \Delta v \cdot e_1, \quad (3)$$

where μ is the coefficient of friction and $\text{proj}_{e_2e_3}$ is a projection operator into the e_2e_3 plane. The projection onto the friction cone is the nonlinearity in the NCP. Together, Equations (1) to (3) form the systemic contact impulse NCP. The post separation contact velocities $c^{(k)}$ are treated as slack variables, and we use the resulting contact impulses Δv in a symplectic Euler time integration scheme, where the updated velocity

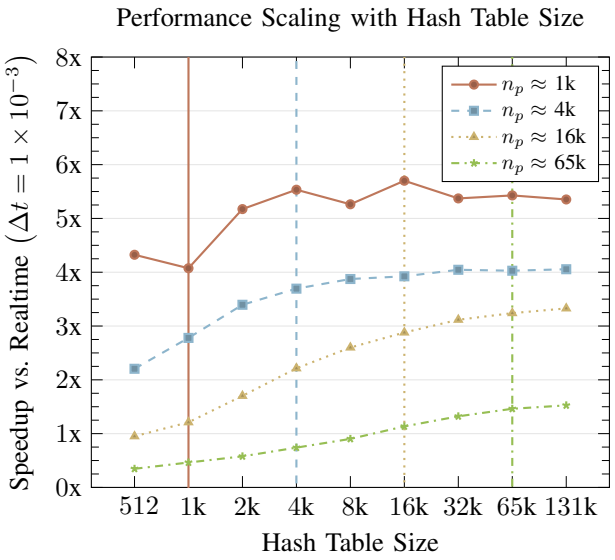

Fig. 3. Plot of speedup over realtime against hash table size (higher numbers are better). To maximize the number of collision candidates, we simulate particles at rest inside a tall cylindrical column. The vertical lines mark the number of particles used in the performance plot of the corresponding style. Based on the shape of these performance curves, we propose a heuristic hash table size of twice the number of particles simulated.

is computed first, per Equation (4), and then used to update positions, per Equation (5).

$$v^{(k+1)} = v^{(k)} + \Delta t \Delta v \tag{4}$$

$$x^{(k+1)} = x^{(k)} + v^{(k+1)} \tag{5}$$

We use a symplectic Euler integrator for its energy preserving properties, particularly in systems of dynamics with rigid body contacts[8].

Due to accumulating errors inherent in simulation, particularly in implicit methods, particles may intersect by small amounts. Since the contact impulse NCP of Equation (1) is specified in terms of velocity and impulse, rather than position, such errors would accumulate if left untreated. To mitigate erroneous penetration, we apply a stabilization scheme proposed by Baumgarte [3], by applying a small corrective impulse along the contact normal of magnitude $\frac{\alpha\psi}{\Delta t}$, where ψ is the penetration depth, and α is a user chosen parameter.

B. Linked Spatial Hashmaps for Broadphase Nearest Neighbor Searches

A naive implementation for detecting collisions between neighboring particles scales as $O(n^2)$ with the number of particles in the system and would contribute a significant source of computation time each timestep. To alleviate this, we implement a linked hashmap data structure to generate lists of candidate collision pairs, which can be built in parallel in GPU global memory. By hashing three-dimensional points according to a discretized spatial hashing rule, we can quickly check discretized regions of state space for particle occupancy and, therefore, significantly filter the list of potential collisions. We build this data structure atomically in GPU memory with

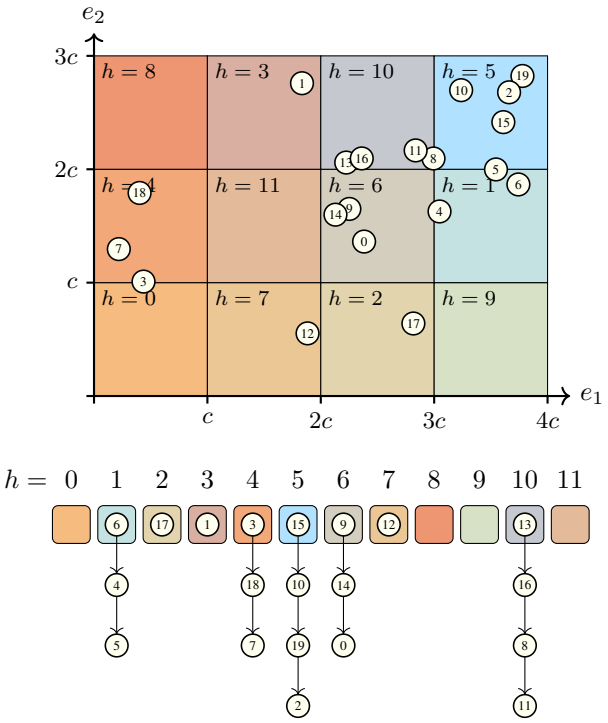

Fig. 4. Graphical representation of the spatial hashmap data structure for fast, parallel broadphase collision checking. Physical coordinates are divided into cells of size c, each with a hash value h computed from the cell's location. A linked hash table, shown in the lower half of the figure, is constructed in parallel according to Algorithm 2, with one thread for each particle in the scene. This data structure generalizes to any number of dimensions; we show a two-dimensional representation for clarity.

a single Atomic Compare-and-Swap (CAS) instruction, using Algorithm 2. Since each particle can belong to at most one cell, the entire set of linked lists can be stored in a single, preallocated array of size n_p. A graphical representation of the data structure is represented in Figure 4. We hash the discretized coordinates of individual particles with Equation (6), a modification of the rule proposed by Teschner et al. [28],

$$h(x_i) = \bigoplus_{j=1}^{3} p_j \left(\text{round}\left(\frac{x_{i,j}}{2r} \right) - q \right) \pmod{n_h}, \tag{6}$$

Algorithm 2: Parallel Spatial Hashmap Construction

foreach particle i **do**
  $\bar{x}_i \leftarrow$ `Round(`$\frac{x_i}{2r}$`)`;
  $h_i \leftarrow$ `SpatialHash(`$\bar{x}_i, n_h$`)`;
  $p \leftarrow$ `AddressOf(`$table[h]$`)`;
  $head \leftarrow table[h_i]$;
  while $head = table[h_i]$ **do**
    $head \leftarrow$ `AtomicCAS(`$p, head, i$`)`;
  end while
  $next[i] \leftarrow head$;
end foreach

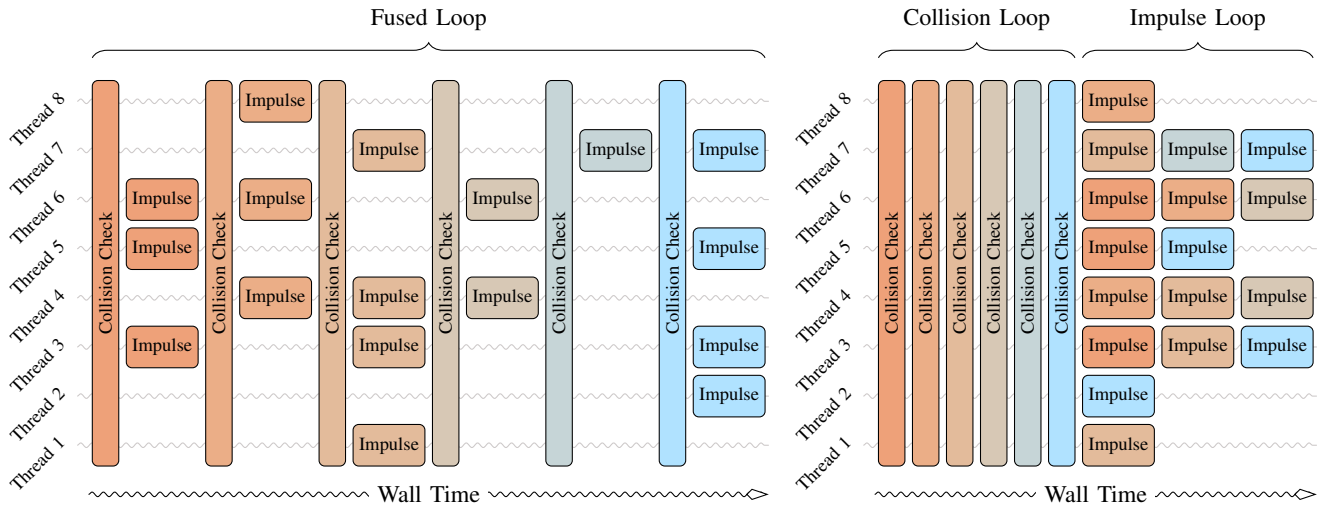

Fig. 5. Graphical illustration of the execution paths for warp divergence during collision detection during a single timestep of simulation. *Left*: Execution pattern of a single fused loop, where all threads in a warp must wait for even a single impulse to be computed. Performance of this approach is shown in Figure 6 as ONELOOP. *Right*: Execution pattern from precomputing all collision checks and storing a list of impulses to be processed and then processing them later. Note that this allows impulses from different collision checks to be computed simultaneously. Performance of this approach is shown in Figure 6 as TWOLOOPSFUSED and TWOLOOPSSPLIT.

where $x_{i,j}$ denotes the jth coordinate of the position $x_i \in \mathbb{R}3$, $p_1 = 73856093$, $p_2 = 19349663$, $p_3 = 83492791$, $q = 100$, n_h is the hashmap size, $\oplus$ is the `xor` bitwise logical operator, and round rounds a real number to the nearest integer.

We find that the size of the hash table has a significant effect on simulator performance and present results in Figure 3. As a heuristic summary of these findings, we propose a hash table size of twice the number of simulated particles.

We query the hash table in parallel with one thread per particle, traversing the linked list of particles with the same spatial hash as filtered candidates for collisions. Since particles can collide across cell boundaries, each thread must also traverse the $3^3 - 1 = 26$ neighboring cell lists. During the linked list traversal, we check particles for collision against the particle associated with the current thread. We discuss a method for accelerating narrowphase collision checking and contact processing on SIMT GPUs in II-D.

C. Collision Geometry using Signed Distance Functions

Our simulation engine models granular materials as particles that are uniform size and spherical, so narrowphase interparticle collision checking between particles i and j is as simple as checking $||x_i - x_j||_2^2 \leq (2r)^2$. However, many interesting scenarios for robotic tasks involve objects of arbitrary geometry, like excavator buckets, bulldozer blades, or robot wheels. We represent general rigid-body geometry using Signed Distance Functions (SDFs). A function $f_G : \mathbb{R}^3 \to \mathbb{R}$ is a signed distance function for a closed, simple surface G if $|f_G(x)|$ is the distance between x and the nearest point in G. More formally, given a distance function d, $|f_G(x)| = \inf\{d(x, y) \mid y \in G\}$. Additionally, the sign of f is positive if x is outside of G and negative if x is inside of G. Using a signed distance function, it is simple to check collisions with spheres and, thus, to perform particle-body collision checks. For a sphere of radius r centered at x, the maximum penetration depth into a geometry G is given by Equation (7),

$$\psi(x) = f_G(x) - r. \tag{7}$$

There are many SDFs which can be represented in simple closed forms for geometric primitives[24]. Of broader interest are geometries described by arbitrary triangle meshes which are commonly used in physical simulation. Computing a SDF for an arbitrary mesh can be quite expensive, depending on the resolution of the mesh surface. To mitigate this, we pre-compute the SDF values on a regular rectilinear three-dimensional grid which contains the bounds of the mesh geometry. To query the SDF, we use a trilinear interpolation scheme across the knot points in this grid. While this discretization naturally introduces approximation errors above and beyond the inherent discretization errors in triangle meshes, such errors are user-tunable through the grid resolution. Additionally, while the memory used by the grid scales $O(n^3)$ with the inverse of the grid resolution, an interpolated lookup in the grid is always $O(1)$, regardless of the resolution or of the complexity of the represented geometry. To check for collisions between a world-frame particle i at coordinates ${}_0x_i$ and a rigid body j at pose ${}_0X^j \in SE(3)$, we transform the particle coordinates into the local coordinates and evaluate Equation (7) as $\psi\left({}_jX^0 {}_0x\right)$, where ${}_jX^0 = \left({}_0X^j\right)^{-1}$.

D. Minimizing Warp Divergence

In CUDA programs, a *warp* is a collection of threads that execute the same instruction at the same time. Since each thread sees different data in the SIMT architecture, conditional instructions may cause two threads in the same warp to take different branches of control flow, a phenomenon called *warp divergence*. Since the program counter must remain the same

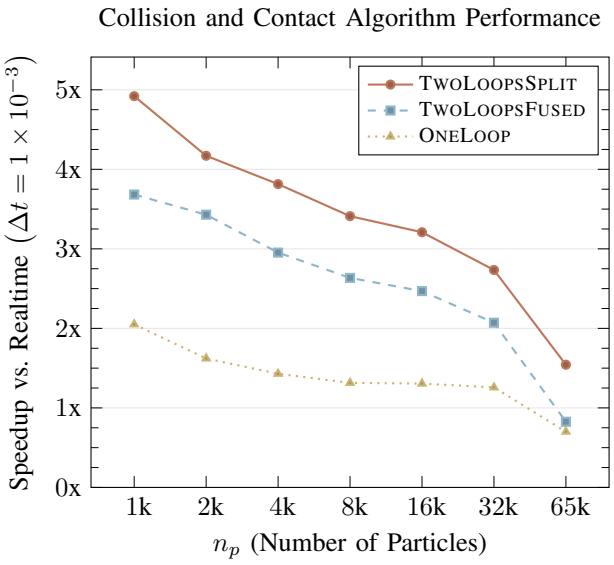

Fig. 6. Speedup vs. realtime (higher numbers are better) for different orderings for processing contacts and collisions. To maximize the number of collision candidates, we simulate particles at rest inside a tall cylindrical column. OneLoop shows a naive implementation, while TwoLoopsFused shows the performance of splitting collision and contact processing but keeping them in a single kernel. TwoLoopsSplit shows the performance of splitting collision and contact processing across kernel calls.

for each thread in the warp, CUDA environments will execute both branches of the conditional, one after the other, and will mask out the effects of instructions on inactive threads. In effect, warp divergence can substantially negatively impact program performance on the GPU, as threads may spend time "doing nothing" while waiting for other threads to finish their branches.

As suggested in Nakahara and Washizawa [21], we minimize warp divergence by deferring computation of contact impulses until after all collisions are detected and only marking particle pairs as "colliding" during an initial narrowphase collision detection step. Due to the nature of the SIMT architecture, splitting the collision checking and contact processing phases into two loops vastly shortens the number of instructions in the "colliding" code path during collision detection. Despite an efficient broadphase filter on nearest neighbor search using spatial hashing (Section II-B), we find experimentally that only about 16% of candidate collisions are actually in collision. Since it is significantly faster to store the indices of colliding pairs than it is to compute the contact response impulse (Section II-A), we can reduce the time spent waiting for threads assigned to non-colliding particles. A graphical representation of the difference in approaches is shown in Figure 5. This performance increase is confirmed experimentally, using a large pile of particles at rest in a tall cylindrical column designed to maximize the number of interparticle collisions that are physically feasible. We show the experimental difference in several approaches to collision-contact computation order in Figure 6. In summary, we find that splitting the collision detection and contact impulse computation across two subsequent CUDA kernel calls is the fastest approach for all tested numbers of particles.

III. Benchmark Environments

Using the simulation methodology described in Section II, we construct several exemplary simulation environments which showcase various aspects of our simulation engine, particularly as it pertains to robotics test and learning environments. We also fully describe the environment we use for performance benchmarks of our engine.

A. Bulldozing

In the Bulldozing task, shown in Figure 7, the the controlled robot is a tracked vehicle with a scoop rigidly attached to the front, modeled after a toy bulldozer STL file. We set up an OpenAI Gym[6] style environment, designed for a high-achieving agent to plan a series of actions to use the bulldozer to move material from one zone to another using a fixed time budget, which we briefly describe in this subsection.

Dynamics We use the simulation engine described in this paper, running on a machine with a 72-core Intel(R) Xeon(R) Gold 6154 CPU and four NVIDIA GeForce RTX 2080 Ti GPUs. We note that our engine currently does not meaningfully support multiple GPUs, besides running multiple independent simulations simultaneously. On this hardware, our simulation is able to run at realtime with tens of thousands of particles. However, to decrease the time required for training, we sacrifice physical accuracy for computation speed by increasing the simulation timestep to $0.01\,\mathrm{s}$. We note that this leads to increased numerical error during integration, leading to nonphysical behavior, particularly the collapse of tall stacks of particles, and a general behavior of steep slopes to "ooze" down to a more shallow configuration. However, we believe that the differences in particulate behavior are subtle and that these errors do not significantly affect the performance of training agents in this environment. For environments and tasks in which particle stacking and stable steep slopes are required, this tradeoff may, of course, not be acceptable.

Observations We compute two image observations of the environment, both on the GPU. The "ego camera" is a perspective depth camera that moves with the robot mesh and points out of the "front windshield" of the bulldozer. The "sky camera" is an orthographic third-person depth camera that faces down from the top of the scene and captures the heights of particle stacks across the entire environment. These images are computed on the same GPU used for simulation, minimizing expensive host-device memory transfers. Additionally, we provide the x and y positions of the bulldozer, as well as the yaw angle θ about the global z-axis.

Actions We use a track steering model parametrized by the box $[-1, 1] \times [-1, 1] \subset \mathbb{R}^2$. The first axis describes the linear velocity of the bulldozer along the direction it is facing, and the second axis describes the angular velocity around the yaw axis of the bulldozer.

Rewards The agent gets a reward of $\frac{100}{n}$, where n is the total number of simulated particles for each particle inside of a "goal box" inside the environment. To provide a smoother reward

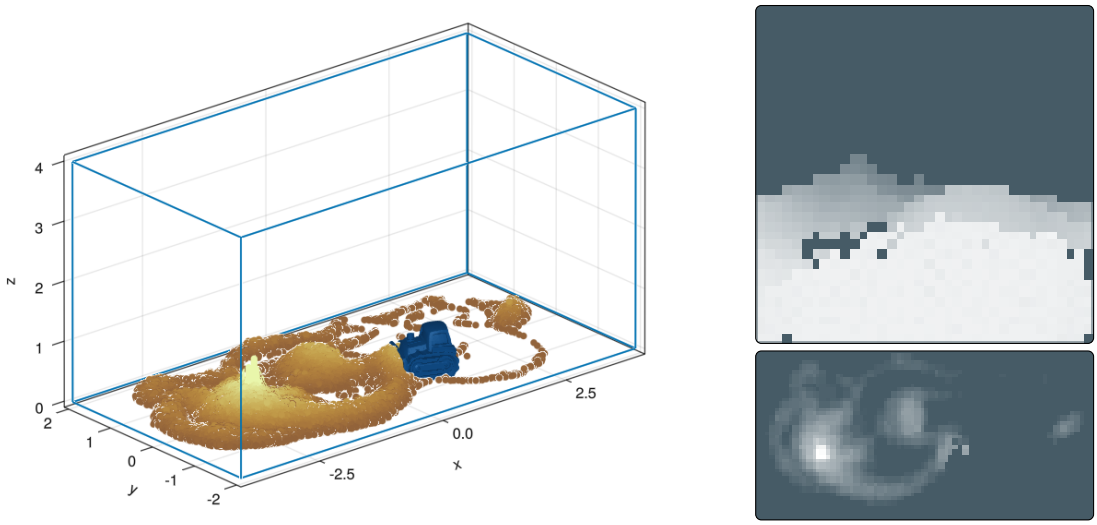

Fig. 7. On the left is an image taken from the bulldozer simulation environment with 50000 granular material particles simulated at 1kHz, running in real-time on an NVIDIA RTX 3080 Ti GPU. Visible here are particles interacting with the non-convex geometry of a toy bulldozer model, as well as mounding effects from stable interparticle contacts with dry coulomb friction. In the top right is displayed a 36-by-36 pixel "ego view" perspective camera showing a depth image from the cockpit of the bulldozer, while on the bottom right is a 72-by-36 pixel depth image taken from a "sky view" orthographic camera. Combined with the lateral position and orientation of the bulldozer, these images are used as the observation space for a reinforcement learning environment.

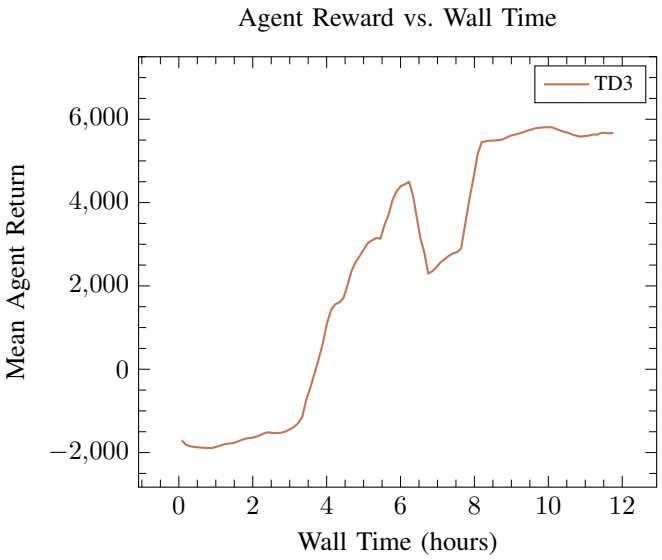

Fig. 8. A plot of agent total episode return (averaged over the 100 most recent episodes) against wall clock time, using the TD3 algorithm. The simulation environment, reward structure, and computer hardware used in this experiment are described in depth in Section III-A.

function, we assign a negative value to each particle outside the box of $-\frac{d}{n}$, where d is the distance from the particle to the closest point on the box.

We train this environment using the TD3 algorithm and show that an agent is able to learn to achieve higher rewards than an initial random policy. We do not present these results as an exemplary demonstration of robot learning, but instead to show that complex environments can be simulated fast enough to train Reinforcement Learning agents in days.

B. Helical Gear Tower

To test our performance and demonstrate our ability to scale to multiple rigid bodies of arbitrary, non-convex geometry, we simulate a tall, cylindrical containment tower with a varying number of involute helical gears rotating at uniformly random velocities, as shown in Figure 10. To keep particles flowing, we implement a cyclic boundary condition at the vertical extents of the cylinder, where particles passing through the bottom cap are transported to the top of the cylinder with the same velocity. However, we do not implement this cyclic boundary for nearest neighbor contact searches. We present results of how performance scales with the number of particles and the number of rigid bodies in Figure 9. While we do not propose a quantitative measurement of the "complexity" of a mesh geometry, we select these helical gears as a benchmark of interaction with arbitrary non-convex geometry, and note that due to our gridded SDF representation of collision geometries discussed in Section II-C, our performance does not change with the shape or complexity of the mesh.

C. Excavation

In the EXCAVATION environment, shown in Figure 1, the robotic manipulator is a 7-DoF Franka Emika Panda robot with a tool rigidly attached that is similar in shape to an excavator bucket. The robot accepts velocity control inputs in joint space. In our testing, we are able to interact with 50000 particles at realtime speeds on a single NVIDIA GeForce RTX 3080 Ti, using a simulation timestep of 5×10^{-4}.

We select the Franka Emika Panda robot for its flexibility and because it is becoming commonplace in the robotic research world. However, the Panda highlights an important limitation of our simulation, the fact that rigid bodies are kinematically driven and do not accumulate reaction forces

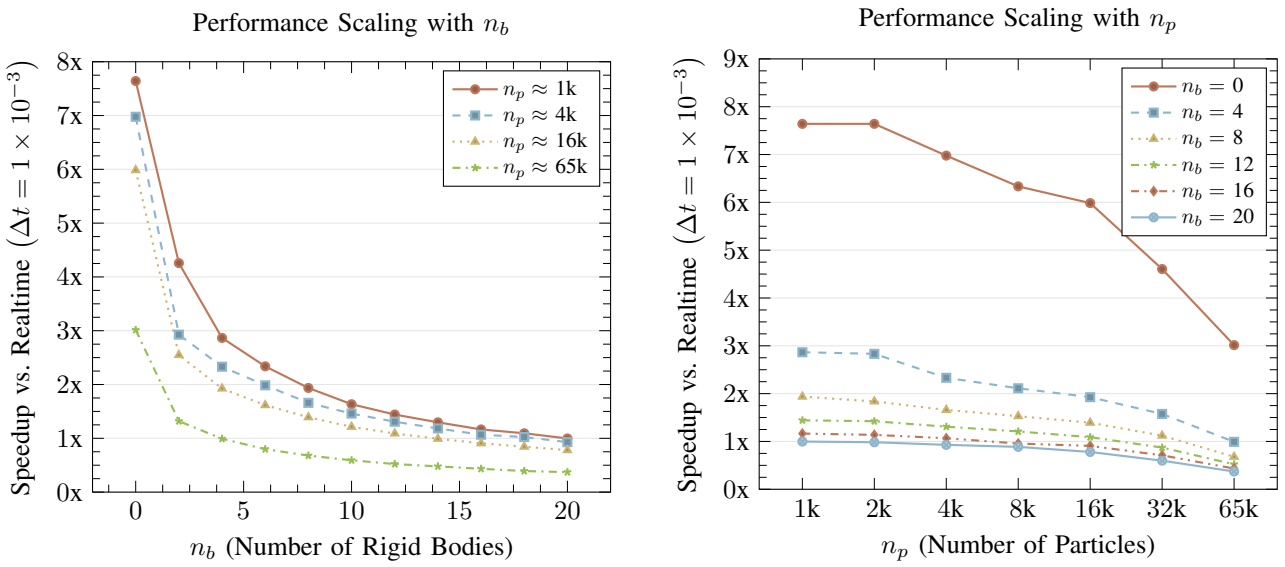

Fig. 9. Plots of speedup vs. realtime (higher numbers are better) against the number of rigid bodies in the scene (left) and the number of particles simulated (right, note log x axis). The benchmark environment simulated is described in Section III-B.

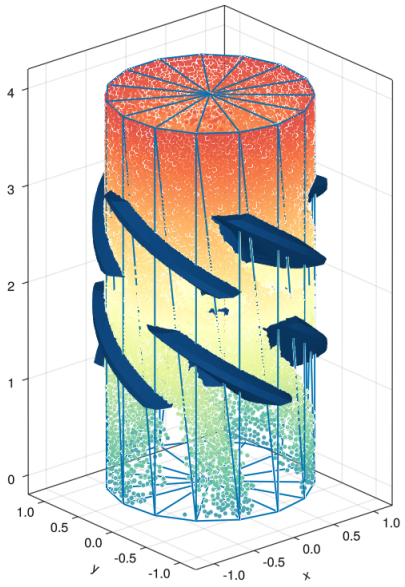

Fig. 10. Image of the helical gear benchmark setup that we use to test performance as it scales with the number of bodies and the number of particles. Particles are colored according to their z-axis coordinate.

from the environment. Particularly in granular manipulation tasks, there is a stark difference in the amount of force required to push a cutting edge through an aggregate (the *penetration force*), and the amount of force required to push a flat surface, like the back of a shovel, through an aggregate (the *deadload force*). Since the Panda robot is designed for safety around humans and in research applications, it has a relatively low maximum applicable force, leading to a wide gap in behavior between simulation and reality, which we intend to address in future improvements to our engine. Currently, users may manually compute the total force on an individual particle or rigid body, and use this in a higher-level decision to terminate an action or training episode, which we propose as a mitigation for this limitation.

IV. Related Works

Simulation of granular dynamics is of great interest for many tasks besides robotics, including Computer Aided Design and Engineering of earthmoving equipment [22], scientific sampling apparatus [25], and production lines [13]. Accordingly, there are several overarching approaches for the simulation of granular dynamics in existing literature, each with various design decisions that affect computational efficiency and accuracy. Broadly, these are Discrete Element Method (DEM), which represents a Lagrangian state of granular materials as a set of distinct bodies, Computational Fluid Dynamics, or Material Point Method (MPM) approaches, which use a hybrid Eulerian-Lagrangian state representation, and Reactive Force Theories, which are extremely efficient and use a fully data-driven approach, but only model the force on a tool exerted by soil. Our presented framework falls firmly into the Discrete Element Method (DEM) category.

The simulation of bulk material by the interactions of discrete particles with Lagrangian state is generally called the Discrete Element Method (DEM). One popular Discrete Element Method (DEM) framework for granular dynamics is Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [29], which offers a "Granular" package for rigid or elastic interparticle physical contact forces. A fork of LAMMPS, called LAMMPS Improved for General Granular and Granular Heat Transfer Simulations (LIGGGHTS) [18], also offers large-scale granular dynamics simulations with arbitrary particle geometries and sizes, as well as interactions with arbitrarily shaped rigid objects. While both LAMMPS

and LIGGGHTS are well-tested and proven simulation codes that offer significant flexibility to model multiple physical models of contact, they have been primarily designed for large clusters of CPUs communicating with networked message-passing interfaces, and are inherently limited in their ability to reach "interactive" rates of simulation for large numbers of particles.

Other solvers, such as Ansys Rocky [1], target granular interactions on GPUs, and are flexible, high quality, and high performance. However, these systems are closed-source simulation codes without published methods, which require expensive licenses, and prohibit user modification of the underlying software. Algoryx Momentum Granular [27] provides real-time simulation of granular materials, along with a two-way coupling with rigid multibody physics, by using an adaptive multiscale particle model, presumably executing on a CPU. While the authors publish their methodology, the implementation is also closed source and available only under a commercial license.

Project Chrono is a multiphysics engine supporting multibody rigid dynamics, computational fluid dynamics, and solid mechanics through finite element analysis. Project Chrono also offers Chrono::GPU [9], a GPU accelerated engine for granular flows. Chrono::GPU does not target realtime interaction, instead supporting elastic contact modes for very high accuracy simulations, which run significantly slower than realtime.

Wheel-soil interaction, also called terramechanics, is a key aspect of modeling wheeled robots traversing off-road terrains. This behavior is especially important during highly dynamic motions or when traversing loose or rocky soils. Previous approaches have found success by directly learning input-output maps of vehicle dynamics on such terrains from real data from trajectories from human experts and then used the learned model directly for Model-Predictive Path Integral control schemes to achieve "drifting" like behavior on dirt racetracks [30].

Additionally, there have been several efforts in robotics to control heavy earth-moving equipment like excavators. Early efforts used a physically based analytical model of soil computing onlly reactive forces of the earth on the bucket mechanism of the excavator [23]. Similarly, more recent efforts have achieved impressive results by using reinforcement learning to control excavator buckets in inhomogeneous soil densities by using a reduced analytical reactive force model, and employing heavy domain randomization to train a reinforcement learning policy for bucket control [7]. A key stated intention for the use of reduced, force-based models in these works is the computational intensity required for a full, Lagrangian-state particle simulation of soil mechanics. Reactive force models are limited in their ability to richly represent the state of the soil under manipulation, and are thus inapplicable to completely simulate tasks such as material transport.

Other approaches include the MPM, which models Lagrangian particle states but integrates their dynamics on Eulerian grids. MPM is an extremely flexible modeling approach for a wide variety of physical phenomena, but in general, is not efficient to run at interactive simulation speeds[14], despite high quality implementations available as part of the DiffTaichi[17] GPU framework.

There are many high quality interfaces for GPU programming in high level languages, including the aforementioned Taichi framework and the Warp[20] library from NVIDIA. We choose the Julia language due to its composable interfaces, high quality GPU libraries[4], and the native compilation of non-GPU code.

V. Conclusion and Future Work

We present a methodology for a simulation framework of hundreds of thousands of particles in dry frictional contact, which in bulk approximate granular media. Our framework is for execution on CUDA GPUs, and we describe methods for high performance on such architectures. While our approach does not model all types of materials, our engine favors speed while modeling a Lagrangian state, which is critical for robotic tasks that need to manipulate the state of soil. A performant implementation of the simulation architecture, which was used for the performance benchmarks in this paper is written in the Julia programming language and is released as open-source software[1]. This framework is able to handle one-way contact coupling with rheonomically constrained rigid bodies of arbitrary geometry, and allows a user-tunable tradeoff between geometry accuracy and CUDA memory usage, while maintaining constant compute time performance for rigid body collision detection.

Our model of granular interaction incorporates several simplifications and is not an appropriate model for all of the various types of granular interactions present in nature. Particularly in robotics, however, where fast simulation allows planning algorithms to predict dynamics in closed-loop control algorithms and enables fast environments for robot learning setups, we believe there is a role for a granular simulation engine that is focused on speed and which can represent arbitrary state configurations of particulate media.

To demonstrate the utility of our simulation framework for tasks in robotics, we show a collection of several benchmark robotic tasks involving the manipulation of granular materials, as well as showing that state of the art reinforcement learning algorithms can be brought to bear against these problems. An open-source implementation of these environments, along with an OpenAI Gym-like API, is available in the linked software repository.

We hope that these contributions spur increased research in the difficult research area of robotic manipulation of granular materials, and see several clear paths toward a more complete simulation framework, which we list in the following paragraphs and intend to address in future work.

Two-Way Coupling For tasks where the mass of the accumulated material is non-negligible compared to the actuation power of the manipulator, a full rigid-body or multiphysics dynamics engine, and a full two-way force coupling is required.

Differentiability Many robotic dynamic planning algorithms and system identification algorithms use Jacobians of the

[1]Link withheld for review anonymity.

dynamics to accelerate optimization. The Lagrangian particle state representation is not amenable to useful direct differentiation, due to its permutation invariance. A carefully chosen state representation could pave a path toward meaningful differentiability.

Non-Homogeneity While the presented framework allows for an easily replaceable interparticle contact model, each particle has the same parameters and contact equations. Since many real-world materials are non-homogenous, efficient computation for such media is an important contribution towards a fully flexible simulator.

Acknowledgments

This work is supported by the NASA Space Technology Research Fellowship, grant number 80NSSC19K1182. The authors thank Lorenzo Fluckiger, Trey Smith, Brian Coltin, Gautam Salhotra, K.R. Zenter, and Shashank Hegde for helpful discussions and feedback during the writing of this paper.

References

[1] Ansys Rocky — Particle Dynamics Simulation Software. https://www.ansys.com/products/fluids/ansys-rocky.

[2] Rika Antonova, Jingyun Yang, Priya Sundaresan, Dieter Fox, Fabio Ramos, and Jeannette Bohg. A bayesian treatment of real-to-sim for deformable object manipulation. *IEEE Robotics and Automation Letters*, 7(3):5819–5826, 2022.

[3] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1(1):1–16, June 1972. ISSN 0045-7825. doi: 10.1016/0045-7825(72)90018-7.

[4] Tim Besard, Christophe Foket, and Bjorn De Sutter. Effective extensible programming: Unleashing Julia on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 2018. ISSN 1045-9219. doi: 10.1109/TPDS.2018.2872064.

[5] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[7] Pascal Egli, Dominique Gaschen, Simon Kerscher, Dominic Jud, and Marco Hutter. Soil-Adaptive Excavation Using Reinforcement Learning. *IEEE Robotics and Automation Letters*, 7(4):9778–9785, October 2022. ISSN 2377-3766. doi: 10.1109/LRA.2022.3189834.

[8] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4397–4404, May 2015. doi: 10.1109/ICRA.2015.7139807.

[9] 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):1813, October 2021. ISSN 2227-9717. doi: 10.3390/pr9101813.

[10] Kenneth A. Farley, Kenneth H. Williford, Kathryn M. Stack, Rohit Bhartia, Al Chen, Manuel de la Torre, Kevin Hand, Yulia Goreva, Christopher D. K. Herd, Ricardo Hueso, Yang Liu, Justin N. Maki, German Martinez, Robert C. Moeller, Adam Nelessen, Claire E. Newman, Daniel Nunes, Adrian Ponce, Nicole Spanovich, Peter A. Willis, Luther W. Beegle, James F. Bell, Adrian J. Brown, Svein-Erik Hamran, Joel A. Hurowitz, Sylvestre Maurice, David A. Paige, Jose A. Rodriguez-Manfredi, Mitch Schulte, and Roger C. Wiens. Mars 2020 Mission Overview. *Space Science Reviews*, 216(8):142, December 2020. ISSN 1572-9672. doi: 10.1007/s11214-020-00762-y.

[11] Isabel M. Rayas Fernández, Christopher E. Denniston, David A. Caron, and Gaurav S. Sukhatme. Informative path planning to estimate quantiles for environmental analysis. *IEEE Robotics and Automation Letters*, 7(4):10280–10287, 2022.

[12] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. Deep Whole-Body Control: Learning a Unified Policy for Manipulation and Locomotion. In *Conference on Robot Learning (CoRL)*, 2022.

[13] S. Geer, M. L. Bernhardt-Barry, E. J. Garboczi, J. Whiting, and A. Donmez. A more efficient method for calibrating discrete element method parameters for simulations of metallic powder used in additive manufacturing. *Granular Matter*, 20(4):77, October 2018. ISSN 1434-7636. doi: 10.1007/s10035-018-0848-4.

[14] Amin Haeri, Dominique Tremblay, Krzysztof Skonieczny, Daniel Holz, and Marek Teichmann. Efficient Numerical Methods for Accurate Modeling of Soil Cutting Operations. In *37th International Symposium on Automation and Robotics in Construction*, Kitakyushu, Japan, October 2020. doi: 10.22260/ISARC2020/0085.

[15] Eric Heiden, Miles Macklin, Yashraj S Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. DiSECt: A differentiable simulation engine for autonomous robotic cutting. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021. doi: 10.15607/RSS.2021.XVII.067.

[16] Peter C. Horak and Jeff C. Trinkle. On the Similarities and Differences Among Contact Models in Robot Simulation. *IEEE Robotics and Automation Letters*, 4(2):493–499, April 2019. ISSN 2377-3766, 2377-3774. doi: 10.1109/LRA.2019.2891085.

[17] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[18] Christoph Kloss, Christoph Goniva, Alice Hager, Stefan Amberger, and Stefan Pirker. Models, algorithms and validation for opensource DEM and CFD–DEM. *Progress in Computational Fluid Dynamics, an International Journal*, 12(2-3):140–152, January 2012. ISSN 1468-

4349. doi: 10.1504/PCFD.2012.047457.

[19] Vincent Lim, Huang Huang, Lawrence Yunliang Chen, Jonathan Wang, Jeffrey Ichnowski, Daniel Seita, Michael Laskey, and Ken Goldberg. Real2Sim2Real: Self-Supervised Learning of Physical Single-Step Dynamic Actions for Planar Robot Casting. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8282–8289, Philadelphia, PA, USA, May 2022. IEEE Press. doi: 10.1109/ICRA46639.2022.9811651.

[20] Miles Macklin. Warp: A High-performance Python Framework for GPU Simulation and Graphics, March 2022.

[21] Yasuhiro Nakahara and Teruyoshi Washizawa. Accelerating DEM simulations on GPUs by reducing the impact of warp divergences, 2015.

[22] Erfan G. Nezami, Youssef M. A. Hashash, Dawei Zhao, and Jamshid Ghaboussi. Simulation of front end loader bucket–soil interaction using discrete element method. *International Journal for Numerical and Analytical Methods in Geomechanics*, 31(9):1147–1162, 2007. ISSN 1096-9853. doi: 10.1002/nag.594.

[23] Borinara Park. Development of a Virtual Reality Excavator Simulator: A Mathematical Model of Excavator Digging and a Calculation Methodology. October 2002.

[24] Inigo Quilez. Inigo Quilez. https://iquilezles.org.

[25] Dario Riccobono, Scott Moreland, Paul Backes, and Giancarlo Genta. Granular Flow Characterization during Sampling Operation for Enceladus Surface Acquisition. pages 564–576, April 2021. doi: 10.1061/9780784483374.053.

[26] Gautam Salhotra, I.-Chun Arthur Liu, Marcus Dominguez-Kuhne, and Gaurav S. Sukhatme. Learning deformable object manipulation from expert demonstrations. *IEEE Robotics and Automation Letters*, 7(4):8775–8782, 2022.

[27] Martin Servin, Tomas Berglund, and Samuel Nystedt. A multiscale model of terrain dynamics for real-time earthmoving simulation. *Advanced Modeling and Simulation in Engineering Sciences*, 8(1):11, December 2021. ISSN 2213-7467. doi: 10.1186/s40323-021-00196-3.

[28] Matthias Teschner, Bruno Heidelberger, Matthias Muller, Danat Pomeranets, and Markus Gross. Optimized Spatial Hashing for Collision Detection of Deformable Objects.

[29] Aidan P. Thompson, H. Metin Aktulga, Richard Berger, Dan S. Bolintineanu, W. Michael Brown, Paul S. Crozier, Pieter J. in 't Veld, Axel Kohlmeyer, Stan G. Moore, Trung Dac Nguyen, Ray Shan, Mark J. Stevens, Julien Tranchida, Christian Trott, and Steven J. Plimpton. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*, 271:108171, February 2022. ISSN 0010-4655. doi: 10.1016/j.cpc.2021.108171.

[30] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, May 2016. doi: 10.1109/ICRA.2016.7487277.

[31] Takashi Yamamoto, Koji Terada, Akiyoshi Ochiai, Fuminori Saito, Yoshiaki Asahara, and Kazuto Murase. Development of Human Support Robot as the research platform of a domestic mobile manipulator. *ROBOMECH Journal*, 6(1):4, April 2019. ISSN 2197-4225. doi: 10.1186/s40648-019-0132-3.