Time Integrating Articulated Body Dynamics Using Position-Based Collocation Methods

Zherong Pan[1] and Dinesh Manocha[2]
[1]University of North Carolina, North Carolina NC 27514, USA,
zherong@cs.unc.edu
[2]Department of Computer Science and Electrical & Computer Engineering, University of Maryland at College Park, Maryland MD 20740, USA,
dm@cs.umd.edu

Abstract. We present a new time integrator for articulated body dynamics. We formulate the governing equations of the dynamics using only the position variables and then recast the position-based articulated dynamics as an optimization problem. Our reformulation allows us to integrate the dynamics in a fully implicit manner without computing high-order derivatives. Therefore, under arbitrarily large timestep sizes, we observe highly stable behaviors using an off-the-shelf numerical optimizer. Moreover, we show that the accuracy of our time integrator can increase by using a high-order collocation method. We show that each iteration of optimization has a complexity of $\mathcal{O}(N)$ using the Quasi-Newton method or $\mathcal{O}(N^2)$ using Newton's method, where N is the number of links of the articulated model. Finally, our method is highly parallelizable and can be accelerated using a Graphics Processing Unit (GPU). We highlight the efficiency and stability of our method on different benchmarks and compare the performance with prior articulated body dynamics simulation methods based on the Newton-Euler equation. Using a larger timestep size, our method achieves up to 4 times speedup on a single-core CPU. With GPU acceleration, we observe an additional $3-6$ times speedup over a 4-core CPU.

1 Introduction

Numerical modeling of articulated bodies is a fundamental problem in robotics. It is important in the design and evaluation of mechanisms, robot arms, and humanoid robots. Furthermore, articulated body simulators are increasingly used to evaluate a controller during reinforcement learning [6, 23], to predict the future state of a robot during online control [29], and to satisfy the dynamics constraints for motion planners [27]. In all these applications, the underlying algorithms are implemented on top of dynamic simulators and may invoke these simulators thousands of times for different parameters and settings [29]. As a result, the performance of these applications is easily affected by these simulators' performances.

Many widely-used articulated body simulation packages [25, 29] are based on implicit time-stepping schemes [26]. These methods model the articulated body's governing equation as an ordinary differential equation (ODE) and then integrate the ODE using high-order numerical schemes. These methods can be arbitrarily accurate but require small timestep sizes. One simple strategy to improve the runtime performance is to use a large timestep size [24]. This strategy has proven successful in some applications, such as controlling humanoid robots [28], where the timestep size used in a controller can be larger than that used in the underlying simulator. A key issue in using a large

timestep size is ensuring that the time integrator is still stable. For example, the stable region of a semi-implicit Euler integrator shrinks as the timestep size increases [4]. To time integrate an articulated body under a large timestep size, a simple and widely-used method is to use an unconditionally stable fully implicit Euler integrator [4]. However, in a conventional articulated body's governing dynamics equation, the use of a fully implicit Euler integrator involves a costly $\mathcal{O}(N^3)$ computation of high-order derivatives, where N is the number of links in an articulated body.

Main Results: We present position-based articulated dynamics (PBAD), a novel optimization-based algorithm for articulated body dynamics simulation. Unlike prior method [26], which represents the velocity as a time derivative and evaluates this derivative analytically, our PBAD formulation represents this velocity using finite differences in the Euclidean space. This Euclidean space discretization allows us to represent all the physical variables as functions of positions. As a result, we can integrate the system implicitly without high-order derivatives. In addition, we show that numerical simulation under our PBAD framework can be recast as a numerical optimization. Therefore, our time integrator is stable under an arbitrarily large timestep size because a numerical optimizer can ensure that the energy value decreases during each iteration through line-search [19] or trust region limitation [18]. Solving these unconstrained minimization problems requires evaluating the energy gradient and/or Hessian and solving a linear system of size $\mathcal{O}(N)$. To this end, we use techniques similar to well-known forward- and inverse-dynamics algorithms [8] and show that the necessary energy gradient and Hessian information can be computed within $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$. Finally, we show that the accuracy of PBAD time integrator can be improved by approximating the velocity using high-order polynomials, leading to a high-order collocation method [12].

We have implemented our algorithm and evaluated the performance on many articulated models with $10 - 200$ DOFs. Compared with a conventional semi-implicit Euler integrator, our PBAD simulator achieves up to 4 times overall speedup with a serial implementation running on a single-core CPU. Finally, all the operations in our unconstrained energy minimization are inherently parallel and we accelerate the simulation on a GPU to obtain $3 - 6$ times additional speedup over a 4-core CPU, as shown in Section 5.2.

The rest of the paper is organized as follows. We first review conventional Lagrangian articulated body dynamics in Section 3 and then introduce our PBAD formulation in Section 4. Next, in Section 5, we present some algorithmic and numeric analysis of our method. Finally, we compare our method with an earlier method [26] on a set of classic benchmarks used by [25, 29] in Section 6. We also show some applications in online/offline control algorithms in Section 6.

2 Related Work

We give a brief overview of previous work in articulated body dynamics, time-integration schemes, and position-based dynamics.

2.1 Articulated Body Dynamics

Articulated body dynamic simulation is a classic, well-studied problem in robotics. Some methods [31, 5, 30] focus on articulated bodies with general constraints, where the configurations of articulated bodies are represented using maximal coordinates. However, tree-structured articulated bodies represented using minimal coordinates have received the most attention. Very efficient algorithms [20, 8] have been developed for forward/inverse-dynamics and these are key steps in a dynamics simulator. These algorithms have been further accelerated using divide-and-conquer [7], adaptivity [11], and GPU-parallelism [32, 33].

2.2 Time Integration Schemes

A time integrator predicts the future configuration of an articulated body given its current configuration. Time integrators vary in their computational cost, stability, and accuracy (see [4, 16] for a review). Widely-used integrators in articulated body simulators [25, 29], such as explicit high-order Runge-Kutta schemes, are linear multi-step methods for ODE, which requires small timestep sizes. Compared with explicit schemes, implicit Runge-Kutta schemes have better stability, some of which are also known as collocation methods [1]. Collocation methods approximate the locus of configuration using high-order polynomials. Unlike these general-purpose integrators, special integrators such as Lie-group integrators [15] and variational integrators [17] can be developed to respect the Lie group structure of articulated bodies, resulting in desirable conservative properties in linear/angular momentum and the Hamiltonian.

2.3 Position-Based Dynamics (PBD)

Our method is inspired by the recent advances in PBD in computer graphics (see [2] for a survey). PBD has been shown to be stable under arbitrarily large timestep sizes and is preferred for interactive applications such as game engines. PBD algorithms have been developed for various dynamics systems such as fluid bodies, deformable bodies, and rigid bodies [5]. In computer graphics, however, rigid bodies are represented using maximal coordinates while in our PBAD formulation, we use minimal coordinates. We have also extended conventional second-order PBD discretizations to arbitrarily high-order collocation methods. The connection between PBD and optimization-based integrators is revealed in [3] and later refined in [10, 21].

3 Background: Lagrangian Articulated Body Dynamics

We briefly review the conventional articulated body dynamics formulation under generalized coordinates (see [20] for more details). Throughout our derivation, we assume that there is only one rigid body. The more general case of multiple rigid bodies can be derived by a concatenation of equations for each rigid body. The configuration of a rigid body $\mathcal{B}$ is parameterized by generalized coordinates, $\mathbf{q}$. $|\mathbf{q}|$ is the number of DOFs and

is proportional to the number of links, N. For an arbitrary point $\mathbf{p} \in \mathcal{B}$ in the body-fixed frame of reference, its corresponding position in a global frame of reference is:

$$\mathbf{P}(\mathbf{q}) = \mathbf{R}(\mathbf{q})\mathbf{p} + \mathbf{t}(\mathbf{q}),$$

where $\mathbf{R}$ is a global rotation and $\mathbf{t}$ is a global translation. The dynamics of $\mathcal{B}$ is governed by the following equation:

$$\int_{\mathbf{p}\in\mathcal{B}} \frac{\partial \mathbf{P}(\mathbf{q})}{\partial \mathbf{q}}^T \left[\rho \ddot{\mathbf{P}}(\mathbf{q}) - \mathbf{f}\right] d\mathbf{p} = 0, \tag{1}$$

where $\mathbf{f}$ are the internal/external forces on $\mathbf{p}$ and ρ is the mass density. If we analytically evaluate the second derivative in Equation 1, we arrive at the following well-known equation:

$$\mathbf{J}^T\mathbf{M}\mathbf{J}\ddot{\mathbf{q}} + \left[\mathbf{J}^T\mathbf{M}\dot{\mathbf{J}} + \mathbf{J}^T \begin{pmatrix} 0 & \\ & [\boldsymbol{\omega}] \end{pmatrix} \mathbf{M}\mathbf{J}\right] \dot{\mathbf{q}} - \mathbf{J}^T\mathbf{f} = 0, \tag{2}$$

where we have $\dot{\mathbf{R}} = [\boldsymbol{\omega}]\mathbf{R}$, $\mathbf{J} = \left(\partial\boldsymbol{\omega}/\partial\mathbf{q}^T \; \partial\mathbf{t}/\partial\mathbf{q}^T\right)^T$, $\mathbf{M}$ being the 6×6 mass matrix. From Equation 2, we can formulate a discrete version to predict the next configuration $\left(\mathbf{q}_{k+1} \; \dot{\mathbf{q}}_{k+1}\right)$ from the current configuration $\left(\mathbf{q}_k \; \dot{\mathbf{q}}_k\right)$. Here we use subscript to denote timestep index, i.e. $\mathbf{q}_k$ is $\mathbf{q}$ at time instance $k\Delta t$. To this end, several widely-used articulated body simulators [29, 25] use a semi-implicit Euler scheme:

$$\frac{\dot{\mathbf{q}}_{k+1} - \dot{\mathbf{q}}_k}{\Delta t} = \left[\mathbf{J}_k^T\mathbf{M}_k\mathbf{J}_k\right]^{-1} \left[\mathbf{J}_k^T\mathbf{f}_k - \left(\mathbf{J}_k^T\mathbf{M}_k\dot{\mathbf{J}}_k + \mathbf{J}_k^T \begin{pmatrix} 0 & \\ & [\boldsymbol{\omega}_k] \end{pmatrix} \mathbf{M}_k\mathbf{J}_k\right) \dot{\mathbf{q}}_k\right], \tag{3}$$

where $[\boldsymbol{\omega}_k]$ is the 3×3 skew-symmetric cross-product matrix. The above scheme usually works well for a small timestep size (usually smaller than 0.01s), but its stability under large timestep size is not guaranteed. This is due to the explicit velocity update in Equation 3, i.e. the right-hand side of Equation 3 is at timestep k. One common method for achieving better stability under a large timestep size is to use the fully implicit Euler scheme by replacing $\left(\mathbf{q}_k \; \dot{\mathbf{q}}_k\right)$ in the right-hand side of Equation 3 with $\left(\mathbf{q}_{k+1} \; \dot{\mathbf{q}}_{k+1}\right)$ and solving for $\mathbf{q}_{k+1}$ using an iterative algorithm. A widely-used iterative algorithm is the (Quasi)-Newton method, which has been used to stably simulate deformable and fluid bodies [24]. However, there are two difficulties in using the (Quasi)-Newton method for fully implicit integration:

- The (Quasi)-Newton method requires the derivatives of the right-hand side of Equation 3 with respect to q_{k+1}, which involves third-order derivatives, $\partial^3\mathbf{R}/\partial\mathbf{q}^3$ and $\partial^3\mathbf{t}/\partial\mathbf{q}^3$, the evaluation complexity of which is $\mathcal{O}(N^3)$.
- The implicit integrator solves a system of nonlinear equations for which even (Quasi)-Newton method could fail to converge under large timestep sizes [10].

4 Position-based Articulated Body Dynamics

In this section, we present our PBAD formulation, which overcomes some of the problems found in prior time integrators. We notice that, from Equation 1, the acceleration

of $\mathbf{P}$ is evaluated analytically to derive Equation 2, which involves up to second-order derivatives. However, if we use a finite difference approximation of $\ddot{\mathbf{P}}$ directly from Equation 1, the analytic derivatives can be eliminated, allowing us to perform a (Quasi)-Newton method without evaluating $\partial^3\mathbf{R}/\partial\mathbf{q}^3$ and $\partial^3\mathbf{t}/\partial\mathbf{q}^3$. For example, if we use second-order finite difference approximation, Equation 1 becomes:

$$\int_{\mathbf{p}\in\mathcal{B}} \frac{\partial\mathbf{P}(\mathbf{q}_{k+1})}{\partial\mathbf{q}_{k+1}}^T \left[\rho\frac{\mathbf{P}(\mathbf{q}_{k+1}) - 2\mathbf{P}(\mathbf{q}_k) + \mathbf{P}(\mathbf{q}_{k-1})}{\Delta t^2} - \mathbf{f}(\mathbf{P}(\mathbf{q}_{k+1}))\right] d\mathbf{p} = 0. \quad (4)$$

Corresponding to Equation 1 under the conventional formulation, Equation 4 is the governing equation under our PBAD formulation. Note that Equation 4 converges to Equation 1 as $\Delta t \to 0$. Equation 4 takes a similar form to the governing equations in previous PBD methods [21, 13] for simulating deformable bodies but is expressed for articulated bodies under minimal coordinates. We can now argue that Equation 4 overcomes the two difficulties. First, if we use the Newton's method to solve Equation 4, we only need to evaluate derivatives up to the second-order, i.e. $\partial^2\mathbf{R}/\partial\mathbf{q}^2$ and $\partial^2\mathbf{t}/\partial\mathbf{q}^2$. Moreover, we will show in Section 5 that, if we use the Quasi-Newton method, only first-order derivatives are needed without modifying the final solutions. Second, the convergence difficulty of the (Quasi)-Newton method under a very large timestep size can be fixed by reformulating Equation 4 as an energy minimization problem:

$$\mathbf{E}(\mathbf{q}) \triangleq \int_{\mathbf{p}\in\mathcal{B}} \left[\frac{\rho}{2\Delta t^2}\|\mathbf{P}(\mathbf{q}_{k+1}) - 2\mathbf{P}(\mathbf{q}_k) + \mathbf{P}(\mathbf{q}_{k-1})\|^2 + \mathbf{Q}(\mathbf{P}(\mathbf{q}_{k+1}))\right] d\mathbf{p}, \quad (5)$$

where $\mathbf{Q}$ is the potential energy for a position-dependent conservative force $\mathbf{f}$. Such a reformulation allows us to use an off-the-shelf, gradient-based optimizer to solve for $\mathbf{q}_{k+1} = \mathbf{argmin}\ \mathbf{E}(\mathbf{q})$. These optimizers use line-search [19] or trust region limitations [18] to ensure that each iteration gets the solution closer to a local minima of $\mathbf{E}(\mathbf{q})$, i.e. the correct $\mathbf{q}_{k+1}$. Although $\mathbf{E}(\mathbf{q})$ in Equation 5 still involves an integral over $\mathcal{B}$, we can derive its analytic form.

4.1 High-Order Position-Based Collocation Method

One advantage of using Equation 1 is that one could use a general linear multistep method (see [4]) to achieve a variable-order of accuracy. We show that our PBAD formulation can also have such flexibility by modifying a high-order collocation method [1]. A collocation method approximates the locus of the configuration of $\mathcal{B}$ using high-order polynomials. Note that, in Equation 4, we assume that, for any $\mathbf{p} \in \mathcal{B}$, its trajectory in the period of time $[(k-1)\Delta t, (k+1)\Delta t]$ is determined by the three collocation points $\mathbf{P}(\mathbf{q}_{k-1}), \mathbf{P}(\mathbf{q}_k), \mathbf{P}(\mathbf{q}_{k+1})$ and a collocation method assumes that $\mathbf{p}$ follows a polynomial curve passing through all the collocation points. For example, in Equation 5, we can fit a quadratic curve from the three points so that it is a second-order collocation method.

To develop higher-order methods, we introduce additional collocation points in between timesteps $(\mathbf{P}(\mathbf{q}_{k+\alpha_1}), \cdots, \mathbf{P}(\mathbf{q}_{k+\alpha_{N-2}}))$ for an Kth-order method, where $0 < \alpha_1 < \cdots < \alpha_{K-1} = 1$. We fit an Kth-order polynomial for any $\mathbf{p} \in \mathcal{B}$ from the

$K+1$ collocation points $\mathbf{P}_* \triangleq \left(\mathbf{P}(\mathbf{q}_{k-1+\alpha_{K-2}}) \cdots \mathbf{P}(\mathbf{q}_{k+\alpha_{K-1}})\right)$. The Kth-order polynomial takes the following form:

$$\mathbf{P}(t) \triangleq \mathbf{P}_*\mathbf{H}\left(1\ t \cdots t^K\right)^T \quad \ddot{\mathbf{P}}(t) \triangleq \mathbf{P}_*\mathbf{H}''\left(1\ t \cdots t^K\right)^T,$$

where $\mathbf{H}, \mathbf{H}''$ are the polynomial basis matrices. We call this a position-based collocation method. A key difference between a position-based collocation method and a conventional collocation method [1] is that we fit polynomials for $\mathbf{P}$ instead of $\mathbf{q}$. In other words, we assume that any $\mathbf{p} \in \mathcal{B}$ follows a polynomial curve in the Cartesian workspace instead of the configuration space. By plugging $\mathbf{P}(t)$ into Equation 1, we obtain:

$$\int_{\mathbf{p}\in\mathcal{B}} \frac{\partial \mathbf{P}(\mathbf{q}_i)}{\partial \mathbf{q}_i}^T \left[\rho\ddot{\mathbf{P}}(i\Delta t) - \mathbf{f}(\mathbf{P}(\mathbf{q}_i))\right] d\mathbf{p} = 0 \quad \forall i = k+\alpha_1, \cdots, k+\alpha_{K-1}. \tag{6}$$

from which we can solve for $\mathbf{q}_* = \left(\mathbf{q}_{k+\alpha_1} \cdots \mathbf{q}_{k+\alpha_{K-1}}\right)$ simultaneously. Given a set of collocation points, we have completed our high-order formulation of PBAD. In practice, we follow [12] and use the roots of the Legendre polynomials as our collocation points. In other words, suppose $L_{K-2}(x)$ is the $(K-2)$th-order Legendre polynomial of the first kind, then $L_{K-2}(2\alpha_i - 1) = 0$ for $i = 1, \cdots, K-2$. Note that, although Equation 6 allows fully implicit integration without high-order derivatives, it does not have a corresponding energy form like Equation 5. However, we can still govern the convergence of a gradient-based optimizer using the following energy form:

$$\mathbf{E}(\mathbf{q}_*) = \sum_{i=k+\alpha_1}^{i=k+\alpha_{K-1}} \| \int_{\mathbf{p}\in\mathcal{B}} \frac{\partial \mathbf{P}(\mathbf{q}_i)}{\partial \mathbf{q}_i}^T \left[\rho\ddot{\mathbf{P}}(i\Delta t) - \mathbf{f}(\mathbf{P}(\mathbf{q}_i))\right] d\mathbf{p}\|^2, \tag{7}$$

where we solve for all the $\mathbf{q}_*$ from $\mathbf{q}_* = \mathbf{argmin}\ \mathbf{E}(\mathbf{q}_*)$. The high-order position-based collocation method (Equation 7) is more general than its second order counterpart (Equation 5) because $\mathbf{f}$ is not integrated to get $\mathbf{Q}$, allowing $\mathbf{f}$ to be non-conservative. Further, Equation 7 still allows simulation in a fully implicit manner without computing third-order derivatives.

5 Optimization Algorithm

In this section, we introduce the algorithm that performs numerical simulations under our PBAD formulation. During the timestep k, an implementation of our PBAD articulated body simulator calls a gradient-based optimizer to solve $\mathbf{q}_* = \mathbf{argmin}\ \mathbf{E}(\mathbf{q}_*)$, where $\mathbf{E}$ takes the form of Equation 5 for second-order collocation methods and conservative force models and $\mathbf{E}$ takes the form of Equation 7 for high-order collocation methods or non-conservative force models. Each timestep is an iterative algorithm whose complexity is not a constant. However, we can analyze the complexity of each iteration and profile the number of iterations empirically.

Our objective functions involve both inertial and potential energy terms. Since the concrete form of potential energy $\mathbf{Q}$ is application-dependent, we focus on the inertial

Algorithm 1 Compute $\mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)$, $\frac{\partial \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)}{\partial \mathbf{q}_b}$ using adjoint method within $\mathcal{O}(N)$. Here $\mathbf{A}$ is a 4×4 matrix. Note that Line 12 is $\mathcal{O}(1)$ because $\partial \mathbf{T}_{i-1}^{i}(\mathbf{q}_b)/\partial \mathbf{q}_b$ is non-zero only at entries corresponding to the ith link.

1: $\mathbf{T}^0(\mathbf{q}_a) \leftarrow \mathbf{I}, \mathbf{T}_0^1(\mathbf{q}_a) \leftarrow \mathbf{I}$
2: $\mathbf{T}^0(\mathbf{q}_b) \leftarrow \mathbf{I}, \mathbf{T}_0^1(\mathbf{q}_b) \leftarrow \mathbf{I}$
3: $\mathbf{I}(\mathbf{q}_a, \mathbf{q}_b) \leftarrow 0$
4: **for** $i = 1, \cdots, N$ **do** ▷ $\mathcal{O}(N)$ forward pass
5: $\quad \mathbf{T}^i(\mathbf{q}_a) \leftarrow \mathbf{T}^{i-1}(\mathbf{q}_a)\mathbf{T}_{i-1}^{i}(\mathbf{q}_a)$
6: $\quad \mathbf{T}^i(\mathbf{q}_b) \leftarrow \mathbf{T}^{i-1}(\mathbf{q}_b)\mathbf{T}_{i-1}^{i}(\mathbf{q}_b)$
7: $\quad \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b) \leftarrow \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b) + \mathbf{I}^i(\mathbf{q}_a, \mathbf{q}_b)$
8: **end for**
9: $\mathbf{A} \leftarrow 0, \frac{\partial \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)}{\partial \mathbf{q}_b} \leftarrow 0$ ▷ $\mathbf{A}$ is 4×4 matrix
10: **for** $i = N, \cdots, 1$ **do** ▷ $\mathcal{O}(N)$ backward pass
11: $\quad \mathbf{A} \leftarrow \mathbf{A} + \frac{\partial \mathbf{I}^i(\mathbf{q}_a, \mathbf{q}_b)}{\partial \mathbf{T}^i(\mathbf{q}_b)}$
12: $\quad \frac{\partial \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)}{\partial \mathbf{q}_b} \leftarrow \frac{\partial \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)}{\partial \mathbf{q}_b} + (\mathbf{T}^{i-1} \frac{\partial \mathbf{T}_{i-1}^{i}(\mathbf{q}_b)}{\partial \mathbf{q}_b}) : \mathbf{A}$ ▷ $\mathcal{O}(1)$
13: $\quad \mathbf{A} \leftarrow \mathbf{A}\mathbf{T}_{i-1}^{i}(\mathbf{q}_b)^T$
14: **end for**

term. Values and derivatives of most widely-used potential energies, such as the gravitational energy, can be evaluated in $\mathcal{O}(N)$ or $\mathcal{O}(N^2)$ and the complexity of algorithm is dominated by the inertial term. During each iteration, we evaluate the value and the partial derivatives of $\mathbf{E}$, which involve an integral over $\mathcal{B}$. We can evaluate this integral analytically. Note that $\mathbf{E}$ in Equation 5 is a linear combination of the following term:

$$\mathbf{I}(\mathbf{q}_a, \mathbf{q}_b) = \int_{\mathbf{p} \in \mathcal{B}} \mathbf{P}(\mathbf{q}_a)^T \mathbf{P}(\mathbf{q}_b) d\mathbf{p}, \tag{8}$$

with different (a, b)-pairs, as shown in our extended report [22]. Similarly, $\mathbf{E}$ in Equation 7 is a linear combination of Equation 8's partial derivatives. Equation 8 can be evaluated analytically as:

$$\mathbf{I}(\mathbf{q}_a, \mathbf{q}_b) = \left[\mathbf{T}(\mathbf{q}_a)^T \mathbf{T}(\mathbf{q}_b)\right] : \tilde{\mathbf{M}} - 1 \qquad \tilde{\mathbf{M}} \triangleq \int_{\mathbf{p} \in \mathcal{B}} \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix}^T d\mathbf{p},$$

where the integrals on the right-hand side ($\tilde{\mathbf{M}}$) is related to the mass and inertia tensor of $\mathcal{B}$ (not exactly the same). This matrix does not depend on $\mathbf{q}_{a,b}$ and can be precomputed. We have also used contract symbols such that $\mathbf{A} : \mathbf{B} = \mathbf{tr}\left[\mathbf{A}^T \mathbf{B}\right]$ and we have used homogeneous coordinates:

$$\mathbf{T}(\mathbf{q}) = \begin{pmatrix} \mathbf{R}(\mathbf{q}) & \mathbf{t}(\mathbf{q}) \\ & 1 \end{pmatrix}.$$

To solve $\mathbf{q}_*$, we consider two optimizers, LBFGS [19] and LM [18]. Given an objective function $\mathbf{E}(\mathbf{q}_*)$, each iteration of LBFGS computes a gradient, $\partial \mathbf{E}(\mathbf{q}_*)/\partial \mathbf{q}_*$, and updates $\mathbf{q}_*$ using a line-search along the gradient direction to ensure the decrease of $\mathbf{E}(\mathbf{q}_*)$. The cost of an LBFGS iteration is dominated by the computation of the gradient which takes $\mathcal{O}(N^2)$ in the case of Equation 7 and $\mathcal{O}(N)$ in the case of Equation 5.

Algorithm 2 Compute $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b{}^2}$ using adjoint method within $\mathcal{O}(N^2)$. Here $\mathbf{A}, \mathbf{B}$ are 4×4 matrices.

▷ Same forward pass as Algorithm 1.
$\mathbf{A} \leftarrow 0$, $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b{}^2} \leftarrow 0$
for $i = N, \cdots, 1$ **do** ▷ $\mathcal{O}(N^2)$ backward pass
  $\mathbf{A} \leftarrow \mathbf{A} + \frac{\partial \mathbf{I}^i(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{T}^i(\mathbf{q}_a)}$
  $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b{}^2} \leftarrow \frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b{}^2} + (\mathbf{T}^{i-1} \frac{\partial^2 \mathbf{T}^i_{i-1}(\mathbf{q}_b)}{\partial \mathbf{q}_b{}^2}) : \mathbf{A}$ ▷ $\mathcal{O}(1)$
  $\mathbf{B} \leftarrow \mathbf{A} \frac{\partial \mathbf{T}^i_{i-1}(\mathbf{q}_b)}{\partial \mathbf{q}_b}^T$
  for $j = i-1, \cdots, 1$ **do**
    $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b{}^2} \leftarrow \frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b{}^2} + (\mathbf{T}^{j-1} \frac{\partial \mathbf{T}^j_{j-1}(\mathbf{q}_b)}{\partial \mathbf{q}_b}) : \mathbf{B}$ ▷ $\mathcal{O}(1)$
    $\mathbf{B} \leftarrow \mathbf{B}\mathbf{T}^j_{j-1}(\mathbf{q}_b)^T$
  end for
  $\mathbf{A} \leftarrow \mathbf{A}\mathbf{T}^i_{i-1}(\mathbf{q}_b)^T$
end for

Optimizer \ Objective	Equation 5	Equation 7
LM	$\mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)$, $\frac{\partial \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b}$, $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_a \partial \mathbf{q}_b}$	$\frac{\partial \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b}$, $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b{}^2}$, $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_a \partial \mathbf{q}_b}$
LBFGS	$\mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)$, $\frac{\partial \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b}$	$\frac{\partial \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b}$, $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_b{}^2}$, $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_a \partial \mathbf{q}_b}$

Table 1: The variables required by different optimizers using different objective functions. Since high-order methods are more frequently used, we use Equation 7 as our objective function in most cases.

Unlike LBFGS, each iteration of LM computes a gradient, $\partial \mathbf{E}(\mathbf{q}_*)/\partial \mathbf{q}_*$, and a $\mathbf{J}^T\mathbf{J}$-approximate Hessian, $\mathbf{J}^T\mathbf{J}(\mathbf{E}(\mathbf{q}_*))$, and updates $\mathbf{q}_*$ using the Newton's method:

$$\mathbf{q}_* \leftarrow \mathbf{q}_* - \left[\mathbf{J}^T\mathbf{J}(\mathbf{E}(\mathbf{q}_*)) + \lambda \mathbf{I}\right]^{-1} \frac{\partial \mathbf{E}(\mathbf{q}_*)}{\partial \mathbf{q}_*},$$

where λ is tuned to ensure the decrease of $\mathbf{E}(\mathbf{q}_*)$. To compute the $\mathbf{J}^T\mathbf{J}$-approximate Hessian, our objective function must be a sum-of-squares, as is the case with Equation 7, or an integral-of-squares, as is the case with Equation 5. The cost of an LM iteration is dominated by solving a linear system of size $|\mathbf{q}| \times |\mathbf{q}|$, and is $\mathcal{O}(N^3)$ assuming a general linear solver.

The two optimization algorithms require different partial derivatives of $\mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)$ (up to second order) during each iteration, as illustrated in Table 1. The values and derivatives of $\mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)$ can be computed efficiently using the adjoint method, which results in algorithms similar to the forward/inverse dynamic algorithms in [8]. To introduce these algorithms, we need notations for multiple rigid bodies. We assume that we have N rigid bodies $\mathcal{B}^1, \cdots, \mathcal{B}^N$, where the parent of $\mathcal{B}^i$ is $\mathcal{B}^{i-1}$. We use superscripts to denote body indices. For each $\mathcal{B}^i$, we denote its transformation as $\mathbf{T}^i$ and we have $\mathbf{T}^i = \mathbf{T}^{i-1}{}^{\mathtt{i}}_{\mathtt{i-1}}$. With these notations, $\mathbf{I}(\mathbf{q}_a, \mathbf{q}_b) = \sum_i \mathbf{I}^i(\mathbf{q}_a, \mathbf{q}_b)$ becomes the summation of all the bodies. We compute $\mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)$ and $\partial \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)/\partial \mathbf{q}_b$ within $\mathcal{O}(N)$ using

Algorithm 3 Compute $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_a \partial \mathbf{q}_b}$ using adjoint method within $\mathcal{O}(N^2)$. Here $\mathbf{E}, \mathbf{F}, \mathbf{G}$ are $4\times4\times4\times4$ tensors, $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are 4×4 matrices, and we use double contraction such that $\mathbf{A} : \mathbf{E} : \mathbf{B} = \sum_{xyzw} [\mathbf{E}_{xyzw}\mathbf{B}_{wz}]$ and we have $\mathbf{A} : \mathbf{CED} : \mathbf{B} = \mathbf{AC} : \mathbf{E} : \mathbf{DB}$. Finally, we define $\mathbf{E}_{xyzw} = \partial^2 \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)/\partial \mathbf{T}_{xy}(\mathbf{q}_a)\partial \mathbf{T}_{zw}(\mathbf{q}_b)$.

▷ Same forward pass as Algorithm 1.
$\mathbf{E} \leftarrow 0$, $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_a \partial \mathbf{q}_b} \leftarrow 0$
for $i = N, \cdots, 1$ **do** ▷ $\mathcal{O}(N^2)$ backward pass
  $\mathbf{E} \leftarrow \mathbf{E} + \frac{\partial^2 \mathbf{I}^i(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{T}^i(\mathbf{q}_a)\partial \mathbf{T}^i(\mathbf{q}_b)}, \mathbf{F} \leftarrow \mathbf{E}, \mathbf{G} \leftarrow \mathbf{E}$
  for $j = i, \cdots, 1$ **do**
    $\frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_a \partial \mathbf{q}_b} \leftarrow \frac{\partial^2 \mathbf{I}(\mathbf{q}_a,\mathbf{q}_b)}{\partial \mathbf{q}_a \partial \mathbf{q}_b} +$
    $(\mathbf{T}^{i-1}(\mathbf{q}_a)\frac{\partial \mathbf{T}^i_{i-1}(\mathbf{q}_a)}{\partial \mathbf{q}_a}) : \mathbf{F} : (\mathbf{T}^{j-1}(\mathbf{q}_b)\frac{\partial \mathbf{T}^j_{j-1}(\mathbf{q}_b)}{\partial \mathbf{q}_b})^T +$
    $(\mathbf{T}^{j-1}(\mathbf{q}_a)\frac{\partial \mathbf{T}^j_{j-1}(\mathbf{q}_a)}{\partial \mathbf{q}_a}) : \mathbf{G} : (\mathbf{T}^{i-1}(\mathbf{q}_b)\frac{\partial \mathbf{T}^i_{i-1}(\mathbf{q}_b)}{\partial \mathbf{q}_b})^T$ ▷ $\mathcal{O}(1)$
    $\mathbf{F} \leftarrow \mathbf{F}\mathbf{T}^j_{j-1}(\mathbf{q}_b)^T, \mathbf{G} \leftarrow \mathbf{T}^j_{j-1}(\mathbf{q}_a)\mathbf{G}$
  end for
  $\mathbf{E} \leftarrow \mathbf{T}^i_{i-1}(\mathbf{q}_a)\mathbf{E}\mathbf{T}^i_{i-1}(\mathbf{q}_b)^T$
end for

Algorithm 1. We compute $\partial^2 \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)/\partial {\mathbf{q}_b}^2$ within $\mathcal{O}(N^2)$ using Algorithm 2 and we compute $\partial^2 \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)/\partial \mathbf{q}_a \partial \mathbf{q}_b$ within $\mathcal{O}(N^2)$ using Algorithm 3.

5.1 Algorithm Complexity of High-Order Collocation Methods

Compared with second-order collocation method that only optimizes $\mathbf{q}_{k+1}$, high-order collocation methods optimize multiple $\mathbf{q}$ in $\mathbf{q}_*$. In addition, we can only use Equation 7 as the objective function. The cost of each iteration of the optimization algorithm is dominated by computing the matrix $\partial^2 \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)/\partial \mathbf{q}_a \partial \mathbf{q}_b$. This matrix has size $|\mathbf{q}_*| \times |\mathbf{q}_*|$ and can be decomposed into $(K-2) \times (K-2)$ blocks of size $|\mathbf{q}| \times |\mathbf{q}|$. Each block is computed using Algorithm 3 and takes $\mathcal{O}(N^2)$, so that the computation of the entire $|\mathbf{q}_*| \times |\mathbf{q}_*|$ matrix takes $\mathcal{O}((K-2)^2 N^2)$.

5.2 GPU Parallelization

Our PBAD formulation is designed to be GPU-friendly. Simulating rigid bodies on a GPU has been previously studied [33, 32]. These methods formulate forward/inverse dynamics algorithms as GPU-scan operations. Our GPU implementation deviates from [33, 32] in two ways. First, our implementation is intended to be used for modeling predictive control [28] and reinforcement learning [6], where we need to generate multiple trajectories at once. This fact provides more opportunities for parallelism. Second, our algorithm is iterative and the number of iterations performed during each timestep tends to be different. In practice, an implementation that runs each timestep in a separate thread could result in starvation, where threads finishing early are waiting for other threads. As a result, we parallelize each iteration of an optimization instead of each timestep. This mechanism is illustrated in Figure 1.

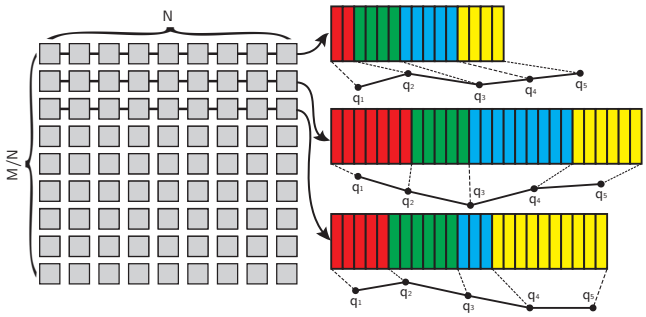

Fig. 1: An illustration of our GPU implementation. The GPU has M cores, each illustrated as a gray box on the left. We use a workgroup of N cores (black arrow) to simulate one trajectory. In this illustration, we compute 3 trajectories that each have 4 timesteps ($\mathbf{q}_2, \cdots, \mathbf{q}_5$). During each call to the GPU, instead of finishing the entire LM optimization, we compute just one iteration of the LM optimization (colored block on the right) so that all the workgroups are running the same computation and no starvation will happen. Different timesteps are illustrated using blocks of different colors. For example, it takes 2 iterations to compute $\mathbf{q}_2$ in the first trajectory and 7 iterations to compute $\mathbf{q}_2$ in the second trajectory (red block).

We choose the LM algorithm in our GPU implementation. Each iteration of LM involves computing $\frac{\partial \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)}{\partial \mathbf{q}_b}$, $\partial^2 \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)/\partial {\mathbf{q}_b}^2$, $\partial^2 \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)/\partial \mathbf{q}_a \partial \mathbf{q}_b$ according to Table 1 and then using a linear system solver. The serial computation of $\partial^2 \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)/\partial {\mathbf{q}_b}^2$ and $\partial^2 \mathbf{I}(\mathbf{q}_a, \mathbf{q}_b)/\partial \mathbf{q}_a \partial \mathbf{q}_b$ takes $\mathcal{O}(N^2)$, which can be costly. We introduce an additional fine-grain parallelism by using a GPU workgroup of N cores to reduce the complexity of computing the partial derivatives to $\mathcal{O}(N)$ using algorithms in our extended report [22]. With the same workgroup of N cores, the complexity of the GPU linear solver is reduced to $\mathcal{O}(N^2)$ using parallel Cholesky factorization [9]. As a result, a GPU with M cores can simulate $\lfloor M/N \rfloor$ trajectories in parallel and the complexity of each iteration is dominated by the linear solver, i.e. is $\mathcal{O}(N^2)$. This method is suitable for modern commodity GPUs with the number of cores $M \gg N$.

Finally, in Section 6, we will show that widely used external force models such as frictional contact forces and fluid drag forces can be formulated as integrable energies, $\mathbf{Q}$, whose values and derivatives can be computed in a similar manner to the inertial terms computed in this section. Putting them together, our method can be used to model the complex locomotion tasks in [6], such as swimming, walking, and jumping.

6 Results & Applications

6.1 Comparison

Throughout this section, we compare our formulation with conventional formulations based on Equation 2 and integrated using the Runge-Kutta method [4]. The same algorithm is implemented in [25, 29]. Note that the definition of order of integration is different for the Runge-Kutta method and the position-based collocation method. The position-based collocation method of order K has accuracy similar to that of the Runge-Kutta method of order $K - 1$. All experiments are performed on a single desktop machine with a 4-core CPU (Intel i7-4790 3.6G) and a 3584-core GPU (Nvidia Titan-X), i.e. $M = 3584$.

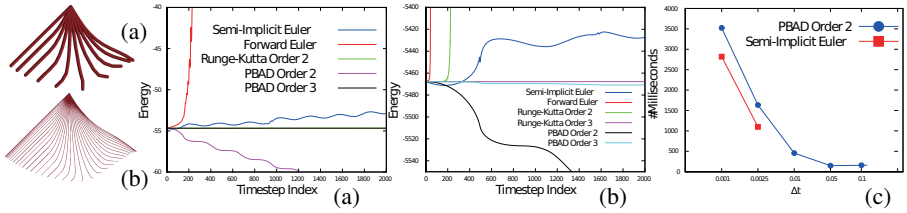

Fig. 2: (a): We plot the total kinetic+potential energy over time during a standard simulation of a 10-link chain that swings downward. Each joint of this chain is a 2-DOF ball joint so that this chain has 20-DOF. Forward Euler integrator for the Newton-Euler equation and semi-implicit Euler integrator are not stable. Being fully implicit, our second-order PBAD solver is stable but quickly loses energy. By increasing the order by one, both the second-order Runge-Kutta and our third-order PBAD solver preserve energy very well. (b): For the more challenging task of a 100-link chain (200-DOF) that swings downward, even the second-order Runge-Kutta method is not stable and we have to use the third-order Runge-Kutta method for better energy preservation. Our second-order PBAD solver is stable but quickly loses energy. Our third-order PBAD solver preserves energy very well. (c): We compare the total computational time for generating a 10s trajectory of a 10-link chain swinging down using a second-order collocation method for PBAD and a semi-implicit Euler integrator for a conventional formulation. PBAD is $1.5-2.1$ times slower at a small timestep size and up to 4 times faster at a large timestep size, such as 0.05s.

Energy Preservation: We compare the accuracy of time integrators for our PBAD formulation and conventional formulation. In Figure 2 (a), we plot the total kinetic+potential energy over time during a standard simulation of a 10-link chain (20-DOF) that swings downward (the same benchmark was used in [11]). The timestep size is 0.0025s. We can see that PBAD is very stable and continuously loses energy (Figure 2 (a) purple). In contrast, low-order explicit integrators such as forward Euler and semi-implicit Euler are not stable. For better accuracy, we can increase the order of integration by one, resulting in a much better performance in terms of energy preservation. In Figure 2 (b), we redo the experiment for a 100-link chain (200-DOF). This is more challenging and low-order explicit integrators are more unstable. The Runge-Kutta method for the Newton-Euler equation is stable at the third order. Although our second-order PBAD solver suffers a fast energy loss, increasing the order by one can significantly improve accuracy.

Timestep Size: In Figure 2 (c), we compare the total computational time for generating a 10s trajectory of a 10-link chain that swings downward using a second-order collocation method for PBAD and a semi-implicit Euler integrator for a conventional formulation. Each timestep of PBAD integration is costlier because multiple iterations of computations are needed to ensure the optimizer converges. For example, when we use timestep sizes of 0.001s and 0.0025s, the total computational time of the PBAD integrator is $1.5-2.1$ times that of the semi-implicit integrator. However, the PBAD integrator can be more efficient under a larger timestep size, while 0.0025s is the largest timestep size that works for the semi-implicit Euler integrator. At a timestep size of 0.05s, the total computational time of the PBAD integrator is 0.21 times that of the semi-implicit integrator, leading to a 4 times speedup.

Optimization Algorithm: We compare the performance of the two optimization algorithms (LM and LBFGS) on CPU. Figure 3 (a, b) shows that, LBFGS generally takes 10 times more iterations than LM. In addition, PBAD integration performed using

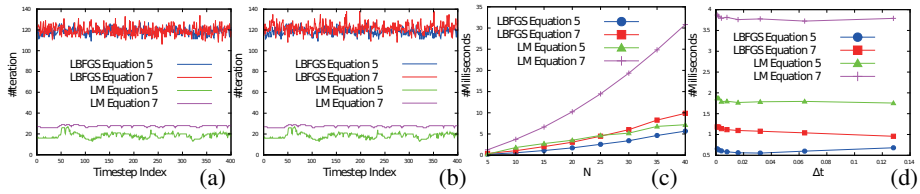

Fig. 3: We compare the performance of the two optimization algorithms (LM and LBFGS) during the simulation of a 10-link (20-DOF) (a) and a 40-link (80-DOF) chain (b) with a large timestep size of 0.05s. The number of iterations used by LBFGS is much larger than that used by LM, although each iteration of LBFGS is cheaper. In addition, the number of iterations is almost independent of the number of links, N. (c): We plot the average time to finish one step of the simulation against the number of links, N. LBFGS is comparable to LM in terms of computational time and the computational time grows almost linearly with N in the range of $N = 10 - 40$. (d): We plot the average time to finish one step of the simulation against the timestep size, Δt. PBAD can be used with very large timestep sizes and we tested $\Delta t = 0.001, 0.002, 0.004, 0.008, 0.016, 0.032, 0.064, 0.128$s. The computation time for each timestep is almost invariant to Δt.

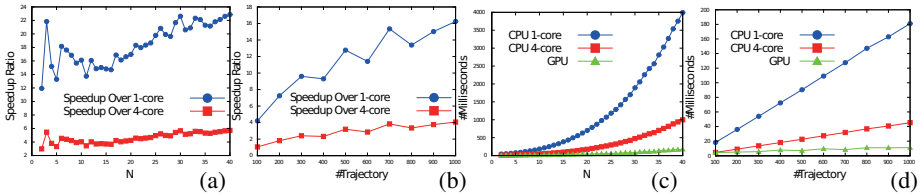

Fig. 4: We compare the performance of CPU and GPU in simulating a chain swinging benchmark. (a): We plot the speedup against the number of links, N. The speedup increases with N and the maximal speedup over a 4-core CPU is 6 times. (b): When $N = 10$, we plot the speedup against the number of trajectories. The speedup also increases with the number of trajectories and the maximal speedup is 4 times. (c): We plot the total computational time against the number of links, N, for generating 100 trajectories of 10 timesteps each. When $N = 40$, the 100 trajectories can be generated in less than 1s on GPU. (d): We plot the total computational time against the number of trajectories.

Equation 7 as the objective function will require more iterations to converge than when using Equation 5. Moreover, the numbers of iterations used by both algorithms are independent of the number of links, N. Considering the number of iterations as an invariant, the cost of LM grows as $\mathcal{O}(N^3)$ and the cost of LBFGS grows as $\mathcal{O}(N)$ on CPU. However, Figure 3 (c) shows that, when the number of links $N < 40$, the total computational time grows almost linearly. In particular, using LM to optimize Equation 7 is costlier than other choices. Figure 3 (c) also shows that the computation times of LBFGS and LM are comparable. Finally, PBAD can be used with very large timestep sizes, such as $\Delta t = 0.128$s, shown in Figure 3 (d), and the average time to compute each timestep is almost invariant to the timestep size. Therefore, large timestep sizes lead to a reduction in total computation time but they also lead to a higher rate of numerical dissipation.

GPU Acceleration: We compare the performance of our PBAD formulation on CPU and GPU. Our GPU implementation only provides acceleration when multiple trajectories are simulated simultaneously for different initial conditions, which is the case with many online/offline control algorithms such as [6, 28]. In Figure 4 (a, b),

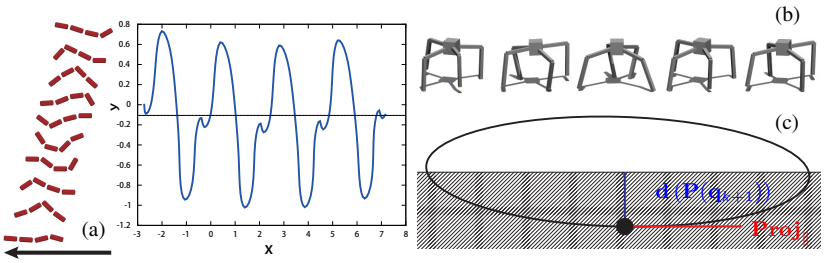

Fig. 5: (a): A 4-linked swimmer is trained to swim forward using CMA-ES. Some optimized swimming gaits and the locus of the center-of-mass are shown. (b): A 4-linked spider is trained to walk forward. Some optimized walking gaits are shown. The notations used by our frictional contact force model, Equation 9, are shown in (c). Our model is penalty-based. Both normal and tangential forces are related to the penetration depth $\mathbf{d}(\mathbf{P}(\mathbf{q}_{k+1}))$ (blue). The tangential forces (red) are modelled as a velocity damping term.

we show the speedup of our GPU implementation over a 4-core CPU. The speedup increases with both the number of links and the number of trajectories to be computed. The speedup is between 3-6 times. The total computational time for generating 100 trajectories of 10 timesteps each is plotted in Figure 4 (c). On GPU, generating these trajectories takes less than 1s for $N \leq 40$. Finally, in Figure 4 (d), we plot the total computational time against the number of trajectories to be computed when $N = 10$. Note that our GPU has 3584 cores and we can compute $\lfloor M/N \rfloor = 358$ trajectories in parallel. Therefore, when the number of trajectories increases from $100 - 300$, more GPU cores are used and the total computational time does not increase. Therefore, the green curve in Figure 4 (d) is almost flat.

6.2 External Force Models and Applications in Controller Optimization

To build a complete robot simulation system, it is essential to model external forces. In this last benchmark, we propose two force models that are compatible with PBAD formulations and parallel GPU implementations.

Our first force model considers a 4-linked chain (9-DOF, including a 6-DOF rigid transformation and 3 hinge joints) immersed underwater, which is under constant fluid drag forces. To model these drag forces, we use the following formulation of potential energy in Equation 5:

$$\mathbf{Q}_{drag}(\mathbf{P}(\mathbf{q}_{k+1}), \mathbf{P}(\mathbf{q}_k)) = D \| \frac{\mathbf{P}(\mathbf{q}_{k+1}) - \mathbf{P}(\mathbf{q}_k)}{\Delta t} \|^2,$$

where D is the drag force coefficient. This term minimizes the velocity of $\mathbf{p}$ and can be considered as a damping force model. An integral of $\mathbf{Q}_{drag}$ over $\mathcal{B}$ can be written as a linear combination of Equation 8 with different (a,b)-pairs as shown in our extended report [22] so that its value and derivatives can be computed using the techniques discussed in Section 5. We use CMA-ES [14] to optimize a controller for the swimmer to move forward and the results are shown in Figure 5 (a).

Our second force model considers a 4-legged spider (18-DOF, including a 6-DOF rigid transformation, 4 ball joints, and 4 hinge joints) trying to move forward on the ground, which is under frictional contact forces. A previous method [26] handles frictional contact forces using complementary conditions, which requires a sequential algorithm. Instead, we use a penalty-based frictional contact model by using the following

potential energy in Equation 5:

$$\mathbf{Q}_{contact}(\mathbf{P}(\mathbf{q}_{k+1}), \mathbf{P}(\mathbf{q}_k)) = D_1 \| \mathbf{d}\left(\mathbf{P}(\mathbf{q}_{k+1})\right) \|^2 + D_2 \| \mathbf{d}\left(\mathbf{P}(\mathbf{q}_{k+1})\right) \|^2 \| \mathbf{Proj}_{\|} \left(\frac{\mathbf{P}(\mathbf{q}_{k+1}) - \mathbf{P}(\mathbf{q}_k)}{\Delta t} \right) \|^2. \quad (9)$$

Here D_1 is the normal force penalty and $\mathbf{d}$ is the penetration depth, which is positive when $\mathbf{P}$ is inside obstacles and zero otherwise, as illustrated in Figure 5 (c). D_2 is the frictional force penalty and $\mathbf{Proj}_{\|}$ is the projection matrix to the tangential directions. The integral of $\mathbf{Q}_{contact}$ over $\mathcal{B}$ is replaced by a summation of a set of discrete contact points. The second term on the right-hand side of Equation 9 approximates frictional forces by requiring tangential velocities to be small when a point $\mathbf{P}$ is inside any of the obstacles. We use policy gradient method [23] to optimize a controller for the spider to move forward; the results are shown in Figure 5 (b).

7 Conclusion, Limitations & Future Work

In this paper, we present the PBAD reformulation of articulated body dynamics. Our reformulation casts the simulation as an energy minimization problem. As a result, off-the-shelf optimizers can be used to stably simulate articulated bodies under very large timestep sizes. Although each timestep of our algorithm requires more iterations than conventional methods, the overall speedup of our PBAD over conventional methods in various benchmarks is up to 4 times under very large timestep sizes, e.g., $\Delta t = 0.1$s. Furthermore, our approach is GPU friendly and can be easily parallelized. We observe an additional $3-6$ times speedup on a commodity GPU over a 4-core CPU. The parallel version of our PBAD solver can accelerate control algorithms such as model predictive control and reinforcement learning by simulating multiple trajectories simultaneously.

Our current formulation still has some limitations. First, numerical dissipation cannot totally be avoided, although we can reduce it using smaller timestep sizes or high-order collocation methods. Second, to recast the articulated body dynamics as an optimization problem and avoid high-order derivatives, we discretize the velocities in a Euclidean workspace, instead of using a Lie-Group structure [17]. As a result, our PBAD method can be less accurate compared with Lie-Group integrators. As part of future work, we would like to study various external force models that are compatible with the our PBAD formulation. A compatible force model should be stable under large timestep sizes. To this end, one method is to formulate the external force implicitly as a function of $\mathbf{q}_{k+1}$ Equation 9. However, the accuracy of these force models have not been well studied.

Acknowledgement

This research is supported in part by ARO grant W911NF16-1-0085, QNRF grant NPRP-5-995-2-415, and Intel.

Bibliography

[1] Ascher, U.M., Petzold, L.R.: Computer methods for ordinary differential equations and differential-algebraic equations, vol. 61. Siam (1998)

[2] Bender, J., Mller, M., Macklin, M.: Position-Based Simulation Methods in Computer Graphics. In: Zwicker, M., Soler, C. (eds.) EG 2015 - Tutorials. The Eurographics Association (2015)

[3] Bouaziz, S., Martin, S., Liu, T., Kavan, L., Pauly, M.: Projective dynamics: Fusing constraint projections for fast simulation. ACM Trans. Graph. 33(4), 154:1–154:11 (Jul 2014)

[4] Butcher, J.: Numerical Methods for Ordinary Differential Equations; 2nd ed. Wiley, Chichester (2008)

[5] Deul, C., Charrier, P., Bender, J.: Position-based rigid body dynamics. Computer Animation and Virtual Worlds 27(2), 103–112 (2014)

[6] Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P.: Benchmarking deep reinforcement learning for continuous control. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48. pp. 1329–1338. ICML'16, JMLR.org (2016)

[7] Featherstone, R.: A divide-and-conquer articulated-body algorithm for parallel o(log(n)) calculation of rigid-body dynamics. part 1: Basic algorithm. The International Journal of Robotics Research 18(9), 867–875 (1999)

[8] Featherstone, R.: Rigid Body Dynamics Algorithms. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)

[9] Galoppo, N., Govindaraju, N.K., Henson, M., Manocha, D.: Lu-gpu: Efficient algorithms for solving dense linear systems on graphics hardware. In: Proceedings of the 2005 ACM/IEEE conference on Supercomputing. p. 3. IEEE Computer Society (2005)

[10] Gast, T.F., Schroeder, C., Stomakhin, A., Jiang, C., Teran, J.M.: Optimization integrator for large time steps. IEEE Transactions on Visualization and Computer Graphics 21(10), 1103–1115 (Oct 2015)

[11] Gayle, R., Lin, M.C., Manocha, D.: Adaptive dynamics with efficient contact handling for articulated robots. In: Robotics: Science and systems. pp. 231–238 (2006)

[12] Guo, B.y., Wang, Z.q.: Legendre–gauss collocation methods for ordinary differential equations. Advances in Computational Mathematics 30(3), 249–280 (Apr 2009)

[13] Hahn, F., Martin, S., Thomaszewski, B., Sumner, R., Coros, S., Gross, M.: Rig-space physics. ACM Trans. Graph. 31(4), 72:1–72:8 (Jul 2012)

[14] Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In: Evolutionary Computation, 1996., Proceedings of IEEE International Conference on. pp. 312–317. IEEE (1996)

[15] Kobilarov, M., Crane, K., Desbrun, M.: Lie group integrators for animation and control of vehicles. ACM Trans. Graph. 28(2), 16:1–16:14 (May 2009)

[16] Krysl, P.: Dynamically equivalent implicit algorithms for the integration of rigid body rotations. Communications in Numerical Methods in Engineering 24(2), 141–156 (2008)

[17] Lee, J., Liu, C.K., Park, F.C., Srinivasa, S.S.: A linear-time variational integrator for multibody systems. arXiv preprint arXiv:1609.02898 (2016)

[18] Levenberg, K.: A method for the solution of certain non-linear problems in least squares. Quarterly of applied mathematics 2(2), 164–168 (1944)

[19] Liu, D.C., Nocedal, J.: On the limited memory bfgs method for large scale optimization. Mathematical Programming 45(1), 503–528 (Aug 1989)

[20] Murray, R.M., Li, Z., Sastry, S.S., Sastry, S.S.: A mathematical introduction to robotic manipulation. CRC press (1994)
[21] Narain, R., Overby, M., Brown, G.E.: ADMM $\supseteq$ projective dynamics: Fast simulation of general constitutive models. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 21–28. SCA '16, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2016)
[22] Pan, Z., Manocha, D.: Time integrating articulated body dynamics using position-based collocation method. In: arXiv:1709.04145 (2018)
[23] Peters, J., Schaal, S.: Policy gradient methods for robotics. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 2219–2225 (Oct 2006)
[24] Schroeder, C.A.: Coupled Simulation of Deformable Solids, Rigid Bodies, and Fluids with Surface Tension. Stanford University (2011)
[25] Smith, R.: Open dynamics engine (2008), http://www.ode.org/
[26] Stewart, D., Trinkle, J.C.: An implicit time-stepping scheme for rigid body dynamics with coulomb friction. In: Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on. vol. 1, pp. 162–169. IEEE (2000)
[27] Stilman, M.: Task constrained motion planning in robot joint space. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 3074–3081 (Oct 2007)
[28] Tassa, Y., Erez, T., Todorov, E.: Synthesis and stabilization of complex behaviors through online trajectory optimization. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 4906–4913 (Oct 2012)
[29] Todorov, E.: Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco. In: 2014 IEEE International Conference on Robotics and Automation (ICRA). pp. 6054–6061 (May 2014)
[30] Tomcin, R., Sibbing, D., Kobbelt, L.: Efficient enforcement of hard articulation constraints in the presence of closed loops and contacts. Computer Graphics Forum 33(2), 235–244 (2014)
[31] Weinstein, R., Teran, J., Fedkiw, R.: Dynamic simulation of articulated rigid bodies with contact and collision. IEEE Transactions on Visualization and Computer Graphics 12(3), 365–374 (May 2006)
[32] Yang, Y., Wu, Y., Pan, J.: Parallel dynamics computation using prefix sum operations. IEEE Robotics and Automation Letters 2(3), 1296–1303 (July 2017)
[33] Yang, Y., Wu, Y., Pan, J.: Unified gpu-parallelizable robot forward dynamics computation using band sparsity. IEEE Robotics and Automation Letters 3(1), 203–209 (Jan 2018)