HAL
open science

Interactive Solid Animation Using Linearized Displacement Constraints

François Faure

▶ To cite this version:

François Faure. Interactive Solid Animation Using Linearized Displacement Constraints. Eurographics Workshop on Computer Animation and Simulation (EGCAS), Aug 1998, Lisbon, Portugal. pp.61–72. inria-00537520

HAL Id: inria-00537520
https://inria.hal.science/inria-00537520v1

Submitted on 18 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Interactive Solid Animation Using Linearized Displacement Constraints

François Faure
Institut für Computergrafik
Technische Universität, Wien

Abstract. We present a new approach for interactive solid animation. It allows a user to efficiently trade-off accuracy for speed, making complicated structures tractable in interactive time. Linearized displacement constraints are used in conjunction with an efficient iterative equation solver to perform the assembly of articulated solids. This allows the initialization of a scene and the correction of numerical integration errors. A robust integration scheme limits the instabilities due to approximations. Applications are shown and discussed.

1 Introduction

Interactivity is a major issue in computer animation. However, interactive articulated solid animation has yet been restricted to small structures, except in special cases. This lack of interactivity may be due to the inheritage of the computer animation techniques from robotics and mechanical engineering. The primary concern of these sciences is physical accuracy rather than computation time. Fast applications such as robot control are applied to simple structures, compared with the objects we want to animate in computer graphics. Virtual reality and large interactive applications require a new approach based on the control of the computation time. Experiments show that when dragging an object, users prefer quick responses, even with low precision, than waiting a few seconds for a precise motion. This implies trading-off accuracy for speed. This idea has been extensively used for rendering. The specific problem of animation is the possible accumulation of error over time, leading to inacceptable results. Therefore, we present an approach based on fast structure assembly and stable integration scheme. By performing the assembly at the end of each time step just before displaying the scene, one can accept relatively large errors inside the time step. This tolerance allows the approximate computation of the time derivatives and the use of large time steps in the numerical integration. As a result, interactivity can be obtained for large scenes.

The remainder of this paper is organized as follows. In section 2, we briefly describe the problems of time integration applied to the simulation of articulated structures, and how they have been dealt with in the field of computer graphics. We then present an alternative approach well-known in mechanical simulation, which seems currently unused in our domain. The following sections present variants of this approach that we have developed in order to meet the needs of computer graphics. Section 3 describes an

assembly algorithm for complex articulated structures. Section 4 presents applications to animation such as inverse kinematics, dynamics, along with first results in trading-off accuracy for speed in computer animation.

2 Background and motivation

Numerical simulation consists in integrating a differential equation over time. From a practical point of view, this requires repeatedly computing the time derivative of the system, and integrating it over a (possibly variable) time step. The laws of physics provide equations on accelerations, which allow the animation of systems of particules as ordinary differential equations (ODEs). In contrast, constrained systems such as articulated structures include joints with associated geometrical equations which must remain satisfied over time. This leads form ODEs to differential algebraic equations (DAEs), which are more difficult to handle. The DAE governing a physically-based articulated body can be expressed in matrix form as:

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

$$\boldsymbol{M}\dot{\boldsymbol{v}} = \boldsymbol{f}(t, \boldsymbol{q}, \boldsymbol{v}) + \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{\lambda} \tag{2}$$

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

where $\boldsymbol{q}$ is the set of coordinates, $\boldsymbol{M}$ the mass matrix associated with the coordinates, $\boldsymbol{f}$ represents the external forces, $\boldsymbol{g}$ the geometrical constraints, and $\boldsymbol{J}$ the derivative of $\boldsymbol{g}$ with respect to the coordinates, also called the Jacobian of the constraints. Matrix $\boldsymbol{D}$ relates the velocities to the derivatives of the coordinates, e.g. angular velocity to quaternion derivatives. The exponent T represents matrix transposition. The vector $\boldsymbol{\lambda}$ gathers the independent components of the constraint forces, acting along the directions of the geometrical constraints. In the remaining of the paper, we use bold letters to denote global vectors and matrices, gathering values related to all the joints or solids of the articulated bodies.

Differentiating twice equation (3) and substituting into equation (2) provides

$$\ddot{\boldsymbol{g}} = \boldsymbol{J}\boldsymbol{M}^{-1}(\boldsymbol{f} + \boldsymbol{J}^T\boldsymbol{\lambda}) + \boldsymbol{a_v} = 0 \tag{4}$$

where $\boldsymbol{a_v}$ is the velocity-dependent part of the relative accelerations. Solving equation (4) provides the constraint forces $\boldsymbol{\lambda}$, then the time derivatives through equations (2) and (1). Using the derivatives, we can compute a new position using:

$$\boldsymbol{x}(t + dt) = \boldsymbol{x}(t) + \boldsymbol{\phi}(t, \boldsymbol{x}(t), dt)$$

where $\boldsymbol{\phi}$ represents an integration scheme such as Euler or Runge-Kutta. Unfortunately, all integration schemes introduce more or less drift, i.e. equation (3) and its first derivative are no more satisfied at time $t + dt$, even if we start from a consistent state at time t. This drift eventually results in solids moving apart from each other without meeting the joint constraints, and it can quickly become visible. The drift is restricted to loop closures if relative coordinates are used.

In order to keep the drift within reasonable values, *Baumgarte stabilization* is very popular[3, 2, 8]. It consists in using a modified version of equation (4), namely:

$$\ddot{\boldsymbol{g}} = \boldsymbol{J}\boldsymbol{M}^{-1}(\boldsymbol{f} + \boldsymbol{J}^T\boldsymbol{\lambda}) + \boldsymbol{a_v} = \gamma_1\dot{\boldsymbol{g}} + \gamma_2\boldsymbol{g}$$

where γ_1 and γ_2 are parameters provided by the user, and the vectors $\boldsymbol{g}$ and $\dot{\boldsymbol{g}}$ straightforwardly computable at each time step. This perturbation acts much like damped springs applied to each constraint. This method has also shown capabilities of assembling articulated solids[2]. Unfortunately, the values of γ_1 and γ_2 are difficult to set. Too weak, they do not prevent the drift from reaching inacceptable values. Too high, they induce instabilities due to time sampling. The optimal compromise can be very difficult to find. In many cases, the stiffness induced by this method makes the use of small time steps necessary, thus reducing the computational efficiency.

More recently, efficient alternative approaches have been proposed, introducing the principle of *post-stabilization*[1, 4]. The basic idea is to proceed in two steps:

- starting from $(\boldsymbol{q}(t), \boldsymbol{v}(t))$, integrate the velocities and accelerations over the time interval dt using your favorite integration scheme, e.g. Runge-Kutta, and denote the result $(\tilde{\boldsymbol{q}}(t+dt), \tilde{\boldsymbol{v}}(t+dt))$
- perform *post-stabilization* in order to meet the geometric constraints and their first derivatives:

$$\boldsymbol{q}(t+dt) = \tilde{\boldsymbol{q}}(t+dt) - \boldsymbol{\delta q} \quad (5)$$

$$\boldsymbol{v}(t+dt) = \tilde{\boldsymbol{v}}(t+dt) - \boldsymbol{\delta v} \quad (6)$$

The computation of the correction terms is explained in section 3.3. This approach frees the user from tuning arbitrary stabilization parameters, and experiments have shown better stability than Baumgarte stabilization[4]. The authors show that if the truncation error of the integration scheme is $O(dt^{p+1})$ then the drift is $O(dt^{2(p+1)})$.

In this paper, we provide three contributions to this approach. First, we generalize the position stabilization (eq.5) to an iterative assembly algorithm. This allows the automatic initialization of scenes including complex geometric constraints.

Second, we include the assembly method in a simple integration scheme which makes no explicit use of velocity. This avoids performing the velocity stabilization (eq.6), which results in more computational efficiency. This also avoids us to derive noisy input data such as coordinates of 3D trakers.

Finally, we investigate the capabilities of post-stabilization for purposes of interactivity instead of precision. It is commonly admitted in mechanical engineering that the fourth-order Runge-Kutta integration scheme is the most efficient in most practical cases[10]. This requires computing four derivatives at each time step. Performing standard post-stabilization requires two additional dynamic solutions. Our approach is perticularly useful when, for purposes of interactivity, we can not afford even *one* dynamic solution within an animation step.

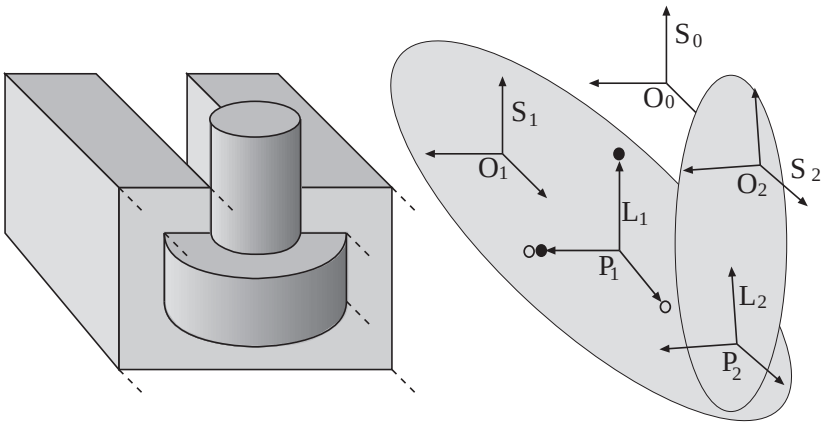

Figure 1: Our model applied to a joint with one translation and one rotation allowed. The two reference frames of this joint are centered on P_1 and P_2, with their direction aligned with the main axes of the joint. The solids are centered on O_1 and O_2, and the absolute coordinate system on O_0. The full and empty disks respectively denote translation and rotation constraints.

3 Fast assembly of articulated structures

Structure assembly can be used either to initialize complicated scenes including closed loops, or to correct positions after time integration. We first present our joint model, which allows us to compute geometric constraints for a wide variety of joints. We then show how to obtain a linearized geometric equation by writing it as a kinematic equation. Then we turn the non-square kinematic equation system into a square dynamics equation system. We finally describe the assembly algorithm.

3.1 Joint model and kinematic equations

We consider two solids represented by their local frames S_1 and S_2 centered on O_1 and O_2, respectively. An example is shown in figure 1. The positions of the solids are defined with respect to a reference frame (O_0, S_0). Let the solids be bound by a joint L_{12}. We represent this joint using two local frames L_1 and L_2 attached to S_1 and S_2 and centered in P_1 and P_2, respectively. The geometric constraints between L_1 and L_2 define the type of the joint. A universal joint requires P_1 and P_2 to remain equal, whereas a plane-to-plane joint requires that one plane fixed in L_1 coincides with one plane fixed in L_2.

The velocities have to be consistent with the constraints. For simplicity, a good choice is to express the kinematic equations at the center of one of the joint frames, this frame being aligned with the motion constraints. Expressing the relative velocity between solids S_i and S_j at point P_i provides:

$$\begin{aligned} v_{ij} &= v_i(P_i) - v_j(P_j) \\ &= v_i(O_i) + \omega_i \times O_iP_i - v_j(O_j) - \omega_j \times (O_jP_j + P_jP_i) \\ \omega_{ij} &= \omega_i - \omega_j \end{aligned}$$

Table 1: Kinematic constraints associated with different types of joints. A cross denotes the presence of a kinematic constraint along the associated direction. Translation and rotation directions are denoted using em t and em r, respectively. We chose arbitrarily the vector i as the main axis of the joint.

joint type	ti	tj	tk	ri	rj	rk
universal	×	×	×			
pin	×	×	×		×	×
cylindrical		×	×		×	×
prismatic		×	×	×	×	×
ball-socket		×	×			
smooth surface contact	×					

The kinematics equations associated with a joint including n_t translation constraints and n_r rotation constraints can be written as:

$$v_{ij}(P).t_k = \dot{c}_k \quad 1 \leq k \leq n_t \tag{7}$$

$$\omega_{ij}.r_k = \dot{c}_k \quad n_t < k \leq n_t + n_r \tag{8}$$

where t_k and r_k are translation and rotation constraint directions . The scalars $\dot{c}$ are the values of the velocity constraints in translation or rotation. These values are null for perfect joints. Table (1) shows the kinematic constraints associated with some common joints. Further work will include dependent constraint directions such as screws.

Generally, there is a whole space of velocities consistent with the constraints, and we want to compute the consistent velocities which are the nearest to given values. The sets of equations (7,8) related to all the joints of the scene can thus be gathered in the following matrix equation:

$$\boldsymbol{J\delta v} = \dot{\boldsymbol{c}} - \boldsymbol{Jv} \tag{9}$$

where $\boldsymbol{v}$ is the current global velocity vector and $\boldsymbol{\delta v}$ is a velocity correction necessary to reach the space of consistent velocitites. The solution of the equation system is explained in section 3.3. Using relative coordinates, only the loop closures require the solution of a kinematic equation, the acyclic constraints being implicitely satisfied.

3.2 Linearized geometric equations

Starting from an inconsistent state, we want to compute new positions satisfying the geometric constraints. Geometric equations are generally difficult to solve because they involve nonlinear equations including sine functions. We obtain a linear equation by integrating a kinematic equation over a virtual time step dt^*:

$$\boldsymbol{J\delta v}dt^* = \boldsymbol{b} \tag{10}$$

where $\boldsymbol{b}$ is the displacement constraints necessary to cancel the errors. We explain at the end of this section how to compute the displacement constraints. The solution of the

equation provides the unknown vector $\boldsymbol{\delta v} dt^*$. The coordinate corrections are straightforwarly obtained using equation (1):

$$\boldsymbol{\delta q} = \boldsymbol{D} \boldsymbol{\delta v} dt^*$$

The solution of the linearized equation system is good approximation of the real solution when small displacements are involved. We do not explicitely use the virtual time step since its value is arbitrary.

Now we show a simple way of computing the displacement constraint . This requires representing the relative rotation of the two joint coordinate frames as $\omega_{ij} dt^*$. The relative translation δ_{ij} and the relative rotation $R(L_i, L_j)$ can be computed using transitivity:

$$\delta_{ij} = P_i P_j = P_i O_i + O_i O + O O_j + O_j P j$$
$$R(L_i, L_j) = R(L_i, S_i).R(S_i, S_0).R(S_0, S_j).R(S_j, L_j)$$

where the operator R is your favorite rotation model (quaternions, matrices, Euler angles...) and the dot denotes the appropriate transitivity operator. We then turn the relative rotation into an (axis,angle) form where the axis is a unit vector, and finally turn it into a three dimensional vector defined as the axis multiplied by the angle, that we call $\bar{\omega}_{ij}$. Projecting the relative rotation and translation to the corresponding constraint directions provides the values of the geometric constraint errors. The displacement constraints are, for each joint, the opposite of the geometric errors:

$$\begin{aligned} b_k = & \quad -\delta_{ij}.t_k \quad & 1 \leq k \leq n_t \\ b_k = & \quad -\bar{\omega}_{ij}.r_k \quad & n_t < k \leq n_t + n_r \end{aligned}$$

3.3 From kinematics to dynamics

Since the number of constraints is independent on the number of coordinates, the Jacobian matrix is typically non-square. A way of regularizing the equation system is to use constraint forces to move the solids. This leads from kinematics to dynamics. In dynamics, each solid S_i obeys the fundamental principle:

$$f_i = M_i \dot{v}_i \tag{11}$$

where f_i and $\ddot{q}_i$ are six-dimensional vectors denoting force and acceleration, and M_i the mass matrix of S_i. A common assumption about constraint forces is that they act along the constraint axes. This derives from the principle of virtual works applied to perfect joints. In this case, the forces applied to the solids by the constraints are simply $\boldsymbol{J^T \lambda}$ where the vector $\boldsymbol{\lambda}$ gathers all the constraint forces. This leads to the equation system:

$$\begin{pmatrix} \boldsymbol{M} & \boldsymbol{-J^T} \\ \boldsymbol{-J} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta \dot{v}} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} 0 \\ \boldsymbol{-b} \end{pmatrix} \tag{12}$$

where the first line is Newton's law restricted to the constraint forces, and the second line is the kinematics equation. Vector $\boldsymbol{b}$ is the relative acceleration correction. It is the

opposite of the relative acceleration which would occur along constrained directions if null constraint forces were applied. Vector $\boldsymbol{\lambda}$ represents the unknown constraint forces necessary to enforce the kinematic constraints. They result in an acceleration correction $\boldsymbol{\delta\dot{v}}$. The equation system (12) is square and models physical interactions between the solids. Absolute coordinates involve a diagonal mass matrix, whereas relative coordinates require computing the entries of a dense matrix. In both cases, matrix $\boldsymbol{M}$ is symmetric positive definite. It is thus possible to perform a substitution of the first line into the second one to obtain a new, reduced equation system:

$$\boldsymbol{A\lambda} = \boldsymbol{b}, \quad with \;\; \boldsymbol{A} = \boldsymbol{JM^{-1}J^T} \tag{13}$$

The corrections can then be computed as

$$\boldsymbol{\delta\dot{v}} = \boldsymbol{M^{-1}J^T(JM^{-1}J^T)^{-1}b}$$

Note that using the identity as a mass matrix is equivalent with performing a left pseudo-inverse solution, well-known in kinematics[6]. This is our motivation for performing a dynamic solution: it is a straightforward generalization of the standard kinematics approaches, and it allows us to compute physically realistic motions. Corrections of velocities or positions are computed in a similar way when performing post-stabilization.

3.4 The fast assembly algorithm

Starting from a state $\boldsymbol{q}$, the assembly is performed by adding to vector $\boldsymbol{q}$ an increment $\boldsymbol{\delta q}$ computed by the function *correction*. This function, which pseudocode is given below, computes iteratively a position correction necessary to meet the constraints. At each iteration, it solves a linear system similar with equation (12), except that position corrections are computed instead of acceleration corrections. This linear system is a fist-order approximation of the geometric equations. Several iterations may thus be necessary. The computation of the correction terminates as soon as a displacement satisfying all the geometric constraints up to a given precision has been computed. This is checked by the boolean function *geometryOk*. The algorithm can also terminate after a given number of iteration have been performed. The procedure *compute_entries* computes the entries of matrices $\boldsymbol{M}, \boldsymbol{J}$ and the displacement constraint $\boldsymbol{b}$ corresponding to given coordinates.

```
correction(q){
    δq = 0
    compute_entries( q, M, J, b )
    while not geometryOk( b ){
        δq += M^-1 J^T (J M^-1 J^T)^-1 b
        compute_entries( q + δq, M, J, b )
    }
    return δq
}
```

In contrast with differential approaches[2, 7], our assembly process is not delayed through time and thus allows the display of accurate geometry at any animation step. In

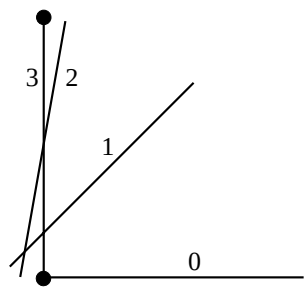

Figure 2: Example of convergence of the assembly method. The geometric constraints state that the endpoints of the bar have to coincide with the centers of the disks.

contrast with kinematical approaches[6, 11], it makes use of mass and it is thus compatible with dynamics.

In practice, the convergence of the algorithm is fast (see example in figure 2), except if the rotations reach high values. In this case the linear approximation is too poor and the system may enter an endless process. To solve this problem, we simply truncate excessive rotations. Empirically, 0.8 radians seems a good value.

At each iteration, the linear equation equation (13) is solved using the biconjugate gradient algorithm[9]. Instead of computing explicitly a decomposition of the matrix $\boldsymbol{JM^{-1}J^T}$, the algorithm solves the equation system by performing a sequence of matrix products. Using absolute coordinates allows the use of matrix sparsity, providing a good efficiency. The biconjugate gradient algorithm performs an iterative minimization of the error. This allows the termination of the algorithm to occur as soon as the desired precision is reached, using various norms, or after a given number of iterations have been performed. Moreover, this algorithm handles indefinite equation systems, and computes a least-square solution in case of inconsistent constraints.

The computation of the correction is twice iterative: each loop traversal involves an iterative solution of a linear equation system. Limiting the number of iterations allow the user to trade-off accuracy for speed, which is useful when applied to complex structures. An example of complex assembly is shown in figure (3). This scenes includes 758 scalar constraints. Six iterations aree used to perform the assembly, each of them limited to 30 conjugate gradient iterations. The computation time is less than one second on a standard SGI O2 workstation.

4 Applications to animation

4.1 Inverse kinematics

We apply our articulated body method to a VR environment including 3D hand-trackers with buttons. This allows us to interactively catch, drag and release objects. A straightforward application is inverse kinematics. A joint binding the tracker and the solid pointed by the traker is created when the button is clicked. As long as the button remains pressed, the position and orientation of the joint is updated according to the position of the traker. Applying the assembly algorithm allows the structure to “follow” the tracker

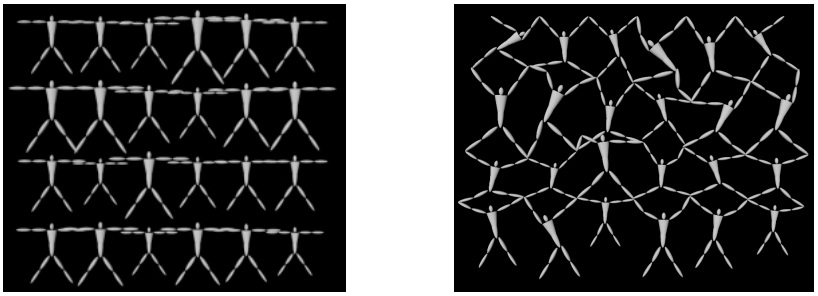

Figure 3: Complex assembly. We want to bind the hands and feet of the different-sized characters.

according to the geometric constraints. A null inverse mass is applied to the tracker, so that only the other objects can be corrected.

4.2 Dynamics

In dynamics, the forces are responsible for the accelerations of the solids. The velocities remain unchanged in case of null forces. So far, we have not yet introduced velocities in the system. Velocities are difficult to compute from 3D traker input since the trackers generally measure only positions. Numerical derivation is dangerous because noisy data induce instability. The problem is even harder when dealing with accelerations. Filtering the data is not a satisfying solution because it introduces disturbing delays. To avoid this problem, we apply Stoermer's integration scheme[9], which makes no explicit use of velocities, using the previous displacements $\boldsymbol{\Delta q}$ instead. This integration scheme can be written as:

$$\boldsymbol{\Delta q}(t+dt) = \boldsymbol{\Delta q}(t) + \ddot{\boldsymbol{q}}dt^2$$
$$\boldsymbol{q}(t+dt) = \boldsymbol{q}(t) + \boldsymbol{\Delta q}(t+dt)$$

This integration scheme requires an initialization of $\boldsymbol{\Delta q}$, e.g. $\boldsymbol{\Delta q}(0) = \dot{\boldsymbol{q}}(0)dt + \frac{1}{2}dt^2\ddot{\boldsymbol{q}}$. It fits perticularly well with our assembly approach. We perform the assembly at the end of each time step so that the geometric errors arising from numerical integration are canceled before displaying the scene. The position increment vector is updated as well as the positions themselves. The pseudocode for a simulation step is as follows:

```
step( dt ){
    Δq += M^-1 f dt^2
    q += Δq
    δq = correction(q)
    q += δq
    Δq += δq
}
```

Note the extreme simplicity of this animation scheme. We do not even compute accelerations compatible with the constraints. Rather, only external forces are considered

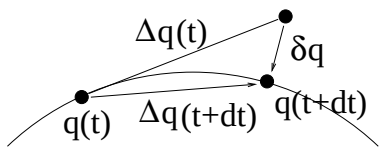

Figure 4: Our method applied to a simple example. A particle is constrained to remain on a fixed circle. It first moves according to its previous motion (in this example, there are no external forces applied). Then a correction δq is applied in order to meet the constraint. The displacement $\boldsymbol{\Delta q}$ is updated accordingly.

in vector $\boldsymbol{f}$, the solid first move as if they were free, and their positions (and displacements) are corrected at the end of each time step. This results in correcting simultaneously acceleration and numerical integration errors, and allows performing large time steps. An illustration is shown in figure 4. Note that the updated "velocity" is the displacement between two positions compatible with the constraints. Contrary with Baumgarte stabilization, this method induces few velocity in the directions of the geometrical constraints. As a result, the method is much more robust to large time steps.

Compared with previous related work[5], the efficiency of our method comes from the use of the biconjugate gradient algorithm, along with a robust integration scheme.

We validated this integration scheme using numerous experiments involving isolated articulated bodies. Energy and momentum remain constant up to machine roundoff precision. Bodies linked to the ground may suffer from energy variations, similarly with what happens using other integration schemes. The limits of this approach are reached when strong forces generate large displacements requiring large corrections. In this case, numerical roundoffs and linear approximations may result in jerky motion, unless short time steps are used. Further work will include acceleration correction.

4.3 Trading-off accuracy for speed

Interactivity is necessary in applications such as virtual reality. Our iterative approach allows the user to tune the level of interactivity by limiting the computation time. The number of assembly iterations can be set to one for high interactivity, while three are generally enough for high precision. The number of conjugate gradient iterations can often be reduced to a surprisingly small number compared with the theoretical number, which is equal to (at most) the number of constraints. Applied to the articulated structure in figure 5 including 56 solids and 261 scalar constraints, 2 geometrical iterations each of them involving 5 conjugate gradient iterations allow us to drag the structure interactively. In contrast, 3 geometrical iterations each of them involving 50 conjugate gradient iterations are necessary to obtain a visually perfect geometric accuracy, resulting in poor interactivity.

Large time steps allow the use of real-world time. In our application, time is read at each entry in the main animation loop, and the time step is deduced from the time of the previous entry. Due to various technical reasons, this time step is not constant. The previous displacements $\boldsymbol{\Delta q}$ used by the integration scheme are scaled by dividing the new time step by the previous one. Since the different time steps have the same order of mag-

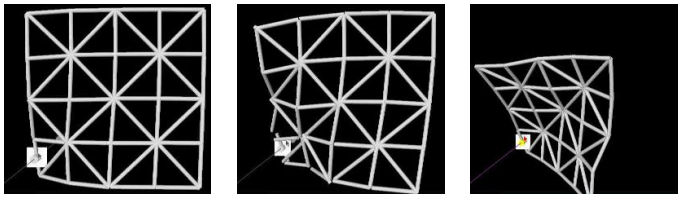

Figure 5: Interactive positioning. Low geometrical precision allows interactivity during motion. Precision is eventually recovered.

Figure 6: Interactive dynamics. The user shakes the plate using a 3D tracker. Low precision allows high interactivity with visually correct results.

nitude, the division does not induce important roundoff errors. We were able to animate the scene shown in figure 6 at 12 frames per second on a standard SGI O2 workstation. This scene includes 28 solids and 108 scalar constraints. Only one geometric iteration and fifteen conjugate gradient iterations are performed for each animation frame. The rendering takes approximately half the computation time.

5 Future work

Our modified post-stabilization approach with tunable accuracy or computation time has shown good capabilities for interactive solid animation. Further improvements should include initial guesses of the conjugate gradient solution. However, we have found few similarity between constraint forces from one step to another using displacement constraints. We expect to find more temporal coherency using acceleration correction. Valuable initial guesses may allow the rapid computation of accelerations more compatible with the constraints. This would reduce the geometrical error at each step and consequently, the number of stabilization iterations. Additionally, weighting the geometric errors in terms of their contribution to the percepted accuracy may reduce the necessary computations.

Aknowledgements

We gratefully acknowledge the support of the European Union's Training and Mobility of Researchers (TMR) programme in funding this work.

References

1. 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.
2. 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.
3. J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics*, 1:1–36, 1972.
4. Hong Sheng Chin. *Stabilization Methods for Simulations of Constrained Multibody Dynamics*. PhD thesis, University of British Columbia, 1995.
5. M.P. Gascuel and J.D. Gascuel. Displacement constraints for interactive modeling and animation of articulated structures. *The Visual Computer*, 10(4):191–204, March 1994.
6. Michael Girard and Anthony A. Maciejewski. Computational modeling for the computer animation of legged figures. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 263–270, July 1985.
7. M. Gleicher. *A Differential Approach to Graphical Manipulation*. PhD thesis, Carnegie Mellon University, 1994.
8. 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.
9. Press, Teukolski, Vetterling, and Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.
10. W. Schielen. *Multibody Systems Handbook*. Springer, Berlin, 1990.
11. Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics*, 13(4):313–336, October 1994.