An Efficient, Error-Bounded Approximation Algorithm for Simulating Quasi-Statics of Complex Linkages

Stephane Redon, Ming C. Lin

An Efficient, Error-Bounded Approximation Algorithm for Simulating Quasi-Statics of Complex Linkages

Stephane Redon and Ming C. Lin

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175, USA
{redon,lin}@cs.unc.edu

Abstract

Design and analysis of articulated mechanical structures, commonly referred to as linkages, is an integral part of any CAD/CAM system. The most common approaches formulate the problem as purely geometric in nature, though dynamics or quasi-statics of linkages should also be considered. Existing optimal algorithms that compute forward dynamics or quasi-statics of linkages have a linear runtime dependence on the number of joints in the linkage. When forces are applied to a linkage, these techniques need to compute the accelerations of all the joints and can become impractical for rapid prototyping of highly complex linkages with a large number of joints.

We introduce a novel algorithm that enables adaptive refinement of the forward quasi-statics simulation of complex linkages. This algorithm can cull away joints whose contribution to the overall linkage motion is below a given user-defined threshold, thus limiting the computation of the joint accelerations and forces to those that contribute most to the overall motion. It also allows a natural trade-off between the precision of the resulting simulation and the time required to compute it. We have implemented our algorithm and tested its performance on complex benchmarks consisting of up to 50,000 joints. We demonstrate that in some cases our algorithm is able to achieve up to two orders of magnitude of performance improvement, while providing a high-precision, error-bounded approximation of the quasi-statics of the simulated linkage.

Keywords: Design and Analysis, Simulation, Virtual Prototyping

1 Introduction

Modeling and design of articulated mechanical structures, commonly referred to as "linkages", is an integral part of any CAD/CAM system. The existing approaches typically focus on sizing mechanical constraints that guide the movement of the end-effector of the system within a workspace. The function of the device is usually prescribed as a set of positions reachable by the end-effector, as well as the mechanical constraints formed by joints that limit relative movement. The objective is to compute all the linkage configurations that can achieve a specific task. Formulated in this way the design problem is purely geometric in nature, often with an emphasis on systems consisting of lower numbers of degrees of freedom to reduce the complexity of the problem.

Complex linkages, such as articulated systems of many links, molecular chains, or articulated-body representations of deformable bodies, often require consideration of additional physical constraints and present new computational challenges. Forward simulation of articulated mechanical systems, *i.e.* computing the joint accelerations and forces depending on external forces and applied torques, is a fundamental issue that should be taken into account in designing highly complex linkages. Especially, a quasi-static analysis can provide insights into mechanisms at or close to equilibrium, or which have small dynamic effects (slow evolution or small-scale motions). However, such computations are frequently expensive to perform and sometimes ignored in rapid prototyping of highly complex mechanical linkages. Existing optimal algorithms that compute forward dynamics or quasi-statics of linkages have a linear runtime dependence on the number of joints in the linkage. When forces are applied to the linkage, these techniques need to compute the accelerations of *all* the joints, which can become impractical for rapid prototyping of complex linkages with a large number of joints.

In this paper, we present a novel algorithm for progressive refinement of the quasi-static simulation of complex linkages, which offers three key advantages:

- **Error-bounded Approximation:** Given any user-defined error threshold, our algorithm is able to determine an approximation of the linkage acceleration within the required error bound.
- **Progressive Refinement:** Our algorithm operates by progressively increasing the quality of the approximate linkage acceleration, thus providing a trade-off between the precision of the approximation and the time required to compute it.
- **Efficiency:** Our algorithm determines the approximate acceleration by computing only *some* of the joints' accelerations and associated coefficients, resulting in a significant performance improvement when the ratio on the number of processed joints to the total number of joints is small, *i.e.* when the required precision is low, or when the forces applied to the linkage have a local effect only on the linkage motion.

To the best of our knowledge, this algorithm is the first technique that offers a method to progressively refine the computation of the quasi-statics of a linkage while guaranteeing the precision of the calculations. We have implemented our algorithm in C++ and tested it on some complex mechanical systems consisting of up to 50,000 links. In some cases, we were able to achieve up to two orders of magnitude of performance improvement over an existing linear-time algorithm, while providing a high-precision, error-bounded approximation of the quasi-statics of the simulated linkage (see an example in Figure 1).

The rest of the paper is organized as follows. Section 2 presents a

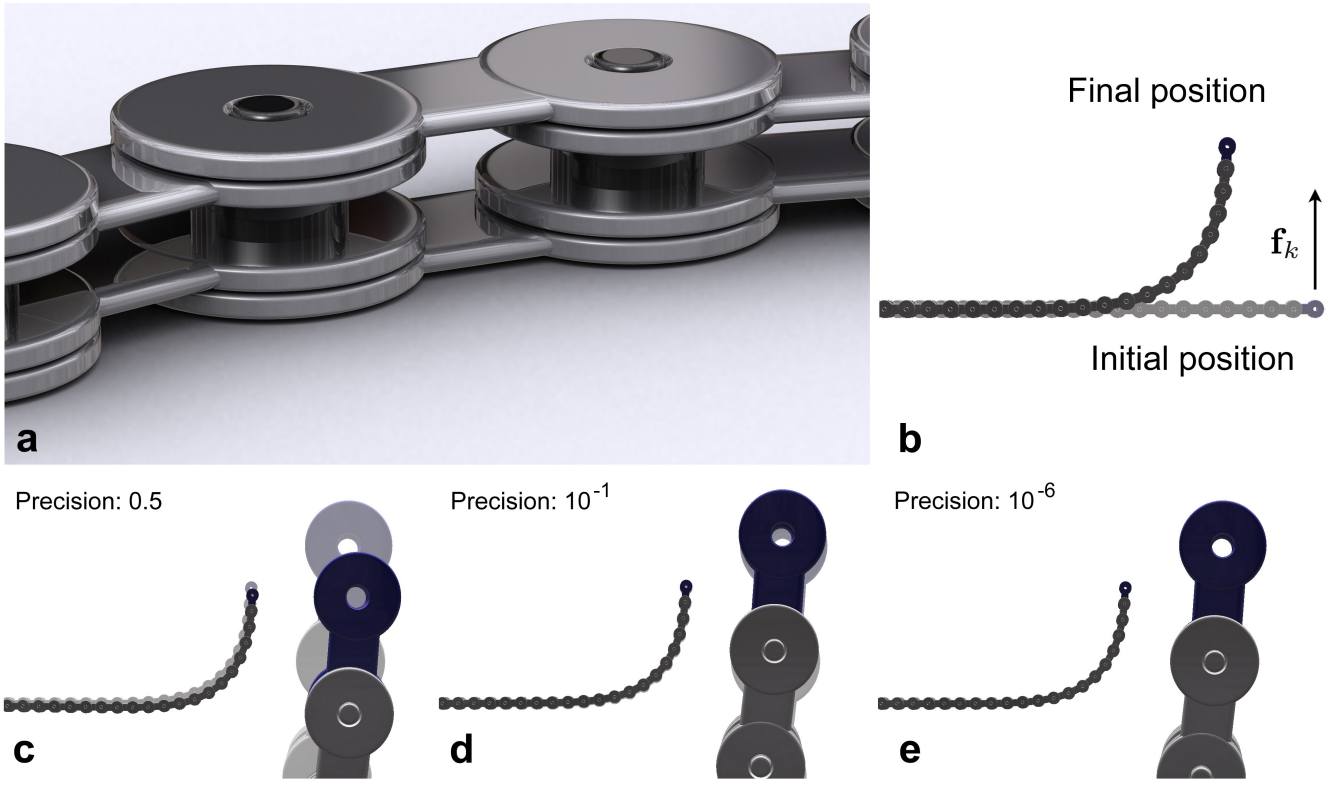

Figure 1: **Error-bounded approximation of a complex linkage's quasi-statics. (a)** *A close-up view of the model.* **(b)** *The initial and final positions of the linkage.* **(c)-(e)** *The simulation results of our algorithm vs. the DCA for varying degrees of precision, with both the view of the chain and a close-up of the end position.*

brief survey of related work on rigid body systems. Section 3 gives an overview of the basic mathematical notation and representations that are fundamental to our approach. Section 4 demonstrates how we obtain an approximate acceleration of a linkage. Section 5 shows how, using the hierarchical reference frames representation we introduce, the linkage state update can be restricted to some coefficients only. Section 6 presents some results obtained with our new approach, and Section 7 concludes by suggesting some potential research directions.

2 Related Work

The computation of the kinematics, quasi-statics, or dynamics of articulated bodies has been a subject of study for many years (*cf* [Hooker and Margulies 1965] for example). We only provide a brief overview here and refer the readers to [Featherstone and Orin 2000] for a recent state-of-the-art survey.

Essentially, an important step in the development of simulation algorithms has been the introduction of methods which have a linear complexity in the number of bodies in the multibody system. Some of these linear methods [Bae and Haug 1987; Brandl et al. 1986; Featherstone 1987; Hollerbach 1980; McMillan and Orin 1995], which rely on a recursive formulation of the motion equations, have now been known for more than two decades. In parallel, several authors have contributed to simplify the motion equations by developing new notations and formulations. Some of these include the spatial notation [Featherstone 1987], and its new version [Featherstone 1999a; Featherstone 1999b], the spatial operator algebra [Rodriguez et al. 1991], as well as Lie-Group formulations [Mueller and Maisser 2003]. More recently, several researchers have proposed parallel algorithms to compute the forward dynamics of articulated bodies using multiple processors [Fijany et al. 1995; Featherstone 1999a; Featherstone 1999b].

To our knowledge, no algorithm has been proposed for obtaining an error-bounded approximation of the quasi-statics of a multibody system. All existing techniques compute the accelerations of *all* joints in the multibody system. They cannot detect nor take advantage of the situations where the interaction between the environment and the linkage is localized, *i.e.* the situations where computing only *some* of the joint accelerations is enough to obtain a useful, error-bounded approximation of the linkage motion.

3 Preliminaries

In this section, we first provide an overview of the divide-and-conquer algorithm (DCA) introduced by Featherstone [Featherstone 1999a; Featherstone 1999b], upon which our algorithm is built. We restrict the presentation to the *quasi-statics* case, where the body velocities are zero at each instant. Then, we give an overview of our approach.

3.1 Overview of the DCA

Consider a system of rigid bodies connected by joints. Assume the initial body positions and the acceleration-independent forces are known. The acceleration-independent forces may include gravity, applied external forces, and active joint forces produced by actuators. The *forward quasi-statics problem* is to determine the body accelerations and the kinematic constraint forces based on the initial body positions and the acceleration-independent forces, assuming the velocities of the rigid bodies are zero at each instant.

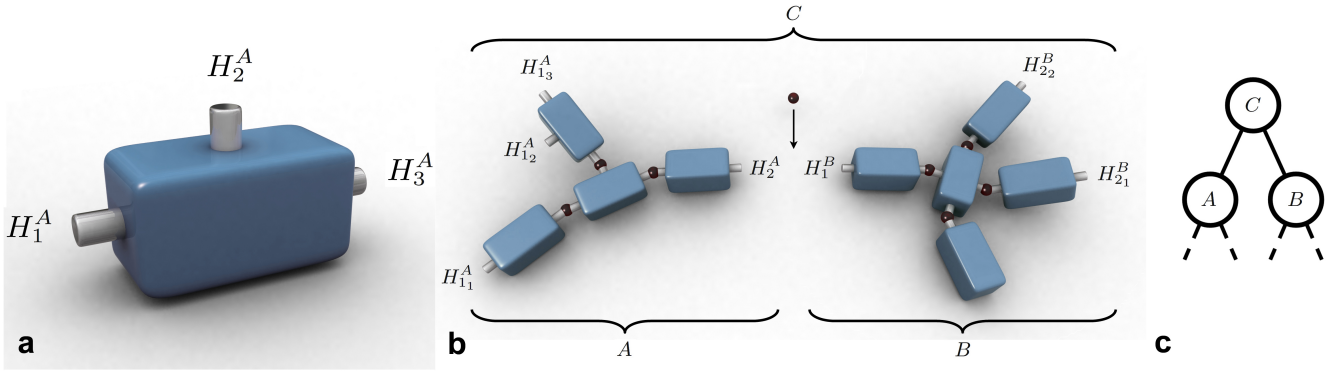

Figure 2: **An articulated body C is formed by connecting two articulated bodies A and B with a joint. (a)** *A rigid body with three handles.* **(b)** *The subassembly A has four handles $H_{1_1}^A$, $H_{1_2}^A$, $H_{1_3}^A$ and H_2^A, while the subassembly B has three handles H_1^B, $H_{2_1}^B$ and $H_{2_2}^B$. The handles H_2^A and H_1^B are consumed during the assembly operation, and only five handles remain in C.* **(c)** *The* assembly tree *describes the hierarchy of articulated bodies.*

To solve this problem (and the more general forward *dynamics* problem, where the body velocities are non zero), Featherstone [Featherstone 1987] defines an articulated body as a system of rigid bodies with a set of *handles*, *i.e.* locations attached to some of the rigid bodies where external forces may be applied. Figure 2(a) shows one rigid body with three handles which, although abstract in nature, have been materialized for clarity. Note that the number of handles m in the articulated body is *independent* of the number of bodies or joints in the articulated body. That is, the handles are a set of arbitrary locations on the articulated body that are used to simplify the equations of motion by focusing on some of the rigid bodies only. More specifically, Featherstone [Featherstone 1987] shows that the handles accelerations can be expressed as affine functions of the forces applied to the handles according to the following *articulated-body equation*:

$$\begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_m \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}_1 & \mathbf{\Phi}_{12} & \cdots & \mathbf{\Phi}_{1m} \\ \mathbf{\Phi}_{21} & \mathbf{\Phi}_2 & \cdots & \mathbf{\Phi}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{\Phi}_{m1} & \mathbf{\Phi}_{m2} & \cdots & \mathbf{\Phi}_m \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_m \end{bmatrix} + \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_m \end{bmatrix} \tag{1}$$

where $\mathbf{a}_i$ is the 6×1 spatial acceleration of handle i, $\mathbf{f}_i$ is the 6×1 spatial force applied to handle i, $\mathbf{b}_i$ is the 6×1 bias acceleration of handle i (the acceleration handle i would have if all the external forces were zero), $\mathbf{\Phi}_i$ is the 6×6 inverse articulated-body inertia of handle i, and $\mathbf{\Phi}_{ij}$ is the 6×6 cross-coupling inverse inertia between handles i and j.

In the DCA [Featherstone 1999a; Featherstone 1999b], an articulated body is recursively defined by assembling two (floating) articulated bodies together. Thus, the complete articulated body is described by a binary *assembly tree*, which specifies how articulated bodies (subassemblies) are repeatedly connected using joints. The leaf nodes in the assembly tree are rigid bodies, and the root node is a special node with only one child, which describes how the complete articulated body relates to the world coordinate system, potentially connecting the articulated body to a fixed base. The root node is the only node where an articulated body can be attached to a fixed base.

The DCA involves four *passes* on the assembly tree, *i.e.* four complete traversals of the assembly tree. Two preliminary passes compute the new body positions from the current values of the joint positions and accelerations, and the coordinate transformation matrices needed by the algorithm, while the next two passes perform the actual dynamics computations:

- **Main pass:** The main pass of the DCA determines the coefficients in the articulated-body equations (1) for each node (*i.e.* articulated body) in the assembly tree, from the leaf nodes to the root node.
- **Back-substitution pass:** Once the articulated-body equations are computed, the back-substitution pass computes the joint accelerations and the kinematic constraint forces, from the root node to the leaf nodes.

3.1.1 Main pass

Assume an articulated body C is formed by assembling two articulated bodies A and B. We call the joint used to assemble A and B the *principal joint* of C. Assume A contains $m+1$ handles, denoted by $H_{1_1}^A, \ldots, H_{1_m}^A, H_2^A$, and B contains $n+1$ handles, denoted by H_1^B, $H_{2_1}^B, \ldots, H_{2_n}^B$. Suppose articulated body C is formed by connecting handle H_2^A in A and handle H_1^B in B. Figure 2(b)-(c) shows an example of such an assembly operation. The goal of the main pass is to obtain the articulated-body equation of C from the articulated-body equations of A and B.

The articulated-body equation of A is:

$$\begin{bmatrix} \mathbf{a}_{1_1}^A \\ \vdots \\ \mathbf{a}_{1_m}^A \\ \mathbf{a}_2^A \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}_{1_{11}}^A & \cdots & \mathbf{\Phi}_{1_{1m}}^A & \mathbf{\Phi}_{1_12}^A \\ \vdots & \ddots & \vdots & \vdots \\ \mathbf{\Phi}_{1_{m1}}^A & \cdots & \mathbf{\Phi}_{1_{mm}}^A & \mathbf{\Phi}_{1_m2}^A \\ \mathbf{\Phi}_{21_1}^A & \cdots & \mathbf{\Phi}_{21_m}^A & \mathbf{\Phi}_2^A \end{bmatrix} \begin{bmatrix} \mathbf{f}_{1_1}^A \\ \vdots \\ \mathbf{f}_{1_m}^A \\ \mathbf{f}_2^A \end{bmatrix} + \begin{bmatrix} \mathbf{b}_{1_1}^A \\ \vdots \\ \mathbf{b}_{1_m}^A \\ \mathbf{b}_2^A \end{bmatrix}.$$

This can be rewritten such as to *single out* the handle H_2^A:

$$\begin{bmatrix} \mathbf{a}_1^A \\ \mathbf{a}_2^A \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}_1^A & \mathbf{\Phi}_{12}^A \\ \mathbf{\Phi}_{21}^A & \mathbf{\Phi}_2^A \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^A \end{bmatrix} + \begin{bmatrix} \mathbf{b}_1^A \\ \mathbf{b}_2^A \end{bmatrix},$$

where $\mathbf{\Phi}_1^A = mat(\mathbf{\Phi}_{1_{ij}}^A)$ is a $6m \times 6m$ *composite* matrix of cross-coupling and inverse articulated-body inertias involving handles

$H^A_{1_1}, \ldots, H^A_{1_m}$, $\mathbf{\Phi}^A_{12} = (\mathbf{\Phi}^A_{21})^T = col(\mathbf{\Phi}^A_{1_i2})$ is a $6m \times 6$ composite matrix of cross-coupling inverse inertias between handles $H^A_{1_1}, \ldots, H^A_{1_m}$ and H^A_2, $\mathbf{\Phi}^A_2$ is the 6×6 inverse articulated-body inertia of handle H^A_2, $\mathbf{f}^A_1$ is a $6m \times 1$ composite vector of external forces applied to the handles $H^A_{1_1}, \ldots, H^A_{1_m}$, $\mathbf{f}^A_2$ is the 6×1 external force applied to handle H^A_2, $\mathbf{b}^A_1$ is a $6m \times 1$ composite vector of bias accelerations of the handles $H^A_{1_1}, \ldots, H^A_{1_m}$, and $\mathbf{b}^A_2$ is the 6×1 bias acceleration of handle H^A_2.

Similarly, the articulated-body equation of B can be written:

$$\begin{bmatrix} \mathbf{a}^B_1 \\ \mathbf{a}^B_2 \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}^B_1 & \mathbf{\Phi}^B_{12} \\ \mathbf{\Phi}^B_{21} & \mathbf{\Phi}^B_2 \end{bmatrix} \begin{bmatrix} \mathbf{f}^B_1 \\ \mathbf{f}^B_2 \end{bmatrix} + \begin{bmatrix} \mathbf{b}^B_1 \\ \mathbf{b}^B_2 \end{bmatrix},$$

where handle H^B_1 is singled out, and the quantities involving the other handles (and potentially the handle H^B_1) are composite vectors or matrices.

Finally, the articulated-body equation of C can be written:

$$\begin{bmatrix} \mathbf{a}^A_1 \\ \mathbf{a}^B_2 \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}^C_1 & \mathbf{\Phi}^C_{12} \\ \mathbf{\Phi}^C_{21} & \mathbf{\Phi}^C_2 \end{bmatrix} \begin{bmatrix} \mathbf{f}^A_1 \\ \mathbf{f}^B_2 \end{bmatrix} + \begin{bmatrix} \mathbf{b}^C_1 \\ \mathbf{b}^C_2 \end{bmatrix},$$

where *all* quantities are composite vectors or matrices involving the remaining $m+n$ handles $H^A_{1_1}, \ldots, H^A_{1_m}$ and $H^B_{2_1}, \ldots, H^B_{2_n}$. Note that the handles H^A_2 and H^B_1 are consumed by the assembly operation, and the forces $\mathbf{f}^A_2$ and $\mathbf{f}^B_1$, that were external forces with respect to the articulated bodies A and B, are now *internal* kinematic constraint forces for the articulated body C.

Assume $\mathbf{S}$ denotes the $6 \times d_J$ joint's motion subspace, where d_J is the number of degrees of freedom of the joint. A typical value for a revolute joint in link coordinates is $\mathbf{S} = (0,0,1,0,0,0)^T$. Assume $\mathbf{Q}$ denotes the $d_J \times 1$ vector of active joint forces. Setting

$$\begin{aligned} \mathbf{V} &= (\mathbf{\Phi}^A_2 + \mathbf{\Phi}^B_1)^{-1} \\ \mathbf{W} &= \mathbf{V} - \mathbf{V}\mathbf{S}(\mathbf{S}^T\mathbf{V}\mathbf{S})^{-1}\mathbf{S}^T\mathbf{V} \\ \boldsymbol{\beta} &= \mathbf{b}^A_2 - \mathbf{b}^B_1 \\ \boldsymbol{\gamma} &= \mathbf{W}\boldsymbol{\beta} + \mathbf{V}\mathbf{S}(\mathbf{S}^T\mathbf{V}\mathbf{S})^{-1}\mathbf{Q}, \end{aligned}$$

it can be shown (*cf* [Featherstone 1999a; Featherstone 1999b]) that the articulated-body equation of C can be obtained from those of A and B:

$$\begin{aligned} \mathbf{\Phi}^C_1 &= \mathbf{\Phi}^A_1 - \mathbf{\Phi}^A_{12}\mathbf{W}\mathbf{\Phi}^A_{21} \\ \mathbf{\Phi}^C_2 &= \mathbf{\Phi}^B_2 - \mathbf{\Phi}^B_{21}\mathbf{W}\mathbf{\Phi}^B_{12} \\ \mathbf{\Phi}^C_{21} &= \mathbf{\Phi}^B_{21}\mathbf{W}\mathbf{\Phi}^A_{21} = (\mathbf{\Phi}^C_{12})^T \\ \\ \mathbf{b}^C_1 &= \mathbf{b}^A_1 - \mathbf{\Phi}^A_{12}\boldsymbol{\gamma} \\ \mathbf{b}^C_2 &= \mathbf{b}^B_2 + \mathbf{\Phi}^B_{21}\boldsymbol{\gamma} \end{aligned} \tag{2}$$

For leaf nodes, which represent rigid bodies, the articulated-body equation coefficients in the case of quasi-statics are:

$$\begin{aligned} \mathbf{\Phi}_i &= \mathbf{\Phi}_{ij} = \mathbf{I}^{-1} \\ \mathbf{b}_i &= \mathbf{I}^{-1}\mathbf{f}_k, \end{aligned} \tag{3}$$

where $\mathbf{I}$ is the rigid body spatial inertia, and $\mathbf{f}_k$ is an acceleration-independent external force applied to the rigid body (any number of handles can be attached to a rigid body). Since we assume the evolution of the rigid-body system is quasi-static, the velocity-dependent term in the bias acceleration is zero.

3.1.2 Back-substitution pass

Once the coefficients of the articulated-body equations have been updated in the main pass, the back-substitution pass determines the joint accelerations and the kinematic constraint forces, from the root node to the leaf nodes.

Based on the values of the forces $\mathbf{f}^A_1$ and $\mathbf{f}^B_2$ computed when processing the parent nodes, the $d_J \times 1$ joint acceleration vector is:

$$\ddot{\mathbf{q}} = (\mathbf{S}^T\mathbf{V}\mathbf{S})^{-1}(\mathbf{Q} - \mathbf{S}^T\mathbf{V}(\mathbf{\Phi}^A_{21}\mathbf{f}^A_1 - \mathbf{\Phi}^B_{12}\mathbf{f}^B_2 + \boldsymbol{\beta})), \tag{4}$$

and the 6×1 kinematic constraint forces are:

$$\begin{aligned} \mathbf{f}^B_1 &= \mathbf{W}\mathbf{\Phi}^A_{21}\mathbf{f}^A_1 - \mathbf{W}\mathbf{\Phi}^B_{12}\mathbf{f}^B_2 + \boldsymbol{\gamma} \\ \mathbf{f}^A_2 &= -\mathbf{f}^B_1 \end{aligned} \tag{5}$$

The root node calculations take two possible forms, depending on the existence of a connection between the articulated body and a fixed base. Since the root node describes the placement of the complete articulated body in the world frame, it has only one articulated-body child B. The external force $\mathbf{f}^B_2$ is assumed to be known (usually $\mathbf{f}^B_2 = \mathbf{0}$, since all acceleration-independent forces have already been accounted for in the bias accelerations).

If the articulated body has a floating base, then $\mathbf{f}^B_1 = \mathbf{0}$ and $\mathbf{a}^B_1 = \mathbf{\Phi}^B_{12}\mathbf{f}^B_2 + \mathbf{b}^B_1$. However, if the articulated body is attached to a fixed base through a joint, the joint acceleration is:

$$\ddot{\mathbf{q}} = (\mathbf{S}^T(\mathbf{\Phi}^B_1)^{-1}\mathbf{S})^{-1}(\mathbf{Q} + \mathbf{S}^T(\mathbf{\Phi}^B_1)^{-1}(\mathbf{\Phi}^B_{12}\mathbf{f}^B_2 + \mathbf{b}^B_1)), \tag{6}$$

and the kinematic constraint force is:

$$\mathbf{f}^B_1 = (\mathbf{\Phi}^B_1)^{-1}(\mathbf{S}\ddot{\mathbf{q}} - \mathbf{\Phi}^B_{12}\mathbf{f}^B_2 - \mathbf{b}^B_1). \tag{7}$$

Again, the velocity-dependent terms have been removed. Note that it is possible to simulate gravity by substituting $\mathbf{b}^B_1 - \mathbf{a}_g$ for $\mathbf{b}^B_1$, which suppresses the need to account for gravity through the action of an external force upon all leaf nodes.

3.2 Overview of Our Algorithm

Assume the initial acceleration-independent forces, the initial body positions and velocities, the initial coordinates transformation matrices, and the initial articulated-body equations are known. We can view the DCA as a two-step process which is iterated to simulate the quasi-statics or dynamics of a linkage:

- **Joint accelerations and forces update:** The joint accelerations and kinematic constraint forces are computed based on the current articulated-body equations.
- **Linkage state update:** The body positions (and velocities in the dynamics case), the coordinates transformation matrices, and the articulated-body equations are updated based on the joint accelerations computed in the first step and the new acceleration-independent forces.

When run on a single processor, the DCA has a linear complexity in the number of joints in the complete articulated body, since each pass in the algorithm needs to traverse the complete assembly tree. There are no mechanisms in the DCA which allow to *predict* the joints with largest accelerations, *i.e.* to determine them without having to compute *all* joint accelerations.

In order to avoid this linear complexity, our algorithm extends the original DCA by modifying each of its two steps in the following way:

- **Approximate linkage acceleration:** We introduce a class of *approximate linkage accelerations*. Given any user-defined error threshold, we are able to determine an approximation of the linkage acceleration, while guaranteeing that the error

caused by the approximation is below the given user-defined threshold. We introduce and calculate some *total acceleration equations*, which help us order the computations in the back-substitution pass and cull away some joints whose contribution to the linkage acceleration is found to be insignificant. This allows us to compute an approximate acceleration which is provably error-bounded, without having to complete the back-substitution pass and compute all the joint accelerations.

- **Simplified linkage state update:** We introduce a *hierarchical reference frames* representation in which we express the state of the linkage and the total acceleration equations. We show that, using this representation, only *some* of the quantities involved in the second step have to be updated, thereby avoiding a linear-time update.

4 Acceleration Approximation

In this section, we introduce a method which allows us to adaptively refine the computation of the joint accelerations and (kinematic constraint) forces, and obtain an approximation of the linkage acceleration which is provably error-bounded.

The key to our approach is the demonstration that it is possible to compute the *total acceleration* of an articulated body, *i.e.* the sum of the (squared) norms of the joint accelerations of all the joints in the articulated body, *without computing each joint acceleration separately*. We use the total acceleration during the back-substitution pass of the DCA to progressively refine the computation of the quasi-statics of the linkage, and stop the refinement when a sufficient precision has been obtained.

As can be seen in Section 3.1, the DCA relies on a bottom-up pass (the main pass), which calculates the articulated-body equations of all the nodes in the assembly tree, to subsequently compute the joint accelerations and forces during a top-down pass (the back-substitution pass). We demonstrate that, similarly, it is possible to calculate some *total acceleration equations* in a bottom-up pass, which allow us to compute the total accelerations in a top-down fashion.

4.1 Definitions

Let C be a linkage, and let $\mathbf{q}^C$ denote the composite vector of the positions of the N_C joints in C:

$$\mathbf{q}^C = \begin{pmatrix} \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_{N_C} \end{pmatrix}.$$

Each joint position $\mathbf{q}_i$ is a $d_{J_i} \times 1$ vector, where d_{J_i} is the number of degrees of freedom for joint i in C, and $\mathbf{q}^C$ is a $d_{J_C} \times 1$ vector, where $d_{J_C} = \sum_{i=1}^{N_C} d_{J_i}$ is the total number of degrees of freedom in C. Similarly, the composite vector $\ddot{\mathbf{q}}^C$ contains the accelerations of the N_C joints in C.

In this paper, we propose to define an approximation of the linkage acceleration as a partial computation of the joint accelerations. Specifically, we define an *approximate acceleration* in the following way.

Definition 1 (Approximate Acceleration)**.** *Consider an articulated body C and its actual acceleration $\ddot{\mathbf{q}}^C$. An* approximate acceleration *$\mathbf{s}^C$ of the exact linkage acceleration $\ddot{\mathbf{q}}^C$ is a composite vector in which some block components are equal to the actual joint accelerations in C, and the other block components are zero. All blocks in $\mathbf{s}^C$ have the same dimension as their counterpart in $\ddot{\mathbf{q}}^C$, and there exists a partition of $\{1,\dots,N_C\}$ into two subsets $\mathscr{E}$ and $\mathscr{Z}$ such as:*

$$\mathbf{s}^C = \begin{pmatrix} \mathbf{s}_1 \\ \vdots \\ \mathbf{s}_{N_C} \end{pmatrix} \quad \textit{and} \quad \begin{cases} \mathbf{s}_i^C = \ddot{\mathbf{q}}_i, & i \in \mathscr{E} \\ \mathbf{s}_j^C = \mathbf{0}, & j \in \mathscr{Z} \end{cases}.$$

The subset $\mathscr{E}$ is the set of indices of the joints whose acceleration has been computed exactly*, while the subset $\mathscr{Z}$ is the set of indices of the joints whose acceleration has been assumed to be* zero*, resulting in a potential approximation of the actual linkage acceleration $\ddot{\mathbf{q}}^C$.*

In order to measure the quality of an approximate acceleration, we use several classical error measures.

Definition 2 (Acceleration Error Measures)**.** *Consider an articulated body C, its actual acceleration $\ddot{\mathbf{q}}^C$, and an approximation $\mathbf{s}^C$ of its acceleration. The* acceleration error $\varepsilon(C)$ *will alternatively denote:*

- *The absolute linkage acceleration error:*

$$\varepsilon(C) = ||\mathbf{s}^C - \ddot{\mathbf{q}}^C||_2 = \sqrt{\sum_{j \in \mathscr{Z}} \ddot{\mathbf{q}}_j^T \ddot{\mathbf{q}}_j}$$

- *The relative linkage acceleration error:*

$$\varepsilon(C) = \frac{||\mathbf{s}^C - \ddot{\mathbf{q}}^C||_2}{||\ddot{\mathbf{q}}^C||_2} = \sqrt{\frac{\sum_{j \in \mathscr{Z}} \ddot{\mathbf{q}}_j^T \ddot{\mathbf{q}}_j}{\sum_{j=1}^{N_C} \ddot{\mathbf{q}}_j^T \ddot{\mathbf{q}}_j}}$$

- *The absolute joint acceleration error:*

$$\varepsilon(C) = \max_{j \in \mathscr{Z}} ||\ddot{\mathbf{q}}_j||_2$$

- *The relative joint acceleration error:*

$$\varepsilon(C) = \frac{\max_{j \in \mathscr{Z}} ||\ddot{\mathbf{q}}_j||_2}{\max_{j=1\dots N_C} ||\ddot{\mathbf{q}}_j||_2}$$

In the rest of this section we show that, given any user-defined maximum acceleration error ε_{max}, and whichever error measure $\varepsilon(C)$ we choose, we are able to find an approximate acceleration $\mathbf{s}^C$ which is sufficiently close to the actual acceleration of the linkage, *i.e.* such as $\varepsilon(C) \leqslant \varepsilon_{max}$, *without having to compute each joint acceleration in C.*

We obtain an approximate linkage acceleration by performing a *partial* back-substitution pass, thereby computing the exact value of *some* joint accelerations, and implicitly assuming that the other joint accelerations are zero. We show that it is possible to *not* complete the back-substitution pass and still determine an approximate linkage acceleration within the required error bound.

The key to our approach is the use of the *total acceleration* of a linkage, defined as follows.

Definition 3 (Total Acceleration)**.** *Consider an articulated body C with actual acceleration $\ddot{\mathbf{q}}^C$. The* total acceleration *$\mathscr{A}(C)$ of C is the*

sum of the squared L_2-norms of the accelerations of the N_C joints in C:

$$\mathscr{A}(C) = ||\ddot{\mathbf{q}}^C||_2^2 = \sum_{i=1}^{N_C} \ddot{\mathbf{q}}_i^T \ddot{\mathbf{q}}_i.$$

Alternatively, we will use an index-free notation:

$$\mathscr{A}(C) = \sum_C \ddot{\mathbf{q}}^T \ddot{\mathbf{q}},$$

where a summation over a linkage C is a shorthand for the summation over all the joints in C.

4.2 Total Acceleration Equations

We demonstrate that the total acceleration of an articulated body is a quadratic function of the external forces acting upon it, as stated in the following lemma.

Lemma 1 (Total Acceleration Equation)**.** *The total acceleration $\mathscr{A}(C)$ of an articulated body C, formed by assembling an articulated body A containing $m+1$ handles and an articulated body B containing $n+1$ handles, is a quadratic function of the external forces acting upon C:*

$$\begin{aligned}\mathscr{A}(C) = &\begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^B \end{bmatrix}^T \begin{bmatrix} \mathbf{\Psi}_1^C & \mathbf{\Psi}_{12}^C \\ \mathbf{\Psi}_{21}^C & \mathbf{\Psi}_2^C \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^B \end{bmatrix} \\ &+ \begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^B \end{bmatrix}^T \begin{bmatrix} \mathbf{p}_1^C \\ \mathbf{p}_2^C \end{bmatrix} + \eta^C,\end{aligned} \tag{8}$$

where $\mathbf{\Psi}_1^C$ is a $6m \times 6m$ matrix, $\mathbf{\Psi}_2^C$ is a $6n \times 6n$ matrix, $\mathbf{\Psi}_{12}^C = (\mathbf{\Psi}_{21}^C)^T$ is a $6m \times 6n$ matrix, $\mathbf{p}_1^C$ is a $6m \times 1$ vector, $\mathbf{p}_2^C$ is a $6n \times 1$ vector, and η^C is in $\mathbb{R}$. We call equation (8) the total acceleration equation *of C.*

Proof. We demonstrate the lemma by induction on the height of the subassembly tree h. For a subassembly tree C of height zero representing a single (floating) rigid body, the total acceleration is zero, since there is no joint in a rigid body, and can thus be expressed as a quadratic function with $\mathbf{\Psi}_1^C = \mathbf{0}$, $\mathbf{\Psi}_2^C = \mathbf{0}$, $\mathbf{\Psi}_{21}^C = (\mathbf{\Psi}_{12}^C)^T = \mathbf{0}$, $\mathbf{p}_1^C = \mathbf{0}$, $\mathbf{p}_2^C = \mathbf{0}$, and $\eta^C = 0$.

Now assume that the result holds for any subassembly tree of height strictly smaller than h, and consider the assembly tree of an articulated body C with height h. Assume that C is formed by assembling two articulated bodies A and B. We will treat the case of the root node, with only one child B, afterwards.

The heights of the assembly trees of A and B are strictly smaller than h, and the total accelerations of A and B can thus be expressed as quadratic functions of the forces respectively acting upon them. Our goal is to show that the total acceleration of C is a quadratic function of the forces acting upon C.

The total acceleration $\mathscr{A}(C)$ of the articulated body C can be computed from those of A and B, respectively denoted by $\mathscr{A}(A)$ and $\mathscr{A}(B)$, and the acceleration $\ddot{\mathbf{q}}_\circ$ of the joint connecting A and B:

$$\begin{aligned}\mathscr{A}(C) &= \sum_C \ddot{\mathbf{q}}^T \ddot{\mathbf{q}} \\ &= \ddot{\mathbf{q}}_\circ^T \ddot{\mathbf{q}}_\circ + \sum_A \ddot{\mathbf{q}}^T \ddot{\mathbf{q}} + \sum_B \ddot{\mathbf{q}}^T \ddot{\mathbf{q}} \\ &= \ddot{\mathbf{q}}_\circ^T \ddot{\mathbf{q}}_\circ + \mathscr{A}(A) + \mathscr{A}(B).\end{aligned}$$

Hence, it suffices to express each of the three terms in the last sum as a quadratic function of the forces acting upon C to prove that the result holds for C as well.

By the inductive hypothesis, the total acceleration $\mathscr{A}(A)$ can be expressed as a quadratic function of the forces acting upon A:

$$\begin{aligned}\mathscr{A}(A) = &\begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^A \end{bmatrix}^T \begin{bmatrix} \mathbf{\Psi}_1^A & \mathbf{\Psi}_{12}^A \\ \mathbf{\Psi}_{21}^A & \mathbf{\Psi}_2^A \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^A \end{bmatrix} \\ &+ \begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^A \end{bmatrix}^T \begin{bmatrix} \mathbf{p}_1^A \\ \mathbf{p}_2^A \end{bmatrix} + \eta^A.\end{aligned}$$

However, equation (5) shows that the force $\mathbf{f}_2^A$ is an affine function of $\mathbf{f}_1^A$ and $\mathbf{f}_2^B$, and thus all forces acting upon A are affine functions of the forces acting upon C:

$$\begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^A \end{bmatrix} = \begin{bmatrix} \mathbf{I_d} & \mathbf{0} \\ -\mathbf{W}\mathbf{\Phi}_{21}^A & \mathbf{W}\mathbf{\Phi}_{12}^B \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^B \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ -\boldsymbol{\gamma} \end{bmatrix},$$

where $\mathbf{I_d}$ is the identity matrix. Consequently, $\mathscr{A}(A)$ can be expressed as a quadratic function of the forces acting upon C.

Similarly, by the inductive hypothesis, the total acceleration $\mathscr{A}(B)$ can be expressed as a quadratic function of the forces acting upon B:

$$\begin{aligned}\mathscr{A}(B) = &\begin{bmatrix} \mathbf{f}_1^B \\ \mathbf{f}_2^B \end{bmatrix}^T \begin{bmatrix} \mathbf{\Psi}_1^B & \mathbf{\Psi}_{12}^B \\ \mathbf{\Psi}_{21}^B & \mathbf{\Psi}_2^B \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^B \\ \mathbf{f}_2^B \end{bmatrix} \\ &+ \begin{bmatrix} \mathbf{f}_1^B \\ \mathbf{f}_2^B \end{bmatrix}^T \begin{bmatrix} \mathbf{p}_1^B \\ \mathbf{p}_2^B \end{bmatrix} + \eta^B\end{aligned}$$

But, since $\mathbf{f}_1^B = -\mathbf{f}_2^A$, the force $\mathbf{f}_1^B$ is an affine function of $\mathbf{f}_1^A$ and $\mathbf{f}_2^B$, and thus the forces acting upon B are affine functions of the forces acting upon C:

$$\begin{bmatrix} \mathbf{f}_1^B \\ \mathbf{f}_2^B \end{bmatrix} = \begin{bmatrix} \mathbf{W}\mathbf{\Phi}_{21}^A & -\mathbf{W}\mathbf{\Phi}_{12}^B \\ \mathbf{0} & \mathbf{Id} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^B \end{bmatrix} + \begin{bmatrix} \boldsymbol{\gamma} \\ \mathbf{0} \end{bmatrix},$$

which shows that the total acceleration $\mathscr{A}(B)$ is a quadratic function of the forces acting upon C.

All that remains is to express $\ddot{\mathbf{q}}_\circ^T \ddot{\mathbf{q}}_\circ$ as a quadratic function of the forces acting upon C. In the case of a generic internal node, $\ddot{\mathbf{q}}_\circ$ is computed using equation (4). To simplify the expression of $\ddot{\mathbf{q}}_\circ^T \ddot{\mathbf{q}}_\circ$ in this case, let us set

$$\begin{aligned}\mathbf{U} &= (\mathbf{S}^T\mathbf{V}\mathbf{S})^{-1}\mathbf{S}^T\mathbf{V} \\ \mathbf{N} &= \mathbf{U}^T\mathbf{U} \\ \mathbf{R} &= (\mathbf{S}^T\mathbf{V}\mathbf{S})^{-1}(\mathbf{Q} - \mathbf{S}^T\mathbf{V}\boldsymbol{\beta}).\end{aligned}$$

We then have:

$$\begin{aligned}\ddot{\mathbf{q}}_\circ^T \ddot{\mathbf{q}}_\circ = &\begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^B \end{bmatrix}^T \begin{bmatrix} \mathbf{\Phi}_{12}^A \mathbf{N} \mathbf{\Phi}_{21}^A & -\mathbf{\Phi}_{12}^A \mathbf{N} \mathbf{\Phi}_{12}^B \\ -\mathbf{\Phi}_{21}^B \mathbf{N} \mathbf{\Phi}_{21}^A & \mathbf{\Phi}_{21}^B \mathbf{N} \mathbf{\Phi}_{12}^B \end{bmatrix} \begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^B \end{bmatrix} \\ &+ 2 \begin{bmatrix} \mathbf{f}_1^A \\ \mathbf{f}_2^B \end{bmatrix}^T \begin{bmatrix} -\mathbf{\Phi}_{12}^A \mathbf{U}^T \mathbf{R} \\ \mathbf{\Phi}_{21}^B \mathbf{U}^T \mathbf{R} \end{bmatrix} + \mathbf{R}^T \mathbf{R},\end{aligned}$$

which is a quadratic function of the forces acting upon C.

The root node is a special case for which A is non-existent, and a supplementary joint acceleration is potentially defined when the articulated body is attached to a fixed base.

If the articulated body is floating, then $\ddot{\mathbf{q}}_\circ = \mathbf{0}$ and $\mathbf{f}_1^B = \mathbf{0}$. Thus $\mathscr{A}(C) = \mathscr{A}(B)$ is a quadratic function of $\mathbf{f}_2^B$ only, *i.e.* the forces acting upon C.

If the articulated body is attached to a fixed base, however, $\ddot{\mathbf{q}}_\circ$ is computed using equation (6), and $\mathbf{f}_1^B$ is computed using equation (7). Both quantities are affine functions of $\mathbf{f}_2^B$, and thus $\mathscr{A}(C) = \ddot{\mathbf{q}}_\circ^T \ddot{\mathbf{q}}_\circ + \mathscr{A}(B)$ is again a quadratic function of $\mathbf{f}_2^B$, *i.e.* the forces acting upon C. □

It should be clear from the proof that the coefficients in $\mathscr{A}(C)$ can be computed from the coefficients in $\mathscr{A}(A)$ and $\mathscr{A}(B)$. Setting

$$\mathbf{Y} = \mathbf{N} + \mathbf{W}(\mathbf{\Psi}_2^A + \mathbf{\Psi}_1^B)\mathbf{W}$$
$$\mathbf{Z} = 2\mathbf{U}^T\mathbf{R} + \mathbf{W}(\mathbf{p}_2^A - \mathbf{p}_1^B) - 2\mathbf{W}(\mathbf{\Psi}_2^A + \mathbf{\Psi}_1^B)\boldsymbol{\gamma},$$

the coefficients of the total acceleration of a generic internal node can be written:

$$\begin{aligned}
\mathbf{\Psi}_1^C &= \mathbf{\Psi}_1^A + \mathbf{\Phi}_{12}^A\mathbf{Y}\mathbf{\Phi}_{21}^A - (\mathbf{\Phi}_{12}^A\mathbf{W}\mathbf{\Psi}_{21}^A + \mathbf{\Psi}_{12}^A\mathbf{W}\mathbf{\Phi}_{21}^A)\\
\mathbf{\Psi}_2^C &= \mathbf{\Psi}_2^B + \mathbf{\Phi}_{21}^B\mathbf{Y}\mathbf{\Phi}_{12}^B - (\mathbf{\Phi}_{21}^B\mathbf{W}\mathbf{\Psi}_{12}^B + \mathbf{\Psi}_{21}^B\mathbf{W}\mathbf{\Phi}_{12}^B)\\
\mathbf{\Psi}_{21}^C &= -\mathbf{\Phi}_{21}^B\mathbf{Y}\mathbf{\Phi}_{21}^A + \mathbf{\Phi}_{21}^B\mathbf{W}\mathbf{\Psi}_{21}^A + \mathbf{\Psi}_{21}^B\mathbf{W}\mathbf{\Phi}_{21}^A = (\mathbf{\Psi}_{12}^C)^T
\end{aligned}$$

$$\begin{aligned}
\mathbf{p}_1^C &= \mathbf{p}_1^A - \mathbf{\Phi}_{12}^A\mathbf{Z} - 2\mathbf{\Psi}_{12}^A\boldsymbol{\gamma}\\
\mathbf{p}_2^C &= \mathbf{p}_2^B + \mathbf{\Phi}_{21}^B\mathbf{Z} + 2\mathbf{\Psi}_{21}^B\boldsymbol{\gamma}
\end{aligned}$$

$$\eta^C = \eta^A + \eta^B + \mathbf{R}^T\mathbf{R} + \boldsymbol{\gamma}^T(\mathbf{\Psi}_2^A + \mathbf{\Psi}_1^B)\boldsymbol{\gamma} - \boldsymbol{\gamma}^T(\mathbf{p}_2^A - \mathbf{p}_1^B) \tag{9}$$

The details of the calculations are omitted here. For the root node, the coefficients in $\mathscr{A}(C)$ are obtained by zeroing the coefficients relative to A in equation (9).

Consequently, the total acceleration equations can be computed in a bottom-up pass: the pass starts from the leaf nodes with all coefficients set to zero, and recursively applies the formulas (9) until the root of the assembly tree is reached. Assuming these total acceleration equations have been calculated, it is now possible to compute the total acceleration of a subassembly during the back-substitution pass, without having to traverse all the descendents of the subassembly to determine the joint acceleration and forces of all the joints. Therefore, when a node C is processed during the back-substitution pass of the DCA, the forces acting upon this node are available, and equation (8) can be used to obtain $\mathscr{A}(C)$ without processing C's descendents.

In summary, the forces acting upon an articulated body can be used during the back-substitution pass to compute not only the principal joint's acceleration and forces, but also the total acceleration of the articulated body, before (or without) processing its descendents.

4.3 Approximate linkage acceleration

Consider an articulated body C, and let $\varepsilon_{max} \geqslant 0$ denote a given user-defined error threshold. Assume for now that the error measure we use is the absolute linkage acceleration error. The goal is to determine an approximate acceleration $\mathbf{s}^C$ for which $\varepsilon(C) \leqslant \varepsilon_{max}$, by performing a partial back-substitution pass: some joint accelerations will be computed exactly, and some others will be implicitly set to zero.

Assume the state of the linkage and the acceleration-independent forces are known. Assume the articulated-body equations and the total acceleration equations are all up-to-date, and we are about to perform the back-substitution pass of the DCA. Since, when the back-substitution pass begins, the forces $\mathbf{f}_1^A$ and $\mathbf{f}_2^B$ acting on the complete articulated body C are assumed to be known, the total acceleration $\mathscr{A}(C) = ||\ddot{\mathbf{q}}^C||_2^2$ can be obtained from equation (8).

Now suppose $\mathscr{A}(C) = 0$. This assumption implies that all the joint accelerations of all the joints in the articulated body are zero, since $\mathscr{A}(C)$ is the sum of the (positive) squared norms of all the joint accelerations. In this case, we have determined all the joint accelerations *without* descending the assembly tree, and we can stop the back-substitution pass at the root node. In this case, the approximate acceleration $\mathbf{s}^C$ is actually equal to the exact one $\ddot{\mathbf{q}}^C$, and $\varepsilon(C) = 0$.

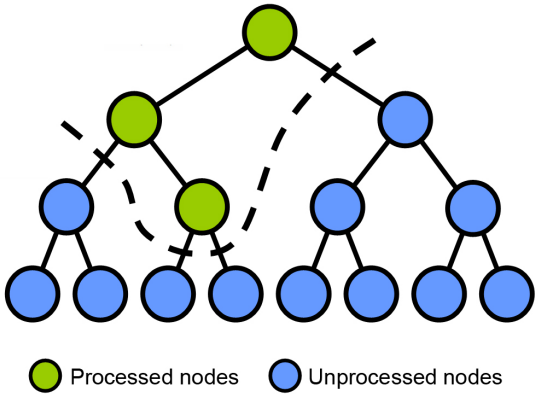

Figure 4: **Partial computation of the joint accelerations.** *Our algorithm computes only* some *joint accelerations in the articulated body, while guaranteeing that the error that results from this partial computation is smaller than a user-defined threshold.*

Suppose now that $\mathscr{A}(C)$ is not zero, but still smaller than $(\varepsilon_{max})^2$. This means that all joint accelerations are sufficiently close to zero. If we stop the back-substitution pass right at the root node and set $\mathbf{s}^C = \mathbf{0}$, we still obtain an approximate acceleration which is close enough to the exact one: $\varepsilon(C) = \sqrt{\mathscr{A}(C)} \leqslant \varepsilon_{max}$.

Assume finally that $\mathscr{A}(C)$ is strictly larger than $(\varepsilon_{max})^2$. We do not stop the traversal of the assembly tree right at the root node. We compute the joint acceleration and forces of C's principal joint, and recursively traverse the descendents of C using a priority queue and maintaining the value of the *current acceleration error* ε_c, *i.e.* the error that would result from stopping the recursion.

The priority queue is used to order the traversal of the subassemblies. A node A has a higher priority than a node B if $\mathscr{A}(A) > \mathscr{A}(B)$. During the traversal, we thus descend first in the portions of the linkages which have the highest total acceleration.

We simultaneously maintain the current acceleration error ε_c. At the beginning of the back-substitution pass, we set $\varepsilon_c = \sqrt{\mathscr{A}(C)}$, where C is the root node of the assembly tree, since this is the acceleration error that would result from stopping the back-substitution before having processed any node. During the back-substitution pass, whenever the acceleration $\ddot{\mathbf{q}}_\circ$ of the principal joint of a node C is computed, we subtract its contribution $\ddot{\mathbf{q}}_\circ^T\ddot{\mathbf{q}}_\circ$ to the current acceleration error:

$$\varepsilon_c \leftarrow \sqrt{(\varepsilon_c)^2 - \ddot{\mathbf{q}}_\circ^T\ddot{\mathbf{q}}_\circ}.$$

Then, if C has children A and B, we compute their total acceleration $\mathscr{A}(A)$ and $\mathscr{A}(B)$ and push them into the priority queue. Since $\ddot{\mathbf{q}}_\circ^T\ddot{\mathbf{q}}_\circ = \mathscr{A}(C) - \mathscr{A}(A) - \mathscr{A}(B)$, the current acceleration error ε_c is always equal to (the square root of) the sum of the total accelerations of the nodes contained in the priority queue:

$$\varepsilon_c = \sqrt{\sum_{C \in q} \mathscr{A}(C)}.$$

Since the norm $\ddot{\mathbf{q}}_\circ^T\ddot{\mathbf{q}}_\circ$ of the principal joint acceleration of any node C is positive, the current acceleration error can only decrease as more nodes are processed. This allows us to stop the back-substitution pass as soon as the current acceleration error becomes

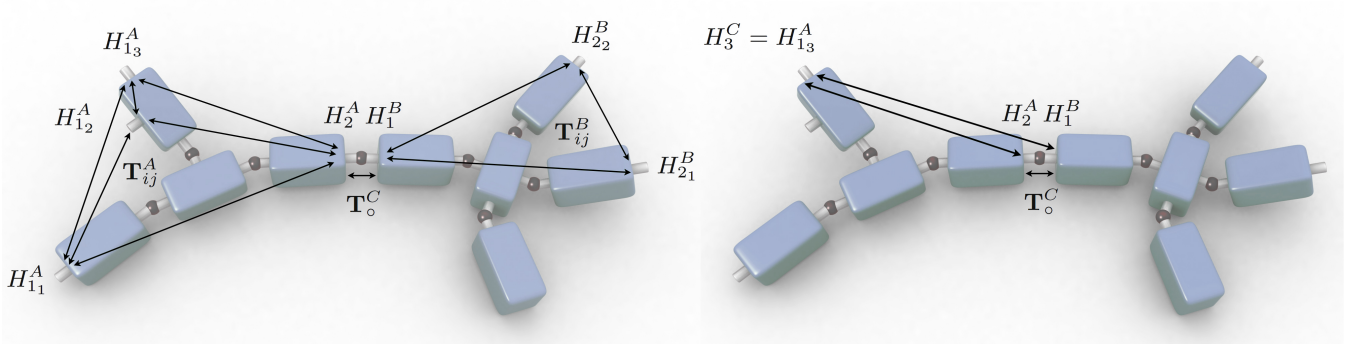

Figure 3: **The three types of transforms maintained during the simulation. Left:** *The cross-transforms* $\mathbf{T}^A_{ij}$ *and* $\mathbf{T}^B_{ij}$ *of A and B and the principal joint transform* $\mathbf{T}^C_\circ$ *of C, from which the cross-transforms of C can be computed.* **Right:** *The principal handles transforms of A and B, which relate the principal handles of A and B to the principal handle of C (assumed to be* H^C_3 *in this case), can be computed from the cross-transforms of A and B and the principal joint transform* $\mathbf{T}^C_\circ$.

smaller than ε_{max}. When the recursion of this algorithm terminates, we have obtained an approximation $\mathbf{s}^C$ of the linkage acceleration $\ddot{\mathbf{q}}^C$ such as $\varepsilon(C) = ||\mathbf{s}^C - \ddot{\mathbf{q}}^C||_2 \leqslant \varepsilon_{max}$.

The case of the relative acceleration error measure can be treated similarly. Since we are able to compute the current acceleration error and, in the beginning of the back-substitution pass, the norm $||\ddot{\mathbf{q}}^C||_2 = \sqrt{\mathscr{A}(C)}$ of the linkage acceleration, we are able to control the relative linkage acceleration error as well.

Finally, since $\max_C ||\ddot{\mathbf{q}}||_2 \leqslant \sqrt{\mathscr{A}(C)}$, we can also control the absolute and relative joint acceleration errors.

In summary, we have shown that, provided the linkage state and the total acceleration equations are up-to-date, we are able to determine an approximate acceleration $\mathbf{s}^C$ up to any level of precision. The time required to compute the approximate acceleration is $O(N_{\mathscr{E}})$, where $N_{\mathscr{E}}$ is *the number of computed joint accelerations*, that is, the size of the traversed sub-tree of the assembly tree (*cf* Figure 4).

This algorithm can be simply written. Let q denote the priority queue. Assume that popping q returns the articulated body with the highest priority, and assume C denotes the node currently processed by the back-substitution pass. Let A and B denote the potential child nodes of C. The algorithm is:

```
ALGORITHM (ACCELERATION APPROXIMATION)
  C ← root node of the assembly tree
  ε_c ← √(A(C))
  q ← push(C)
  while (ε_c > ε_max) {
    C ← pop(q)
    compute joint acceleration q̈_∘ and forces of C's principal joint
    ε_c ← √((ε_c)^2 − q̈_∘^T q̈_∘)
    for any articulated child A of C {
      compute A(A)
      q ← push(A)
    }
  }
```

When this algorithm terminates (which is guaranteed, provided the user-defined threshold ε_{max} is greater than or equal to zero), an approximate linkage acceleration has been determined such that the total acceleration error is smaller than ε_{max}.

5 Simplified Linkage State Update

As the linkage configuration and the forces applied to a linkage evolve, the coefficients describing the linkage state and the total acceleration equations need to be updated before performing the next back-substitution pass. In this section, we introduce a *hierarchical reference frames* representation in which we express the state of the linkage and the total acceleration equations. We show that, using this representation, only *some* of the quantities involved in the second step need to be updated, thereby avoiding a linear-time update of the equation coefficients. Our hierarchical reference frames representation resembles the hierarchy of transforms in the Chain-Tree data structure introduced by Lotan et al. [Lotan et al. 2002] to describe the kinematics of protein chains. However, our representation is able to handle branches in the linkages.

5.1 Hierarchical reference frames

Consider a linkage C formed by assembling two articulated bodies A and B. We call the handles used to connect A and B the *principal handles* of A and B. The root node has no principal handle, since it is not connected to any other assembly.

As in Featherstone [Featherstone 1999a], each handle has a reference frame in which all the quantities related to that handle are expressed. During the simulation, we do not require maintaining the positions of the bodies in the world frame, but only three types of 4×4 homogeneous coordinates transformation matrices that we define as follows:

- **Principal joint transforms:** consider an articulated body C with a left child node A and a right child node B. By convention, we call *principal joint transform* the coordinates transformation matrix $\mathbf{T}^C_\circ$ from the reference frame of B's principal handle to the reference frame of A's principal handle. This transform is a function of the position of C's principal joint only. The principal joint transform of the root node is the coordinates transformation matrix from the reference frame of B's principal handle to the world frame.
- **Handles cross-transforms:** consider an articulated body C with handles $H^C_1, \ldots, H^C_h$, where h is the number of handles of C. We maintain the coordinates transformation matrices $\mathbf{T}^C_{ij}$ from the reference frame of each handle H^C_j to the reference frame of other handles H^C_i, for $1 \leqslant i, j \leqslant h$.

- **Principal handles transforms:** in each node, we maintain the coordinates transformation matrix from the reference frame of the node's principal handle to the reference frame of its *parent* node's principal handle. For example, if B is a node with parent C, we maintain the coordinates transformation matrix $\mathbf{T}_B^C$ from the reference frame of B's principal handle to the reference frame of C's principal handle. The root node has no such transform, and the principal handle transform of the only child of the root node is equal to the root node's principal joint transform, *i.e.* a transform from the reference frame of B's principal handle to the world frame.

These 4×4 homogeneous coordinate transformation matrices are converted at will into 6×6 spatial motion or spatial force transformation matrices when necessary [Featherstone 1999a].

Note that these three types of coordinates transforms are the only ones we need to maintain. Especially, the position of a rigid body is never expressed in the world frame, except when an external force is applied to this rigid body. We note that these world transforms are not necessary in order to display the linkage, when a graphics library which adopts other hierarchical representations is used (eg. OpenGL).

These transforms can be computed recursively from the leaf nodes to the root node of the assembly tree. Figure 3 shows these transforms in the case of the articulated body assembled in Figure 2. Each handle has a reference frame rigidly attached to it, and the transform associated to a pair of handles are represented by arrows. We detail the computations in the rest of this section.

5.1.1 Handle's cross-transforms

The handles cross-transforms are computed from the leaf nodes to the root node of the assembly tree. If C is a leaf node of the assembly tree, its cross-transforms are constant and do not have to be ever updated, since the handles are fixed in the rigid body's reference frame. If C has some children A and B, however, its cross-transforms are computed from those of A and B, and the position of C's principal joint. Since the handles of C are either handles of A or B, we only need to compute the cross-transforms which involve one handle in A and one handle in B. Indeed, the cross-transforms from handles in A to handles in A have already been computed in A, as have the cross-transforms from handles in B to handles in B already been computed in B. The cross-transforms $\mathbf{T}_{ij}^C$ from a handle H_j^C in B to a handle H_i^C in A are computed using the principal joint transform $\mathbf{T}_\circ^C$.

Let H_l^A denote A's principal handle, H_u^B denote B's principal handle, and assume $H_i^C = H_k^A$ and $H_j^C = H_v^B$. Then:

$$\mathbf{T}_{ij}^C = \mathbf{T}_{kl}^A \mathbf{T}_\circ^C \mathbf{T}_{uv}^B, \tag{10}$$

where $\mathbf{T}_{kl}^A$ is the cross-transform from H_l^A to H_k^A, and $\mathbf{T}_{uv}^B$ is the cross-transform from H_v^B to H_u^B. The cross-transform $\mathbf{T}_{ji}^C$ is computed by inverting $\mathbf{T}_{ij}^C$.

5.1.2 Principal handles transforms

Similarly, the principal handles transforms are computed from the leaf nodes to the child nodes of the root node of the assembly tree. Assume C is a linkage with children A and B. Without loss of generality, assume we want to compute the transform $\mathbf{T}_B^C$ from the principal handle of B to the principal handle of C. If the principal handle of C is a handle of B, then $\mathbf{T}_B^C$ is equal to one of B's cross-transforms. However, if the principal handle of C is a handle of A, we need to compute $\mathbf{T}_B^C$ using the principal joint transform $\mathbf{T}_\circ^C$. Assume $H_i^C = H_k^A$ is the principal handle of C, and let H_l^A denote the principal handle of A. Then:

$$\mathbf{T}_B^C = \mathbf{T}_{kl}^A \mathbf{T}_\circ^C. \tag{11}$$

The principal handle transform $\mathbf{T}_A^C$ can be computed in a similar way.

5.2 Simplified Coefficients Update

During the simulation we need to maintain, for each node in the assembly tree, the node's transforms, the node's inverse inertias and bias accelerations, and the node's total acceleration coefficients.

For any node C, most coefficients only depend on their counterparts in A and B and on the current principal joint transform of C. This is the case of the cross-transforms, the principal handles transforms, the inverse inertias, and the quadratic coefficients of the total acceleration equations.

These quantities depend on the joints positions and the acceleration-independent forces, as can be seen from their recursive expressions (2) and (9). Precisely:

- The principal joint transform $\mathbf{T}_\circ^C$ of a node's principal joint depends on the principal joint's position $\mathbf{q}_\circ$.
- The cross-transforms $\mathbf{T}_{ij}^C$ of an internal node depend on the position of the node's principal joint, and the cross-transforms of its child nodes.
- The principal handle transform $\mathbf{T}_B^C$ of a node B with parent C depends on the node's cross-transforms, and potentially the principal joint transform of the node's parent if C's principal handle is not a handle of B.
- The inverse inertias $\mathbf{\Phi}_1^C$, $\mathbf{\Phi}_2^C$, $\mathbf{\Phi}_{12}^C$ and $\mathbf{\Phi}_{21}^C$ and the quadratic coefficients $\mathbf{\Psi}_1^C$, $\mathbf{\Psi}_2^C$, $\mathbf{\Psi}_{12}^C$ and $\mathbf{\Psi}_{21}^C$ of a node C depend on the position of the node's principal joint, and the inverse inertias and quadratic coefficients of the child nodes. The inverse inertias and the quadratic coefficients of a leaf node are constant.
- The bias accelerations $\mathbf{b}_1^C$ and $\mathbf{b}_2^C$, the linear coefficients $\mathbf{p}_1^C$ and $\mathbf{p}_2^C$, and the constant coefficient η^C of a node have a similar dependency on the joint positions, the bias accelerations, the linear and the constant coefficients of their child nodes, but they also depend on the acceleration-independent forces. If an active joint force $\mathbf{Q}$ or an external force $\mathbf{f}_k$ applied to a leaf node change, these quantities and the corresponding ones in all the parent nodes have to be updated.

Assuming an explicit integration scheme, the joints positions $\mathbf{q}_\circ$ only depend on their acceleration $\ddot{\mathbf{q}}_\circ$. Since, during the (partial) back-substitution pass, an approximate linkage acceleration $\mathbf{s}^C$ has been obtained by computing exactly some joint accelerations, and approximating the accelerations of the other joints as zero, we only need to update the positions $\mathbf{q}_\circ$, and the principal joint transforms $\mathbf{T}_\circ^C$, of the joints whose acceleration has been computed. If $N_{\mathscr{E}}$ joints accelerations have been computed, the complexity of this update is $O(N_{\mathscr{E}})$.

Similarly, since the handles cross-transforms, the principal handles transforms, the inverse inertias and the quadratic coefficients of a node only depend on the principal joint's position and their counterparts in the node's children, we only need to update these coefficients in the nodes whose principal joint's acceleration has been computed, and the complexity of this update is $O(N_{\mathscr{E}})$.

Because the remaining coefficients also depend on the principal joints positions, we need to update them *at least* where the joints accelerations have been computed. Thus, the complexity of this update is at least $O(N_{\mathscr{E}})$. However, we also need to take into account changes in the acceleration-independent forces.

We do so by first retrieving the appropriate 4×4 transforms and then converting them to 6×6 spatial force transforms.

Assume the leaf node is C_l and that the nodes on the path from the root to C_l are C_0 (the root node), $C_1, \ldots, C_{l-1}$ and C_l. The transform $\mathbf{T}_l^0$ from the principal handle of C_l to the world frame is obtained by accumulating the principal handle transforms on the path to C_l:

$$\mathbf{T}_l^0 = \mathbf{T}_1^0 \ldots \mathbf{T}_l^{l-1},$$

and the transform $\mathbf{T}_0^l$ from the world frame to the leaf node's principal handle is obtained by inverting $\mathbf{T}_l^0$. The transforms from the world to all the leaf node's handles are then obtained by multiplying $\mathbf{T}_l^0$ by the appropriate handles cross-transforms of C_l, and are converted to 6×6 spatial force transforms to express $\mathbf{f}_k$ in the reference frames of the leaf node's handles. The complexity of an update in an applied external force $\mathbf{f}_k$ is thus also $O(log(N_J))$.

Although each separate update would have a $O(log(N_J))$ complexity, it is possible to perform the updates level by level as in [Lotan et al. 2002], from the leaf nodes to the root node, and avoid processing a same node twice. Assuming there are N_U nodes where either an active joint force $\mathbf{Q}$ or an applied external force $\mathbf{f}_k$ is updated, the complexity of the coefficients update is then $O(N_U log(N_J/N_U))$. This complexity grows to $O(N_J)$ when N_J nodes require an update in an acceleration-independent force $\mathbf{Q}$ or $\mathbf{f}_k$.

Thanks to this hierarchical state representation, it is thus possible to limit the update of the coefficients in the linkage state and the total acceleration equations to *some* nodes only. The number of these required nodes is at least $O(N_{\mathscr{E}})$, where $N_{\mathscr{E}}$ is the number of joint accelerations which have been computed during the back-substitution pass to obtain the approximate acceleration, and depend on the number of acceleration-independent forces that have to be updated.

Before the beginning of the *first* time-step, the coefficients in the linkage state and the total acceleration equations are computed for *all* nodes in the assembly tree. Then, given an user-defined error threshold ε_{max}, our algorithm repeatedly computes an approximate acceleration, with an acceleration error smaller than ε_{max}, and performs the simplified coefficient update based on the approximate acceleration and the changes in the acceleration-independent forces.

6 Results and Discussion

In this section, we present several preliminary benchmarks used to test our algorithm. The algorithm has been implemented in C++, and the benchmarks have been performed on a 2.8 GHz Pentium PC with 2GBytes of RAM. In all benchmarks, the error metric we used was the relative joint acceleration error, which is independent of both the linkage size and the magnitude of the linkage acceleration.

6.1 Bicycle Chain

We first provide an informal, "visual" description of the effect of the acceleration error threshold on the quasi-static simulation of a linkage. For this first test, we used a bicycle chain composed of 30 links, and applied a constant force on the last link of the chain for 10,000 time steps. Figure 1(a) shows a close-up of the chain, Figure 1(b) shows the initial and final positions of the linkage, when computed with the original DCA, and Figure 1(c)-(e) show the impact of the error bounds on the quality of the simulation provided by our approximation algorithm.

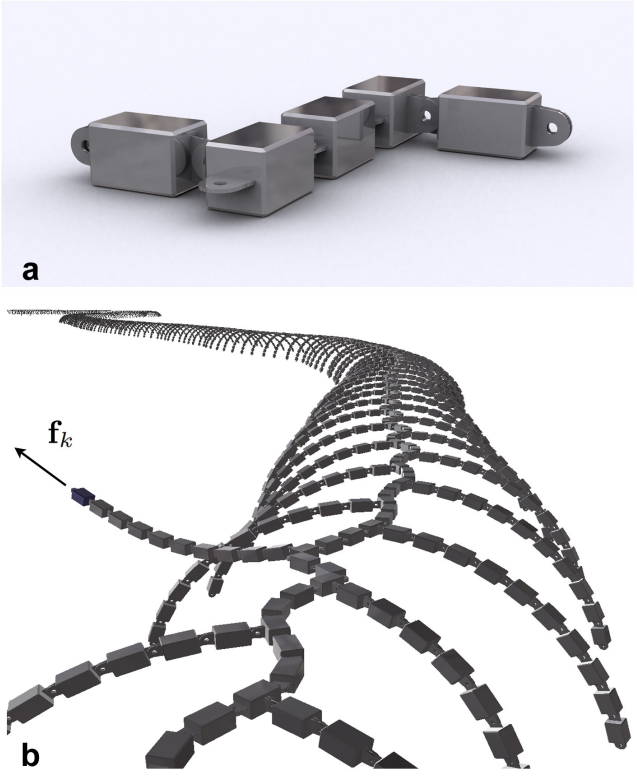

Figure 5: **A millipede-like robot with 13,000 links. (a)** *The building blocks of the millipede robot.* **(b)** *A force $\mathbf{f}_k$ is applied on a leg of the millipede during the simulation.*

6.2 Millipede Robot

Our second benchmark is a millipede-like robot. The robot has 1,000 legs with 10 links each and a spine of 3,000 links, for a total of 13,000 links and 12,999 joints. Figure 5(a) shows a close-up on the building blocks of the millipede, and Figure 5(b) shows a snapshot of the simulation.

In this benchmark, we have applied a single force to a leg of the millipede during 4,000 time steps. We have performed the simulation both with the DCA and our algorithm for various precisions ($\varepsilon_{max} = 2^{-i}$, $0 \leqslant i \leqslant 19$), and we have measured the average running times for each simulation, per time step. We have also measured the average number of joint accelerations that our algorithm had to compute in order to guarantee that the relative joint acceleration error was below ε_{max}. The results are plotted in Figure 6. As expected, as more and more joint accelerations need to be computed when ε_{max} decreases, the average running time of our algorithm increases as well. When too many nodes have to be updated (*i.e.* when the error threshold becomes smaller than 10^{-4}), our algorithm becomes less efficient than the DCA due to the higher cost of a node update. Below this threshold, however, our algorithm can quickly obtain an approximate simulation.

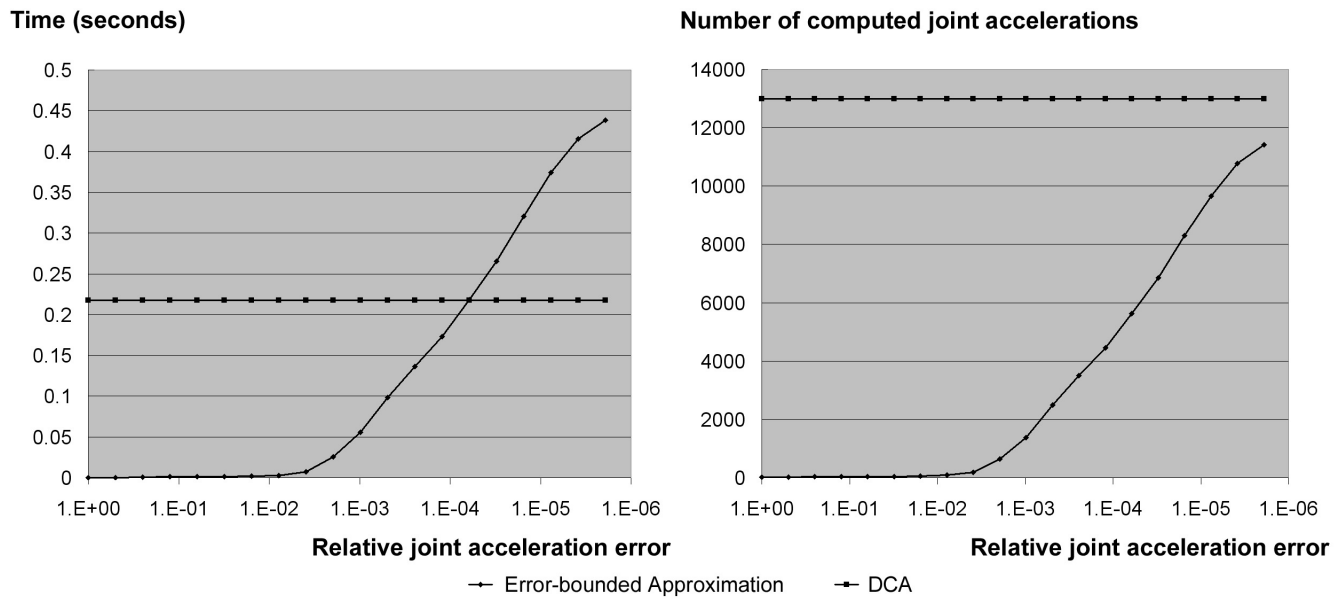

Figure 6: **Average running time of our algorithm when a force is applied to the millipede robot.** *As the maximum error decreases, more and more joint accelerations have to be computed and the average running time of our algorithm increases. When too many nodes have to be updated, our algorithm becomes less efficient than the DCA. Below this threshold, however, our algorithm can quickly obtain an approximate simulation.*

6.3 Molecule model

Our third benchmark involves a randomly generated molecule-like linkage with 50,000 rotational degrees of freedom (*cf* Figure 7).

In this benchmark, we have varied the number of forces N_f applied to the linkage during the simulation (1, 10, 100, and 1,000), as well as the desired precision ($\varepsilon_{max} = 2^{-i}$, $0 \leqslant i \leqslant 19$). For a given number of forces N_f, we have selected a random initial linkage configuration, have performed 4,000 simulation steps with the DCA, and have determined the average time required to perform one time step with the DCA. This average running time was approximately independent of the number of applied forces and found to be about 875 milliseconds per time step. We have then performed 4,000 simulation steps with our algorithm for each precision ε_{max}, using the same initial configuration each time the simulation was restarted. For each pair of parameters (N_f, ε_{max}), we have determined the average running time of our algorithm per time step, and the average number of nodes for which the joint acceleration has been computed during the back-substitution pass. The results are plotted in the graphs shown in Figure 8, and some of the data are also given in Table 1.

6.4 Discussion

Several remarks can be made:

- **Performance *vs* precision:** The average running time and the average number of computed joint accelerations increase with the required precision. It can be noted that the running time and the number of nodes processed by our algorithm do not seem to increase much after the required precision is smaller than a certain threshold. This observation suggests that it might be possible to define some *motion complexity measures*, which would characterize the behavior of complex linkages and could provide some insight when designing linkages. For example, comparing Figures 6 and 8 suggests that the impact of a force on a molecule-like linkage is more *local* than in the case of the millipede robot: for a given precision, the ratio of the number of joints affected by an applied force to the total number of joints is smaller in the case of the molecule-like linkage.

- **Performance *vs* number of applied forces:** There is a clear correlation between the number of acceleration-independent forces that have to be updated at each time step, and the performance of our algorithm. As the number of applied forces increases, more and more joint accelerations have to be computed for a given precision, and more and more nodes have to be processed in the lazy linkage update step.

- **Performance improvement:** Even when the required precision is high (about 10^{-6}), our algorithm offers a significant performance improvement over the DCA. In the case of the molecule-like linkage, when 1,000 forces are applied at each time step, the average running time of our algorithm when $\varepsilon_{max} = 2^{-19} \simeq 10^{-6}$ is about 50 milliseconds, resulting in about 17.5 times performance improvement. When only one force is applied, the average running time of our algorithm with the same precision is about 2.5 milliseconds, providing a factor of 350 times speed-up.

Relative Displacement Error. Because an error in a *single* joint acceleration may induce a large error in *many* body positions expressed in world coordinates, especially with large linkages and rotational degrees of freedom, we need to make sure that, although we compute the joint accelerations only up to some precision, the approximation still allows us to obtain a good approximation of the linkage configuration in the world frame. In this regard, the molecule-like linkage is the most challenging of the three benchmarks presented here. In order to quantify the quality of the simulation provided by our algorithm in this case, we have thus also measured the *relative displacement error* of the simulated linkage.

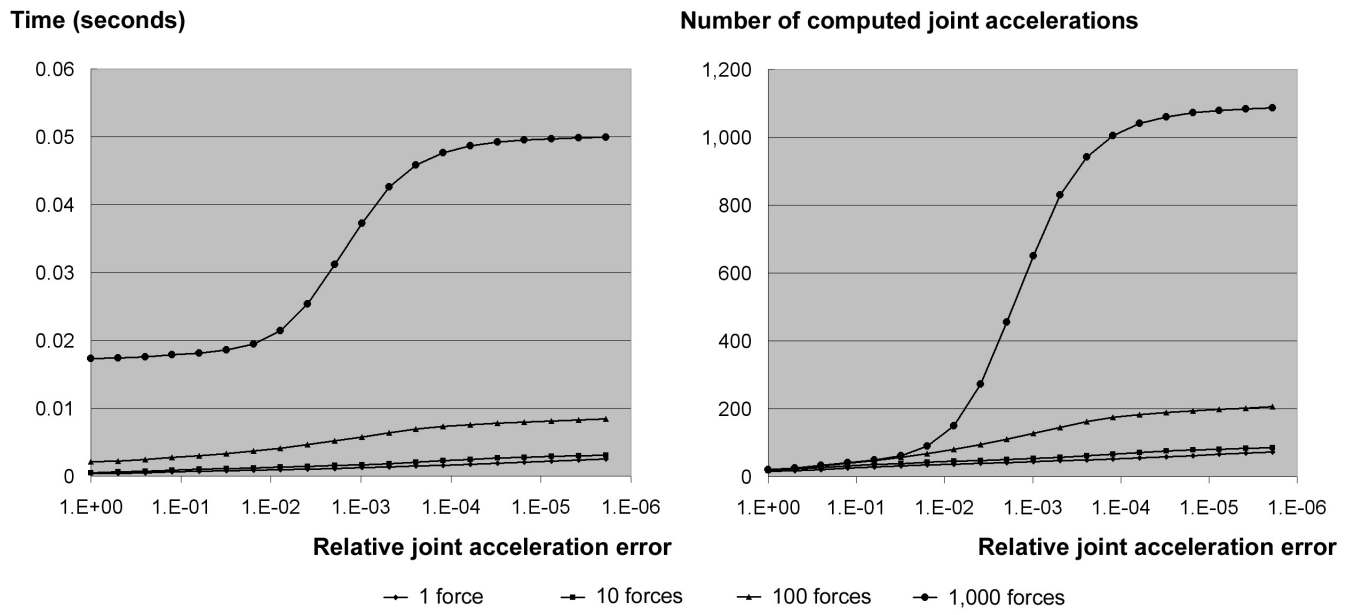

Figure 8: **Performance of our algorithm on a molecule-like model with 50,000 rotational degrees of freedom, when the number of forces applied to the linkage and the required precision vary.** *As the precision enforced on the computation of the joint accelerations increases, the average running time of our algorithm increases as well (left), because more joint accelerations have to be computed (right). Even with high precision requirements, however, our algorithm provides up to a factor of 350 times speed-up over the DCA.*

	Error Threshold					DCA
N_f	2^{-3}	2^{-7}	2^{-11}	2^{-15}	2^{-19}	
Time (milliseconds)						
1	0.6	0.9	1.4	1.9	2.4	875.1
10	0.9	1.3	1.8	2.6	3.1	873.0
100	2.7	4.1	6.4	7.8	8.4	875.0
1,000	17.8	21.4	42.6	49.2	49.9	880.0
Number of computed joint accelerations						
1	25	37	47	59	73	50,000
10	32	45	56	75	85	
100	39	80	145	189	206	
1,000	41	149	830	1,060	1,086	
Relative displacement error						
1	6E-05	2E-06	1E-07	4E-09	2E-09	0.0
10	1E-04	1E-04	4E-07	4E-08	1E-08	
100	8E-04	6E-04	7E-05	3E-06	1E-07	
1,000	6E-03	2E-03	7E-05	2E-06	6E-08	

Table 1: **Performance of our algorithm with the molecule model.** *The table shows the average running time of our algorithm (in milliseconds), the number of joint accelerations computed, and the relative displacement error when the maximum error threshold ε_{max} and the number of forces N_f applied to the linkage vary. By rapidly determining the joints that contribute most to the linkage acceleration, our algorithm is able to obtain a performance gain up to two orders of magnitude over the DCA, with an error threshold of $2^{-19} \simeq 10^{-6}$.*

Whenever a simulation with parameters (N_f, ε_{max}) completed, we determined the displacement $\mathbf{d}_i(N_f, \varepsilon_{max})$ incurred by the center of gravity of each rigid body i in the linkage during the simulation, in world coordinates. By comparing with the displacement $\mathbf{d}_i(N_f)$ obtained with the DCA, we determined the relative displacement error $e(N_f, \varepsilon_{max})$ as follows:

$$e(N_f, \varepsilon_{max}) = \frac{\max_i ||\mathbf{d}_i(N_f, \varepsilon_{max})||}{\max_i ||\mathbf{d}_i(N_f)||}$$

Figure 9 shows how the average relative displacement error per time step evolves when the precision on the joints accelerations increases. As expected, increasing the precision of the computation of the joint accelerations increases the precision on the configuration of the molecule expressed in world coordinates. Specifically, enforcing a precision of about 10^{-6} on the joint accelerations allows us to obtain a final linkage configuration with a relative displacement error smaller than 10^{-6}. Considering the previous performance benchmarks, we conclude that an application for which this level of precision is sufficient can benefit substantially from our algorithm.

7 Conclusion

We have introduced a novel algorithm for efficiently computing an approximation of the forward quasi-statics of an acyclic linkage. Given any user-defined error threshold and acceleration error measure, our algorithm is able, at each time step, to rapidly determine the joints which contribute most to the overall linkage motion, and determine an approximate linkage acceleration within the required error bound. Our algorithm allows a natural trade-off between the precision of the resulting simulation and the time required to compute it. We have implemented our algorithm and tested its performance on complex benchmarks consisting of up to 50,000 links. We demonstrate that in some cases our algorithm is able to

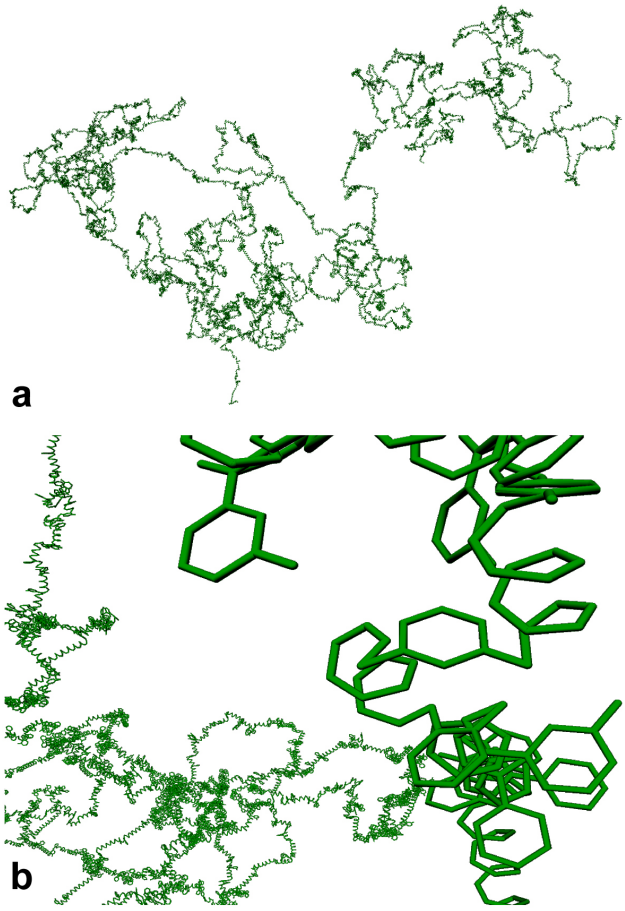

Figure 7: **A randomly generated molecule-like linkage with 50,000 rotational degrees of freedom. (a)** *The complete model.* **(b)** *A close-up view.*

achieve up to two orders of magnitude of performance improvement, while providing a high-precision, error-bounded approximation of the quasi-statics of the simulated linkage.

We would like to extend this work to mechanisms with loops. In our current framework, indeed, loops can only be handled through corrective penalty forces, *i.e.* forces applied to correct drifts where the linkage should be closed. Because the general version of the DCA can handle loops, however (at the price of an increased computational complexity [Featherstone 1999b]), we believe that it might be possible to extend our algorithm to handle loops analytically, and not only through penalty forces.

An important extension of this work is the generalization of the proposed algorithm to simulate the full *dynamics* of complex linkages. This extension is non-trivial, because of the presence of velocity-dependent coefficients in the articulated-body equations and the total acceleration equations. Independent of the reference frames in which they are expressed, the body velocities *relatively to the world frame* have to be available at each time, which, *a priori*, would require updating *all* the equations coefficients.

We have recently shown that a simplified update mechanism is still possible in the dynamics case, and have proposed an adaptive articulated-body dynamics algorithm [Redon et al. 2005a]. Nonetheless, we believe progressive quasi-statics can be useful in several areas, and the present work suggests several future research

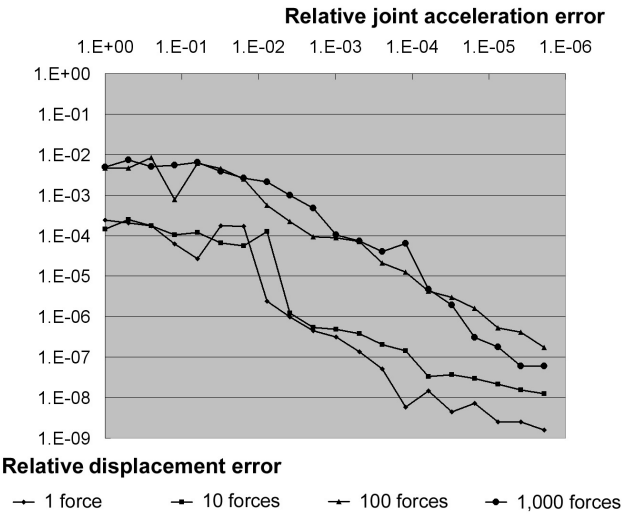

Figure 9: **Influence of the relative joint acceleration error on the relative displacement error.** *As the precision enforced on the computation of the joint acceleration increases, the precision in the final configuration of the linkage expressed in the world frame increases as well.*

directions:

- **Approximate collision detection and response:** We would like to complement this work with new approximate methods for collision detection and response. For example, we would like to investigate partial update mechanisms for bounding-volume hierarchies typically used in collision detection [Redon et al. 2002; Redon et al. 2005b].

- **Progressive quasi-statics with force feedback:** We believe the progressive quasi-statics framework introduced in this paper might be useful for fast approximate force-feedback simulation of complex linkages, which typically require update frequencies greater than 1000Hz (as opposed to 25Hz for visual simulation).

- **Perception studies:** The precision-performance trade-off allowed by our framework seems to make it possible to study human perception of articulated body quasi-static motion, through precise evaluations of the perceptibility of errors in the motion of an articulated body. This would allow for effective and pertinent simplification of visual and haptic (force-feedback) articulated-body simulation.

Finally, beside CAD/CAM, we would also like to investigate the application of our approach in drug design, digital actors, robotics and automation, computational biology, and design of medical instrumentation.

Acknowledgements

The authors would like to acknowledge Dr. Roy Featherstone for his help with the DCA. This project is supported in part by the Army Research Office, Intel Corporation, the National Science Foundation, and the Office of Naval Research.

References

BAE, D., AND HAUG, E. 1987. A recursive formulation for constrained mechanical systems dynamics: Part i. open-loop systems. *Mechanical Structures and Machines, Vol. 15, No. 3, pp. 359-382*.

BRANDL, H., JOHANNI, R., AND OTTER, M. 1986. A very efficient algorithm for the simulation of robots and similar multibody systems without inversion of the mass matrix. *IFAC/IFIP/IMACS Symposium, pp. 95-100*.

FEATHERSTONE, R., AND ORIN, D. E. 2000. Robot dynamics: Equations and algorithms. *IEEE Int. Conf. Robotics and Automation, pp. 826-834*.

FEATHERSTONE, R. 1987. *Robot Dynamics Algorithms*. Kluwer, Boston, MA.

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

FEATHERSTONE, R. 1999. A divide-and-conquer articulated body algorithm for parallel o(log(n)) calculation of rigid body dynamics. part 2: Trees, loops, and accuracy. *International Journal of Robotics Research 18(9):876-892*.

FIJANY, A., SHARF, I., AND D'ELEUTERIO, G. 1995. Parallel o(log n) algorithms for computation of manipulator forward dynamics. *IEEE Transactions on Robotics and Automation 11(3):389400*.

HOLLERBACH, J. 1980. A recursive lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity. *IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-10, No. 11*.

HOOKER, W., AND MARGULIES, G. 1965. The dynamical attitude equations for an n-body satellite. *Journal of Astronautical Sciences, Vol. 12*.

LOTAN, I., SCHWARZER, F., HALPERIN, D., AND LATOMBE, J.-C. 2002. Efficient maintenance and self-collision testing for kinematic chains. *Symposium on Computational Geometry*.

MCMILLAN, S., AND ORIN, D. E. 1995. Efficient computation of articulated-body inertias using successive axial screws. *IEEE Trans. on Robotics and Automation, vol. 11, pp. 606-611*.

MUELLER, A., AND MAISSER, P. 2003. A lie-group formulation of kinematics and dynamics of constrained mbs and its application to analytical mechanics. *Multibody System Dynamics, vol. 9, no. 4, pp. 311-352(42)*.

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2002. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum 21 (3) (Eurographics 2002 Proceedings)*.

REDON, S., GALOPPO, N., AND LIN, M. C. 2005. Adaptive dynamics: Algorithms and analysis. *UNC Chapel Hill Technical Report TR05-006*.

REDON, S., KIM, Y. J., LIN, M. C., AND MANOCHA, D. 2005. Fast continuous collision detection for articulated models. *Journal of Computing and Information Science in Engineering, 5(2), 2005*.

RODRIGUEZ, G., JAIN, A., AND KREUTZ-DELGADO, K. 1991. Spatial operator algebra for manipulator modelling and control. *Int. J. Robotics Research, vol. 10, no. 4, pp. 371-381*.