Investigating the Mobility of Light Autonomous Tracked Vehicles Using a High Performance Computing Simulation Capability

Dan Negrut[*], Hammad Mazhar[*], Daniel Melanz[*†], David Lamb[†], Paramsothy Jayakumar[†], Michael Letherwood[†], Abhinandan Jain[‡], Marco Quadrelli[‡]

[*] Department of Mechanical Engineering
University of Wisconsin - Madison
Madison, WI 53706

[†] U.S. Army Tank Automotive Research, Development and Engineering Center
Warren, MI 48397

[‡] Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

ABSTRACT

This paper is concerned with the physics-based simulation of light tracked vehicles operating on rough deformable terrain. The focus is on small autonomous vehicles, which weigh less than 100 lb and move on deformable and rough terrain that is feature rich and no longer representable using a continuum approach. A scenario of interest is, for instance, the simulation of a reconnaissance mission for a high mobility lightweight robot where objects such as a boulder or a ditch that could otherwise be considered small for a truck or tank, become major obstacles that can impede the mobility of the light autonomous vehicle and negatively impact the success of its mission. Analyzing and gauging the mobility and performance of these light vehicles is accomplished through a modeling and simulation capability called Chrono::Engine. Chrono::Engine relies on parallel execution on Graphics Processing Unit (GPU) cards.

1 INTRODUCTION

Engineers are increasingly relying on simulation to augment and, in some cases, replace costly and time consuming experimental work. However, current simulation capabilities are sometimes inadequate to capture phenomena of interest. In tracked vehicle analysis, for example, the interaction of the track with granular terrain has been difficult to characterize through simulation due to the prohibitively long simulation times associated with *many-body dynamics* problems. This is the generic name used here to characterize dynamic systems with a large number of bodies encountered, for instance, when one adopts a discrete representation of the terrain in vehicle dynamics problems. However, these many-body dynamics problems can now capitalize on recent advances in the microprocessor industry that are a consequence of Moore's law, of doubling

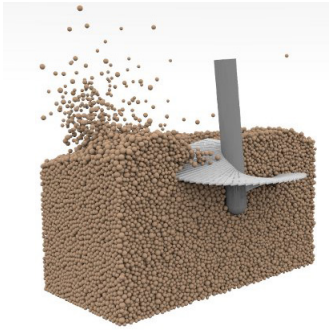
the number of transistors per unit area roughly every 18 months. Specifically, until recently, access to massive computational power on parallel supercomputers has been the privilege of a relatively small number of research groups in a select number of research facilities, thus limiting the scope and impact of high performance computing (HPC). This scenario is rapidly changing due to a trend set by general-purpose computing on graphics processing unit (GPU) cards. NVIDIA's CUDA library [21] allows one to use the streaming multiprocessors available in high-end graphics cards. In this setup, a latest generation NVIDIA GPU Kepler card will reach 1.5 Teraflops by the end of 2012 owing to a set of 1536 scalar processors working in parallel, each following a Single Instruction Multiple Data (SIMD) execution paradigm. Despite having only 1536 scalar processors, such a card is capable of managing tens of thousands of parallel threads at any given time. This overcommitting of the GPU hardware resources is at the cornerstone of a computing paradigm that aggressively attempts to hide costly memory transactions with useful computation, a strategy that has lead, in frictional contact dynamics simulation, to a one order of magnitude reduction in simulation time for many-body systems [34, 20].

The challenge of using parallel computing to reduce simulation time and/or increase system size stems, for the most part, from the task of designing and implementing many-body dynamics specific parallel numerical methods. Designing parallel algorithms suitable for frictional contact many-body dynamics simulation remains an area of active research. Results reported in [16] indicate that the most widely used commercial software package for multibody dynamics simulation, which draws on a so called penalty or regularization approach, runs into significant difficulties when handling simple problems involving hundreds of contact events, and thus cases with thousands of contacts become intractable. Unlike these penalty or regularization approaches where the frictional interaction is represented by a collection of stiff springs combined with damping elements that act at the interface of the two bodies [11, 28, 29, 22], the approach embraced herein draws on a different mathematical framework. Specifically, the parallel algorithms rely on time-stepping procedures producing weak solutions of the differential variational inequality (DVI) problem that describes the time evolution of rigid bodies with impact, contact, friction, and bilateral constraints. When compared to penalty-methods, the DVI approach has a greater algorithmic complexity, but avoids the small time steps that plague the former approach.

The task of presenting this class of algorithms and their parallel implementation is organized as follows. Section 2 provides a brief description of the general equations that capture the dynamics of many-body systems. This section also contains an outline of the parallel method embraced to numerically solve the equations of motion. One of the challenging components of the solution method is the collision detection step required to determine the set of contacts active in the many-body system. These contacts, crucial in producing the frictional contact forces at work in the system, are determined in parallel using an approach outlined in Section 3. A scalable rendering pipeline that can leverage thousands of CPU cores for visualization purposes is discussed in Section 4. The engineer-

ing application used to demonstrate this parallel simulation capability is that of a light tracked vehicle that operates on granular terrain and negotiates an obstacle course. To further illustrate the versatility of the simulation capability, the vehicle is assumed to be equipped with a drilling device used to penetrate the terrain. Both the vehicle dynamics and the drilling process are seamlessly analyzed within the same HPC-enabled simulation capability. A schematic of the vehicle is provided in Fig. 1(a). A cut-away image of the drilling tool is shown in isolation in Fig. 1(b).

(a) Light autonomous vehicle negotiating a pile of rubble.



(b) Cutaway view: Anchor penetrating granular material [18].

Figure 1: Two examples in which hundreds of thousands of bodies interact mutually through contact and friction.

2 THE MANY-BODY DYNAMICS PROBLEM

2.1 General Considerations

The modeling approach adopted in order to abstract and represent the dynamics of the vehicle/terrain interaction is based on a differential variational inequality (DVI) methodology. Compared to penalty or regularization approaches

[11, 28, 29, 22], it allows for larger integration step sizes. The formulation of the equations of motion, that is, the equations that govern the time evolution of a multibody system, is based on the so-called absolute, or Cartesian, representation of the position and attitude of each rigid body in the system. The state of the system is denoted by the generalized positions $\mathbf{q} = \left[\mathbf{r}_1^T, \epsilon_1^T, \ldots, \mathbf{r}_{n_b}^T, \epsilon_{n_b}^T\right]^T \in \mathbb{R}^{7n_b}$ and their time derivatives $\dot{\mathbf{q}} = \left[\dot{\mathbf{r}}_1^T, \dot{\epsilon}_1^T, \ldots, \dot{\mathbf{r}}_{n_b}^T, \dot{\epsilon}_{n_b}^T\right]^T \in \mathbb{R}^{7n_b}$, where n_b is the number of bodies, $\mathbf{r}_j$ is the absolute position of the center of mass of the jth body, and the quaternions (Euler parameters) ϵ_j are used to represent rotation and to avoid singularities. Instead of using quaternion derivatives in $\dot{\mathbf{q}}$, it is more advantageous to work with angular velocities expressed in the local (body-attached) reference frames; in other words, the method described will use the vector of generalized velocities $\mathbf{v} = \left[\dot{\mathbf{r}}_1^T, \bar{\omega}_1^T, \ldots, \dot{\mathbf{r}}_{n_b}^T, \bar{\omega}_{n_b}^T\right]^T \in \mathbb{R}^{6n_b}$. Note that the generalized velocity can be easily obtained as $\dot{\mathbf{q}} = \mathbf{L}(\mathbf{q})\mathbf{v}$, where $\mathbf{L}$ is a linear mapping that transforms each $\bar{\omega}_i$ into the corresponding quaternion derivative $\dot{\epsilon}_i$ by means of the linear algebra formula $\dot{\epsilon}_i = \frac{1}{2}\mathbf{G}^T(\mathbf{q})\bar{\omega}_i$, with 3x4 matrix $\mathbf{G}(\mathbf{q})$ as defined in [12].

2.1.1 Bilateral Constraints

Bilateral constraints represent kinematic relationships between two rigid bodies in the system. For example, spherical joints, prismatic joints, or revolute joints can be expressed as holonomic algebraic equations constraining the relative positions of two bodies. A set B of constraints leads to a collection of scalar equations

$$\Psi_i(\mathbf{q}, t) = 0, \quad i \in \mathcal{B}\,, \tag{1}$$

the number of which depends on the type of constraints in set B. Bilateral constraints must also be satisfied at the velocity level,

$$\frac{d\Psi_i(\mathbf{q}, t)}{dt} = 0 \Rightarrow \quad \frac{\partial \Psi_i}{\partial \mathbf{q}}\dot{\mathbf{q}} + \frac{\partial \Psi_i}{\partial t} = \nabla_q \Psi_i^T \dot{\mathbf{q}} + \frac{\partial \Psi_i}{\partial t} = \nabla_q \Psi_i^T \mathbf{L}(\mathbf{q})\mathbf{v} + \frac{\partial \Psi_i}{\partial t} = 0, \tag{2}$$

which is obtained by taking one time derivative of Equation 1.

2.1.2 Unilateral Constraints and Friction

Unilateral constraints enforce contact constraints between rigid bodies in the system. It is assumed that a gap function, $\Phi(\mathbf{q})$, can be defined for each pair of near-enough bodies. This gap function describes the distance between the closest points on the two bodies of interest.

Unilateral contact constraints also introduce friction forces into the system. When a contact is active, or $\Phi_i(\mathbf{q}) = 0$, a normal force acts on each of the two bodies at the contact point. When a contact is inactive, or $\Phi_i(\mathbf{q}) > 0$, no normal force exists. This represents a complementarity condition. Consider two bodies A and B in contact as shown in Fig. 2. Let $\mathbf{n}_i$ be the normal at the contact pointing toward the exterior of the body of lower index, which by convention is considered to be body A. Let $\mathbf{u}_i$ and $\mathbf{w}_i$ be two vectors in the

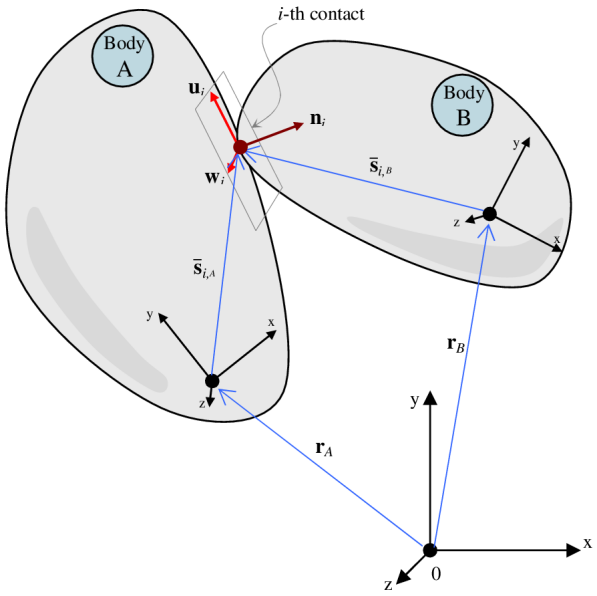


Figure 2: Contact i between two bodies $A, B \in \{1, 2, \ldots, n_b\}$.

contact plane such that $\mathbf{n}_i, \mathbf{u}_i, \mathbf{w}_i \in \mathbb{R}^3$ are mutually orthonormal vectors. The frictional contact forces are defined by the multipliers $\widehat{\gamma}_{i,n} \geq 0$, $\widehat{\gamma}_{i,u}$, and $\widehat{\gamma}_{i,w}$, which lead to the normal component of the friction force, $\mathbf{F}_{i,N} = \widehat{\gamma}_{i,n}\mathbf{n}_i$ and the tangential component of the force $\mathbf{F}_{i,T} = \widehat{\gamma}_{i,u}\mathbf{u}_i + \widehat{\gamma}_{i,w}\mathbf{w}_i$.

The Coulomb friction model, which draws for contact i on the friction coefficient μ_i, is used to write the following constraints:

$$\widehat{\gamma}_{i,n} \geq 0, \quad \Phi_i(\mathbf{q}) \geq 0, \quad \Phi_i(\mathbf{q})\widehat{\gamma}_{i,n} = 0, \tag{3}$$

$$\mu_i\widehat{\gamma}_{i,n} \geq \sqrt{\widehat{\gamma}_{i,u}^2 + \widehat{\gamma}_{i,w}^2} \quad , \quad ||\mathbf{v}_{i,T}|| \left(\mu_i\widehat{\gamma}_{i,n} - \sqrt{\widehat{\gamma}_{i,u}^2 + \widehat{\gamma}_{i,w}^2}\right) = 0, \tag{4}$$

$$\langle \mathbf{F}_{i,T}, \mathbf{v}_{i,T} \rangle = -||\mathbf{F}_{i,T}|| \, ||\mathbf{v}_{i,T}|| \tag{5}$$

Equation 3 captures the complementarity condition previously described. Equations 4 and 5 relate the magnitude and direction of the friction force to the multipliers and tangential velocity of the contact. These remaining equations can be expressed in an equivalent manner using the maximum dissipation principle. This frames the Coulomb friction model as a minimization problem, which can be seen in Equation 6.

$$(\widehat{\gamma}_{i,u}, \widehat{\gamma}_{i,w}) = \underset{\sqrt{\widehat{\gamma}_{i,u}^2 + \widehat{\gamma}_{i,w}^2} \leq \mu_i\widehat{\gamma}_{i,n}}{\operatorname{argmin}} \mathbf{v}_{i,T}^T \left(\widehat{\gamma}_{i,u}\mathbf{u}_i + \widehat{\gamma}_{i,w}\mathbf{w}_i\right). \tag{6}$$

The nature of the friction cone can be seen if yet another form of the friction force equations is considered. The friction force of the i-th contact can be expressed

as follows, where Υ is a cone in three dimensions whose slope is $\tan^{-1}\mu_i$.

$$\mathbf{F}_i = \mathbf{F}_{i,N} + \mathbf{F}_{i,T} = \widehat{\gamma}_{i,n}\mathbf{n}_i + \widehat{\gamma}_{i,u}\mathbf{u}_i + \widehat{\gamma}_{i,w}\mathbf{w}_i \ \in \ \Upsilon, \tag{7}$$

2.1.3 Equations of Motion of Systems with Frictional Contact

The time evolution of the dynamical system is governed by the following differential variational inequality [7]:

$$\begin{array}{rcl}
\dot{\mathbf{q}} & = & \mathbf{L}(\mathbf{q})\mathbf{v} \\
\mathbf{M}\dot{\mathbf{v}} & = & \mathbf{f}\,(t,\mathbf{q},\mathbf{v}) + \sum\limits_{i\in\mathcal{B}} \widehat{\gamma}_{i,b}\nabla\Psi_i + \\
 & & + \sum\limits_{i\in\mathcal{A}} \left(\widehat{\gamma}_{i,n}\,\mathbf{D}_{i,n} + \widehat{\gamma}_{i,u}\,\mathbf{D}_{i,u} + \widehat{\gamma}_{i,w}\,\mathbf{D}_{i,w}\right) \\
i \in \mathcal{B} & : & \Psi_i(\mathbf{q},t) = 0 \\
i \in \mathcal{A} & : & \widehat{\gamma}_{i,n} \geq 0 \ \perp \ \Phi_i(\mathbf{q}) \geq 0, \qquad \text{and} \\
(\widehat{\gamma}_{i,u},\widehat{\gamma}_{i,w}) & = & \underset{\mu_i\widehat{\gamma}_{i,n}\geq\sqrt{\widehat{\gamma}_{i,u}^2+\widehat{\gamma}_{i,w}^2}}{\operatorname{argmin}} \mathbf{v}^T\left(\widehat{\gamma}_{i,u}\,\mathbf{D}_{i,u} + \widehat{\gamma}_{i,w}\,\mathbf{D}_{i,w}\right).
\end{array} \tag{8}$$

The tangent space generators $\mathbf{D}_i = [\,\mathbf{D}_{i,n},\,\mathbf{D}_{i,u},\,\mathbf{D}_{i,w}] \in \mathbb{R}^{6n_b\times 3}$ are sparse and are defined given a pair of contacting bodies A and B as

$$\mathbf{D}_i^T = \begin{bmatrix} \mathbf{0} & \ldots & -\mathbf{A}_{i,p}^T & \mathbf{A}_{i,p}^T\mathbf{A}_A\tilde{\mathbf{s}}_{i,A} & \mathbf{0} & \ldots \\ \mathbf{0} & \ldots & \mathbf{A}_{i,p}^T & -\mathbf{A}_{i,p}^T\mathbf{A}_B\tilde{\mathbf{s}}_{i,B} & \mathbf{0} & \ldots \end{bmatrix}, \tag{9}$$

where $\mathbf{A}_A$ is the orientation matrix associated with body A, $\mathbf{A}_{i,p} = [\mathbf{n}_i, \mathbf{u}_i, \mathbf{w}_i]$ is the $\mathbb{R}^{3\times 3}$ matrix of the local coordinates of the ith contact, the vectors $\bar{\mathbf{s}}_{i,A}$ and $\bar{\mathbf{s}}_{i,B}$ are the contact point positions in body coordinates (see Fig. 2). A tilde $\tilde{\mathbf{x}}$ over a vector $\mathbf{x} \in \mathbb{R}^3$ represents the skew symmetric matrix associated with the outer product of two vectors [12].

2.2 Discretization Scheme for Numerical Solution

Note that Eqs. 3 and 4 express complementarity conditions between the normal force and gap function, and associated with the stick-slip transition, respectively. The presence of these complementarity conditions is the trademark of a DVI formulation, whose numerical solution in the context of rigid body dynamics can be traced back to [19, 15, 17]. The DVI formulations have been classified by differential index in [23] and recent time-stepping schemes have included both acceleration-force linear complementarity problem (LCP) approaches [8, 24, 35] and velocity-impulse LCP-based time-stepping methods [31, 5, 6, 30]. The LCPs, obtained as a result of the introduction of inequalities in time-stepping schemes for DVI, coupled with a polyhedral approximation of the friction cone must be solved at each time step in order to determine the system state configuration as well as the Lagrange multipliers representing the reaction forces [15, 31]. If the simulation entails a large number of contacts and rigid bodies, as in the case of part feeders, packaging machines, and granular flows, the computational burden of classical LCP solvers can become significant. Indeed,

a well-known class of numerical methods for LCPs based on *simplex methods*, also known as *direct* or *pivoting* methods [10], may exhibit exponential worst-case complexity [9]. They may be impractical even for problems involving as few as several hundred bodies when friction is present [4, 33]. Moreover, the three-dimensional Coulomb friction case leads to a nonlinear complementarity problem (NCP): the use of a polyhedral approximation to transform the NCP into an LCP introduces artificial anisotropy in friction cones [31, 35, 5]. This discrete and finite approximation of friction cones is one of the reasons for the large dimension of the problem that needs to be solved in multibody dynamics with frictional contact.

In order to circumvent the limitations imposed by the use of classical LCP solvers and the limited accuracy associated with polyhedral approximations of the friction cone, a parallel fixed-point iteration method with projection on a convex set has been proposed, developed, and tested in [7]. The method is based on a time-stepping formulation that solves at every step a cone constrained optimization problem [2]. The time-stepping scheme, proved to converge in a measure differential inclusion sense to the solution of the original continuous-time DVI, sets off at time t_l by assuming that a set of contacts, $\mathcal{A}$, exists between bodies in the system, and a set of bilateral constraints, $\mathcal{B}$, is also active. The governing differential equations then assume the form of a DVI problem. The equation of motion is discretized so that an approximation to the solution can be found at discrete instants in time. Given a position $\mathbf{q}^{(l)}$ and velocity $\mathbf{v}^{(l)}$ at the time step $t^{(l)}$, the numerical solution is found at the new time step $t^{(l+1)} = t^{(l)} + h$ by solving the following optimization problem with equilibrium constraints [32]:

$$\begin{aligned}
\mathbf{M}(\mathbf{v}^{(l+1)} &- \mathbf{v}^{(l)}) = h\mathbf{f}(t^{(l)}, \mathbf{q}^{(l)}, \mathbf{v}^{(l)}) + \sum_{i\in\mathcal{B}} \gamma_{i,b}\nabla\Psi_i + \\
&+ \textstyle\sum_{i\in\mathcal{A}} \left(\gamma_{i,n}\,\mathbf{D}_{i,n} + \gamma_{i,u}\,\mathbf{D}_{i,u} + \gamma_{i,w}\,\mathbf{D}_{i,w}\right), \qquad (10)
\end{aligned}$$

$$i \in \mathcal{B}: \quad \tfrac{1}{h}\Psi_i(\mathbf{q}^{(l)}, t) + \nabla\Psi_i^T\mathbf{v}^{(l+1)} + \tfrac{\partial\Psi_i}{\partial t} = 0 \qquad (11)$$

$$i \in \mathcal{A}: \quad 0 \le \tfrac{1}{h}\Phi_i(\mathbf{q}^{(l)}) + \mathbf{D}_{i,n}^T\mathbf{v}^{(l+1)} \perp \gamma_n^i \ge 0, \qquad (12)$$

$$(\gamma_{i,u}, \gamma_{i,w}) = \underset{\mu_i\gamma_{i,n}\ge\sqrt{\gamma_{i,u}^2+\gamma_{i,w}^2}}{\operatorname{argmin}} \mathbf{v}^{(l+1),T}\left(\gamma_{i,u}\,\mathbf{D}_{i,u} + \gamma_{i,w}\,\mathbf{D}_{i,w}\right) \qquad (13)$$

$$\mathbf{q}^{(l+1)} = \mathbf{q}^{(l)} + h\mathbf{L}(\mathbf{q}^{(l)})\mathbf{v}^{(l+1)}. \qquad (14)$$

Here, γ_s represents the constraint impulse of a contact constraint; that is, $\gamma_s = h\widehat{\gamma}_s$, for $s = n, u, w$. The $\frac{1}{h}\Phi_i(\mathbf{q}^{(l)})$ term achieves constraint stabilization; its effect is discussed in [3]. Similarly, the term $\frac{1}{h}\Psi_i(\mathbf{q}^{(l)})$ achieves stabilization for bilateral constraints. The scheme converges to the solution of a measure differential inclusion [2] when the step size $h \to 0$.

The proposed approach casts the problem as a monotone optimization problem through a relaxation over the complementarity constraints, replacing Eq. (12) with

$$i \in \mathcal{A} : 0 \le \frac{1}{h}\Phi_i(\mathbf{q}^{(l)}) + \mathbf{D}_{i,n}^T\mathbf{v}^{(l+1)} - \mu_i\sqrt{(\mathbf{v}^T\,\mathbf{D}_{i,u})^2 + (\mathbf{v}^T\,\mathbf{D}_{i,w})^2} \perp \gamma_n^i \ge 0.$$

The solution of the modified time-stepping scheme will approach the solution of the same measure differential inclusion for $h \to 0$ as the original scheme [2], yet, in some situations, for large h, μ, or relative velocity $\mathbf{v}^{(l+1)}$, i.e., when not in an asymptotic regime, this relaxation can introduce motion oscillations. It was shown in [7] that the modified scheme is a cone complementarity problem (CCP), which can be solved efficiently by an iterative numerical method that relies on projected contractive maps. Omitting for brevity some of the details discussed in [7, 34], we note that the algorithm makes use of the following vectors:

$$\tilde{\mathbf{k}} \equiv \mathbf{M}\mathbf{v}^{(l)} + h\mathbf{f}(t^{(l)}, \mathbf{q}^{(l)}, \mathbf{v}^{(l)}) \tag{15}$$

$$\mathbf{b}_i \equiv \left\{ \tfrac{1}{h}\Phi_i(\mathbf{q}^{(l)}), 0, 0 \right\}^T \quad i \in \mathcal{A}, \tag{16}$$

$$b_i \equiv \tfrac{1}{h}\Psi_i(\mathbf{q}^{(l)}, t) + \tfrac{\partial \Psi_i}{\partial t}, \quad i \in \mathcal{B}. \tag{17}$$

The solution, in terms of dual variables of the CCP (the multipliers), is obtained by iterating the following contraction maps until convergence, where Π_{Υ_i} represents the orthogonal projection on the friction cone associated with contact i [32]:

$$\forall i \in \mathcal{A}: \quad \gamma_i^{r+1} = \Pi_{\Upsilon_i}\left[\gamma_i^r - \omega\eta_i\left(D_i^T\mathbf{v}^r + \mathbf{b}_i\right)\right] \tag{18}$$

$$\forall i \in \mathcal{B}: \quad \gamma_i^{r+1} = \Pi_{\Upsilon_i}\left[\gamma_i^r - \omega\eta_i\left(\nabla\Psi_i^T\mathbf{v}^r + b_i\right)\right]. \tag{19}$$

At each iteration r, before repeating (18) and (19), also the primal variables (the velocities) are updated as

$$\mathbf{v}^{r+1} = \mathbf{M}^{-1}\left(\sum_{z\in\mathcal{A}} D_z\gamma_z^{r+1} + \sum_{z\in\mathcal{B}} \nabla\Psi_z\gamma_z^{r+1} + \tilde{\mathbf{k}}\right). \tag{20}$$

2.3 Parallel Implementation

The dynamics of a large multibody system whose bodies interact through contact, friction, and bilateral constraints can be simulated in time via the CCP algorithm previously described. A sequential implementation of this algorithm is described by the following pseudo-code:

Algorithm 1: Inner Iteration Loop

1. For $i \in \mathcal{A}(\mathbf{q}, \delta)$, evaluate $\eta_i = 3/\text{Trace}(\,\mathbf{D}_i^T\mathbf{M}^{-1}\,\mathbf{D}_i)$.
2. For $i \in \mathcal{B}$, evaluate $\eta_i = 1/(\nabla\Psi_i^T\mathbf{M}^{-1}\nabla\Psi_i)$.
3. Warm start: if some initial guess γ^* is available for multipliers, then set $\gamma^0 = \gamma^*$, otherwise $\gamma^0 = \mathbf{0}$.
4. Initialize velocities: $\mathbf{v}^0 = \sum_{i\in\mathcal{A}}\mathbf{M}^{-1}\,\mathbf{D}_i\gamma_i^0 + \sum_{i\in\mathcal{B}}\mathbf{M}^{-1}\nabla\Psi_i{\gamma_{i,b}}^0 + \mathbf{M}^{-1}\tilde{\mathbf{k}}$.

5. For $i \in \mathcal{A}(\mathbf{q}^{(l)}, \delta)$, compute changes in multipliers for contact constraints:
$$\gamma_i^{r+1} = \lambda\, \Pi_{\Upsilon_i}\left(\gamma_i^r - \omega\eta_i\left(\mathbf{D}_i^T\mathbf{v}^r + \mathbf{b}_i\right)\right) + (1-\lambda)\gamma_i^r \;;$$
$$\Delta\gamma_i^{r+1} = \gamma_i^{r+1} - \gamma_i^r \;;$$
$$\Delta\mathbf{v}_i = \mathbf{M}^{-1}\,\mathbf{D}_i\Delta\gamma_i^{r+1}.$$

6. For $i \in \mathcal{B}$, compute changes in multipliers for bilateral constraints:
$$\gamma_i^{r+1} = \lambda\, \left(\gamma_i^r - \omega\eta_i\left(\nabla\Psi_i^T\mathbf{v}^r + b_i\right)\right) + (1-\lambda)\gamma_i^r \;;$$
$$\Delta\, \gamma_i^{r+1} = \gamma_i^{r+1} - \gamma_i^r \;;$$
$$\Delta\mathbf{v}_i = \mathbf{M}^{-1}\nabla\Psi_i\Delta\gamma_i^{r+1}.$$

7. Apply updates to the velocity vector:
$$\mathbf{v}^{r+1} = \mathbf{v}^r + \textstyle\sum_{i\in\mathcal{A}}\Delta\mathbf{v}_i + \sum_{i\in\mathcal{B}}\Delta\mathbf{v}_i$$

8. $r := r + 1$. Repeat from 5 until convergence, or until $r > r_{max}$.

The stopping criterion is based on the value of the velocity update. The overall algorithm that provides an approximation to the solution of Eqs. 10 through 14 relies on Algorithm 1 and requires the following steps:

Algorithm 2: Outer, Time-Stepping, Loop

1. Set $t = 0$, step counter $l = 0$, provide initial values for $\mathbf{q}^{(l)}$ and $\mathbf{v}^{(l)}$.

2. Perform collision detection between bodies, obtaining $n_{\mathcal{A}}$ possible contact points within a distance δ. For each contact i, compute $\mathbf{D}_{i,n}$, $\mathbf{D}_{i,u}$, $\mathbf{D}_{i,w}$; for each bilateral constraint compute the residual $\Phi_i(\mathbf{q})$, which also provides $\mathbf{b}_i$.

3. For each body, compute forces $\mathbf{f}(t^{(l)}, \mathbf{q}^{(l)}, \mathbf{v}^{(l)})$.

4. Use Algorithm 1 to solve the cone complementarity problem and obtain unknown impulse γ and velocity $\mathbf{v}^{(l+1)}$.

5. Update positions using $\mathbf{q}^{(l+1)} = \mathbf{q}^{(l)} + h\mathbf{L}(\mathbf{q}^{(l)})\mathbf{v}^{(l+1)}$.

6. Increment $t := t + h$, $l := l + 1$, and repeat from step 2 until $t > t_{\text{end}}$

A parallel implementation that leveraged the parallel computing power of commodity GPUs was considered based on the two algorithms outlined above. Solution of the CCP problem proceeds as a collection of functions, or kernels, which are executed on the GPU. First, some pre-processing steps are executed. Applied forces are calculated in a body-parallel fashion, and contacts are pre-processed in a contact-parallel fashion to compute the normal direction and friction plane directions. Next, the inner iteration loop is entered and a series of four kernels is executed until convergence. In a contact-parallel manner, the unilateral constraints are processed. In a constraint-parallel manner, the

bilateral constraints are processed. In a reduction-slot-parallel manner, speed updates are summed to a single resultant per body. Finally, in a body-parallel manner, speed updates are applied to each body. Once a certain number of iterations has been performed or convergence has been achieved, the generalized velocities are integrated forward in time in a body-parallel fashion to get the set of generalized positions. Details of the parallel reduction of speed-updates can be found in [20]. Pseudo-code for the parallel implementation can be seen below. Details regarding data structures and computational flow of the parallel implementation can also be found in [34, 20]. The details regarding the parallel collision detection are provided in Section 3.

Parallel Kernels for Solution of Dynamics Problem

1. Parallel Collision Detection
2. (Body parallel) Force kernel
3. (Contact parallel) Contact preprocessing kernel
4. Inner Iteration Loop:
   (a) (Contact parallel) CCP contact kernel
   (b) (Bilateral-Constraint parallel) CCP constraint kernel
   (c) (Reduction-slot parallel) Velocity change reduction kernel
   (d) (Body parallel) Body velocity update kernel
5. (Body parallel) Time integration kernel

---

3 PARALLEL COLLISION DETECTION

The implemented 3D collision detection algorithm performs a two-level spatial subdivision using axis-aligned bounding boxes. The first partitioning occurs at the CPU level and yields a relatively small number of *large boxes*. The second partitioning of each of these boxes occurs at the GPU level yielding a large number of *small bins*. The GPU 3D collision detection, which handles spheres, ellipsoids, and planes, occurs in parallel at the bin level. Any other geometries are represented as a collection of these primitives using a padding (decomposition) process detailed in [13]. Several kernel calls build on each other to eventually enable, in a one-thread-per-bin GPU parallel fashion, an exhaustive collision detection process in which thread i checks for collisions between all the bodies that happen to intersect the associated bin i. This requires $\mathcal{O}(b_i^2)$ computational effort, where b_i represents the number of bodies touching bin i. The value of b_i is controlled by an appropriate selection of the bin size. Figure 3 illustrates a typical collision detection scenario and is used in what follows to outline the nine stages of the proposed approach. Note that

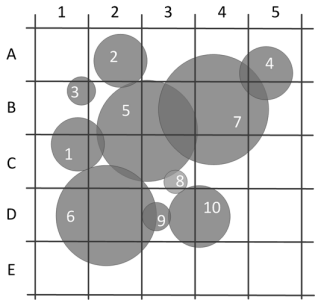


Figure 3: Two-dimensional example used to introduce the nine stages of the collision detection process. The grid is aligned to a global Cartesian reference frame.

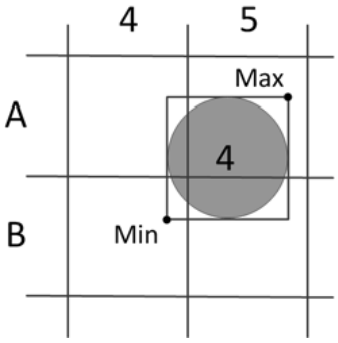


Figure 4: Minimum and maximum bounds of object, based on spatial subdivision in Fig. 3.

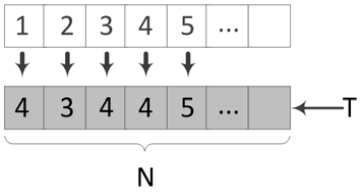


Figure 5: Array **T** with N entries, based on spatial subdivision in Fig. 3.

Figure 6: Result of prefix sum operation on **T**, based on spatial subdivision in Fig. 3. Each entry represents an object's offset based on the number of bins it touches.

the actual implementation is for 3D collision detection and does not require the bodies to be spheres.

Stage 1. The process begins by counting for each object the number of bins it intersects. As Fig. 4 shows, an object (body) can intersect, or touch, more than one bin. The minimum and maximum bounding points of each object are determined and placed in their respective bins. For example, Fig. 4 shows that object 4's minimum point lies in B4 and its maximum point in A5. The entire object must fit between the minimum and maximum points; therefore the number of bins that the object intersects can be determined quickly by counting the number of bins between the two points in each axis and multiplying them. In this case the number is 4. For each body, this number is saved into an array **T** (see Fig. 5), of size equal to the number of bodies N.

Stage 2. An inclusive parallel prefix sum is carried out on **T** [27]. The CUDA-based Thrust library implementation [14] of the scan algorithm operates on **T** to return in **S** (see Fig. 6) the memory offset information.

Stage 3. An array **B** (see Fig. 7), is first allocated of size equal to the value of the last element in **S**. This value is equal to the total number of object-bin intersections. Each element in **B** is set to a key-value pair of two unsigned integers. The key is the bin id and the value is the object id. In this stage, the

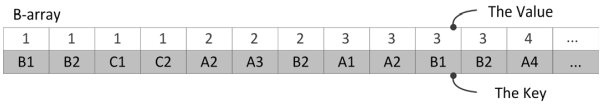


Figure 7: Array **B**, based on spatial subdivision in Fig. 3.

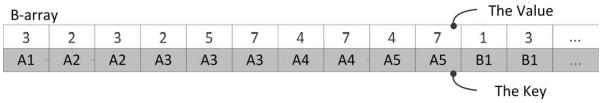


Figure 8: Sorted array **B**, based on spatial subdivision in Fig. 3.

memory offsets contained in **S** are used so that the thread associated with each body can write data to the correct location in **B**.

Stage 4. In this stage, the key-value array **B** is sorted by key, that is, by bin id. This effectively inverts the body-to-bin mapping to a bin-to-body mapping by grouping together all bodies in a given bin for further processing. The stage draws on the GPU-based radix sort from the Thrust library [14].

Stage 5. Next, the start of each bin in the sorted array **B** is identified in parallel. The number of threads used to this end is equal to the number of elements in **B**; i.e., the number of object-bin interactions. Each thread reads the current and previous bin value; if these values differ, then the start of a bin has been detected. The starting positions for each bin are written into an array **C** of key-value pairs of size equal to the number of bins in the 3D grid. When the start of a bin is found in array **B**, the thread and bin id are saved as the key and value, respectively. This pair is written to the element in **C** indexed by the bin id. Note that not all bins are active. Inactive bins; i.e., bins touched by zero or one bodies, are set to 0xffffffff, the largest possible value for an unsigned integer on a 32-bit, X86 architecture. Figure 9 shows the outcome of this stage.

Stage 6. The array **C** is next radix-sorted [14] by key. Consequently, inactive bins (identified by the 0xffffffff entries, represented for brevity as 0xfff in Fig. 10) "migrate" to the end of the array.

Stage 7. The total number of active bins is determined next by finding the index in the sorted array **C** of the first occurrence of 0xffffffff. Determining this index allows memory and thread usage to be allocated accurately thus having no threads wasted on inactive bins. One GPU thread is assigned in

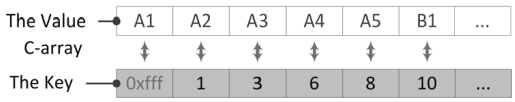


Figure 9: Array **C**, based on spatial subdivision in Fig. 3.

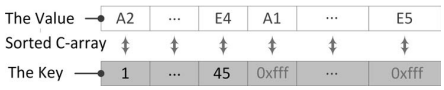


Figure 10: Sorted array **C**, based on spatial subdivision in Fig. 3.

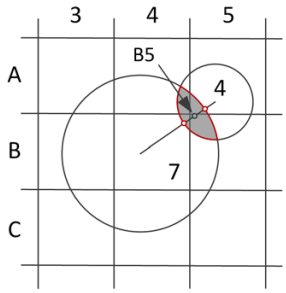


Figure 11: Center of collision volume. Based on spatial subdivision in Fig. 3.

this stage to each active bin to perform an exhaustive, brute-force, bin-parallel collision detection for the purpose of *only counting* the collision events. By carefully selecting the bin size, the number of objects being tested for collisions is expected to be small; i.e., on average, b_i is in the range of 3 to 4 objects per bin. After counting the total number of collisions in its bin, the thread writes that tally into an unsigned integer array **D** of size equal to the number of active bins.

More involved, the algorithm for counting and subsequently computing ellipsoid collision information is described in detail in [25]. For spheres, the algorithm checks for collisions by calculating the distance between the centers of the objects. Contacts can occur only when the distance between the spheres' centers is less than or equal to the sum of their radii. Because one object could be contained within more than one bin, checks were implemented to prevent double counting. Since the midpoint of a collision volume can be contained only within one bin, only one thread (associated with that bin) will register/count a collision event. For example, in order to determine the midpoint of the collision volume the algorithm relies on the vector from the centroid of object 4 to the centroid of object 7; see Fig. 11. The points where this vector intersects each object defines a segment; the location of the middle of this segment is used to decide the unique bin that claims ownership of the contact. If one object is completely inside the other, the midpoint of the collision volume is the centroid of the smaller object. Using this process, the number of collisions are counted for each bin and written to **D**.

Stage 8. An inclusive parallel prefix scan operation [14] is performed on **D**. This returns an array **E** whose last element is the total number of collisions in the uniform grid, a value that allows an exact amount of memory to be allocated in the next stage.

Stage 9. The final stage of the collision detection algorithm computes the actual contact information. To this end, an array of contact information structures **F** is allocated with a size equal to the value of the last element in **E**. The collision pairs are then found by using the algorithm outlined in Stage 7. Instead of

simply counting the number of collisions, actual contact information is *computed and written* to its respective place in **F**.

4 RENDERING PIPELINE

Adequately understanding the results of a simulation would be very arduous without an element of visualization since, as systems become more complex, the sheer volume of components and numerical outputs makes the results exceedingly difficult to interpret; consequently, rendering is a critical last step to the modeling and simulation process. To address this issue, a high performance visualization pipeline has been generated that allows for a simple means to create general-purpose renderings of arbitrary models. It supports a variety of simulation data files (csvs, custom-format, etc.) to remotely and easily generate an animation during or immediately after a simulation is computed. The "high-performance" attribute of this pipeline stems from its ability to scale up to 1000s of CPU cores as demonstrated by its use on the Euler supercomputer available to this research group [26].

Creating this pipeline poses several technical problems, the most conspicuous of which being how to "automate" the process as well as how to handle massively complex scenes. These issues have been addressed by utilizing Renderman [1] in conjunction with in-house developed code with the overall goal of implementing a full fledged distributed-computing rendering solution. The developed solution leverages Renderman's REYES (Renders Everything You Ever Saw) algorithm to handle complexity, uses computer clusters to process jobs, and utilizes the Renderman Bytestream and Shading Language to generate procedural scenes. Handling geometrically complex scenes with limited hardware resources, such as rendering a model with millions of granular bodies, is an insurmountable challenge for many commercial renderers. However, Renderman was particularly designed to handle this problem with the REYES algorithm. At the high-level, the REYES algorithm is a micropolygon renderer, which only performs shading computations for a subset of visible polygons at any given time, loading just this relevant scene data into memory. This data can be "bucketed" according to grids of pixels of the output image and rendered independently. The REYES pipeline is illustrated in Fig. 12.

Speed is another critical attribute of the rendering pipeline; this is an attribute where, given the size of the many-body systems considered, distributed-computing becomes an absolute necessity. To put it in perspective, in order to render a two-hour movie at 24 frames per second in one year, each frame can afford only three minutes of render time, a relatively short timeframe for complex scenes. Computer clusters combined with Renderman offer two vital means to parallelize rendering: simultaneous image rendering and distributed bucket rendering. Simultaneous image rendering tasks individual compute nodes with rendering a single image from the animation, the number of frames rendered in parallel scaling linearly with the number of nodes. Distributed bucket rendering allows for parallel rendering of the same image, where pixel buckets are rendered

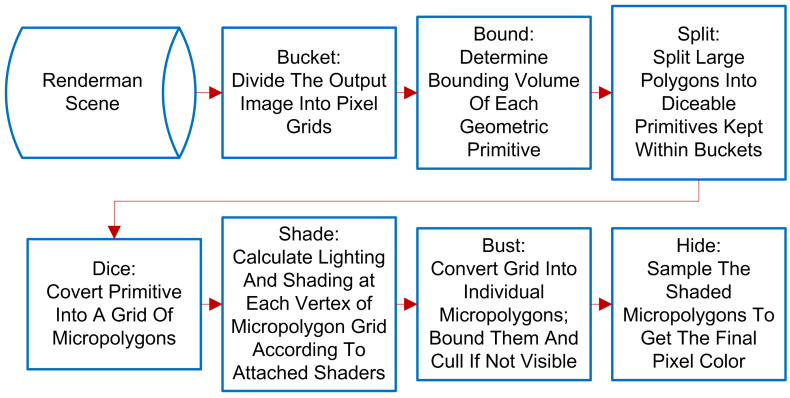


Figure 12: Schematic of the REYES pipeline.

independently on separate nodes and then stitched back together to form the final image. The benefits of these approaches are immediately apparent in the speedup factor, but also with the flexibility in rendering approach, where one can tailor the computation for a particular need (such as distributed bucket rendering for an immensely-visually complex still image).

Finally, in order to simplify the rendering process for users without a background in graphics, the pipeline, which is illustrated in Fig. 13, must automate image generation as much as possible while retaining the ability to render arbitrary visual effects; the Renderman Bytestream and Shading Language provide the means to meet this demand. The Renderman Bytestream simply allows us to pipe Renderman calls into the renderer at runtime thus facilitating procedural calls as the scene is being rendered. These procedural calls are determined by interpreting data with a simulation-specific metadata file that is either generated or defined by the user. This small metadata file configures the formatting options (such as resolution, input data format, etc.) and the salient features of objects in the simulation (geometry, appearance, etc.). Attaching user-specified Renderman shaders to the objects enables customization of simulation object appearance. Shaders are highly-functional compiled bits of code that, at a high-level, programmatically control how a micropolygon is perturbed or colored; the Renderman Shading Language is powerful and flexible enough to make it possible to define any visual effect. The user can draw from a library of shaders (created by our group) or define their own and assign them in the metadata file, consequently retaining full control over the appearance of their model. One last feature of the pipeline is the notion of "injecting" simulation data into predefined scenes. Setting up the aesthetic components of a scene can be a huge time-investment and typically requires artistic ability (lighting, cinematography, mise-en-scene); work an engineer typically does not want to deal with. The implemented software infrastructure offers a set of directives that one can insert

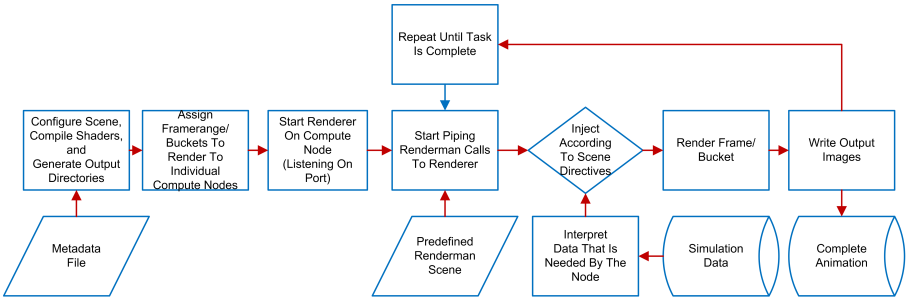


Figure 13: Schematic of the customized Renderman pipeline.

into existing Renderman scenes that, when encountered at render time, will be overridden with the emission of corresponding procedural calls (such as piping the interpreted simulation data). Thus, the users can specify a scene into which they want to "inject" their data, either drawing from a library of scenes (made available on Euler) or defining their own. Ultimately, this pipeline abstracts away the need to deal with commercial graphics applications, thus making it possible to host simulation rendering as a remote, "controllably-automatic" service for anyone without a background in graphics.

5 NUMERICAL EXPERIMENTS

5.1 Light Tracked Vehicle Mobility Simulation

This simulation captures the dynamics of a complex system comprised of many bilateral and unilateral constraints. Using a combination of joints and linear actuators, a tracked vehicle model was created and then simulated navigating over either flat rigid terrain or deformable terrain made up of gravel-type granular material. The vehicle is modeled to represent a small, lightweight tracked vehicle much like an autonomous robot that could be sent to another planet or used to navigate dangerous terrain.

There are two tracks, each with 61 track shoes (see Fig. 1(a)). Each track shoe is made up of two cylinders and three rectangular plates and has a mass of .34 kg. Each shoe is connected to its neighbors using one pin joint on each side, allowing the tracks to rotate relative to each other only along one axis. Within each track there are five rollers, each with a mass of 15 kg, one idler and one sprocket both with a mass of 15 kg. The chassis is modeled as a rectangular box with a mass of 200 kg and moments of inertia were computed for all parts using a CAD package. The purpose of the rollers is to keep the tracks separated and support the weight of the vehicle as it moves forward. The idler is necessary as it keeps the track tensioned. It is usually modeled with a linear spring/actuator but for the purposes of demonstration it was fixed using a revolute joint, to the

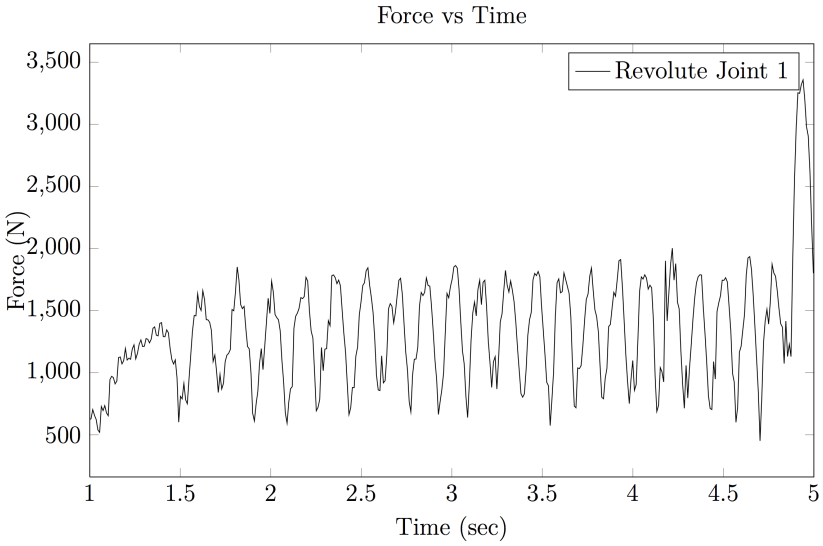


Figure 14: Magnitude of force experienced by one revolute joint.

vehicle chassis. The sprocket is used to drive the vehicle and is attached using a revolute joint to the chassis. Torque is applied to drive the track, with each track driven independently of the other. When the sprocket rotates, it comes into contact with the cylinders on the track shoe and turns the track with a gear like motion.

The track for the vehicle was created by first generating a ring of connected track shoes. This ring was dropped onto a sprocket, five rollers, and an idler which was connected to the chassis using a linear spring. The idler was pushed with 2000 N of force until the track was tensioned and the idler had stopped moving. This pre-tensioned track was then saved to a data file and loaded for the simulation of the complete vehicle.

5.2 Simulation Results for Tracked Vehicle

In this simulation scenario, the tracked vehicle was dropped onto a flat surface and a torque was applied to the sprocket to drive it forwards; the forces on several revolute joints connecting the track shoes were analyzed as they traveled around the sprocket. Figure 14 shows the forces in one revolute joint after the track has dropped onto the flat surface. Transient behavior is observed when the torque is applied to the sprocket at 1 second and the track shoe connected to this joint comes into contact with the sprocket at 5 s. The oscillatory behavior of the joint forces can be attributed to several factors. First, the tension in the track was very high; there was no spring/linear actuator attached to the idler, so high tension forces could not be dampened. Secondly, the combination of a high pre-tensioning force (2000 N) and lack of a linear actuator on the idler

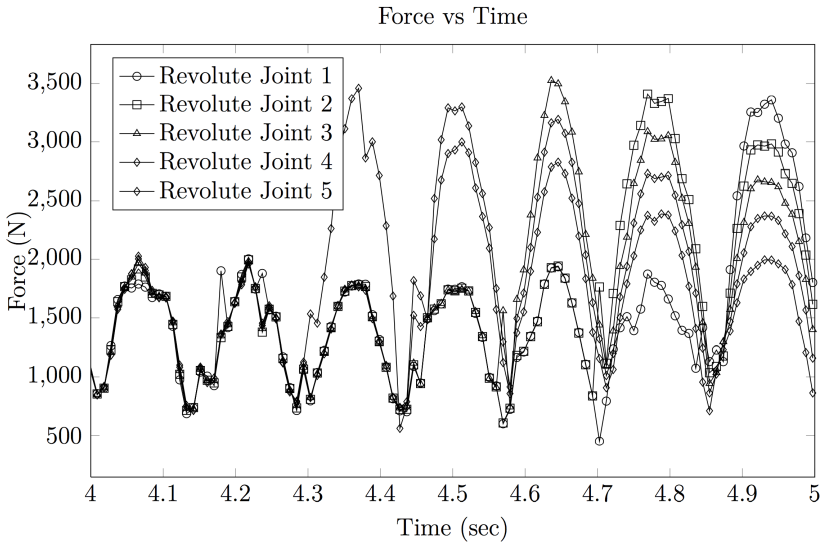


Figure 15: Magnitude of force experienced by 5 revolute joints.

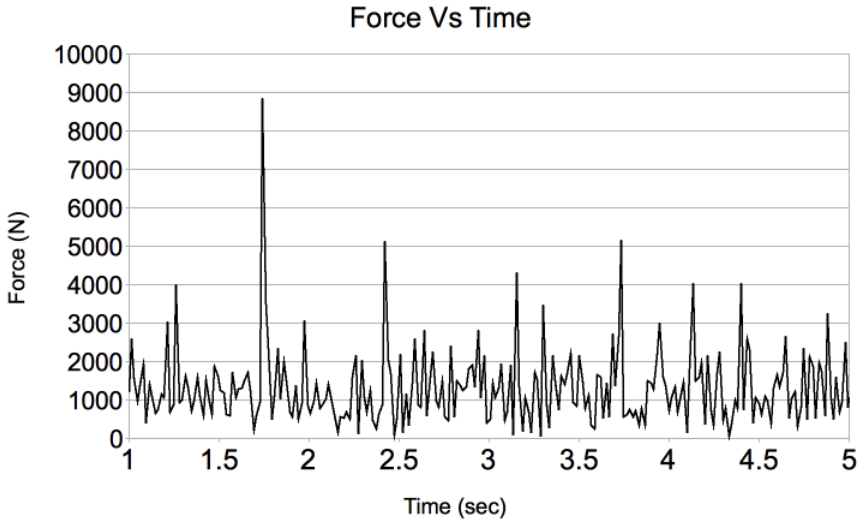


Figure 16: Magnitude of force experienced by one revolute joint on granular terrain.

resulted in high revolute joint forces.

Figure 15 shows the joint forces for several revolute joints as their associated track shoes go around the sprocket. This plot shows that the forces in the joint are highest when the track shoe first comes into contact with the sprocket. As the track shoe moves around the sprocket, the force decreases as subsequent track shoes and their revolute joints help distribute the load. It should be noted that the gearing motion between the track shoes and the sprocket was not ideal as it was not very smooth. In a more realistic model, forces between track shoes would be overlapping so that the movement of the tracks is more smooth and the forces experienced by the revolute joints are smaller.

Figure 15 shows the joint forces for one revolute joint where the tracked vehicle was simulated as it moved over a bed of 84,000 granular particles. The particles were modeled as large pieces of gravel with a radius of .075 m, and a density of 1900 kg/m^3. A torque of 100 N-m was applied to both sets of tracks to move the vehicle. Note that unlike the case where the vehicle moves on a flat section of ground, the forces experienced by the revolute joints are much noisier. Individual grains move under the tracks as the vehicle moves causing large vibrations to travel through the shoes. These vibrations would be reduced when modeling a more complaint terrain material that can dissipate energy on contact.

5.3 Anchoring in Granular Material

The purpose of this effort is to study the performance of different anchor designs and provide a recommendation on which is better suited to the task of anchoring. The anchors will be tested against a range of parameters relating to soil, environment, and anchor penetration angles/velocities to better understand their corresponding performance characteristics.

The anchor was modeled using three types of regular primitives. The tip of the anchor was modeled using a sphere, the shaft was modeled using a cylinder, and the helix was modeled using 67 thin boxes swept along a helical spline. This configuration is optimal compared to using a triangulated mesh for the anchor. The triangular mesh requires several thousand triangles, decreasing the performance of the collision detection and increasing the memory requirements for the simulation. Using primitives has an added benefit; modifying the geometry of the anchor becomes straightforward, thus the pitch of the anchor along with its diameter and thickness can be varied easily allowing parametric studies to be completed.

The anchor had several constraints and forces which were used to control its motion. First, there was a bilateral constraint restricting the planar motion of the anchor, forcing it to move straight up or down. This constraint also prevented the anchor from rotating in all axes except the vertical axis. A pressing force was used to press the anchor into the material. This simulated an actuator which may be attached to the other end of the anchor, forcing it to penetrate. A torque was applied to the anchor to cause it to rotate and screw into the material, and a vertical force was used to pull the anchor out of the material at

the end of the simulation.

The numerical experiment consisted of a granular bed made up of spheres of randomly varying radii that was pre-settled and loaded at the start of each simulation. For each simulation only the parameters associated with the anchor were varied. The anchor's mass was 10 kg with a radius of 0.5m. The granular material had a mass of 0.005 kg and a radius randomly varying between 0.025 m and 0.036 m. The granular material had a friction coefficient of 0.4, and gravity was set to -9.806 m/s^2. The time step was 5×10^{-4} s, with 1000 CCP iterations performed per time step.

5.4 Simulation Results for Anchoring System

Several sets of parametric tests were simulated using the anchor model: the torque applied to the anchor was varied, and the pullout force applied after anchoring was varied. Fig. 17 shows an anchor with the same mass that has 4 different torques applied to it. For each test at the end the pullout force remained constant at 300N. The plot shows that at 2 seconds, when the anchoring torque was applied, the anchor with the highest torque went in the deepest, fastest, which is as expected. With the constant pulling force that was applied at 7 seconds, only the anchor with the lowest anchoring torque, 600 N-m, was pulled upwards, the mass of the granular material above the three other anchors was too high for the force to have any effect.

Fig. 18, with a close up in Fig. 19, shows a different set of simulations. Here the anchoring torque was kept constant, but the pullout force was changed. The purpose of the test was to gauge the magnitude of the pullout force required for a given applied torque. The plot shows that only a force of 2000N was able to pullout easily, gaining velocity as it moved upwards. The pullout force of 1600N was able to start pulling out slowly and at a constant velocity.

6 CONCLUSIONS AND FUTURE WORK

This work describes developments that expand parallel simulation capabilities in multibody dynamics. The many-body dynamics problem of interest has been modeled as a cone complementarity problem whose parallel numerical solution scales linearly with the number of bodies in the system. These developments have directly resulted in the ability to simulate complex tracked vehicles operating on granular terrain. The parallel simulation capability was demonstrated in the context of an application that emphasizes the interplay between light-vehicle/track/terrain dynamics, where the vehicle feature length becomes comparable with the dimensions associated with the obstacles expected to be negotiated by the vehicle. The simulation capability is anticipated to be useful in gauging vehicle mobility early in the design phase, as well as in testing navigation/control strategies defined/learned on the fly by small autonomous vehicles as they navigate uncharted terrain profiles.

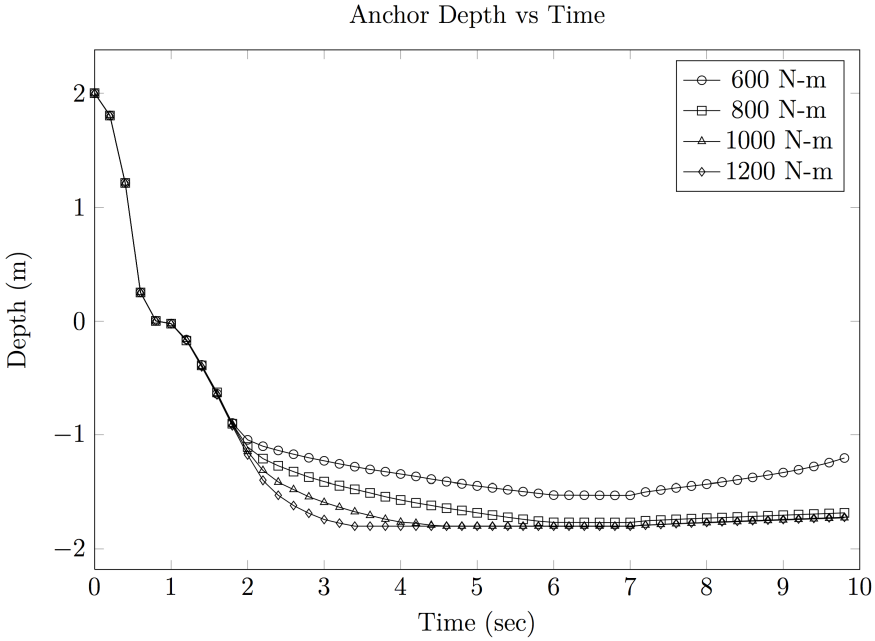


Figure 17: Anchor with different applied torques and a constant pullout force of 300N.

In terms of future work, the convergence issue induced by the multiscale attribute of the vehicle-terrain interaction problem remains to be addressed. Additionally, technical effort will focus on extending the entire algorithm to run on a cluster of GPU-enabled machines, further increasing the size of tractable problems. The modeling approach remains to be augmented with a dual discrete/continuum representation of the terrain to accommodate large scale simulations for which an exclusively discrete terrain model would unnecessarily burden the numerical solution.

7 ACKNOWLDEGMENTS

The research effort of the first two authors was partially supported by funding provided by the National Science Foundation under NSF Project CMMI-0840442 and through TARDEC grant W911NF-11-D-0001-0048. M. Quadrelli, A. Jain and H. Mazhar were partially supported through Jet Propulsion Laboratory sub-contract 1435056 for research carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Funding for the third author was provided by a US Army SMART fellowship.

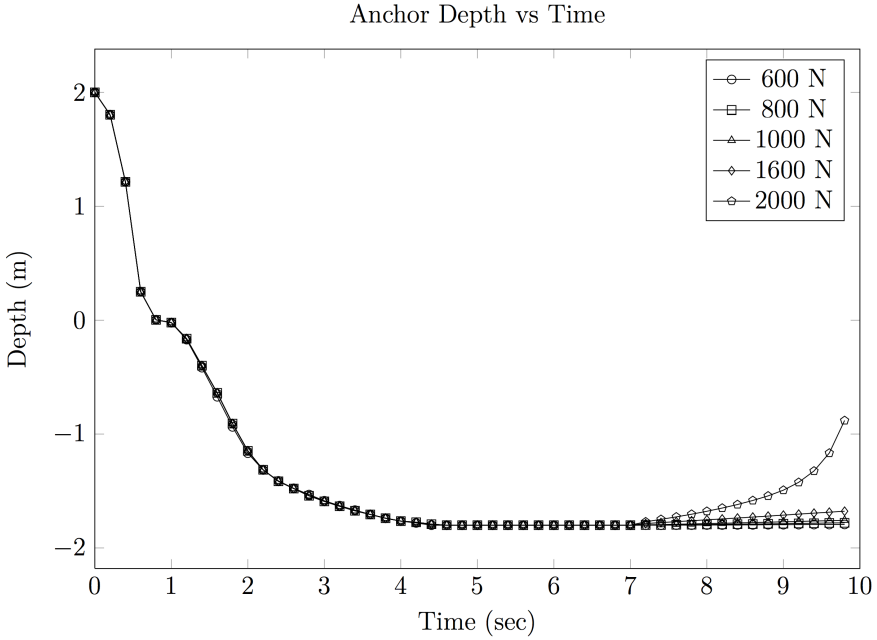


Figure 18: Anchor with different pullout forces and a constant torque of 1000N.

References

[1] Pixar Animation Studios: RenderMan Interface Specification, version 3.2, 2000.

[2] Anitescu, M. Optimization-based simulation of nonsmooth rigid multibody dynamics. *Mathematical Programming 105*, 1 (2006), 113–143.

[3] Anitescu, M., and Hart, G. D. A constraint-stabilized time-stepping approach for rigid multibodydynamics with joints, contact and friction. *International Journal for Numerical Methods in Engineering 60(14)* (2004), 2335–2371.

[4] Anitescu, M., and Hart, G. D. A fixed-point iteration approach for multibody dynamics with contact and friction. *Mathematical Programming, Series B 101(1)* (2004), 3–32.

[5] Anitescu, M., and Potra, F. A. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics 14* (1997), 231–247.

[6] Anitescu, M., Potra, F. A., and Stewart, D. Time-stepping for three-dimensional rigid-body dynamics. *Computer Methods in Applied Mechanics and Engineering 177* (1999), 183–197.

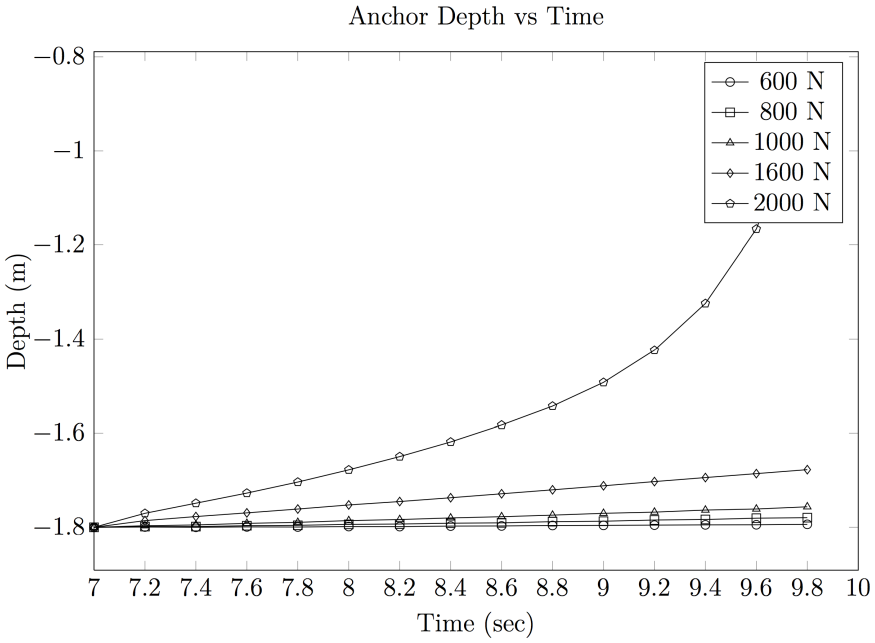


Figure 19: Closeup of Fig. 18, Anchor with different pullout forces and a constant torque of 1000N.

[7] Anitescu, M., and Tasora, A. An iterative approach for cone complementarity problems for nonsmooth dynamics. *Computational Optimization and Applications 47*, 2 (2010), 207–235.

[8] Baraff, D. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica 10* (1993), 292–352.

[9] Baraff, D. Fast contact force computation for nonpenetrating rigid bodies. In *Computer Graphics (Proceedings of SIGGRAPH)* (1994), pp. 23–34.

[10] Cottle, R. W., and Dantzig, G. B. Complementary pivot theory of mathematical programming. *Linear Algebra and Its Applications 1* (1968), 103–125.

[11] Donald, B. R., and Pai, D. K. On the motion of compliantly connected rigid bodies in contact: A system for analyzing designs for assembly. In *Proceedings of the Conf. on Robotics and Automation* (1990), IEEE, pp. 1756–1762.

[12] Haug, E. J. *Computer-Aided Kinematics and Dynamics of Mechanical Systems Volume-I.* Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[13] Heyn, T. *Simulation of Tracked Vehicles on Granular Terrain Leveraging GPU Computing.* M.S. thesis, Department of Mechanical Engineering, University of Wisconsin–Madison, http://sbel.wisc.edu/documents/TobyHeynThesis_final.pdf, 2009.

[14] Hoberock, J., and Bell, N. Thrust: A Parallel Template Library. Available online at http://code.google.com/p/thrust/, 2009.

[15] Lotstedt, P. Mechanical systems of rigid bodies subject to unilateral constraints. *SIAM Journal of Applied Mathematics 42*, 2 (1982), 281–296.

[16] Madsen, J., Pechdimaljian, N., and Negrut, D. Penalty versus complementarity-based frictional contact of rigid bodies: A CPU time comparison. Tech. Rep. TR-2007-06, Simulation-Based Engineering Lab, University of Wisconsin, Madison, 2007.

[17] Marques, M. D. P. M. *Differential Inclusions in Nonsmooth Mechanical Problems: Shocks and Dry Friction*, vol. 9 of *Progress in Nonlinear Differential Equations and Their Applications.* Birkhäuser Verlag, Basel, 1993.

[18] Mazhar, H. *Parallel Multi-Body Dynamics on Graphics Processing Unit (GPU) Cards.* M.S. thesis, Department of Mechanical Engineering, University of Wisconsin–Madison, http://sbel.wisc.edu/documents/HammadMazharMSthesisFinal.pdf, 2012.

[19] Moreau, J. J. Standard inelastic shocks and the dynamics of unilateral constraints. In *Unilateral Problems in Structural Analysis* (New York, 1983), G. D. Piero and F. Macieri, Eds., CISM Courses and Lectures no. 288, Springer–Verlag, pp. 173–221.

[20] Negrut, D., Tasora, A., Anitescu, M., Mazhar, H., Heyn, T., and Pazouki, A. Solving large multi-body dynamics problems on the GPU. *GPU Gems Vol. 4* (2011), 269–280.

[21] NVIDIA. CUDA Programming Guide. Available online at http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf, 2012.

[22] Pang, J.-S., Kumar, V., and Song, P. Convergence of time-stepping method for initial and boundary-value frictional compliant contact problems. *SIAM Journal of Numerical Analysis 43*, 5 (2005), 2200–2226.

[23] Pang, J.-S., and Stewart, D. Differential variational inequalities. *Mathematical Programming 113*, 2 (2008), 345–424.

[24] Pang, J.-S., and Trinkle, J. C. Complementarity formulations and existence of solutions of dynamic multi-rigid-body contact problems with Coulomb friction. *Mathematical Programming 73*, 2 (1996), 199–226.

[25] Pazouki, A., Mazhar, H., and Negrut, D. Parallel ellipsoid collision detection with application in contact dynamics-DETC2010-29073. In *Proceedings to the 30th Computers and Information in Engineering Conference* (2010), S. Fukuda and J. G. Michopoulos, Eds., ASME International Design Engineering Technical Conferences (IDETC) and Computers and Information in Engineering Conference (CIE).

[26] SBEL. Euler: A CPU/GPU–Heterogeneous Cluster at the Simulation-Based Engineering Laboratory, University of Wisconsin-Madison. http://sbel.wisc.edu/Hardware, 2012.

[27] Sengupta, S., Harris, M., Zhang, Y., and Owens, J. Scan primitives for GPU computing. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware* (2007), Eurographics Association, p. 106.

[28] Song, P., Kraus, P., Kumar, V., and Dupont, P. Analysis of rigid-body dynamic models for simulation of systems with frictional contacts. *Journal of Applied Mechanics 68(1)* (2001), 118–128.

[29] Song, P., Pang, J.-S., and Kumar, V. A semi-implicit time-stepping model for frictional compliant contact problems. *International Journal of Numerical Methods in Engineering 60*, 13 (2004), 267–279.

[30] Stewart, D. E. Rigid-body dynamics with friction and impact. *SIAM Review 42(1)* (2000), 3–39.

[31] Stewart, D. E., and Trinkle, J. C. An implicit time-stepping scheme for rigid-body dynamics with inelastic collisions and Coulomb friction. *International Journal for Numerical Methods in Engineering 39* (1996), 2673–2691.

[32] Tasora, A. A Fast NCP Solver for Large Rigid-Body Problems with Contacts. In *Multibody Dynamics: Computational Methods and Applications*, C. Bottasso, Ed. Springer, 2008, pp. 45–55.

[33] Tasora, A., Manconi, E., and Silvestri, M. Un nuovo metodo del simplesso per il problema di complementarit lineare mista in sistemi multibody con vincoli unilateri. In *Proceedings of AIMETA 05* (Firenze, Italy, 2005).

[34] Tasora, A., Negrut, D., and Anitescu, M. Large-scale parallel multi-body dynamics with frictional contact on the Graphical Processing Unit. *Journal of Multi-body Dynamics 222*, 4 (2008), 315–326.

[35] Trinkle, J., Pang, J.-S., Sudarsky, S., and Lo, G. On dynamic multi-rigid-body contact problems with Coulomb friction. *Zeitschrift fur angewandte Mathematik und Mechanik 77* (1997), 267–279.