Fast Iterative Refinement of Articulated Solid Dynamics

François Faure

Fast Refinable Equation Solution for Articulated Solid Dynamics

François Faure
francois@cg.tuwien.ac.at

November 12, 1998

A new dynamics algorithm for articulated solid animation is presented. It provides enhancements of computational efficiency and accuracy control with respect to previous solutions. Iterative refinment allows us to perform interactive animations which could be only computed off-line using previous methods.

The efficiency results from managing two sets of constraints associated with the kinematic graph, and proceeding in two steps. First, the acyclic constraints are solved in linear time. An iterative process then reduces the closed loop errors while maintaining the acyclic constraints. This allows the user to efficiently trade-off accuracy for computation time. We analyze the complexity, and investigate practical efficiency compared with other approaches. In contrast with previous research, we present a single method which is computationally efficient for acyclic bodies as well as for mesh-like bodies.

The accuracy control is provided by the iterative improvement performed by the algorithm, and also from the existence of two constraint priority levels induced by the method.

Used in conjunction with a robust integration scheme, this new algorithm allows the interactive animation of scenes containing a few thousand geometric constraints including closed loops. It has been successfully applied to real-time simulations.

1 Introduction

Articulated solids are widely used in computer animation, since they allow us to model character skeletons as well as lifeless objects. Among the available techniques, physically-based animation is appealing since it allows the

automatic computation of realistic motions. However, the computational complexity of the currently available methods has not allowed the interactive animation of large scenes. As a result, physically-based articulated bodies have not been extensively used in interactive applications such as virtual reality, and tuning an animation for movie creation is still a tedious task. In this paper, we propose a new method which efficiently trades off accuracy for speed. Integrated in a virtual reality environment, it allows us to physically interact with scenes containing hundreds or thousands of geometric constraints.

A major difficulty with physically-based articulated solid animation comes from closed loops, because closed loops induce geometrical constraints which have to be solved explicitly. Typically, constraining the end-effector of a robot arm to follow a given path creates a closed loop. Equations have to be written and solved in order to meet the constraints at the loop closure. Consider a character climbing on a rope ladder. We model the two strings of the ladder using articulated bars, and the steps using rigid bars. Each step creates a closed loop, since it is attached to both strings. Depending on the length of the ladder, the scene can include an arbitrarily large number of closed loops. The computational efficiency of the currently available methods strongly depend on the number of constraints and on the relative number of loop closures. Some methods are efficient when applied to scenes including few closed loops[9, 5], and become inefficient as the number of closed loops increases. Other methods, e.g. [10] perform comparatively well with many closed loops but are relatively inefficient when applied to near acyclic scenes. In contrast, our new approach is efficient for any kind of scene. It handles acyclic constraints in linear time using a well known approach, and performs an iterative error minimization at the loop closures. We show that unlike previous methods, the complexity smoothly ranges from linear to quadratic in terms of the total number of constraints, according to the relative number of closed loops.

Our new approach enhances previous methods[9, 5] with an important feature, namely, the control of the computation time. Two criteria, precision or number of iterations, can be used to terminate the computation. The ability of bounding the computation time is the key point for the interactive simulation of complex environments. Some experiments presented in section 4.2 illustrate the computational efficiency obtained by adjusting the desired precision to the actual user needs. Interactivity is especially important for virtual reality applications. We noticed that when interacting with articulated bodies, the user by far prefers getting quick approximate responses, than waiting several seconds until a precise motion has been computed. We

present applications to a variety of scenes in section 5.

The remainder of this paper is organized as follows. In the next section, we summarize previous work and provide background on dynamics equations. In section 3, we present our new algorithm. Theoretical and practical comparisons with previous approaches are shown in section 4. Applications are presented in section 5.

2 Background

2.1 Previous work

The field of articulated solids have been thoroughly investigated by mechanical engineering and robotics research. Paul studied the kinematics of manipulators[14]. Numerous algorithms for direct or inverse dynamics have been proposed[21, 9, 3, 13] and a variety of simulation systems are available[16].

In the field of animation synthesis, Wilhelms and Barsky[19] presented a general method to compute the motion of an articulated structure subject to external forces. Armstrong and Green[1] proposed a fast algorithm for acyclic structures including rotational joints. Isaacs and Cohen[11] showed how to simultaneously handle given forces and given motions in an acyclic body. They extended this approach to complex kinematic constraints such as closed loops[12]. Barzel and Barr[6] presented a dynamics approach which allows the automatic assembly of articulated structures. Witkin et al. enhanced interactivity by creating and deleting objects and constraints on-the-fly[20]. Baraff presented a method solving the inequality and complementarity constraints of the Coulomb friction model[4]. Gleicher developed a general constraint-based approach for interactive scene modeling and animation[10]. Baraff presented a fast algorithm for acyclic structures along with an extension to closed loops [5].

Currently, optimality of articulated body dynamics computation remains an open question. If we use a to denote the number of acyclic constraints and l the number of loop constraints, Featherstone's and Baraff's methods[9, 5] run in time $O(a + al + l^3)$, while Gleicher's method[10] runs in time $O(a^2 + al + l^2)$. Featherstone handles acyclic scenes using a recursive algorithm which avoids redundant computations. Baraff exploits matrix structure and sparsity. Both methods handle loop closures by creating and solving an additional equation system which is responsible for the terms al and l^3 in the complexities. These approaches are thus efficient when applied to near acyclic structures but become inefficient as the number of closed loop

increases. In contrast, Gleicher[10] exploits matrix sparsity without considering the graph structure, and performs an iterative solution. Since no cubic term is involved in the time complexity, this approach is well suited for structures including a large number of closed loops.

2.2 Problem formulation

Numerous ways of formulating the dynamics of articulated bodies can be found in standard texts, e.g. [21, 9, 16]. The formulation we use, which has been extensively presented by Baraff[5], is comparatively simple and involves only sparse matrices, which is an important feature when dealing with large structures.

In the remaining of the paper, we use bold letters to denote vector and matrices related to the entire structure (global values). Global vectors are represented by lower-case letters, whereas global matrices are represented by upper-case letters. A summary of the notations used in this paper is provided in appendix A.

Articulated structures are made of bodies and joints that we can represent by the nodes of a kinematic graph. In this paper, we focus on rigid bodies. The joints are associated with geometric constraints which must remain satisfied over the animation. As an example, binding the endpoints P_1 and P_2 of two solids results in a geometric constraint $P_1P_2 = 0$, which can be represented by three scalar constraints, each of them associated with an independent direction. We can gather all the scalar constraints of a structure within a vector equation $\boldsymbol{g}(\boldsymbol{q},\mathrm{t}) = \boldsymbol{0}$ where $\boldsymbol{q}$ represents the coordinates of the bodies. We focus exclusively on absolute coordinates since they allow the use of sparse matrices[5].

In physically-based animation, forces are responsible for the motion of the bodies. Solids obey Newton-Euler's law $M\dot{v} = f$ where M is 6×6 symmetric positive definite matrix and f a six-dimensional vector representing the net force and torque applied to the solid. This results in a global equation $\boldsymbol{M}\dot{\boldsymbol{v}}_{\boldsymbol{s}} = \boldsymbol{f}_{\boldsymbol{s}}$ where $\boldsymbol{M}$ is a block-diagonal matrix, $\dot{\boldsymbol{v}}_{\boldsymbol{s}}$ and $\boldsymbol{f}_{\boldsymbol{s}}$ represent the accelerations and net forces applied to the solids. The subscript s denotes vectors associated with the solids. Their dimension is six times the number of solids.

Constraint forces are necessary to maintain the constraints. They act along their associated geometrical directions. Since their magnitudes depend on the geometry, velocity and given forces applied to the solids, we have to compute them at each time step. We represent the magnitudes by Lagrange multipliers of the geometric constraints, gathered in a global vector $\boldsymbol{\lambda}$. The

computation of these values, which is the main topic of this paper, is detailed in section 3. The net force $\boldsymbol{f_s}$ applied to the solids is the sum of the actions of given forces $\boldsymbol{f_{ext}}$ and of the Lagrange multipliers: $\boldsymbol{f_s = f_{ext} + J^T \lambda}$ where $\boldsymbol{J}$ is the Jacobian matrix of the constraints, and T denotes matrix transposition.

The Jacobian matrix $\boldsymbol{J}$ codes for the dependency of the constraint values on the motions of the solids[5]. The relative velocity $\boldsymbol{v_c}$ along the constraint directions is related to velocity of the solids by the relation $\boldsymbol{\delta v_c = J v_s}$. We use the subscript c to denote vectors associated with constraints. The dimension of these vectors is the number of scalar constraints in the structure. Matrix $\boldsymbol{J}$ is a rectangular matrix, with sparse block structure since each joint acts only on two solids.

The physical motion of an articulated solid structure can be mathematically represented by the following differential algebraic equation:

$$\boldsymbol{\dot{q} = Q v} \tag{1}$$

$$\boldsymbol{M \dot{v}_s = f_{ext} + J^T \lambda} \tag{2}$$

$$\boldsymbol{g(q,}t\boldsymbol{) = 0} \tag{3}$$

Equation (1) represents the relation between the velocities $\boldsymbol{v}$ and the coordinates $\boldsymbol{q}$, since rotations can be modeled using different sets of parameters. Equation (2) represents Newton-Euler's law, and equation (3) the geometric constraints. In order to compute the constraint forces $\boldsymbol{\lambda}$, we differentiate twice the geometric equation, to obtain:

$$\boldsymbol{\ddot{g} = 0 = J \dot{v}_s + a_v} \tag{4}$$

where $\boldsymbol{a_v}$ is the velocity-dependent part of the relative accelerations. The nonholonomic constraints, if any, have to be included in the differentiated equation (4). Combining equations (2) and (4) allow us to write the dynamics equation:

$$\begin{pmatrix} \boldsymbol{M} & \boldsymbol{-J^T} \\ \boldsymbol{-J} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta \dot{v}_s} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} 0 \\ \boldsymbol{-b} \end{pmatrix} \tag{5}$$

where $\boldsymbol{b}$ accounts for external forces and velocity-dependent accelerations, and represents the relative acceleration error that would result form null constraint forces. We call $\boldsymbol{b}$ the initial constraint error. Vector $\boldsymbol{\delta \dot{v}_s}$ represents the acceleration correction due to the constraint forces. Adding it to the unconstrained accelerations $\boldsymbol{M^{-1} f_{ext}}$ provides the constrained accelerations. The first line of equation (5) is Newton's law restricted to

the constraint forces. The second line represents the kinematic constraints. Solving this linear system allows us to compute the accelerations of the solids, then to integrate over a time step dt.

Numerical integration induces errors resulting in the possible violation of the geometric equation (3) at time $t + dt$. A widely used method to restrict this drift within acceptable bounds is to bias the constraint errors in function of the current position errors and velocity errors[7, 6, 12]. This results in a different $\boldsymbol{b}$ in the dynamics equation (5). Other stabilization methods consist in projecting after integration the velocities and the positions to subspaces compatible with the constraints[2, 8]. In this case, equations similar with the dynamics equation (5) have to be solved, except that they deal with velocity changes or small displacements instead of accelerations. The dynamics equation has also been used to normalize kinematic equations[10]. In all these cases, the efficient solution of the dynamics equation (5) is a key point for fast animation, and this paper focuses on this topic. In the remaining of the paper, we consider some initial error and the Lagrange multipliers necessary to cancel it, regardless of whether the error represent acceleration, velocity or small displacement.

Baraff[5] presents an algorithm to solve equation (5) in linear time in the case of an acyclic structure (without closed loops). It is based on the linear-time decomposition of the matrix in the form $\boldsymbol{LDL^T}$ where $\boldsymbol{L}$ is a sparse lower triangular matrix and $\boldsymbol{D}$ is a block-diagonal matrix. Once this decomposition is achieved, the dynamics equation can be solved in linear time for any initial error. He then presents an extension to closed loops where a reduced equation system related to closed loops only is computed and solved. Featherstone proposed a similar approach[9]. It is efficient when applied to scenes containing a small number of closed loops.

Another algorithm has been presented by Bae and Haug[3]. A topological analysis of the loops in the kinematic graph is used to perform a recursive computation of the solution. This method is not straightforwardly extendible to an iterative approach.

The mass matrix $\boldsymbol{M}$ of the dynamics equation is easily invertible since it is block-diagonal and each block is a symmetric positive definite matrix. It is thus possible to perform a substitution of the first line into the second and solve a reduced-size equation system:

$$\boldsymbol{JM^{-1}J^T\lambda = b} \tag{6}$$

Baraff[5] points out that though $\boldsymbol{J}$ and $\boldsymbol{M}$ are sparse, the product $\boldsymbol{JM^{-1}J^T}$ may be dense. Gleicher[10] solves equation (6) using a conju-

gate gradient algorithm. This algorithm only addresses the matrix through its product with a vector. The product can be performed in three steps, allowing the use of matrix sparsity. This quadratic-time approach is efficient applied to structures including a large number of closed loops.

3 The iterative structured solution

We present our method for solving the dynamics equation (5), that we call the iterative structured solution. It aims to gather the advantages of the previously presented methods. The principle of the method is first intuitively explained. The associated derivation of the dynamics equation is then presented. We finally provide additional implementation detail. As previously mentioned, the method can be applied to compute corrections of accelerations, velocity or position. In the following, the term *motion* and the vector $\boldsymbol{x_s}$ denote any of these concepts, and the term *force* and the vector $\boldsymbol{\lambda}$ denote the corresponding dynamic actions.

3.1 Principle

The basic idea of our method is to solve the closed loops constraints iteratively, while maintaining the acyclic constraints met. Termination occurs as soon as a given precision at the closed loop constraints is reached, or sooner if we trade-off accuracy for speed. The method is illustrated in figure 1. For clarity, small displacements are used to represent the motions.

We start from an initial error that we want to cancel using Lagrange multipliers, by solving the dynamics equation (5). This state is represented in figure 1.a, where the arrow represents any external influence responsible for an initial error.

In the first step, we cancel the initial acyclic constraint errors. The closed loop errors are updated according to the motion corrections. This state is illustrated in figure 1.b. At this point, all external influences have been taken into account and the remaining question is to meet the closed loop constraints.

In the second step, we perform a constrained minimization of the updated closed loop errors. The search space is restricted to constraint forces which induce no error in the acyclic constraints. All closed loop constraints are simultaneously handled by the global minimization. Figure 1.c illustrates a step of this iterative solution. The method terminates when a given precision or number of iterations is reached, as illustrated in figure 1.d.

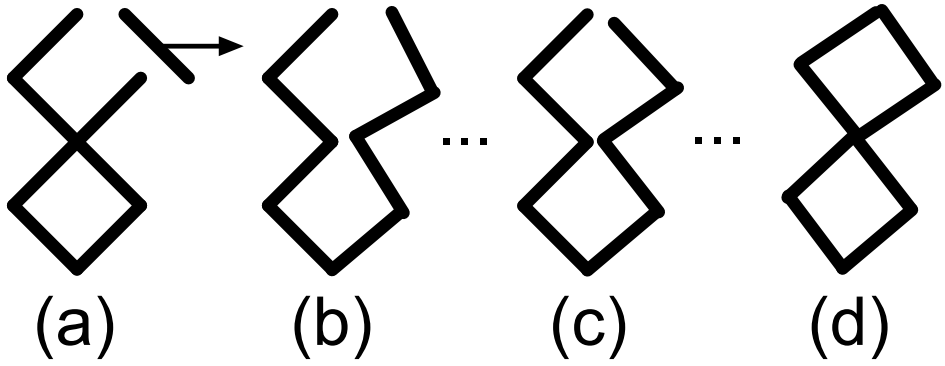


Figure 1: The iterative structured solution. In (a), external forces represented by the arrow are responsible for constraint errors. In (b), the acyclic errors have been canceled, the constrained minimization of the closed loop errors can start. An intermediate step of the constrained minimization is shown in (c). The minimization terminates in (d).

3.2 Derivation

We call $\boldsymbol{J_a}$ the Jacobian matrix related to the acyclic constraints, and $\boldsymbol{J_l}$ the Jacobian matrix related to the loop constraints. The dynamics equation (5) of an acyclic body, or equivalently:

$$\begin{pmatrix} \boldsymbol{M} & \boldsymbol{J_a^T} \\ \boldsymbol{J_a} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{x_s} \\ \boldsymbol{-\lambda_a} \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{b_a} \end{pmatrix} \tag{7}$$

can be solved by performing the following matrix decomposition:

$$\begin{pmatrix} \boldsymbol{M} & \boldsymbol{J_a^T} \\ \boldsymbol{J_a} & \boldsymbol{0} \end{pmatrix} = \boldsymbol{LDL^T} \tag{8}$$

where $\boldsymbol{L}$ is a sparse lower triangular matrix and $\boldsymbol{D}$ is a block-diagonal matrix. Once this decomposition is achieved, the motion corrections and constraint forces of the acyclic articulated body can be computed in linear time for any initial error $\boldsymbol{b_a}$.

We now show how this can be used to solve the closed loop constraints. Separating acyclic and closed loop constraints turns equation (6) into:

$$\begin{pmatrix} \boldsymbol{J_a} \\ \boldsymbol{J_l} \end{pmatrix} \boldsymbol{M^{-1}} \begin{pmatrix} \boldsymbol{J_a^T} & \boldsymbol{J_l^T} \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda_a} \\ \boldsymbol{\lambda_l} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b_a} \\ \boldsymbol{b_l} \end{pmatrix} \tag{9}$$

which can be written as:

$$\begin{pmatrix} \boldsymbol{A_{aa}} & \boldsymbol{A_{al}} \\ \boldsymbol{A_{la}} & \boldsymbol{A_{ll}} \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda_a} \\ \boldsymbol{\lambda_l} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b_a} \\ \boldsymbol{b_l} \end{pmatrix} \tag{10}$$

where $\boldsymbol{A_{xy}} = \boldsymbol{J_x M^{-1} J_y^T}$ for x and y in $\{a, l\}$. The acyclic matrix decomposition (8) allows the computation of the product of $\boldsymbol{A_{aa}^{-1}}$ with any appropriate-sized vector in linear time. It is therefore feasible to perform a substitution within equation (10) which leads to the following equivalent equation system:

$$\boldsymbol{\lambda_a} = \boldsymbol{A_{aa}^{-1}(b_a - A_{al}\lambda_l)} \tag{11}$$

$$\boldsymbol{(A_{ll} - A_{la}A_{aa}^{-1}A_{al})\lambda_l} = \boldsymbol{b_l - A_{la}A_{aa}^{-1}b_a} \tag{12}$$

Equation (12) can be written in a more compact form as:

$$\boldsymbol{\hat{A}\lambda_l = \hat{b}} \quad \textit{with } \boldsymbol{\hat{A} = A_{ll} - A_{la}A_{aa}^{-1}A_{al}},\ \boldsymbol{\hat{b} = b_l - A_{la}A_{aa}^{-1}b_a} \tag{13}$$

The solution of equation (13) provides the closed loop forces $\boldsymbol{\lambda_l}$, allowing the computation of the right-hand term of equation (11). The acyclic forces $\boldsymbol{\lambda_a}$ can in turn be computed using the decomposition of the acyclic matrix $\boldsymbol{A_{aa}}$. The motion corrections are straightforwardly deduced from the constraint forces $\boldsymbol{\lambda_a}$ and $\boldsymbol{\lambda_l}$.

Equation (13) is a system of l equations and l unknowns, where l is the number of scalar closed loop constraints. The reduced constraint matrix $\boldsymbol{\hat{A}}$ allows the computation of the relative motions along the constrained closed loop directions in response to given forces along these directions, *taking into account the underlying acyclic structure*. It can be seen as the response function of the acyclic structure to forces along the closed loop constraint directions. Vector $\boldsymbol{\hat{b}}$ is the updated closed loop constraint computed at the end of the first step of our method, as described in the previous section. Consequently, performing an iterative minimization of $\|\boldsymbol{\hat{A}\lambda_l - \hat{b}}\|$ over $\boldsymbol{\lambda_l}$ achieves the constrained minimization of the second step of our method. We perform this minimization using the biconjugate gradient algorithm[15]. This algorithm addresses the matrix (or its transpose) only through its product with a vector, which we compute step by step in linear time as explained in the following section.

3.3 Implementation

The sparse matrices involved in our method are made of blocks associated with solids and joints. The blocks are therefore stored in the the nodes of the kinematic graph. A similar decomposition of the global vectors is achieved.

The key point of the iterative minimization is the ability of computing efficiently the product $\boldsymbol{x}_l = \hat{\boldsymbol{A}}\boldsymbol{\lambda}_l$ where $\boldsymbol{x}_l$ represents the relative motion along the closed loop constraint directions due to the closed loop constraint force $\boldsymbol{\lambda}_l$, taking into account the reaction of the acyclic constraints. Expanding the decomposition of matrix $\hat{\boldsymbol{A}}$ in equation (13) gives:

$$\hat{\boldsymbol{A}} = \boldsymbol{J}_l(1 - \boldsymbol{M}^{-1}\boldsymbol{J}_a^T\boldsymbol{A}_{aa}^{-1}\boldsymbol{J}_a)\boldsymbol{M}^{-1}\boldsymbol{J}_l^T \tag{14}$$

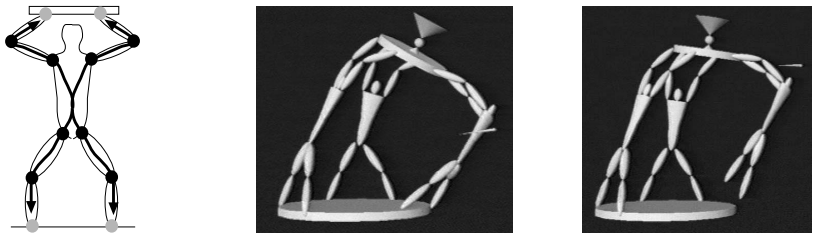
where $\mathbf{1}$ represents the identity. This allows us to compute the acyclic response using vector products computed from the right to the left. Matrix $\boldsymbol{A}_{aa}^{-1}$ is not computed explicitly, we use its decomposition to directly compute its product with a vector instead.

We define a procedure *acyclic_solve*($\boldsymbol{b}_a$,$\boldsymbol{x}_s$) which computes in linear time the solution of the acyclic equation system (eq. 7) for any given $\boldsymbol{b}_a$, and returns the corresponding motion correction $\boldsymbol{x}_s$. The solution of the equation system is not only the acyclic constraint forces $\boldsymbol{\lambda}_a$ but also the associated motions $\boldsymbol{x}_s$. This provides us directly with the product $\boldsymbol{M}^{-1}\boldsymbol{J}_a^T\boldsymbol{A}_{aa}^{-1}\boldsymbol{b}_a$ involved in $\hat{\boldsymbol{A}}$ (eq.14). Pseudocode for this product can be found in[5], except that the values of $\boldsymbol{x}_s$ have to be extracted during the final step instead of the values of $\boldsymbol{\lambda}_a$.

We define a procedure *acyclic_response*($\boldsymbol{f}_l$,$\boldsymbol{x}_l$) which computes the product $\boldsymbol{x}_l = \hat{\boldsymbol{A}}\boldsymbol{f}_l$. The procedure uses two auxiliary vectors $\boldsymbol{x}_0$ and $\boldsymbol{b}_0$, and consists of five steps:

acyclic_response($\boldsymbol{f}_l$,$\boldsymbol{x}_l$)
$\boldsymbol{x}_0 = \boldsymbol{M}^{-1}\boldsymbol{J}_l^T\boldsymbol{f}_l$
$\boldsymbol{x}_l = \boldsymbol{J}_l\boldsymbol{x}_0$
$\boldsymbol{b}_0 = \boldsymbol{J}_a\boldsymbol{x}_0$
acyclic_solve($\boldsymbol{b}_0$,$\boldsymbol{x}_0$)
$\boldsymbol{x}_l \mathrel{-}= \boldsymbol{J}_l\boldsymbol{x}_0$

The procedure *acyclic_response* is given to the biconjugate gradient algorithm as a black box matrix product procedure. The product with the transposed matrix, also required, is achieved similarly since $\hat{\boldsymbol{A}}$ is symmetric. Due to numerical roundoff, the algorithm sometimes provides an incorrect result. We overcome this difficulty by verifying that the computed forces result in the given motions. If necessary, a new solution is started using the wrong result as a new initial guess. In practice, we never encountered wrong solutions repeatedly.



(a) (b) (c)

Figure 2: Constraint hierarchy. In (a), the kinematic structure of each character. In (b), secondary constraints bind the hands and the feet of the characters. In (c), only secondary constraints are violated.

3.4 Constraint hierarchy

The constrained minimization of the closed loop errors induces a constraint hierarchy. In any case, the acyclic constraints are met up to machine precision. This allows the user to define an acyclic graph of prioritary constraints such as in figure 2a, where acyclic and closed loop joints are represented as black and gray circles, respectively. The kinematic graph of the character is shown using arrows, from root to leaves. In 2b, a user interactively moves a structure made out of three of these characters. The white cone shows the position of a 3D tracker manipulated by the user. A closed loop joint has been created between the tracker and the body of one character. In 2c, the constraints have become inconsistent but each character remains internally connected.

4 Efficiency

We now address the question of the efficiency of our approach for interactive computer animation. We first compare the theoretical time and space complexities of different approaches. We then compare practical results over a class of examples.

Implementing and testing all methods proposed in the literature is out of the scope of our work. Nonetheless, we propose a rough classification of the methods and investigate the efficiency of one member of each class. We consider only methods able to handle closed loops and distinguish the following classes.

Some methods compute and solve a dense matrix related to both acyclic and closed loop constraints, e.g. [21, 19, 11, 12, 6]. Using our dynamics for-

	time complexity $O()$	space complexity $O()$
full dense	$(a+l)^3$	a^2+al+l^2
full iterative	a^2+al+l^2	$a+l$
structured dense	$a+al+l^3$	$a+l+l^2$
our method	$a+al+l^2$	$a+l$

Table 1: comparison between different methods for solving a system including a acyclic constraints and l closed loop constraints.

mulation, this corresponds to explicitly computing and solving the equation system (6). We call this approach *full dense solution*. Other methods solve equation (6) using a conjugate gradient algorithm, e.g. [10]. This allows the use of matrix sparsity and the iterative solution of the equation. We call this approach *full iterative solution*. Another approach consists in using acyclic solutions to compute a dense equation system related to the closed loop constraints only [9, 5]. We call this approach *structured dense solution*. Finally, our constrained minimization of the closed loop errors is an iterative method exploiting linear-time solution for acyclic constraints.

4.1 Theoretical complexity

We consider an articulated structure including a acyclic scalar constraints and l closed loop scalar constraints. The first step of our method, which deals only with acyclic constraints, is achieved in linear time. Each acyclic response computation involves sparse matrix products and one linear acyclic solution. It is thus achieved in time $O(a+l)$. The number l of unknowns is the theoretically maximal number of iterations of the biconjugate gradient algorithm, each of them involving two acyclic response computations. Table 1 summarizes the complexities of the approaches.

The theoretical results suggest that the iterative structured solution is the most efficient approach for the animation of large scenes. Its time complexity smoothly ranges from linear to quadratic in terms of the relative amount of closed loop constraints. Its memory requirement is proportional to the size of the articulated structure. Moreover, similarly with the full iterative method, the progressive improvement of the solution through iterations allows bounding either precision or computation time.

4.2 Practical efficiency

In order to investigate practical efficiency, we use a parametrizable rope ladder swinging in a gravity field as a case study. Two strings, modeled by articulated bars, are bound by additional orthogonal bars creating closed loops (fig. 3). This allows us to create scenes with various size and proportion of closed loops. All the joints are point-to-point constraints (spherical joints) including three scalar constraints. The whole articulated structure is made of repeated patterns, and parameterized by the size of the pattern and the amount of patterns.

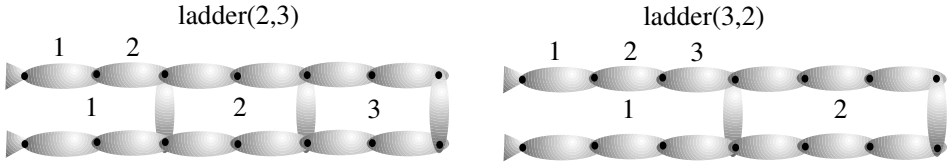


Figure 3: Two examples of the ladder structure. The bars are connected by joints represented by dots. On the left of the ladder, the bars are connected to fixed points represented by cones.

Comparing iterative and direct algorithms in terms of computation time is somewhat arbitrary, since the result depends on the precision desired in the iterative solutions. Reaching high precision, say, kinematic errors of magnitude $1.0e^{-6}$ when animating unit length solids, may require an experienced user, especially using iterative algorithms. However, as far as computer animation is concerned, much lower precision is necessary to get invisible errors. Here we investigate the application of our method to computer animation. Consequently, using unit-length bars, we terminate the iterations as soon as a precision of $1.0e^{-3}$ is reached for each kinematic constraint, providing us with the same visual result as a higher precision would.

Table 2 summarizes the numerical results, measured on an SGI O2 workstation, with a 180 MHz R5000 processor. The dense matrix solutions are achieved using Cholesky decomposition, which is to our knowledge the fastest among all the available algorithms for such dense matrices. This is not necessarily a realistic choice, because this method fails on indefinite matrices, which occur quite frequently in practice. In such a case, other matrix decompositions should be performed.

Our approach applied to this example tends to run up to 10 or 20 times

ladder	(12,1)	(12,4)	(1,48)	(1,96)	(6,96)
a, l	75, 3	300, 12	432, 144	864, 288	3744, 288
full dense	16	405	1733	13034	
full iterative	4.4	50	121	370	2501
structured dense	3.1	21	323	1350	4612
our method	4.5	15	35	68	208

Table 2: Computation times, in *ms*, of different solution algorithms. The parameters used to generate various ladder structures appear on the top line. The number of acyclic and closed loop constraints appear on the second line.

faster than the other methods. While using other examples for comparison may result in fewer improvement, our approach tends to be the most efficient as the size of the scene grows for every example we tested. The number of iterations grows less than linearly along with the size of the scene. As a result, the practical complexity is less than quadratic. This can be explained by the reduced interdependence between closed loop constraints located far from each other. A more sophisticated analysis of the convergence of the conjugate gradient algorithm can be found in [17].

An additional feature of the iterative methods, not exploited in this comparison, is the ability to start from an initial guess. Since in most cases the forces vary slowly from one simulation step to another, except when collisions occur, starting from the previous result can greatly improve the efficiency of iterative methods[18].

5 Applications

We apply our method to a VR system. A joint between a 3d tracker and its nearest solid is created/deleted when clicking/releasing a button. Aside from the dynamics solution algorithm, another important element of an animation system is the integration scheme, which is out of the scope of this paper. Our integration scheme is related to the method of Asher and Chin[2, 8], briefly described in section 2.2. The length of the time step is computed at each entry in the main loop, using the machine clock. Rendering takes about half the total computation time in the following examples.

Figure 4 shows a scene including two sea weeds and fishes. Each sea weed is made of 36 solids. This scene is purely acyclic when we start the animation, and the dynamics solution is computed in linear time without iterating. The frame rate of this animation involving 216 scalar constraints

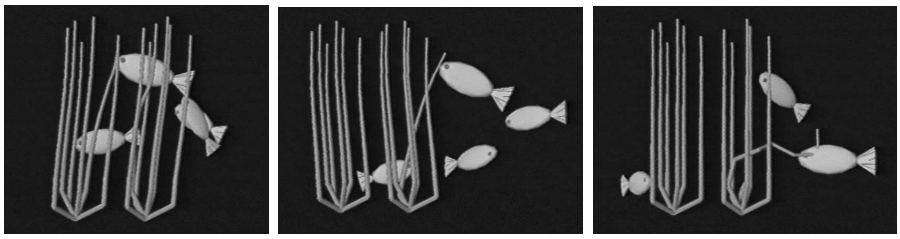


Figure 4: A near acyclic scene.

reaches approximately 9 images/second. Creating a joint between the mouth of the fish and a weed results in one closed loop joint including three constraints. Two iterations are necessary in practice, bringing the frame rate to 8 i/s.

Moderately cyclic scenes such as the pendulum in figure 5 greatly benefit from our method. In this example, each string is modeled using ten bars. There are 269 scalar constraints among which 18 closed loop constraints. Only three iterations (involving six acyclic solutions, due to the biconjugate

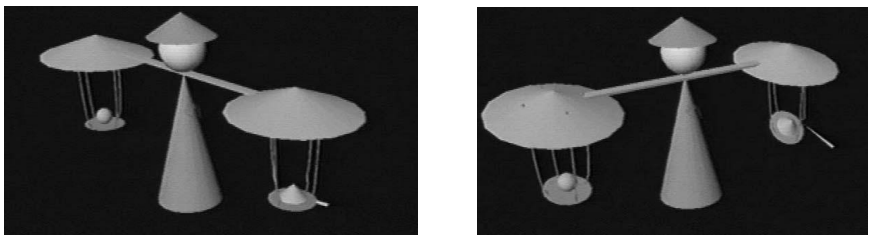
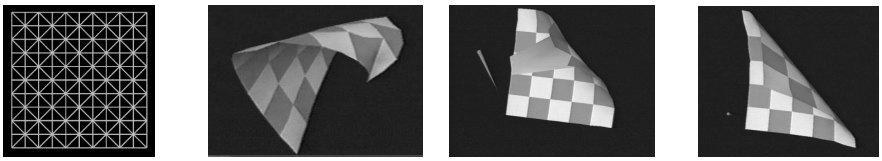


Figure 5: A moderately cyclic scene. The cone represents our manipulator.

gradient) are necessary to have the pendulum smoothly swinging in a gravity field, resulting in a frame rate of 8 i/s. In comparison, computing the reduced matrix involved in the structured dense method requires 18 acyclic solutions. The moderately cyclic scenes are the ones for which our method provides the most improvement with respect to the previous methods. Many scenes involving humanoids belong to this category such as the scene in figure 2.b for which 12 i/s are obtained.

Highly cyclic scenes such as triangular meshes can also be animated interactively using our method. Though particle systems with implicit integration may be more efficient applied to the following example, it provides a comparison between our approach and a full iterative method. The scene in

figure 6 includes 320 solids with 1557 scalar constraints including 600 closed loop constraints. Control points on the bars are used to define a deformable surface with constant area. In this shape modeling application, only the final shape matters, we can thus tolerate large errors while modeling. This allows interactivity. Once we stop moving, a few seconds are necessary for the structure to fully recover its geometric constraints. For the same degree of interactivity, our methods propagates the motion faster than the standard conjugate gradient algorithm, due to the linear-time acyclic solutions. The idea of handling acyclic subgraphs using linear-time methods might thus also benefit to particle animation with implicit integration.



(a) (b) (c) (d)

Figure 6: An application to shape modeling using a highly cyclic structure. In (a), the underlying structure in the initial state. In (b), the final shape obtained. In (c), an intermediate position obtained using a standard conjugate gradient. Note the deformation. In (d), an equivalent intermediate position obtained using our method.

6 Conclusion

We have presented a new algorithm for the solution of constrained dynamics equations. We believe that its efficiency along with its ability to perform iterative refinements meet the needs of the computer graphics community. The efficiency has been shown in theory and in practice. In contrast with previous approaches, the time complexity smoothly ranges from linear to quadratic depending on the relative amount of closed loop constraints. Our method combines a number of features desirable for computer animation which were not previously available simultaneously in a unified algorithm. It handles acyclic structures efficiently. In presence of closed loops, it provides the user with the ability of trading-off accuracy for computational efficiency, it handles overconstrained systems, and it allows the use of two constraint priority levels. The low complexity of the algorithm, along with the ability offered to the user to tune computation time, allow the interactive animation of large scenes.

For further efficiency improvements, future work should introduce the weighting of the closed loop constraints in terms of their visual importance, so that the iterative refinement minimizes the most visible errors with higher priority. For purposes of generality, the lower level of the body structure will be extended from acyclic articulated solids to any object able to compute its reaction when given forces are applied to it. This will allow us to combine solids, deformable bodies with various physical laws, and procedurally animated characters.

A Notations

exponent

symbol	meaning
$\boldsymbol{T}$	matrix or vector transposition

subscripts

symbol	meaning
$\boldsymbol{a}$	vectors or matrices related to the acyclic constraints
$\boldsymbol{l}$	vectors or matrices related to the closed loop constraints
$\boldsymbol{c}$	vectors or matrices related to all the constraints
$\boldsymbol{s}$	vectors or matrices related to the solids

scalars

symbol	meaning
a	number of acyclic scalar constraints
l	number of closed loop scalar constraints
c	total number of scalar constraints
s	number of solids times six

matrices

symbol	dimension	meaning
$\boldsymbol{J}$	$c \times s$	Jacobian in terms of the absolute motions $(\boldsymbol{J^T}) = (\boldsymbol{J_a^T J_l^T})$
$\boldsymbol{J_a}$	$a \times s$	Jacobian of the acyclic constraints
$\boldsymbol{J_l}$	$l \times s$	Jacobian of the closed loop constraints
$\boldsymbol{M}$	$s \times s$	mass matrix of the solids (block-diagonal)
$\boldsymbol{A}$	$c \times c$	constraint matrix ($\boldsymbol{A = J M^{-1} J^T}$)
$\hat{\boldsymbol{A}}$	$l \times l$	reduced constraint matrix

vectors (all column-vectors)

symbol	dimension	meaning
$\boldsymbol{\delta x}$	q	motion correction
$\boldsymbol{x_s}$	s	absolute solid motions (accel., impulsions, small displacements)
$\boldsymbol{f_s}$	s	net forces applied to the solids
$\boldsymbol{f_{ext}}$	s	external forces applied to the solids
$\boldsymbol{b}$	c	net constraints $(\boldsymbol{b^T}) = (\boldsymbol{b_a^T b_l^T})$
$\boldsymbol{b_a}$	a	net acyclic constraints
$\boldsymbol{b_l}$	l	net closed loop constraints
$\hat{\boldsymbol{b}}$	l	reduced constraints
$\boldsymbol{\lambda}$	c	unknown constraint forces $(\boldsymbol{\lambda^T}) = (\boldsymbol{\lambda_a^T \lambda_l^T})$
$\boldsymbol{\lambda_a}$	a	unknown acyclic constraint forces
$\boldsymbol{\lambda_l}$	l	unknown closed loop constraint forces
$\boldsymbol{x_c}$	c	relative motions along constraint directions
$\boldsymbol{x_l}$	l	relative closed loop motions
$\boldsymbol{f_l}$	l	closed loop constraint forces

References

[1] W. W. Armstrong and M. Green. The dynamics of articulated rigid bodies for purposes of animation. In M. Wein and E. M. Kidd, editors, *Graphics Interface '85 Proceedings*, pages 407–415. Canadian Inf. Process. Soc., 1985.

[2] U. M. Ascher, H.Chin, L.R.Petzold, and S. Reich. Stabiliation of constrained mechanical systems with daes and invariant manifold. *Journal of Mechanics of Structures and Machines*, 23(2):135–157, 1995.

[3] Dae-Sung Bae and Edward J. Haug. A recursive formulation for constrained mechanical system dynamics: part 2, closed loop systems. *Journal of Mechanics of Structures and Machines*, 15(4), 1987.

[4] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 23–34. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[5] David Baraff. Linear-time dynamics using lagrange multipliers. In *SIGGRAPH 96 Conference Proceedings*, Computer Graphics Proceedings, Annual Conference Series, pages 137–146. ACM SIGGRAPH, Addison Wesley, August 1996. ISBN 0-201-94800-1.

[6] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 179–188, August 1988.

[7] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics*, 1:1–36, 1972.

[8] Hong Sheng Chin. *Stabilization Methods for Simulations of Constrained Multibody Dynamics*. PhD thesis, University of British Columbia, 1995.

[9] R. Featherstone. *Robot Dynamics Algorithms*. Kluwer, 1987.

[10] M. Gleicher. *A Differential Approach to Graphical Manipulation*. PhD thesis, Carnegie Mellon University, 1994.

[11] Paul M. Isaacs and Michael F. Cohen. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 215–224, July 1987.

[12] Paul M. Isaacs and Michael F. Cohen. Mixed methods for complex kinematic constraints in dynamic figure animation. *The Visual Computer*, 4(6):296–305, December 1988.

[13] M.W.Walker and D.E.Orin. Efficient dynamic computer simulation of robotic mechanisms. *Journal of Dynamic Systems*, (104):205–211, 1982.

[14] Richard P. Paul. *Robot Manipulators: Mathematics, Programming, and Control.* MIT Press, Cambridge, MA, 1981.

[15] Press, Teukolski, Vetterling, and Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992.

[16] W. Schielen. *Multibody Systems Handbook.* Springer, Berlin, 1990.

[17] J. Shewchuck. An introduction to the conjugate gradient algorithm without the agonizing pain. Technical Report CMU-CS-TR-94-125, Carnegie Mellon University, 1994. see also http://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.ps.

[18] C. van Overveld and B. Barenburg. All you need is force: a constraint-based approach for rigid body dynamics in computer animation. In D.Terzopoulos and D.Thalmann, editors, *Proceedings of Computer Animation and Simulation '95*, Springer Computer Science, pages 80–94. Springer, 1995.

[19] J. Wilhelms and B. A. Barsky. Using dynamic analysis to animate articulated bodies such as humans and robots. In M. Wein and E. M. Kidd, editors, *Graphics Interface '85 Proceedings*, pages 97–104. Canadian Inf. Process. Soc., 1985.

[20] Andrew Witkin, Michael Gleicher, and William Welch. Interactive dynamics. In Rich Riesenfeld and Carlo Sequin, editors, *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, volume 24, pages 11–21, March 1990.

[21] J. Wittenburg. *Dynamics of Systems of Rigid Bodies.* B.G. Teubner, Stuttgart, Germany, 1977.