Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2010)
M. Otaduy and Z. Popovic (Editors)

Linear-Time Dynamics for Multibody Systems with General Joint Models

Weiguang Si[†1] and Brian Guenter[‡2]

[1]University of California, Los Angeles
[2]Microsoft Research

Abstract

Most current linear-time forward dynamics algorithms support only simple types of joints due to difficulties in computing derivatives of joint transformations up to order two. We apply the D symbolic differentiation algorithm to a recursive formulation of forward dynamics to get a highly efficient linear-time forward dynamics algorithm supporting multibody systems with general scleronomic joints. With this new algorithm we can easily build a tree-topology multibody system with complex joint models and perform forward dynamics efficiently. The source code for the algorithm is freely available for non-commercial use.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

This paper describes a new linear-time algorithm for forward dynamics of tree structured systems with rigid bodies connected by scleronomic joints, which are defined as arbitrary functions of the minimal generalized coordinates of the system. In a minimal coordinate formulation the generalized coordinates usually correspond directly to the variables parameterizing the kinematics of the system, which makes these formulations immune to drift. However, since derivatives of the joint transformations up to second order are required for dynamics simulation, they have, in the past, primarily been applied to simple joint models whose derivatives are easy to compute, such as revolute and universal joints. Many interesting joint types, such as surface-surface, surface-point, curve-curve, and curve-point, have easily specified kinematics but complicated derivatives which are difficult to compute. Our new algorithm extends an existing recursive forward dynamics algorithm [Plo97] to allow for joints of arbitrary complexity. The recursive equations, including derivatives, are computed symbolically using D* [Gue07] , transformed to C++ code, and compiled to generate an efficient executable. For the simple joint types the new algorithm is generally faster than widely used existing forward dynamics algorithms. The source code for the algorithm is freely available for non-commercial use and can be downloaded at the authors' webpage.

2. Related Work

Early work on dynamics simulation focused on the problem of computational efficiency [Fea83, Kha87]. However, the difficulty of computing complex derivatives has made it difficult to apply these algorithms to general scleronomic joints. Computer graphics researchers have employed different strategies to compute derivatives of complex functions [Gue07, Kas92, Coh92, PSE*00, GHDS03]. But efficient dynamics algorithms which can support general scleronomic joints have been a challenge.

Featherstone [Fea83] developed the Articulated Body Algorithm (ABA), which is a linear-time dynamics algorithm. The first version of this algorithm supported only single-degree-of-freedom joints, but an improved version [Fea87] included general scleronomic joint models. This algorithm is widely used in dynamics simulation. However, due to the complexity in computing derivatives of complex joints, most of the implementations of Featherstone's algorithm support only simple types of joints, which are called *lower pairs* [Reu76], such as prismatic joints, spherical joints, etc.

† e-mail: forswg@cs.ucla.edu
‡ e-mail: bguenter@microsoft.com

DOI: 10.2312/SCA/SCA10/031-037

Symbolic dynamics has also received much attention in both industry and academia. Michael et al. [SR94] developed SD/FAST, which is widely used in physically-based simulation. Autolev [KL00], a symbolic manipulation package that uses Kane's method to develop equations of motion and perform simulations, is also frequently used for rigid body simulation and motion analysis. However, both of these software support only simple joint models. Guenter et al. [GL09] proposed a symbolic Lagrangian multibody dynamics algorithm which supports general scleronomic joints with empirically measured $O(n)$ inverse dynamics and $O(n^3)$ forward dynamics. However, the $O(n^3)$ time complexity and long symbolic preprocessing time for large-DOF systems make it most useful for smaller systems.

Kry and Pai [KP03] used reduced coordinates to evolve a continuous contact between smooth piecewise parametric surfaces. The computation of derivatives is very involved as they handle general object geometry by subdivision surfaces. Lee and Terzopoulos [LT08] modeled more general joints for multibody dynamics, their method features the efficient computation of derivatives by representing joints using spline. However, the joint types are still limited to special forms that can be modeled by splines. Rodriguez [Rod87] proposed a linear-time forward dynamics algorithm by invoking results from Kalman filtering. Ploen [Plo97] rederived this algorithm using Lie groups and Lie algebras. His formulation is clear and easy to understand but only deals with simple joints. We reformulate this algorithm, extending it to handle complex joints. We use D* to compute the complex derivatives and to symbolically perform all operations, applying algebraic simplification to the resulting expressions. The D* code generator converts the symbolic expression into a C++ program which is then compiled and executed. This algorithm runs in linear time and supports general scleronomic joint models. Symbolic preprocessing time is less than one minute even for systems with hundreds of degrees of freedom and the resulting executable is computationally efficient.

3. Geometric Preliminaries

The mechanical systems under consideration are tree structured linkages of rigid bodies. The relative orientation and position of two linked bodies are defined by $\mathrm{SE}(3)$, the Special Euclidean Group of rigid-body motions, which consists of matrices

$$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix},$$

written as an ordered pair $(\mathbf{R},\mathbf{t})$, where $\mathbf{R} \in \mathrm{SO}(3), \mathbf{t} \in \mathbb{R}^3$, and $\mathrm{SO}(3)$ is the group of 3×3 rotation matrices.

Given a moving body frame $\mathbf{T}(t) = (\mathbf{R},\mathbf{t})$, its *generalized velocity* expressed in the instantaneous body frame is a *twist*

$$\hat{\mathbf{v}} = \mathbf{T}^{-1}\dot{\mathbf{T}} = \begin{bmatrix} [\omega] & v \\ 0 & 0 \end{bmatrix}, \tag{1}$$

where

$$[\omega] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}.$$

The *Lie algebra* of $\mathrm{SE}(3)$, denoted as $\mathrm{se}(3)$, consists of matrices of the form $\hat{\mathbf{v}}$. We also represent $\hat{\mathbf{v}}$ as a vector $\mathbf{v} = [\omega^T, v^T]^T$. The $\vee$ operator maps a twist to its vector form, i.e. $\hat{\mathbf{v}}^\vee = \mathbf{v}$.

For $\mathbf{T} \in \mathrm{SE}(3)$ and $\mathbf{g} = [\omega^T, v^T]^T$, the adjoint mapping is defined as $\mathrm{Ad}_{\mathbf{T}}\hat{\mathbf{g}} = \mathbf{T}\hat{\mathbf{g}}\mathbf{T}^{-1}$, or in matrix form as

$$\mathrm{Ad}_{\mathbf{T}}\mathbf{g} = \begin{bmatrix} \mathbf{R} & 0 \\ [\mathbf{t}]\mathbf{R} & \mathbf{R} \end{bmatrix} \begin{bmatrix} \omega \\ v \end{bmatrix}. \tag{2}$$

In Section 4 we show how to use the adjoint mapping to transform between coordinate frames.

We can also define the adjoint mapping using the Lie bracket: $\mathrm{ad}_{\hat{\mathbf{g}}} : \mathrm{se}(3) \mapsto \mathrm{se}(3)$, defined as $\mathrm{ad}_{\hat{\mathbf{g}}_1}\hat{\mathbf{g}}_2 = \hat{\mathbf{g}}_1\hat{\mathbf{g}}_2 - \hat{\mathbf{g}}_2\hat{\mathbf{g}}_1$ or

$$\mathrm{ad}_{\mathbf{g}_1}\mathbf{g}_2 = \begin{bmatrix} [\omega_1] & 0 \\ [v_1] & [\omega_1] \end{bmatrix} \begin{bmatrix} \omega_2 \\ v_2 \end{bmatrix}. \tag{3}$$

The dual space of $\mathrm{se}(3)$, denoted as $\mathrm{se}^*(3)$, represents *generalized force*, written as $\mathbf{f} = [\mathbf{m}^T, \mathbf{h}^T]^T$, where $\mathbf{m} \in \mathbb{R}^3$ is a moment and $\mathbf{h} \in \mathbb{R}^3$ a force. In matrix form, the corresponding dual operators of adjoint mappings $\mathrm{Ad}^*_{\mathbf{T}} : \mathrm{se}^*(3) \mapsto \mathrm{se}^*(3)$ and $\mathrm{ad}^*_{\mathbf{g}} : \mathrm{se}^*(3) \mapsto \mathrm{se}^*(3)$ are the transposes of $\mathrm{Ad}_{\mathbf{T}}$ and $\mathrm{ad}_{\mathbf{g}}$:

$$\mathrm{Ad}^*_{\mathbf{T}} = \mathrm{Ad}^T_{\mathbf{T}}, \qquad \mathrm{ad}^*_{\mathbf{g}} = \mathrm{ad}^T_{\mathbf{g}}.$$

In this notation the Newton-Euler equations of rigid body motion are [PBP95]:

$$\mathbf{f} = \mathbf{J}\dot{\mathbf{v}} - \mathrm{ad}^*_{\mathbf{v}}\mathbf{J}\mathbf{v}, \tag{4}$$

where $\mathbf{f} \in \mathrm{se}^*(3)$ is the generalized force applied to the rigid body and $\mathbf{v} \in \mathrm{se}(3)$ is the generalized velocity. Here $\mathbf{J} \in \mathbb{R}^{6\times 6}$ is the *generalized inertia* of the rigid body:

$$\mathbf{J} = \begin{bmatrix} \mathcal{I} - m[\mathbf{r}]^2 & m[\mathbf{r}] \\ -m[\mathbf{r}] & m\mathbf{I} \end{bmatrix}, \tag{5}$$

where m is the mass, $\mathcal{I}$ is the inertia tensor of the rigid body about the center of mass, $\mathbf{r} \in \mathbb{R}^3$ is the position of the center of mass, and $\mathbf{I}$ denotes the 3×3 identity matrix.

4. Dynamics of Tree-Topology Multibody Systems with General Scleronomic Joints

Consider a general tree-topology multibody system where links and joints are numbered from 1 to n. Let $\lambda(i)$ be the index of the parent link of link i. We use $\mathbf{T}_i$ to represent the configuration of body frame i with respect to the inertial reference frame. Letting $\mathbf{G}_i = f_{\lambda(i),i}$ denote the relative configuration of i with respect to $\lambda(i)$ gives:

$$\mathbf{T}_i = \mathbf{T}_{\lambda(i)}\mathbf{G}_i. \tag{6}$$

Substituting (6) into (1), we can obtain

$$\begin{aligned}\hat{\mathbf{v}}_i &= \mathbf{T}_i^{-1}\dot{\mathbf{T}}_i \\ &= \left(\mathbf{T}_{\lambda(i)}\mathbf{G}_i\right)^{-1}\left(\dot{\mathbf{T}}_{\lambda(i)}\mathbf{G}_i + \mathbf{T}_{\lambda(i)}\dot{\mathbf{G}}_i\right) \\ &= \mathbf{G}_i^{-1}\hat{\mathbf{v}}_{\lambda(i)}\mathbf{G}_i + \mathbf{G}_i^{-1}\dot{\mathbf{G}}_i. \end{aligned} \quad (7)$$

Using the adjoint mapping, we can write (7) in vector form:

$$\mathbf{v}_i = \mathrm{Ad}_{f_{\lambda(i),i}^{-1}}\mathbf{v}_{\lambda(i)} + \mathbf{S}_i\dot{\mathbf{q}}_i, \quad (8)$$

where

$$\mathbf{S}_i = \left[\left(\mathbf{G}_i^{-1}\frac{\partial \mathbf{G}_i}{\partial q_i^1}\right)^{\vee} \quad \left(\mathbf{G}_i^{-1}\frac{\partial \mathbf{G}_i}{\partial q_i^2}\right)^{\vee} \quad \cdots \quad \left(\mathbf{G}_i^{-1}\frac{\partial \mathbf{G}_i}{\partial q_i^{d_i}}\right)^{\vee}\right]. \quad (9)$$

Here the number of DOFs of joint i is d_i, i.e $\mathbf{q}_i = \begin{bmatrix} q_i^1 & q_i^2 & \cdots & q_i^{d_i}\end{bmatrix}^T$; $\mathbf{S}_i$ is therefore a $6 \times d_i$ matrix.

Using the relationship $\frac{d}{dt}\mathrm{Ad}_{\mathbf{T}^{-1}} = -\mathrm{ad}_{\mathbf{v}}\mathrm{Ad}_{\mathbf{T}^{-1}}$ (see Appendix C) , where $\mathbf{v}$ is defined in (1), we have

$$\dot{\mathrm{Ad}}_{f_{\lambda(i),i}^{-1}} = -\mathrm{ad}_{\mathbf{S}_i\dot{\mathbf{q}}_i}\mathrm{Ad}_{f_{\lambda(i),i}^{-1}}. \quad (10)$$

Take derivatives of both sides of (8) to obtain:

$$\dot{\mathbf{v}}_i = \mathrm{Ad}_{f_{\lambda(i),i}^{-1}}\dot{\mathbf{v}}_{\lambda(i)} - \mathrm{ad}_{\mathbf{S}_i\dot{\mathbf{q}}_i}\mathrm{Ad}_{f_{\lambda(i),i}^{-1}}\mathbf{v}_{\lambda(i)} + \dot{\mathbf{S}}_i\dot{\mathbf{q}}_i + \mathbf{S}_i\ddot{\mathbf{q}}_i, \quad (11)$$

where

$$\dot{\mathbf{S}}_i = \frac{\partial \mathbf{S}_i}{\partial q_i^1}\dot{q}_i^1 + \ldots + \frac{\partial \mathbf{S}_i}{\partial q_i^{d_i}}\dot{q}_i^{d_i}. \quad (12)$$

Let $\mu(i)$ be the set of child link indices of link i, i.e. $\mu(i) = \{k|\lambda(k) = i\}$. Then from (4) we have

$$\begin{aligned}\mathbf{f}_i &= \sum_{k\in\mu(i)} {}^i\mathbf{f}_k + \mathbf{J}_i\dot{\mathbf{v}}_i - \mathrm{ad}^*_{\mathbf{v}_i}\mathbf{J}_i\mathbf{v}_i - \mathbf{f}_{e,i} \quad (13)\\ &= \sum_{k\in\mu(i)} \mathrm{Ad}^*_{f_{i,k}^{-1}}\mathbf{f}_k + \mathbf{J}_i\dot{\mathbf{v}}_i - \mathrm{ad}^*_{\mathbf{v}_i}\mathbf{J}_i\mathbf{v}_i - \mathbf{f}_{e,i}, \quad (14)\end{aligned}$$

where $\mathbf{f}_i \in \mathrm{se}^*(3)$ is the generalized force applied by link $\lambda(i)$ to link i, ${}^i\mathbf{f}_k = \mathrm{Ad}^*_{f_{i,k}^{-1}}\mathbf{f}_k$ denotes $\mathbf{f}_k$ relative to frame i, and $\mathbf{f}_{e,i}$ is the external force (e.g., gravity) on link i.

The $d_i \times 1$ vector of joint torques, τ_i, can be extracted from the generalized forces

$$\tau_i = \mathbf{S}_i^T\mathbf{f}_i. \quad (15)$$

From equations (9) and (12), notice that $\mathbf{S}_i$ and $\dot{\mathbf{S}}_i$ need first and second partial derivatives. Both matrices are necessary for forward and inverse dynamics. For simple joints, these derivatives are trivial to compute, e.g. for revolute joints rotating around z-axis, $\mathbf{S}_i = [0,0,1,0,0,0]^T$ and $\dot{\mathbf{S}}_i = [0,0,0,0,0,0]^T$; for non-simple joints, computing these derivatives is much more complex. However, computing $\mathbf{S}_i$ and $\dot{\mathbf{S}}_i$ with D* is straightforward.

The motion equations for a tree-topology multi-body system are

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \phi(\mathbf{q}) = \tau. \quad (16)$$

Detailed formulation is provided in Appendix A. To perform forward dynamics, we need to solve (16) to obtain

$$\ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1}(\tau - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} - \phi(\mathbf{q})) \quad (17)$$

As the inverse mass matrix $\mathbf{M}(\mathbf{q})^{-1}$ has a square factorization, the forward dynamics procedure can be rewritten recursively (refer to Appendix B for more details). We can get the following $O(n)$ recursive algorithm:

1. Forward recursion: for $k = 1$ to n

$$\mathbf{v}_k = \mathrm{Ad}_{f_{\lambda(k),k}^{-1}}\mathbf{v}_{\lambda(k)}, \quad (18)$$

$$\mathbf{a}_k = -\mathrm{ad}_{\mathbf{S}_k\dot{\mathbf{q}}_k}\mathbf{v}_k + \dot{\mathbf{S}}_k\dot{\mathbf{q}}_k, \quad (19)$$

$$\mathbf{v}_k = \mathbf{v}_k + \mathbf{S}_k\dot{\mathbf{q}}_k, \quad (20)$$

$$\mathbf{b}_k = -\mathrm{ad}^*_{\mathbf{v}_k}\mathbf{J}_k\mathbf{v}_k - \mathbf{f}_{e,i}. \quad (21)$$

2. Forward recursion: for $k = 1$ to n

$$\alpha_k = \mathrm{Ad}_{f_{\lambda(k),k}^{-1}}\alpha_{\lambda(k)} + \mathbf{a}_k, \quad (22)$$

$$\bar{\alpha}_k = \mathbf{J}_k\alpha_k + \mathbf{b}_k. \quad (23)$$

3. Backward recursion: for $k = n$ to 1

$$\mathbf{p}_k = \sum_{i\in\mu(k)} \mathrm{Ad}^*_{f_{k,i}^{-1}}\mathbf{p}_i + \bar{\alpha}_k, \quad (24)$$

$$\bar{\mathbf{p}}_k = \mathbf{S}_k^T\mathbf{p}_k, \quad (25)$$

$$\bar{\tau}_k = \tau_k - \bar{\mathbf{p}}_k. \quad (26)$$

4. Backward recursion: for $k = n$ to 1

$$\bar{\mathbf{J}}_k = \mathbf{J}_k + \sum_{i\in\mu(k)} (\mathrm{Ad}^*_{f_{k,i}^{-1}}\bar{\mathbf{J}}_i\mathrm{Ad}_{f_{k,i}^{-1}} - \mathrm{Ad}^*_{f_{k,i}^{-1}}\bar{\mathbf{J}}_i\mathbf{S}_i\Omega_i^{-1}\mathbf{S}_i^T\bar{\mathbf{J}}_i\mathrm{Ad}_{f_{k,i}^{-1}}), \quad (27)$$

$$\Omega_k = \mathbf{S}_k^T\bar{\mathbf{J}}_k\mathbf{S}_k, \quad (28)$$

$$\mathbf{Y}_{\lambda(k),k} = \mathrm{Ad}^*_{f_{\lambda(k),k}^{-1}}(\mathbf{I} - \bar{\mathbf{J}}_k\mathbf{S}_k\Omega_k^{-1}\mathbf{S}_k^T), \quad (29)$$

$$\Pi_{\lambda(k),k} = \mathrm{Ad}_{f_{\lambda(k),k}^{-1}}\bar{\mathbf{J}}_k\mathbf{S}_k\Omega_k^{-1}. \quad (30)$$

5. Backward recursion: for $k = n$ to 1

$$\mathbf{z}_k = \sum_{i \in \mu(k)} (\mathbf{Y}_{k,i}\mathbf{z}_i + \Pi_{k,i}\bar{\tau}_i), \tag{31}$$

$$\mathbf{c}_k = \bar{\tau}_k - \mathbf{S}_k^T \mathbf{z}_k, \tag{32}$$

$$\bar{\mathbf{c}}_k = \Omega_k^{-1}\mathbf{c}_k. \tag{33}$$

6. Forward recursion: for $k = 1$ to n

$$\eta_k = \mathbf{Y}_{\lambda(k),k}^T \eta_{\lambda(k)} + \mathbf{S}_k \bar{\mathbf{c}}_k, \tag{34}$$

$$\ddot{\mathbf{q}}_k = \bar{\mathbf{c}}_k - \Pi_{\lambda(k),k}^T \eta_{\lambda(k)}. \tag{35}$$

The inverse of matrix Ω_k can be computed symbolically, using Gaussian elimination, or numerically using any standard matrix library. Matrix Ω_k is of size $d_k \times d_k$, where d_k is the number of DOFs of the joint k. The largest possible size is 6×6 for scleronomic joints as we need at most 3 DOFs for rotation and at most 3 DOFs for translation. For the revolute joint, the matrix size is only 1×1; for the surface-surface constrained joint, it is 4×4. These small matrices can be efficiently inverted using symbolic Gaussian elimination and this can improve runtime performance significantly, especially when there are many trivial matrix elements (e.g. 0, 1.), which is often the case.

For the fully symbolic method recursions 1 ~ 6 are all evaluated symbolically. For the mixed symbolic-numerical method the computation is divided into two parts: recursions 1 ~ 3 are computed symbolically while recursions 4 ~ 6 are computed numerically. We describe results for both methods in Section 5. The mixed symbolic-numerical method has lower symbolic processing complexity than the fully symbolic method, reducing memory use and processing time in the preprocessing phase. This method also avoids symbolic matrix inverse, large improvements in efficiency can be made with effort. However, since the symbolic preprocessing is done only once this is usually not an issue, unless the system is so complex that memory usage exceeds total available memory.

5. Experiments and Results

Algorithms employing D* symbolic differentiation are executed in a 2-phase process. The first phase is symbolic preprocessing, code generation, and compilation; this preprocessing is done only once per system. The second phase is execution of the resulting code; this phase determines the running-time property of an algorithm and this is the time we use to compare with other algorithms. Table 1 shows the comparison results of different algorithms for different examples. The experiment is based on our implementation of the Featherstone's algorithm and SD/FAST software downloaded from [SR94]. We compared our algorithm (Method I, processing all recursions symbolically) with Featherstone's algorithm and SD/FAST under the same conditions. We also show the running time of the method which deals with our forward dynamics formulation in a mixed symbolic-numerical way (Method II). All timings were measured on a 2.67 GHz Intel Core 2 Quad processor with 2 GB of RAM. From Table 1 we can see that our algorithm is faster than both Featherstone's algorithm and SD/FAST in most cases. Our algorithm also supports non-simple joint types, such as surface-surface constrained joints. The running time of SD/FAST for a 100 rigid-link revolute joint chain is not available because it takes more than tens of hours to compile the C code with optimization.

The accompanying video shows some simulation results of Method I. Fig.1 shows the example of a 20 torus chain,

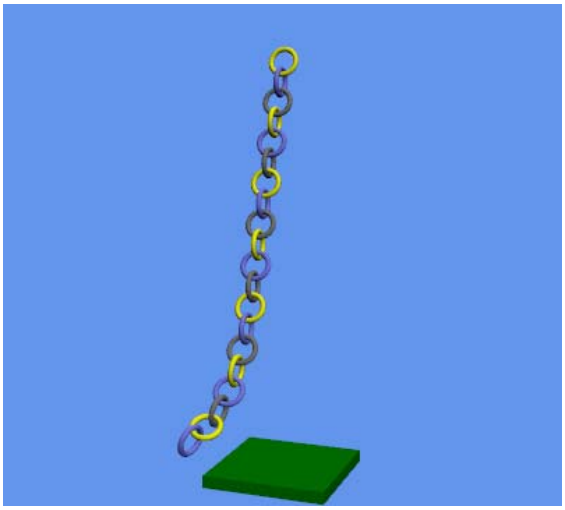

Figure 1: *Torus chain example*

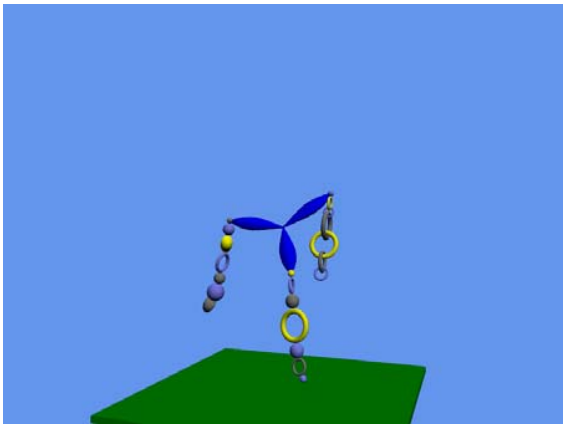

Figure 2: *Windmill example*

Example	RC20	RC50	RC100	TC10	TC20	WM
DOFs	20	50	100	38	78	73
Method I	0.020	0.050	0.112	0.055	0.115	0.108
Featherstone	0.025	0.064	0.127	NA	NA	NA
SD/FAST	0.016	0.136	—	NA	NA	NA
Method II	1.0	2.4	4.9	1.05	2.2	2.4

Table 1: *Running Time Comparison: RC20 stands for revolute joint chain of 20 rigid links; RC50 stands for revolute joint chain of 50 rigid links; RC100 stands for revolute joint chain of 100 rigid links; TC10 stands for torus chain of 10 tori; TC20 stands for torus chain of 20 tori; WM stands for a tree structured windmill example. All the running time entries are in milliseconds.*

which consists of 20 tori connected by surface-surface constrained joints. In Fig.2 is a windmill example which has sphere-sphere, sphere-ellipsoid, sphere-torus, torus-torus, and torus-ellipsoid surface joint connections. Fig.3 shows an example of a torus chain sliding along a Bezier spline with 7 control points under the effect of gravity. The spline passes through the center of the first torus, while the main axis of the first torus is always aligned with the tangential direction of the spline. Fig.4 shows an example of a multibody system sliding around a Bezier surface with 25 control points under the effect of gravity. The sliding board of the system is aligned with the tangential plane of the surface. It is easy to model and simulate these complex mechanisms because parameterizing the kinematics of the complex mechanisms is straightforward. The difficulty is in computing the derivatives, which is handled automatically by D*.

As our dynamics algorithm deals with all kinds of scleronomic joints in a unified way, there is no limit in the joint types of dynamics systems as long as the transformation matrix of the joint can be represented as a function of independent generalized coordinates.

In general the symbolic method results in faster code than the mixed symbolic-numerical method but the size of the symbolic expressions is larger which makes the symbolic processing time and memory requirements larger. The running time of the mixed symbolic-numerical method is also strongly dependent on the numerical matrix solver used. We use our own implementation of Jacobi's method, which is slow but which we can freely distribute. Faster alternatives, which we cannot redistribute, are Lapack and the Intel Performance Math Kernel Library. These solvers can easily be substituted by the user for our solver, if desired.

6. Discussion and Conclusion

Any collision handling techniques for minimal coordinate approaches, for example, [KM98], can be used for simulating collision response.

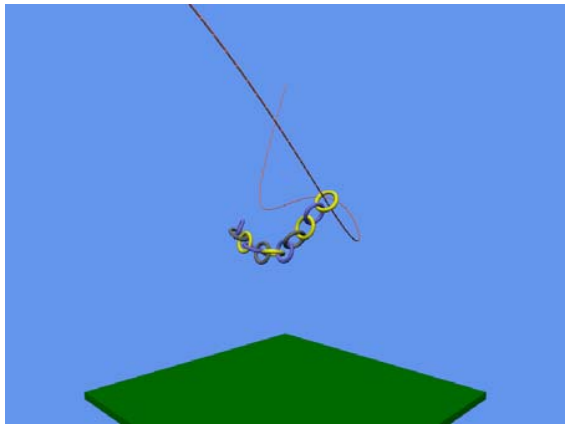

Figure 3: *A torus chain sliding along a spline*

Figure 4: *Spline surface example*

Like Featherstone's algorithm our algorithm requires generalized coordinates which parameterize system kinematics. This is not generally a limitation; many complex mechanisms are easily parameterized. Singular parameterizations, such as gimbal lock in a ball joint parameterized with Euler angles, can be handled by switching between parameterizations with different singular points. For example, [Gra98], uses this method to parameterize the ball joint without singularity.

Using this new algorithm we can easily build and simulate systems with complex joints. The $O(n)$ running time makes it practical to simulate large, complex systems in real time. For example, one could better model human figure motion by using more accurate joints, such as a four bar linkage knee joint, a surface-surface type scapula joint, etc. The source code for the algorithm is freely available for non-commercial use and can be downloaded at the authors' webpage.

Acknowledgements

We would like to thank Sung-Hee Lee for providing the implementation of Featherstone's algorithm as well as and his valuable discussions with us.

Appendix A: Formulation of Motion Equation

We formulate the equations of motion for a tree-topology multibody system of n joints using the following global matrix representation:

$$\mathbf{v} = \begin{bmatrix}\mathbf{v}_1^T & \mathbf{v}_2^T & \cdots & \mathbf{v}_n^T\end{bmatrix}^T \in \mathbb{R}^{6n};$$

$$\mathbf{v}_\lambda = \begin{bmatrix}\mathbf{v}_{\lambda(1)}^T & \mathbf{v}_{\lambda(2)}^T & \cdots & \mathbf{v}_{\lambda(n)}^T\end{bmatrix}^T \in \mathbb{R}^{6n};$$

$$\mathbf{q} = \begin{bmatrix}\mathbf{q}_1^T & \mathbf{q}_2^T & \cdots & \mathbf{q}_n^T\end{bmatrix}^T \in \mathbb{R}^{\sum_i d_i};$$

$$\mathbf{S} = \mathrm{diag}(\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_n) \in \mathbb{R}^{6n\times\sum_i d_i};$$

$$\mathbf{M}_\lambda = \mathrm{diag}(\mathrm{Ad}_{f_{\lambda(1),1}^{-1}}, \mathrm{Ad}_{f_{\lambda(2),2}^{-1}}, \cdots, \mathrm{Ad}_{f_{\lambda(n),n}^{-1}}) \in \mathbb{R}^{6n\times 6n};$$

$$\mathrm{ad}_{\mathbf{S}\dot{\mathbf{q}}} = \mathrm{diag}(-\mathrm{ad}_{\mathbf{S}_1\dot{\mathbf{q}}_1}, -\mathrm{ad}_{\mathbf{S}_2\dot{\mathbf{q}}_2}, \cdots, -\mathrm{ad}_{\mathbf{S}_n\dot{\mathbf{q}}_n}) \in \mathbb{R}^{6n\times 6n}.$$

From (8) and (11), we obtain the global matrix representations of the generalized velocities and their derivatives:

$$\mathbf{v} = \mathbf{S}\dot{\mathbf{q}} + \mathbf{M}_\lambda\mathbf{v}_\lambda; \tag{36}$$

$$\dot{\mathbf{v}} = \mathbf{S}\ddot{\mathbf{q}} + \dot{\mathbf{S}}\dot{\mathbf{q}} + \mathbf{M}_\lambda\dot{\mathbf{v}}_\lambda + \mathrm{ad}_{\mathbf{S}\dot{\mathbf{q}}}\mathbf{M}_\lambda\mathbf{v}_\lambda. \tag{37}$$

Defining $\Theta_\lambda \in \mathbb{R}^{6n\times 6n}$ as

$$\Theta_\lambda(i,j) = \begin{cases}\mathbf{I}_{6\times 6} & \text{if } \lambda(i) = j\\ \mathbf{0}_{6\times 6} & \text{otherwise}\end{cases}, \tag{38}$$

it follows that

$$\mathbf{v}_\lambda = \Theta_\lambda\mathbf{v}. \tag{39}$$

Taking derivatives on both sides of (39) we obtain

$$\dot{\mathbf{v}}_\lambda = \Theta_\lambda\dot{\mathbf{v}}. \tag{40}$$

Substituting (39) and (40) into (36) and (37) respectively gives

$$\mathbf{v} = \mathbf{G}_\lambda\mathbf{S}\dot{\mathbf{q}}, \tag{41}$$

$$\dot{\mathbf{v}} = \mathbf{G}_\lambda\mathbf{S}\ddot{\mathbf{q}} + \mathbf{G}_\lambda\dot{\mathbf{S}}\dot{\mathbf{q}} + \mathbf{G}_\lambda\mathrm{ad}_{\mathbf{S}\dot{\mathbf{q}}}\Gamma_\lambda\mathbf{v}, \tag{42}$$

where Γ_λ is

$$\Gamma_\lambda = \mathbf{M}_\lambda\Theta_\lambda \tag{43}$$

and $\mathbf{G}_\lambda$ is

$$\mathbf{G}_\lambda = (\mathbf{I} - \Gamma_\lambda)^{-1} \tag{44}$$

$$= \mathbf{I} + \Gamma_\lambda + \cdots + \Gamma_\lambda^{n-1}. \tag{45}$$

We obtain (45) as Γ_λ is a nilpotent matrix.

We denote

$$\mathbf{f} = \begin{bmatrix}\mathbf{f}_1^T & \mathbf{f}_2^T & \cdots & \mathbf{f}_n^T\end{bmatrix}^T \in \mathbb{R}^{6n};$$

$$\mathbf{J} = \mathrm{diag}(\mathbf{J}_1, \mathbf{J}_2, \cdots, \mathbf{J}_n) \in \mathbb{R}^{6n\times 6n};$$

$$\mathrm{ad}_{\mathbf{v}}^* = \mathrm{diag}(-\mathrm{ad}_{\mathbf{v}_1}^*, -\mathrm{ad}_{\mathbf{v}_2}^*, \cdots, -\mathrm{ad}_{\mathbf{v}_n}^*) \in \mathbb{R}^{6n\times 6n};$$

$$\mathbf{f}_e = \begin{bmatrix}-\mathbf{f}_{e,1}^T & -\mathbf{f}_{e,2}^T & \cdots \mathbf{f}_{e,n}^T\end{bmatrix}^T.$$

From (14) we can obtain the global matrix representation of the generalized forces:

$$\mathbf{f} = \Gamma_\lambda^T\mathbf{f} + \mathrm{ad}_{\mathbf{v}}^*\mathbf{J}\mathbf{v} + \mathbf{J}\dot{\mathbf{v}} + \mathbf{f}_e, \tag{46}$$

we can solve the above equation and obtain

$$\mathbf{f} = \mathbf{G}_\lambda^T\mathrm{ad}_{\mathbf{v}}^*\mathbf{J}\mathbf{v} + \mathbf{G}_\lambda^T\mathbf{J}\dot{\mathbf{v}} + \mathbf{G}_\lambda^T\mathbf{f}_e. \tag{47}$$

According to (15), it is also easy to get the following equation

$$\tau = \mathbf{S}^T\mathbf{f}. \tag{48}$$

Substituting (41),(42) and (47) into (48), we have

$$\begin{aligned}\tau = {} & \mathbf{S}^T\mathbf{G}_\lambda^T\mathbf{J}\mathbf{G}_\lambda\mathbf{S}\ddot{\mathbf{q}} + \mathbf{S}^T\mathbf{G}_\lambda^T(\mathbf{J}\mathbf{G}_\lambda\mathrm{ad}_{\mathbf{S}\dot{\mathbf{q}}}\Gamma_\lambda\mathbf{G}_\lambda\mathbf{S}\\ & + \mathbf{J}\mathbf{G}_\lambda\dot{\mathbf{S}} + \mathrm{ad}_{\mathbf{v}}^*\mathbf{J}\mathbf{G}_\lambda\mathbf{S})\dot{\mathbf{q}} + \mathbf{S}^T\mathbf{G}_\lambda^T\mathbf{f}_e.\end{aligned} \tag{49}$$

Finally, the motion equations for a tree-topology multibody system are

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \phi(\mathbf{q}) = \tau, \tag{50}$$

where

$$\mathbf{M}(\mathbf{q}) = \mathbf{S}^T\mathbf{G}_\lambda^T\mathbf{J}\mathbf{G}_\lambda\mathbf{S}, \tag{51}$$

$$\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) = \mathbf{S}^T\mathbf{G}_\lambda^T(\mathbf{J}\mathbf{G}_\lambda\mathrm{ad}_{\mathbf{S}\dot{\mathbf{q}}}\Gamma_\lambda\mathbf{G}_\lambda\mathbf{S} + \mathbf{J}\mathbf{G}_\lambda\dot{\mathbf{S}} + \mathrm{ad}_{\mathbf{v}}^*\mathbf{J}\mathbf{G}_\lambda\mathbf{S}), \tag{52}$$

$$\phi(\mathbf{q}) = \mathbf{S}^T\mathbf{G}_\lambda^T\mathbf{f}_e. \tag{53}$$

Appendix B: Algorithm for Forward Dynamics

We derive the linear time forward dynamics algorithm from the motion equations. We solve (50) to obtain

$$\ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1}\bar{\tau}, \tag{54}$$

where

$$\bar{\tau} = \tau - \mathbf{S}^T\mathbf{G}_\lambda^T(\mathbf{J}\mathbf{G}_\lambda\mathbf{a} + \mathbf{b}), \tag{55}$$

$$\mathbf{a} = \mathrm{ad}_{\mathbf{S}\dot{\mathbf{q}}}\Gamma_\lambda\mathbf{G}_\lambda\mathbf{S}\dot{\mathbf{q}} + \dot{\mathbf{S}}\dot{\mathbf{q}}, \tag{56}$$

$$\mathbf{b} = \mathrm{ad}_{\mathbf{v}}^*\mathbf{J}\mathbf{G}_\lambda\mathbf{S}\dot{\mathbf{q}} + \mathbf{f}_e. \tag{57}$$

Notice that the inverse mass matrix $\mathbf{M}(\mathbf{q})^{-1}$ has a square factorization

$$\mathbf{M}(\mathbf{q})^{-1} = (\mathbf{I} - \mathbf{S}^T \mathbf{Y}_\lambda \Pi_\lambda)^T \Omega^{-1} (\mathbf{I} - \mathbf{S}^T \mathbf{Y}_\lambda \Pi_\lambda), \quad (58)$$

where Ω is a block diagonal matrix, and $\mathbf{Y}_\lambda$ and Π_λ are upper triangular matrices. Precise definitions of $\mathbf{Y}_\lambda$, Π_λ and Ω will be given later and can also be found in [Plo97].

Using this factorization we rewrite the forward dynamics procedure recursively:

1. Compute $\mathbf{a}$ and $\mathbf{b}$ according to equations (56) and (57)
2. $\alpha = \mathbf{G}_\lambda \mathbf{a}$
3. $\bar{\alpha} = \mathbf{J}\alpha + \mathbf{b}$
4. $\mathbf{p} = \mathbf{G}_\lambda^T \bar{\alpha}$
5. $\bar{\mathbf{p}} = \mathbf{S}^T \mathbf{p}$
6. $\bar{\tau} = \tau - \bar{\mathbf{p}}$
7. $\mathbf{c} = (\mathbf{I} - \mathbf{S}^T \mathbf{Y}_\lambda \Pi_\lambda) \bar{\tau}$
8. $\bar{\mathbf{c}} = \Omega^{-1} \mathbf{c}$
9. $\ddot{\mathbf{q}} = (\mathbf{I} - \mathbf{S}^T \mathbf{Y}_\lambda \Pi_\lambda)^T \bar{\mathbf{c}}$

Before we get an $O(n)$ recursive forward dynamics algorithm from direct expansion of the above procedure, we need to analyze the structure of some matrices.

We define $\lambda^p(i)$ as $\underbrace{\lambda \circ \lambda \circ \cdots \circ \lambda}_{p \text{ times}}(i)$, then according to equation (45), we have

$$\mathbf{G}_\lambda(i,j) = \begin{cases} \mathrm{Ad}_{f_{\lambda(i),i}^{-1}} \cdots \mathrm{Ad}_{f_{j,\lambda^{p-1}(i)}^{-1}} & \text{if } \lambda^p(i) = j \text{ for some } p \\ \mathbf{I}_{6\times 6} & \text{if } i = j \\ \mathbf{0}_{6\times 6} & \text{otherwise} \end{cases} \quad (59)$$

From equation (44) we know that $\Gamma_\lambda = \mathbf{I} - \mathbf{G}_\lambda^{-1}$, so

$$\Gamma_\lambda \mathbf{G}_\lambda = \mathbf{G}_\lambda - \mathbf{I} \quad (60)$$

Now it is straightforward to get the $O(n)$ recursive algorithm, which is specified in equations 18 ~ 35.

Appendix C: Time Derivative of Adjoint Mapping

$$\begin{aligned} \frac{d}{dt}\mathrm{Ad}_{\mathbf{T}^{-1}} &= \begin{bmatrix} \dot{\mathbf{R}}^T & 0 \\ -\dot{\mathbf{R}}^T [\mathbf{t}] - \mathbf{R}^T [\dot{\mathbf{t}}] & \dot{\mathbf{R}}^T \end{bmatrix} \\ &= \begin{bmatrix} -[\omega]\mathbf{R}^T & 0 \\ [\omega]\mathbf{R}^T [\mathbf{t}] - [v]\mathbf{R}^T & -[\omega]\mathbf{R}^T \end{bmatrix} \\ &= -\begin{bmatrix} [\omega] & 0 \\ [v] & [\omega] \end{bmatrix} \begin{bmatrix} \mathbf{R}^T & 0 \\ -\mathbf{R}^T [\mathbf{t}] & \mathbf{R}^T \end{bmatrix} \\ &= -\mathrm{ad}_{\mathbf{v}} \mathrm{Ad}_{\mathbf{T}^{-1}} \end{aligned}$$

References

[Coh92] COHEN M. F.: Interactive spacetime control for animation. *SIGGRAPH Comput. Graph. 26*, 2 (1992), 293–302.

[Fea83] FEATHERSTONE R.: The calculation of robot dynamics using articulated-body inertias. *International Journal of Robotics Research 2* (May 1983), 13–30.

[Fea87] FEATHERSTONE R.: *Robot Dynamics Algorithm*. Kluwer Academic Publishers, Norwell, MA, USA, 1987. Manufactured By-Publishers, Kluwer Academic.

[GHDS03] GRINSPUN E., HIRANI A. N., DESBRUN M., SCHRÖDER P.: Discrete shells. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 62–67.

[GL09] GUENTER B., LEE S.-H.: *Symbolic Lagrangian Multibody Dynamics*. Tech. rep., Microsoft Research, 2009.

[Gra98] GRASSIA F. S.: Practical parameterization of rotations using the exponential map. *J. Graph. Tools 3*, 3 (1998), 29–48.

[Gue07] GUENTER B.: Efficient symbolic differentiation for graphics applications. *ACM Trans. Graph. 26*, 3 (2007), 108.

[Kas92] KASS M.: Condor: constraint-based dataflow. *SIGGRAPH Comput. Graph. 26*, 2 (1992), 321–330.

[Kha87] KHATIB O.: A unified approach for motion and force control of robot manipulators: The operational space formulation. *Robotics and Automation, IEEE Journal of 3*, 1 (February 1987), 43–53.

[KL00] KANE T., LEVINSON D.: *Dynamics Online: Theory and Implementation with AUTOLEV*. OnLine Dynamics, Inc., 2000.

[KM98] KOKKEVIS E., METAXAS D.: Efficient dynamic constraints for animating articulated figures. *Multibody System Dynamics 2*, 2 (1998), 89–114.

[KP03] KRY P. G., PAI D. K.: Continuous contact simulation for smooth surfaces. *ACM Trans. Graph. 22*, 1 (2003), 106–129.

[LT08] LEE S.-H., TERZOPOULOS D.: Spline joints for multibody dynamics. *ACM Trans. Graph. 27*, 3 (2008), 1–8.

[PBP95] PARK F. C., BOBROW J. E., PLOEN S. R.: A lie group formulation of robot dynamics. *Int. J. Rob. Res. 14*, 6 (1995), 609–618.

[Plo97] PLOEN S. R.: *Geometric Algorithms for the Dynamics and Control of Multibody Systems*. PhD thesis, University of California, Irvine, 1997.

[PSE*00] POPOVIĆ J., SEITZ S. M., ERDMANN M., POPOVIĆ Z., WITKIN A.: Interactive manipulation of rigid body simulations. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 209–217.

[Reu76] REULEAUX F.: *The Kinematics Of Machinery: Outlines Of A Theory Of Machines*. Kessinger Publishing L.L.C., Whitefish, MT, USA, 1876.

[Rod87] RODRIGUEZ G.: Kalman filtering, smoothing, and recursive robot arm forward and inverse dynamics. *Robotics and Automation, IEEE Journal of 3*, 6 (December 1987), 624–639.

[SR94] SHERMAN M., ROSENTHAL D.:, 1994. http://www.sdfast.com/.