Adaptive Dynamics of Articulated Bodies: Implementation Details

Stephane Redon[*] Nico Galoppo[†] Ming C. Lin[‡]

Department of Computer Science
University of North Carolina at Chapel Hill

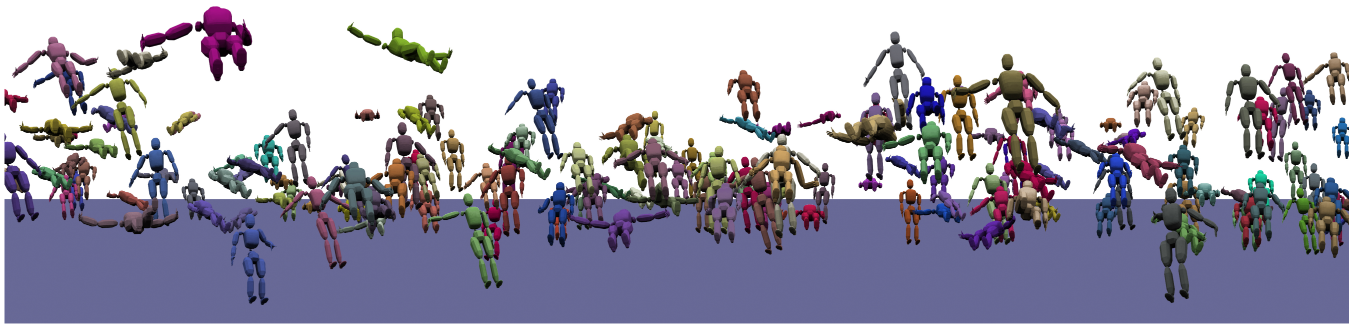

Figure 1: **Adaptive dynamics of articulated characters**. In this complex scene, 200 human characters, represented by $17,800$ rigid bodies and $19,000$ degrees of freedom, are suddenly pushed away from the camera due to applied forces. Our adaptive dynamics algorithm allows an animator to progressively reduce the number of simulated joints in the characters as their distance to the camera increases, while automatically determining *which* joints should be animated to best approximate the characters motion. Depending on the total amount of simplification specified by the animator, a potentially significant speed-up can be achieved over typical linear-time forward dynamics algorithms.

1 Introduction

We provide some implementation details related to the paper "Adaptive Dynamics of Articulated Bodies" [Redon et al. 2005]. Thanks to the recursive nature of the DCA [Featherstone 1999a; Featherstone 1999b], upon which our algorithm is built, the code is highly modular and an object-oriented language is highly appropriate. We have implemented the algorithm in C++.

Modularity: Two fundamental classes of the dynamics library are `cArticulatedBody` and `cJoint`. An articulated body is recursively defined as a pair of articulated bodies connected by a joint, and thus `cArticulatedBody` contains pointers to two children articulated bodies and a `cJoint` object. The class `cJoint` is actually a virtual base class from which specialized joint classes are derived (*e.g.* `cRevoluteJointZAxis` to implement a 1-dof rotational around a local z axis). This code specialization allows us to optimize the code based upon the type of joint and the known dimension of the motion subspace (*e.g.* using loop unrolling).

Variables mappings: When two articulated bodies A and B are connected to form a new articulated body C, their handles are renamed to simplify the various dynamics expressions. In order to ease the implementation as well, a `cArticulatedBody` object stores pointers which emulate the variables renaming. For example, if H_6^A in A becomes H_2^A in C, the variable `b2A` in C is a pointer to `b[6]` in A, which contains the bias acceleration $\mathbf{b}_6^A$ of handle H_6^A. The renaming pointers are assigned only once, when a new articulated body is created from two others.

Caching: Depending on the amount of memory available to the dynamics library, many quantities that are used in more than one function can be cached (*e.g.* $\mathbf{W}$ and γ).

2 Coordinates transformations

All dynamics quantities are expressed in the reference frame of their respective handle, and the matrices and tensors can thus have several reference frames associated to them. For example, $\mathbf{\Phi}_{12}^A$ is applied to a force expressed in the reference frame of handle H_2^A to obtain an acceleration expressed in the reference frame of handle H_1^A. This requires us to transform some quantities before using them. For example, $\mathbf{\Phi}_2^A$ and $\mathbf{\Phi}_1^B$ have to be in the same pair of bases before adding them and compute $\mathbf{V} = (\mathbf{\Phi}_2^A + \mathbf{\Phi}_1^B)^{-1}$. A hierarchical state representation is used to compute and store the necessary coordinates transformation matrices [Redon and Lin 2005]. The spatial transformation rules are given in [Featherstone 1999a], and can be highly optimized for each type of joint, thanks to the code specialization allowed by the joint class hierarchy.

3 Linear Coefficients Tensors

We have introduced rank-three and rank-four *linear coefficients tensors* to obtain the values of the linear dynamics coefficients in constant time for the nodes in the passive region [Redon et al. 2005]. For example, the composite bias acceleration $\mathbf{b}_1^C$ of a passive node C is $(\mathbf{b}_1^C)_a = (\mathbf{B}_1^C)_{abc}(\mathbf{v}_1^C)_b(\mathbf{v}_1^C)_c$, where $\mathbf{v}_1^C$ is the composite velocity of H_1^C. In order to efficiently perform this evaluation, it is worth noticing that this amounts to evaluating six quadratic polynomials per handle. Computing and storing these polynomials instead of the tensors $\mathbf{B}_1^C$ allows for a significant speedup in the computation of the linear dynamics coefficients. Similarly, the rank-four tensor $\mathbf{E}^C$ can be replaced by one quartic in the components of $\mathbf{v}^C$ to obtain η^C. The various polynomial operations can easily be implemented using a combination of formal calculus software and IDE macros.

References

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.*

REDON, S., AND LIN, M. C. 2005. An efficient, error-bounded approximation algorithm for simulating quasi-statics of complex linkages. In *Proceedings of ACM Symposium on Solid and Physical Modeling*.

REDON, S., GALOPPO, N., AND LIN, M. C. 2005. Adaptive dynamics of articulated bodies. In *ACM Transactions on Graphics (SIGGRAPH 2005 Proceedings)*.

[*]Stephane Redon is now at INRIA. E-mail: stephane.redon@inria.fr.
[†] E-mail: nico@cs.unc.edu. [‡] E-mail: lin@cs.unc.edu