Interactive animation of structured deformable objects

Mathieu Desbrun *Peter Schröder* *Alan Barr*

Caltech

Abstract

In this paper, we propose a stable and efficient algorithm for animating mass-spring systems. An integration scheme derived from implicit integration allows us to obtain interactive realistic animation of any mass-spring network. We alleviate the need to solve a linear system through the use of a predictor-corrector approach: We first compute a rapid approximation of the implicit integration, then we correct this estimate in a post-step process to preserve momentum. Combined with an inverse dynamics process to implement collisions and other constraints, this method provides a simple, stable and tunable model for deformable objects suitable for virtual reality. An implementation in a VR environment demonstrates this approach.

1 Introduction

Interactive animation of deformable objects in virtual reality systems has been a challenging problem for years. Although many fast methods of animation have been proposed, few techniques are currently able to dynamically animate even simple deformable objects at interactive rates. Moreover, animation in an immersive environment needs to be *very* robust and stable, to be "bullet-proof" against any action of the user. Thus, two apparently exclusive properties need to be satisfied: we need both efficiency and stability, without compromising the visual result. This paper proposes an approach that finds a balance between these two goals, enabling robust interactive animation as Fig. 1 illustrates.

1.1 Background and motivation

One of the simplest physically-based models over the last decade, and thus, the most likely to achieve real-time performances, is the mass-spring system [13, 12, 2]. A deformable body is approximated by a set of masses linked by springs in a fixed topology. This model can be seen as a discrete approximation of a finite-element method for integrating the Lagrange partial derivative equation of motion [21]. Easy to implement, highly parallelizable, and involving few computations, it seems a perfect candidate for simple virtual reality applications. Recently, improvements to this model have been made, such as an inverse dynamics step to bound the stretch of springs[18]. Adaptive time stepping to preserve the system's global energy has also been proposed [9].

Unfortunately, all of these approaches suffer from the same problem: the time step squared must be inversely pro-

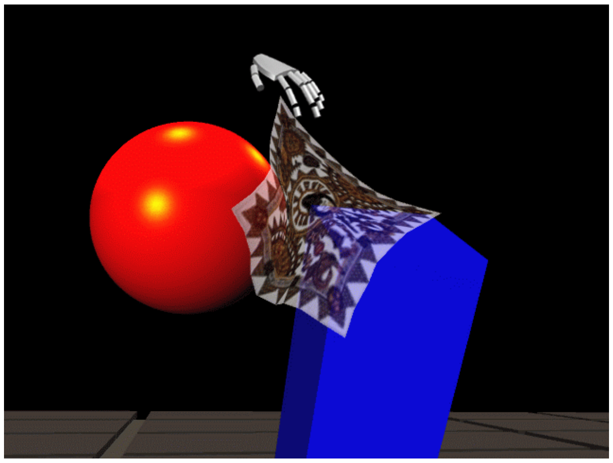
Figure 1: *Picture captured during live sessions in a VR environment: A silk scarf is moved around in a scene with obstacles in real-time.*

portional to the stiffness. Even if this is not an issue for off-line computations, it prevents many use in real-time applications since very small time steps are required to ensure stability. Various ways to overcome this problem have been used. An extensive dissipative force, opposite to the velocity of a mass point, provides a good way to maintain the stability of the integration. However, this method introduces an implausibly low terminal velocity, resulting in slow motions as if the medium was made of oil. Gravity has often been modified (lowered) to avoid large forces in the system, but once again, it introduces unbearable alteration in realism.

Other approaches have been taken to animate deformable objects of fixed topology. Elasticity and visco-elasticity have been modeled with success [20, 22, 21], but these methods suffer from the same time step handicap. Global methods, gaining efficiency by restraining the possible deformation [16, 23], are perfect for interactive manipulation, but are unfortunately of limited realism.

To the authors' knowledge, only two existing models achieve real-time computations for deformable structured objects. The first one is derived from the finite element theory, and takes advantage of linear elasticity to allow real-time deformation of any meshed objects [3]. This model is not dynamic, but is rather a collection of static postures, which greatly limits its applications. The other model is the recent development of neuro-animators [8]: after a learning period, a large neural network can emulate a simple physical system. This recent approach has not been proven practical for large coupled systems such as cloth.

The use of implicit integration, which can stably take large time steps, has been proposed [1] in the context of cloth animation. This method offers extremely low computational times, which indicates the possibility of real-time animation of simple objects. Inspired by this approach, we propose in this paper a fast and stable way to animate any mass-spring system.

1.2 Overview

We adopt the idea of the implicit integration, but we propose a predictor-corrector approach to alleviate the computational burden of solving a linear system at each time step. We introduce an approximation of the implicit integration to predict the next position, and then correct this approximation to ensure various physical invariants. The resulting algorithm is similar to a regular explicit Euler integration scheme, with a significant improvement: stability is ensured for arbitrarily large time steps. Finally, we mix this method with a inverse dynamics relaxation technique both to adjust the approximation made in the integration and to mimic nonlinear behavior.

This remainder of this paper is organized as follows: Section 2 details the principle of implicit integration applied to a 1D mass-spring system, along with its advantages and disadvantages. We present our integration scheme in Section 3, by explaining the approximation made to obtain stability and efficiency, and showing how to eliminate most of the errors created. Section 4 briefly mentions an existing post-step modification we use to implement constraints, collisions, and to enhance the realism. We finally give pseudo-code of our method in Section 5, before showing some results in section 6 and concluding in section 7.

2 Implicit integration: 1D case

While implicit time integration was used in early work on deformable bodies [20], only two recent papers extensively focus on the advantages of such a method [10, 1]. In order to understand exactly what the implicit integration scheme does, we start by detailing the simplest 1D case of mass-spring system.

2.1 Notation

In this paper, we will use the following notation:

- $\mathbf{x}$ is the geometric state of the system, consisting of all the positions $\mathbf{x}_i$ of the mass points: $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)^T$.
- $\mathbf{v}$ is the vector containing all of the velocities: $\mathbf{v} = \dot{\mathbf{x}}$.
- $\mathbf{F}_i$ denotes the internal forces (due to springs) acting on a mass point i, whereas $\mathbf{F}_i^{ext}$ will denote the external forces such as gravity.
- Superscript indices indicate the time beginning with an arbitrary time t_0. For instance, $\mathbf{x}_i^n = \mathbf{x}_i(t_0 + n\,dt)$.
- We will also use the backward difference operator: $\Delta^{n+1}\mathbf{x} = \mathbf{x}^{n+1} - \mathbf{x}^n$.

Physical model in 1D

Suppose we have a 1D model in which each discrete mass point i of mass m at position $\mathbf{x}_i$ is linked to its two immediate neighbors by a spring of stiffness k (Figure 2).

Figure 2: *1D case: masses linked by springs.*

Integration scheme

To animate such a system, the following *explicit Euler integration* scheme can be used:

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \mathbf{F}_i^n \frac{dt}{m}$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \mathbf{v}_i^{n+1}\,dt.$$

Note that in the explicit Euler method, the forces at time t_n contribute to the velocities at time t_{n+1}. Higher-order schemes, like Runge-Kutta, are better in terms of numerical accuracy for smooth solutions. However, since we often have to handle collisions (which gives rise to discontinuities in the motion during animation), these schemes are not appropriate.

Despite its ease of implementation, the explicit Euler scheme requires an integration time step dt which must be inversely proportional to the square root of the stiffness (this criterion is more generally know as the Courant condition [24]). Otherwise, the system will blow up rapidly since assuming the internal forces as constant over too large a time step may often induce a wild change in position. In practice, we effectively notice a stable behavior of the system only for small time steps. This is the general problem of stiff sets of equations: stability can be achieved only at very small time scale with explicit schemes [17].

Another scheme, called *implicit Euler integration*, has proven to be much better adapted to such a problem [10, 1]. The idea is to replace the forces at time t by the forces at time $t + dt$:

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \mathbf{F}_i^{n+1} \frac{dt}{m} \qquad (1)$$
$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \mathbf{v}_i^{n+1}\,dt$$

This simple substitution enforces stability in a distinctive way: now, the new positions are not blindly reached, but they correspond to a state where the force field is coherent with the displacement found. We still consider the forces to be constant within the time step, but in theory, *whatever the value of the time step*, the output state of the system will have consistent forces that will not give rise to instabilities. To put it in a different way, we can say that an explicit scheme takes a step into the unknown only knowing the initial conditions, while an implicit scheme tries to land on the next position correctly.

To implement this scheme, we must compute $\mathbf{F}^{n+1}$ without yet knowing the positions of the masses at time $t+dt$. Fortunately, we can write a first-order approximation (which is actually exact for springs):

$$\mathbf{F}^{n+1} = \mathbf{F}^n + \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \Delta^{n+1}\mathbf{x}. \qquad (2)$$

As the internal forces of the system $\mathbf{F}_i$ are already proportional to the gradient of an internal energy, we note that the matrix $H = \frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ is actually the negated hessian matrix of the system. In our present 1D case, we have:

$$H = k \begin{bmatrix} -1 & 1 & 0 & 0 & . & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & . & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & . & 0 & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & . & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & . & 0 & 0 & 1 & -1 \end{bmatrix}$$

Now, substituting Equ. (2) into Equ. (1), and by writing: $\Delta^{n+1}\mathbf{x} = (\mathbf{v}^n + \Delta^{n+1}\mathbf{v})\, dt$, we find:

$$\Delta^{n+1}\mathbf{v} = (\mathbf{I} - \frac{dt^2}{m} H)^{-1} (\mathbf{F}^n + dt\, H \mathbf{v}^n) \frac{dt}{m}. \qquad (3)$$

2.2 Difference between explicit and implicit

As H is constant in this 1D case, the implicit scheme does not appear really different from the explicit one. In the next sections, we take a closer look at the differences.

Addition of artificial viscosity

First, we can see that extra forces $\tilde{\mathbf{F}}^n = dt\, H \mathbf{v}^n$ are added to the set of internal forces $\mathbf{F}^n$. For a given mass point i, we remark that the additional force $\tilde{\mathbf{F}}_i$ can be written as follows, thanks to the structure of H:

$$\tilde{\mathbf{F}}_i = k\, dt \sum_{j|(i,j) \in \text{Edges}} (\mathbf{v}_j - \mathbf{v}_i).$$

The sum of the relative velocity with the neighbors has been used several times in the past as **artificial viscosity** to create a dissipative force in mass-spring or particle systems [14, 4], or in Rayleigh damping forces. It intuitively means that the motion of a mass point is influenced by the motion of its neighbors: a particle will tend to *follow the local displacement*. In the implicit integration, this articifial viscosity is proportional to both the time step and the stiffness of the material. As these two parameters are responsible for the instabilities in an integration process, we "cushion" the force field by using the right amount of viscosity, therefore adding stability.

Filtering of the force field

Once an artificial viscosity has been added, we find the corresponding change of velocity through multiplication of the resulting force by the inverse of a constant matrix. This

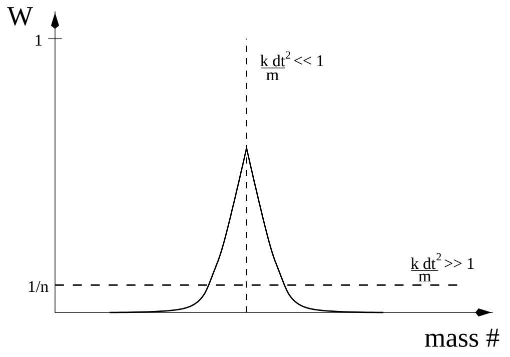

Figure 3: *A typical filter W, and two extreme cases: extremely rigid (Constant filter, equivalent to pure translation) and extremely loose (Dirac filter, equivalent to the explicit Euler integration).*

matrix $(\mathbf{I} - \frac{dt^2}{m} H)$ has many nice properties. First, as H is a hessian matrix, it is symmetric. More important, H has a zero eigenvalue for the eigenvector $(1,1,...,1)^T$ as a global translation of the mass-spring system does not create a change in the internal forces. As a consequence, the matrix $W = (\mathbf{I} - \frac{dt^2}{m} H)^{-1}$ has an eigenvalue of 1 for the same eigenvector. We can then write:

$$\forall j, \; \sum_{i=1}^{n} W_{ij} = 1,$$

meaning that each line sums to 1. The multiplication of the force field by W then corresponds to a **filtering**, as mentioned by Kass [10]. The resulting forces are a discrete convolution between the force field and this set of filters W. Due to the simple structure of H in 1D, we note that the filters are Fibonacci sequences: because of the tridiagonal structure of H, there is a recurrence relation between every three successive terms of the filters.

In the extreme case $kdt^2 << 1$, we will get almost a Dirac filter, meaning that for low stiffness or small time steps, the implicit integration is basically equivalent to an explicit one. In the other extreme case where either the stiffness or the time step is significant, we will have an almost constant filter, meaning that the mass points will all translate together as in a rigid body motion. In between, filters will look like the one depicted in Fig. 3. We then get a better understanding of the efficiency of such an integration: the stiffer springs are, the wider filters are. The resulting force on a mass point will take into account all the forces around, smoothing the possible large difference of net force between a mass point and its immediate neighbor. Another way to express the same idea is to say that this scheme propagates the information through the whole material during a single time step, instead of just affecting the immediate neighbors in the explicit integration, and thus does not require a limit on the time step size.

2.3 Discussion

Implicit integration seems to perfectly suit the animation community: it offers a way to create stable animations, without having to tune unintuitive damping coefficients. More importantly, it allows the time step to be set much higher without problem, which reduces the computational time for a given portion of animation. Therefore, as shown in the 1D case, the explicit integration seems to be of no interest. Nevertheless, we have to mention that we do gain *stability* by losing *accuracy*. Indeed, using an implicit integration smoothes the force field and adds viscosity, which introduces drift compared to the real theoretical evolution of the system. Moreover, as soon as the system is in 2D or 3D, the hessian matrix changes at each time step even for springs, which requires the solution of a large linear system.

In an animation context, these two last flaws are not real limitations. First, damping forces are frequently added to a system for both realism and stability anyway, so the fact that an implicit integration adds them directly without any need for tuning is probably better for the user. And even if solving a large linear system takes time, the advantages in the overall computational time are really significant as shown in [1].

3 An extension to 2D and 3D

In this section, we propose a different way to integrate the motion of any mass-spring systems. A simple approximation allows us to advance the masses' position with the ease of an explicit Euler scheme, and the nice properties of an implicit time integration scheme. We give pseudo-code for the overall algorithm in Fig. 6.

3.1 Physical model

We now consider a general mass-spring system in 2D or 3D, which is any set of mass points linked by springs. For the sake of generality, we will consider that a mass point i is linked to all the others with springs of rest length l^0_{ij} and stiffness k_{ij}. This latter stiffness value is set to zero if the actual model does not contain a spring between masses i and j.

3.2 Implicit integration

To use implicit integration, we must compute the hessian matrix of the system. In contrast to the 1D case, we no longer have a constant and simple matrix. For instance, for a spring between mass i and mass j, we have:

$$\frac{\partial \mathbf{F}_{(i,j)}}{\partial \mathbf{x}_i} = -k_{ij}\left[\frac{||\mathbf{x}_i-\mathbf{x}_j||-l^0_{ij}}{||\mathbf{x}_i-\mathbf{x}_j||}\mathbf{I}_3 + l^0_{ij}\,\frac{(\mathbf{x}_i-\mathbf{x}_j)^T(\mathbf{x}_i-\mathbf{x}_j)}{||\mathbf{x}_i-\mathbf{x}_j||^3}\right]$$

Thus we need to compute a $3n \times 3n$ matrix with sums of such expressions, and then solve the linear system at each time step. This amounts to an application of the method used in [1] to a mass-spring system. In this paper, we propose an alternative approach which eliminates the need to solve a linear system through an approximation of this matrix.

3.3 Splitting the problem in two

As computing the derivative of the spring forces seems very expensive, we prefer splitting the force in two parts, a linear and a nonlinear one:

$$\mathbf{F}_{(i,j)} = \mathbf{F}^{\text{linear}}_{(i,j)} + \mathbf{F}^{\text{non-linear}}_{(i,j)}$$

$$\mathbf{F}^{\text{linear}}_{(i,j)} = -k_{ij}\,(\mathbf{x}_i-\mathbf{x}_j), \qquad \mathbf{F}^{\text{non-linear}}_{(i,j)} = k_{ij}\,l^0_{i,j}\,\frac{(\mathbf{x}_i-\mathbf{x}_j)}{||\mathbf{x}_i-\mathbf{x}_j||}$$

In the next two sections, we show how to efficiently perform an approximate integration of these two parts by taking advantage of the implicit integration detailed above.

3.4 Integrating the linear forces

The linear part represents a force that would act in the same mass-spring system if all the rest lengths were zero, thus shrinking the simulated object. But, as seen in section 2, forces that are linear in position are easy to integrate: the hessian matrix is constant. In our framework, we can directly write the matrix H as a $n \times n$ matrix[1] as follows:

$$\begin{cases} H_{ij} = k_{ij} & \text{if } i \neq j \\ H_{ii} = -\sum_{j\neq i} k_{ij} \end{cases} \tag{4}$$

As in the 1D case, we can compute the filter matrix once and use it to provide a stable integration regardless of what the time step is, using Equ. (3). Integrating the linear forces is thus very simple.

3.5 Integrating the nonlinear forces

The other part, non linear, is less intuitive, but has the nice feature of always being of the same magnitude. It means that $\mathbf{F}^{\text{non-linear}}_{(i,j)}$ between t and $t+dt$ will just *rotate*, without varying in magnitude. In our integration predictor, we simply decide to overlook the rotation, and suppose that this non-linear part will stay constant (which amounts to considering the hessian matrix as being null).

Two main reasons have motivated our choice for this approximation: first, as these forces do not change of magnitude, the prediction mainly introduces *an error in angle*. The second reason is that we can simply *balance* this error in angle later, with a straightforward post-step displacement as we are going to explain in the next section.

3.6 Correction of momentum

To validate a time integration scheme, it is important to check that linear and angular momenta are preserved. In our case, it will enable us to *correct* the integration.

Preservation of linear momentum

One of the most important features in animation is the preservation of linear momentum, as any error here will directly affect the motion in an awkward way. Fortunately, the implicit integration scheme preserves this quantity, in spite of the artificial viscosity and the filtering. Indeed, we note that the sum of all the artificial viscosity forces is zero:

$$\sum_{i=1}^{n}\tilde{\mathbf{F}}_i = dt\sum_{i=1}^{n}\left(\sum_{j=1}^{n}k_{ij}\,(\mathbf{v}_j-\mathbf{v}_i)\right)$$

[1]Actually, the real Hessian matrix should be $3n\times 3n$, but as each entry is a constant times the 3×3 identity matrix, we store only the constant as both memory requirement and computational time are then optimized.

$$= dt \sum_{i<j} k_{ij} \left[(\mathbf{v}_j - \mathbf{v}_i) + (\mathbf{v}_i - \mathbf{v}_j) \right] = \mathbf{0}$$

Similarly, the filtering of the force field doesn't affect the linear momentum since, knowing that $\sum_{j=1..n} W_{ij} = 1$, we can write:

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + dt \frac{\sum_{j=1}^n \mathbf{F}_j W_{ij}}{m}$$
$$= \mathbf{v}_i^n + dt \frac{\mathbf{F}_i}{m} + \frac{dt}{m} \sum_{j=1}^n (\mathbf{F}_j - \mathbf{F}_i) W_{ij}.$$

But then:

$$\sum_{i=1..n} \left(\sum_{j=1..n} (\mathbf{F}_j - \mathbf{F}_i) W_{ij} \right) = \sum_{\text{all } (i,j)} (\mathbf{F}_j - \mathbf{F}_i) W_{ij} = \mathbf{0}$$

as W is symmetric. Therefore, the motion of the center of gravity is the same as in the explicit case.

Preservation of angular momentum

Another important physical quantity is the angular momentum. As with linear momentum, any loss may be noticed immediately. Regrettably, our integration scheme doesn't preserve this quantity. In practice, the stiffer the springs, the bigger the loss, which is not surprising since we made a deliberate approximation that introduces angular errors.

Fortunately, we can balance this loss by a post-correction on the position. Once the internal forces have been filtered (i.e., multiplied by the constant matrix W), we can compute the resulting global torque δT :

$$\delta T = \sum_{i=1}^n (\mathbf{x}_G - \mathbf{x}_i) \wedge \mathbf{F}_i^{\mathit{filtered}}$$

where $\mathbf{x}_G$ is the center of gravity. Actually, as the sum of all internal forces must be zero (action/reaction law), we use the more direct expression:

$$\delta T = \sum_{i=1}^n \mathbf{F}_i^{\mathit{filtered}} \wedge \mathbf{x}_i.$$

This torque should be zero, so we need to modify the integration output to balance it. We can simply add the correction force:

$$\mathbf{F}_i^{\mathit{correc}} = (\mathbf{x}_G - \mathbf{x}_i) \wedge \delta T \, dt$$

on each mass point to compensate for the angular velocity change. This expression is found by making the assumption of a unit inertia matrix, and by linearly approximate the rotation. Although more accurate would be easy to use, we found these approximations to be satisfactory. Note that the sum of all these correction forces is zero, so that this force field just corrects the angular momentum without affecting the linear momentum. In practice, we add $\mathbf{F}_i^{\mathit{correc}} dt^2/m$ to the position of every mass point. We do not have a zero-error scheme, as we only compensate for the *overall* torque error, and may miss local variations of torques. But tests show that the behavior begins to be implausible only for high stiffness or very large time steps, which will be addressed in section 4.

3.7 Discussion

Once the angular momentum has been re-adjusted, the animation obtained using the above scheme is satisfactory for moderate stiffness. However, as local torques have been overlooked, this simplified scheme performs badly for high stiffness without a post-correction process: even if the animation remains stable, we obtain wrinkled meshes. We thus have to add a post-correction, which is the subject discussed in the next section.

4 Post-step modification: inverse dynamics

4.1 Motivation

Springs are certainly not a perfect physical model for real clothes or real deformable objects. Roughly speaking, their elongation is proportional to the force applied, which may result in implausibly large deformation. The common force/deformation curve for a material is nonlinear as sketched in Fig. 4. So we must modify the behavior of our mass-spring system to make it more realistic. One way to

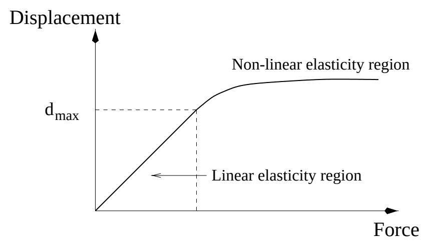

Figure 4: *Force/Displacement curve simulated by our model*

achieve this is to add a *post-correction* phase after a time step. This post-correction can then be considered as a *constraint enforcement*: all the mass points are first advanced normally, then we modify their positions to enforce a desired constraint. Various approaches have proposed to iterate small displacements until constraints are met [7, 15, 6].

4.2 Inverse dynamics process

In our context, we use an adequate and straightforward post-step modification of mass points to eliminate large stretch as defined in [18], where more details can be found. The underlying idea is the following: Each time a spring is overstretched, bring the two extreme mass points together along their axis while preserving the position of the center of gravity of these two masses. If one of the two mass points is constrained at a given position, just move the other one to ensure a reasonable elongation. By doing this for each spring and iterating, the resulting position both satisfies the external constraints (if the mass-spring system is grabbed for instance) and simulates a nonlinear behavior as springs are shrunk if there is an unwanted stretch. As this method is an inverse dynamics process that does not involve forces, stability is not an issue. Since it is similar to a Jacobi iteration, the convergence properties may not be ensured; however, in our case where accurate convergence is not needed, this does not cause problems.

4.3 Implementation

In practice, we define a normalized threshold d_{max} which represents the limit of proportionality in the desired force/deformation curve for our springs. Then we iterate over the springs exceeding this threshold and shrink them as explained in the previous paragraph. The criterion to end the iteration can be chosen in various ways: either after a predefined number of iterations (and it seems sufficient according to our tests), or until convergence is reached (but we may then lose real-time performance), or even until convergence is reached or time is up (as we need to display new positions at interactive rates). As this phase is only a "polishing" process, stopping the iterations before final convergence doesn't create noticeable visual artifacts. This simple procedure provides the final touch to complete our model: we now have a way to simply deal with constraints due for instance to collision or user interaction. And as this inverse dynamics process (or any other having the same properties) keeps the mass-spring system from being over-stretched, it enhances the realism of the overall animation.

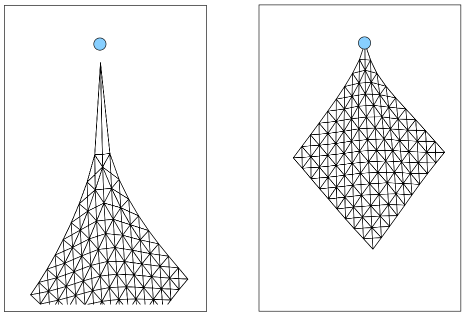

Figure 5: *Left: after one step of implicit integration, large deformation appears near the grabbed mass point, and constraints are sometimes not met. Right: after the inverse dynamics process, we ensure both a more natural look of the material and the enforcement of constraints.*

5 Animation algorithm

We sum up the whole process in this following pseudo-code in Fig. 6. An inspection of this algorithm, putting the very first off-line computation of W aside, shows that the theoretical complexity should be quadratic in the number of mass points because W is not sparse as H is. However, the convolution of the force field with W is actually performed quite efficiently compared to the evaluation of the forces themselves. The observed overall complexity turns out to be linear for the range tested (number of mass points$<$ 1000), regardless of stiffness or time step.

```
Precompute $W = (\mathbf{I}_n - \frac{dt^2}{m}H)^{-1}$
At each time step $dt$
    //Compute internal forces $\mathbf{F}_i$
    //due to springs and artificial viscosity.
    $\mathbf{x}_G = \mathbf{0}$
    For each mass point $i$
        $\mathbf{F}_i = \mathbf{0}$
        $\mathbf{x}_G + = \mathbf{x}_i$
        For each mass point $j$ such as $(i,j)$ linked by a spring
            $\mathbf{F}_i + = k_{ij}\,(||\mathbf{x}_i - \mathbf{x}_j|| - l_0^{ij})\,\frac{\mathbf{x}_i - \mathbf{x}_j}{||\mathbf{x}_i - \mathbf{x}_j||}$
            $\mathbf{F}_i + = k_{ij}\,dt\,(\mathbf{v}_j - \mathbf{v}_i)$
    $\mathbf{x}_G / = n$
    $\delta T = \mathbf{0}$
    // Integrate the approximation (predictor, see section 3)
    For each mass point $i$
        $\mathbf{F}_i^{filtered} = \sum_j \mathbf{F}_j W_{ij}$
        $\delta T + = \mathbf{F}_i^{filtered} \wedge \mathbf{x}_i$
        $\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \left[\mathbf{F}_i^{filtered} + \mathbf{F}_i^{ext}\right]\frac{dt}{m}$
        $\mathbf{x}_i^{new} = \mathbf{x}_i + \mathbf{v}_i^{n+1}\,dt$
    // Post correction of angular momentum (corrector, see section 3.6)
    For each mass point $i$
        $\mathbf{F}_i^{\mathrm{correc}} = (\mathbf{x}_G - \mathbf{x}_i) \wedge \delta T\,dt$
        $\mathbf{x}_i^{new} + = \mathbf{F}_i^{\mathrm{correc}}\,\frac{dt^2}{m}$
    // Now, use the inverse dynamics (see section 4)
    $nbIter = 0$
    do
        Post-step inverse dynamics (as in [18] for instance)
        $nbIter = nbIter + 1$
    until ($error < \varepsilon$) or ($nbIter > nbIterMax$) or ($time\ is\ up!$)
    // Update real velocity and position
    $\mathbf{v}_i^{n+1} = (\mathbf{x}_i^{n+1} - \mathbf{x}_i^n)/dt$
    $\mathbf{x}_i = \mathbf{x}_i^{new}$
```

Figure 6: *Pseudo-code of our algorithm.*

6 Results

We implemented this algorithm in 2D and 3D. We tested it with cloth-like material first, then with volumetric objects. In 3D, we used an immersive environment, namely the Responsive Workbench [11], to be able to virtually manipulate the animated objects. We allow the user to grab the object with a stylus (a mouse with 6DOF), and move it around the scene containing a ground plane and diverse obstacles, with friction on them all. It is important to note here that no air friction or low gravity has been needed to achieve perfectly stable results.

Quality of results

The visual results indicate that the approximated implicit integration mixed with the post-step inverse dynamics process achieves simulation of deformable objects very well. Even without force-feedback devices, one can really feel as if manipulating either a piece of cloth, or a soft object. Figures 7 and 8 show various snapshots obtained during such virtual reality sessions, with different stiffnesses ranging from zero (no internal forces, so only the inverse dynamics is applied, resulting in simulation of a mesh made of rigid rods, like chain mail) to 10^6. We also tried to add damping in the system, as W is still constant in this case. Our tests of whether such an additional parameter is worthy for an animator were inconclusive. Needless to say that such a range of stiffness simulated using an explicit scheme would have required time steps of the order of 10^{-6} minimum to be stable. In our experiments, we used $dt = 0.02$, corresponding to 50 frames/s.

Comparison with a real implicit scheme

We compared our computational times with [1]. It appears that on average, our algorithm is 10 times faster than the conjugate gradient iterations for the pictures shown in this paper (between 120 and 400 masses). It is important to mention that when an iterative solver is used as in [1], the number of iterations required to converge varies a lot depending on the user interaction. Although the number of iterations of our post-correction may slightly depend on how much the piece of cloth has been moved in a time step, it seems that our technique is significantly more robust. Also, we maintain a fixed time step throughout the animation, which is vital for real-time applications. The speed up factor is thus hard to define precisely, but it decreases rapidly with the number of mass points used. It seems reasonable to consider our technique as appropriate only for objects with less than 1000 DOF.

Implementation details

Our implementation includes friction on obstacles, and uses a very simple plastic collision model, perfectly fine for cloth, where mass points are pulled back on the surface of the obstacles and have their normal velocity cancelled. However, as the restitution coefficient is zero, this is not appropriate for other 3D objects. We thus tested volumetric objects onyl in surgery-like conditions, where an organ is partially fixed and can only be pressed or pinched. In those conditions, any structured deformable object can be simulated with great efficiency.

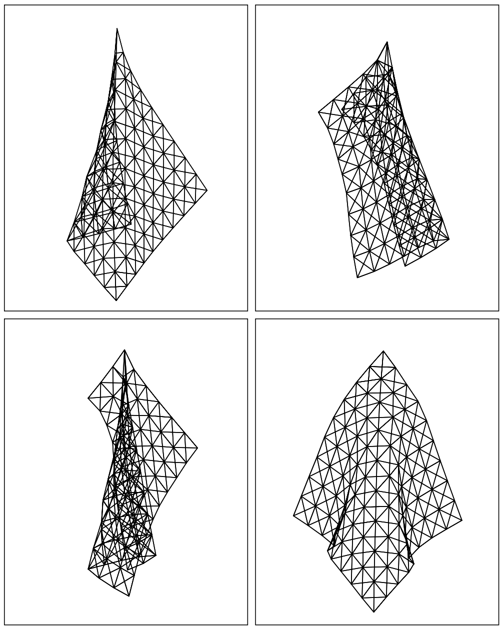

Figure 7: *Different hanging postures of a cloth-like material animated with our algorihtm*

7 Conclusion & Discussion

We have presented a new algorithm to animate efficiently any mass-spring system. As we can perform the time integration for any time step size with a constant computational time, our method is perfectly designed for real-time interaction in virtual reality environments. The algorithm is an approximated implicit integration scheme, but we preserve important physical quantities such as linear and angular momenta, vital for realism. This technique can thus be classified as an *implicit predictor/corrector* scheme. It handles constraints and collisions in a very simple way, and uses nonlinear springs that enhance the visual behavior of the motion.

We do not claim to have great generality. Although this approach is very convenient for virtual reality where bullet-proof and real-time methods are needed, this kind of approximation cannot be used for more accurate simulations. But we believe that a clear undestanding of implicit integration may be the key to more general approaches. Seeing the integration process as a *smoothing process*, to get rid of high frequencies, is particularly important for anyone willing to obtain robust animation with a fixed time step (it is also closely related to Laplacian smoothing in modeling, as explored in [19, 5]). The correction step introduced is also interesting as it provides a useful sanity-check that will ensure the preservation of vital invariants, as symplectic methods do.

Color pictures from this paper can be found at:
`http://www.cs.caltech.edu/~mathieu`

8 Acknowledgments

The authors would like to thank Mark Meyer for a part of the implementation, Steven Schkolne for proof-reading, Fehmi Cirak and François Faure for useful comments on physics, and Frédéric Cazals and Gilles Debunne for advice. The research reported in this paper was supported in part by DOE (W-7405-ENG-48), NSF (ACI-9721349, EIA-9871235), and the NSF STC for Computer Graphics and Visualization. Other support was provided by DesignWorks/USA and through a Packard Fellowship.

9 References

[1] David Baraff and Andrew Witkin. Large steps in cloth simulation. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, July 1998.

[2] John E. Chadwick, David R. Haumann, and Richard E. Parent. Layered construction for deformable animated characters. *Computer Graphics*, 23(3):243–252, July 1989.

[3] Stéphane Cotin, Hervé Delingette, and Nicolas Ayache. Real time volumetric deformable models for surgery simulation. In *Proceedings of Visualization in Biomedical Computing*, volume Lectures Notes in Computer Science, volume 11, September 1996.

[4] Mathieu Desbrun and Marie-Paule Gascuel. Animating soft substances with implicit surfaces. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 287–290. ACM SIGGRAPH, Addison Wesley, August 1995. Los Angeles, CA.

[5] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH 99 Conference Proceedings*, to appear in August 1999. Los Angeles, CA.

[6] François Faure. Interactive solid animation using linearized displacement constraints. *9th Eurographics Workshop on Computer Animation and Simulation*, September 1998.

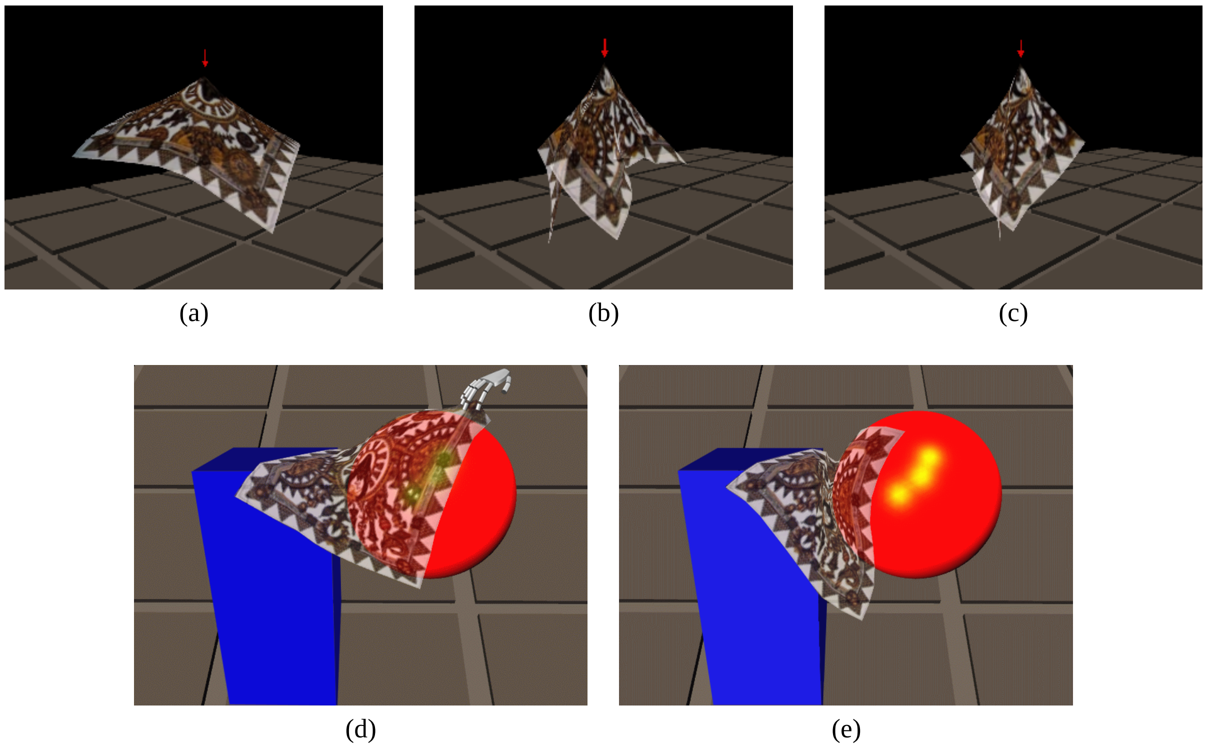
Figure 8: *(a)–(c): Various rest positions for different stiffnesses obtained during a real-time session. (d)&(e): The scarf is put on two obstacles, and slides in between. Time step used:* 0.02.

[7] Jean-Dominique Gascuel and Marie-Paule Gascuel. Displacement constraints for interactive modeling and animation of articulated structures. *The Visual Computer*, 10(4):191–204, March 1994. An early version of this paper appeared in the *Third Eurographics Workshop on Animation and Simulation*, Cambridge, UK, Sept 92.

[8] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 9–20. ACM SIGGRAPH, Addison Wesley, July 1998.

[9] Ammar Joukhadar. Adaptive time step for fast converging dynamic simulation system. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, volume 2, pages 418–424, November 1996.

[10] Michael Kass. An introduction to continuum dynamics for computer graphics. In *SIGGRAPH Course Notes*. ACM SIGGRAPH, 1995.

[11] W. Krueger and B. Froehlich. The responsive workbench (virtual work environment). *IEEE Computer Graphics and Applications*, 14(3):12–15, May 94.

[12] A. Luciani, S. Jimenez, J-L. Florens, C. Cadoz, and O. Raoult. Computational physics: a modeler simulator for animated physical objects. In *Eurographics'91*, Vienna, Austria, September 1991.

[13] Gavin Miller. The motion dynamics of snakes and worms. *Computer Graphics*, 22(4):169–177, August 1988. Proceedings of SIGGRAPH'88 (Atlanta, Georgia).

[14] Gavin Miller and Andrew Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics*, 13(3):305–309, 89. This paper also appeared in SIGGRAPH'89 Course notes number 30.

[15] C. Van Overveld. An iterative approach to dynamic simulation of 3-D rigid-body motions for real-time interactive computer animation. *The Visual Computer*, 7:29–38, 1991.

[16] Alex Pentland and John Williams. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics*, 23(3):215–222, July 1989. Proceedings of SIGGRAPH'89 (Boston, MA, July 1989).

[17] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. *Numerical Recipes in C, second edition*. Cambridge University Press, New York, USA, 1992.

[18] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface*, pages 147–154, June 1995.

[19] Gabriel Taubin. A signal processing approach to fair surface design. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 351–358. ACM SIGGRAPH, Addison Wesley, August 1995.

[20] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *Computer Graphics*, 21(4):205–214, July 1987. Proceedings of SIGGRAPH'87 (Anaheim, California).

[21] Demetri Terzopoulos, John Platt, and Kurt Fleisher. Heating and melting deformable models (from goop to glop). In *Graphics Interface'89*, pages 219–226, London, Ontario, June 1989.

[22] Demetri Terzopoulos and Andrew Witkin. Physically based model with rigid and deformable components. *IEEE Computer Graphics and Applications*, pages 41–51, December 1988.

[23] Andrew Witkin and William Welch. Fast animation and control for non-rigid structures. *Computer Graphics*, 24(4):243–252, August 1990. Proceedings of SIGGRAPH'90 (Dallas, Texas, August 1990).

[24] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method*. McGraw-Hill, 1991.