Interactive Animation of Cloth-like Objects in Virtual Reality

Mark Meyer — Caltech
Gilles Debunne — iMAGIS/IMAG
Mathieu Desbrun — USC/Caltech
Alan H. Barr — Caltech

Abstract

Modeling and animation of cloth has experienced important developments in recent years. As a consequence, complex textile models can be used to realistically drape objects or human characters in a fairly efficient way. However, real-time realistic simulation remains a major challenge, even if applications are numerous, from rapid prototyping to e-commerce. In this paper, we present a stable, real-time algorithm for animating cloth-like materials. Using a hybrid explicit/implicit algorithm, we perform fast and stable time integration of a physically-based model with rapid collision detection and response, as well as wind or liquid drag effects to enhance realism. We demonstrate our approach through a series of examples in VR environments, proving that real-time animation of cloth, even on low-end computers, is now achievable.

1 Introduction

Interactive animation of non trivial objects, with complex behavior like cloth flapping, remains a challenge in Computer Graphics and in VR. Although some algorithms exist to animate objects in real-time, they are currently restricted to rigid objects [9], or very specific kinds of elastic objects without the integration of dynamics. Moreover, animation in immersive environments put constraints on the algorithm as everything needs to be bullet-proof against any user action, while also ensuring a constant frame rate. This paper proposes a simple method that leads to animation of complex behaviors for cloth-like objects, while guaranteeing both a low computational cost and unconditional stability - making it ideal for real-time, interactive applications like Virtual Reality.

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 performance, is the mass-spring system [19, 16, 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 [26]. Easy to implement, highly parallelizable, and involving few computations, it seems a perfect candidate for virtual reality applications. Recently, improvements to this model have been made, such as an post-integration step to bound the stretch of springs [23], as well as adaptive time stepping to preserve the system's global energy [13].

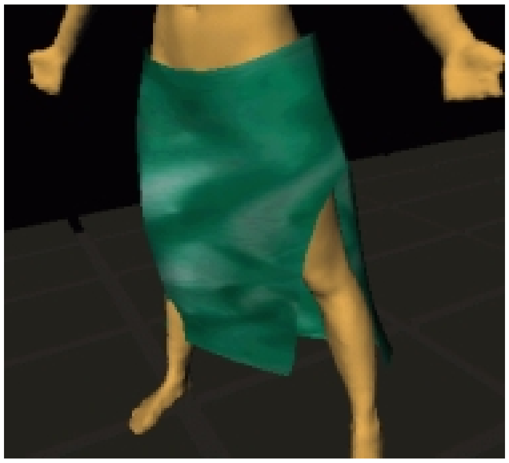

Figure 1. *Picture captured during a live session using our cloth model. The skirt was wrapped and seamed in real time.*

Unfortunately, all of these approaches suffer from the same problem: the time step must be inversely proportional to the square root of the stiffness. While this may not be an issue for off-line computations, it prevents much use in real-time applications since very small time steps are required to ensure stability. Various ways to overcome this problem have been proposed. 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 molasses. Gravity has often been modified (lowered) to avoid large forces in the system, but once again, it introduces an 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 [7, 25, 27, 26], but these methods suffer from the same time step handicap. Global methods, gaining efficiency by restricting the possible deformations [21, 31], are perfect for interactive manipulation, but, unfortunately, offer limited realism.

To the authors' knowledge, only few existing approaches achieve real-time computations for deformable structured objects. The first is derived from finite element theory, and takes advantage of linear elasticity to allow real-time deformation of any meshed object [3, 12]. However, this model is not dynamic, but rather a collection of static postures, greatly limiting its potential applications. A second approach is the recent development of neuro-animators [11]: 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. Debunne *et al.* [4] have recently introduced a technique for animating soft bodies in real time. However, as the technique requires the objects to have finite volume, it is not applicable to thin objects 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. However, implicit integration has the drawback of solving a large linear system at each time step. Inspired by this approach, we recently developed [6] an algorithm that alleviates this shortcoming.

In this paper, we extend this system to include nonlinear external forces and collisions with complex dynamic objects. The end result is a fast and stable algorithm to animate arbitrarily connected mass-spring systems in complex, dynamic environments.

1.2 Overview

We build upon our previous work and present a novel combination of implicit integration, explicit integration, and post-step correction to efficiently simulate cloth-like objects in real-time. We will show that the internal forces, often very large for common materials like fabrics, have to be integrated using an implicit integration. Additional external forces, however, can be explicitly integrated without problem. We will demonstrate how to mix these two integration techniques to obtain real-time animation.

We will first review our physical model in Section 2, and explain why usual integration techniques are completely inadequate for Virtual Reality. In Section 3, we will offer a review of a new technique originally introduced in [6] that will allow us to animate simple deformable objects in real-time. We will extend this technique in Section 4 by adding external forces, such as collision or wind drag, while still focusing on very low computational times to ensure real-time applications. Finally, we will recap the entire algorithm in Section 5, present results in Section 6 and discuss our conclusions in Section 7.

2 Physical Model

As mentioned in the introduction, we simply cannot afford a complex physically-based model if real-time applications are needed. We therefore choose a simple mass-spring system, as it seems to be the most straightforward model in terms of computational complexity. For the sake of generality, we will consider that a mass point i is linked to all the others with (linear) springs of rest length l_{ij}^0 and stiffness k_{ij}. This stiffness value is set to zero if the actual model does not contain a spring between masses i and j. In the remainder of this paper, we will focus on 2-manifold deformable objects like clothes, paper sheets, or membranes, but our results are still valid for any 3D object. We will also use the following notation:

- $\mathbf{x}$ is the (time varying) 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.
- 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: $\Delta^{n+1}\mathbf{x} = \mathbf{x}^{n+1} - \mathbf{x}^n$.

2.1 Problems with explicit integration

To animate such a simple 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 and its apparent low computational complexity, the explicit Euler scheme requires an integration time step dt inversely proportional to the square root of the stiffness (this criterion is more generally know in physics as the Courant condition [32]). Otherwise, the system will diverge rapidly since assuming the internal forces as constant over too large a time step may induce a wild change in position. In practice, we effectively notice a stable behavior of the system only for very small time steps. This is the general problem of *stiff sets of equations*: stability can be achieved only at a very small time scale with explicit schemes [22]. Therefore, the use of explicit schemes in Virtual Reality is often unrealistic in practice: computational times are commonly large, forcing the overall animation to be between 10 and 1000 times slower than real-time.

2.2 Introducing implicit integration

Another scheme, called *implicit Euler integration*, has proven to be much more adapted to such a problem [14, 1, 6, 5]. The basic idea is to replace the forces at time t by the forces at time $t+dt$:

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

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. Using an approximation of the force at the next time step introduces a sort of feedback to the integration process, strongly increasing stability. 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 a different way, we can say that an explicit scheme takes a step into the unknown knowing only 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):

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

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. Details on the underlying mathematics of implicit integration can be found in [6]. Turning the crank on the equations, and using the backward operator $\Delta^{n+1}\mathbf{x} = (\mathbf{v}^n + \Delta^{n+1}\mathbf{v})\,dt$, the update rule using implicit integration is given by the following:

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

Note that this is a linear system. We will later refer to the matrix $(\mathbf{I} - \frac{dt^2}{m}H)^{-1}$ as W for the sake of simplicity ($\mathbf{I}$ is the identity matrix).

2.3 Practical differences between explicit/implicit

Our previous paper described all the equations resulting from this implicit technique, and this provides an intuitive interpretation of what exactly happens when implicit integration is used instead of explicit integration [6]. Roughly speaking, two main factors make implicit integration stable.

Addition of artificial viscosity

First, implicit integration “implicitly” adds a set of extra forces to the internal forces of the system. These extra forces turn out to be equivalent to an *artificial viscosity* proportional to both stiffness and time step size, just like Rayleigh damping forces. It intuitively means that each mass point will tend to follow the local displacement of its neighbors, avoiding wild local instabilities. Notice that since our artificial viscosity depends on both the stiffness and time step size (the two factors responsible for instabilities), we are able to add just the right amount of viscosity to each individual mass point.

Filtering of the force field

Secondly, using implicit integration also amounts to a *filtering* of the force field (multiplying by the matrix W). This can be understood quite intuitively by a signal processing analogy: given a sampling rate of a signal, only a fixed range of frequencies can be faithfully reproduced (Shannon’s theorem). When it comes to animation, a simulation with a fixed time step dt can only handle deformation frequencies up to a certain limit. Past this limit, the sampling rate is not sufficient and instabilities may (and most probably will) occur. Therefore, one way to ensure stability is to filter the force field so that no high frequencies can remain. As the matrix W is based on both the stiffness and the time step, it uses just the right amount of filtering automatically, without any parameter tweaking by the user. We thus guarantee an unconditionally stable animation, whatever the user may do.

2.4 Discussion

This technique, introduced only recently in Computer Graphics, has proven to have very good performance. Baraff and Witkin for instance used implicit integration in the context of cloth animation with great success [1]. Previous techniques using explicit integrations, however sophisticated they may be, have to use time step sizes of the order of 10^{-7} second, while implicit integration can often handle exactly the same animation with a time step size of typically 10^{-2}. Although standard implicit integration requires solving a linear system at each time step, it is usually sparse and can thus be efficiently solved. The performance benefits of using implicit integration for deformable objects, in consequence, can be extremely significant.

3 A rapid implicit-based integration scheme

As we have just seen in the previous section, there are definite advantages in using implicit integration in a VR context where stable and quick results are needed. Unfortunately, a direct implementation requires solving a linear system [1]. Even if this linear system is most commonly very sparse, techniques such as conjugate gradient have an overhead that prevents any real-time solving. We propose in this section a variant of this implicit integration technique. By approximating the implicit integration in an explicit way, and then correcting the errors created, we can efficiently simulate an implicit integration with a low computational cost. Once again, more details about this technique can be found in [6].

3.1 Approximate implicit integration

Contrary to the 1D case, the Hessian matrix H is not constant in 3D. Due to the non-zero rest lengths of springs, the forces are not linear. Therefore, we have to solve a different linear system at each time step of our animation, which, as briefly mentioned above, is simply impossible in real-time with current computers.

To overcome this difficulty, we can linearize the force field by temporarily assuming a zero rest length for all springs. By doing so, we obtain a constant Hessian matrix, as well as a constant matrix W. Therefore, we compute W once as a pre-process to our animation and alleviate the need to solve a linear system at each step. For the reader's convenience, we reproduce here the coefficients of the matrix H, which can be found in [6]:

$$\left\{ \begin{array}{ll} H_{ij} = k_{ij} & \text{if } i \neq j \\ H_{ii} = -\sum_{j \neq i} k_{ij} & \end{array} \right. \tag{4}$$

Integrating the motion at each time step then amounts to a simple matrix-vector multiplication, just as in explicit integration. This results in a very efficient way to perform an implicit integration [6]. However, the approximation we made about the rest length of the springs will inevitably induce more or less significant errors in the integration. The next section presents a way to compensate for these errors, guaranteeing a physically correct result anyway.

3.2 Correction of momentum

As we know that our approximate implicit integration creates inaccuracies, we need to double-check basic physical properties to ensure a plausible result. Linear and angular momenta, for instance, must always be zero during a time integration of internal forces. Preserving these invariants will then enable us to correct the previous approximation.

Linear momentum is actually preserved with our technique, even though the implicit integration is only approximated. Angular momentum, however, is not. Unfortunately, any loss of angular momentum is easily noticeable during an animation. In practice, the stiffer the springs, the bigger the loss of momentum, which is not surprising since we made a deliberate approximation that introduces angular errors [6]. We therefore need to compensate for this loss.

An easy solution is to compute the exact amount of angular momentum added to the system, and then balance this excess or loss by giving a "little push" to the masses. Once again using a linear approximation, we add a correction vector to each mass position such that:

- the sum of all these corrections is zero, leaving the linear momentum unaffected,
- the induced angular rotation it creates balances the angular velocity error.

These correction vectors are very simple to compute [6], and are computationally inexpensive in an animation algorithm. The overall torque error is easily taken care of this way.

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 final correction, which is the subject discussed in the next section.

3.3 Post-step modification

3.3.1 Motivation

Springs are certainly not a perfect physical model for real deformable objects. Roughly speaking, their elongation is proportional to the force applied (linear elasticity), which may result in implausibly large deformations. The common force/deformation curve for a material is nonlinear, so we must modify the behavior of our mass-spring system to account for this. One way to achieve this is to add a *post-correction* phase after a time step. This will also perform the final correction discussed above.

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 been proposed to iterate small displacements until constraints are met [10, 20, 8].

3.3.2 Implementation

In our context, we use an adequate and straightforward post-step modification of mass points to eliminate large stretch as defined in [23] and in [6]. The underlying idea is simple: each time a spring is over-stretched, we 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, we simply 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 kinematics 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. In practice we can stop this process whenever the next frame is needed.

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 modification process keeps the mass-spring system from being over-stretched, it enhances the realism of the overall animation. Figure 2 shows different hanging postures obtained using the above algorithm. As mentioned, no implausible stretching appears, and the postures seem natural.

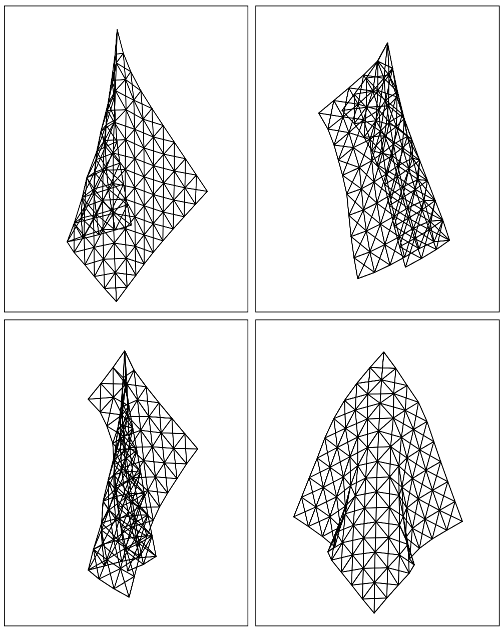

Figure 2. *Different hanging postures of a piece of fabric animated using our rapid implicit-based integration technique. Notice that there are no overly stretched springs. This leads to a natural looking behavior for our cloth. The different postures were obtained from a single simulation by shaking the cloth and sliding it over obstacles before allowing it to come to rest.*

4 Adding external forces

While our model behaves realistically when internal forces and gravity are taken into account, it cannot yet interact with more complex environments through either additional external forces like wind or through collisions with complex objects. In this section, we extend the possibilities of the previous algorithm by adding new features that will improve the visual motion complexity in a virtual environment.

4.1 Drag force for wind or fluid flow

The addition of nonlinear external forces, such as wind [30], would greatly enhance the realism of the simulation, especially for fabrics. In order to incorporate these nonlinear forces into standard implicit integration, we would need to compute the hessian matrix of the forces at every time step, a very expensive operation. As these nonlinear forces are external, we also cannot use our simple implicit-based algorithm either since their is no obvious invariant to preserve. However, as the external forces are usually much smaller than the internal forces of the object, we propose to *explicitly* integrate these external forces. As we will demonstrate in the result section, most external forces can be handled explicitly with standard Euler integration without any instability problems.

We use a nonlinear wind formulation in accordance with fluid laws in physics. Each mass point having approximately the same small surface area[1], we assign a drag force due to wind to a mass point i such as:

$$\mathbf{F}_i^{\text{drag}} = K_{\text{drag}}\,||\mathbf{v}_i^{\text{wind}}||\ \left(\mathbf{n}_i \cdot \mathbf{v}_i^{\text{wind}}\right)\mathbf{n}_i. \qquad (5)$$

In this equation, K_{drag} is a user specified coefficient, $\mathbf{v}_i^{\text{wind}}$ is the velocity of the wind (or any other fluid) at the position of mass point i, and $\mathbf{n}_i$ is the normal to the simulated surface at this same point. Since all the distances between mass point i and its neighbors stay approximately constant, the normal vector can be easily approximated by the following equation [5]:

$$\mathbf{n}_i = \frac{\sum_{\text{j neighbors of i}}(\mathbf{x}_i - \mathbf{x}_j)}{||\sum_{\text{j neighbors of i}}(\mathbf{x}_i - \mathbf{x}_j)||}$$

If the former sum turns out to be zero, then $\mathbf{n}_i$ can simply be measured as a normalized cross product of the edges between mass i and any two neighboring masses j and k: $\mathbf{n}_i = ((\mathbf{x}_j - \mathbf{x}_i) \wedge (\mathbf{x}_k - \mathbf{x}_i))/||((\mathbf{x}_j - \mathbf{x}_i) \wedge (\mathbf{x}_k - \mathbf{x}_i))||$.

Perfect respect of physical laws would require $\mathbf{v}_{\text{wind}}$ to be a *relative velocity*, the difference between the wind velocity and the mass point velocity. After many tests, however, we noticed that using only the wind velocity (instead of the

[1] Once again, we restrict our explanations to fabric-like material. Real 3D objects would be considered as exposed to wind only for mass points on the surface of the object, internal mass points would not be affected.

relative velocity) results in realistic effects and doesn't significantly alter the visual results. Additionally, using the relative velocity can create problems of stability as now, this speed can be arbitrarily large depending on how fast the user waves the object. To ensure stability whatever happens, we decided to used the simplified, yet satisfactory wind velocity alone.

4.2 Collision detection and response

Interactions with objects in a virtual scene is an important effect for a deformable object. Fortunately, we can handle object interactions as if they were special point-to-point constraints (see Section 3.3). Nevertheless, it is a costly process to constantly check whether or not a mass point is outside of all the obstacles in the scene. Almost as costly is the process of forcing the mass point back onto the surface of the obstacle it has penetrated.

In this section, we propose to accelerate both collision detection and collision response to be able to deal with multiple, complex, dynamic objects in our virtual environment. This results in a rapid collision detection and response system, which completes our model.

4.2.1 Collision detection

Collision detection has been well researched in Computer Graphics, both in the form of collision detection itself as well as in ray tracing. Many acceleration techniques take advantage of temporal and spatial coherence or hierarchies. This can create inconsistencies in the update rate as certain configurations will require much more computation than others. In interactive applications, inconsistent update rates and noticeable drops in performance are unacceptable. Therefore, the update rate should be both as fast and as consistent as possible.

We developed a simple, efficient voxel-based collision detection algorithm similar to McNeely *et al.* [17]. This algorithm provides us with fast, consistent update rates requiring little computation, but, as a side effect, requires a fair amount of memory.

As a preprocess, we enclose each object by a bounding box in object space. We then discretize this box into voxels whose size satisfies a user-defined accuracy for collisions. The voxels are then classified as inside, outside, or surface voxels according to whether they are totally inside the object, totally outside the object or have the surface passing through them.

At runtime, the objects and the cloth can be moved around in real time. To check whether a mass point has collided with an object, we transform the mass point into the object's space and determine which voxel it is in. If the voxel is an outside voxel, no collision has occurred. On the other hand, if the voxel is an inside or surface voxel, we must respond to the collision as described in the next section.

4.2.2 Collision response

Once a collision has been detected we must respond to ensure a non-penetrating result. In addition to storing a type (inside/outside/surface) in each voxel, we also store the closest point, *CP*, to the surface from the center of this voxel, *VC* (see Figure 3(a)), along with the normal **N** to the object at *CP*. In order to reduce the storage required for each voxel, the vector **N** can be approximated by

$$\mathbf{N} \approx (-1)^{\delta}\,(CP - VC)/||CP - VC||$$

(which is exact for any continuous surface), where δ is 1 if *VC* is outside the object and 0 otherwise. The calculation of the closest point and normal is done as a preprocess, similar to the typing of each voxel.

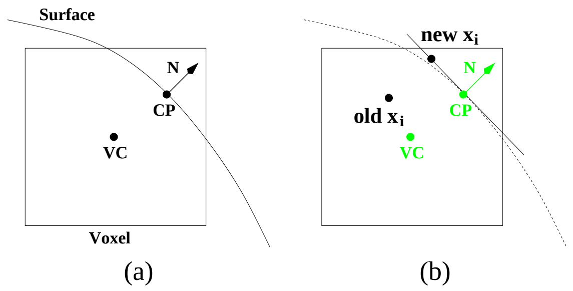

Figure 3. *(a) For each voxel, we store the closest surface point CP from the center of the voxel VC, along with the normal* **N** *at CP. (b) For a mass point inside this voxel, the closest point will be found as the projection onto the tangent plane of the surface at CP.*

Once the closest point from each voxel is known, we still must define how to project a mass point $\mathbf{x}_i$ that has penetrated back onto the surface. An initial attempt, consisting in directly moving the mass point to *CP*, resulted in a jittered motion, because as the point *CP* may be the closest point to *VP*, it is most likely not the closest point to $\mathbf{x}_i$ itself. Instead, we chose to approximate the surface by its tangent plane, in essence, a linearized representation of the surface inside each voxel. Given the normal **N** and the closest point *CP*, we project the mass point from its current position $\mathbf{x}_i$ to the tangent plane by the following operation:

$$\text{new } \mathbf{x}_i = \text{old } \mathbf{x}_i + (\mathbf{N} \cdot (CP - \mathbf{x}_i))\mathbf{N}. \qquad (6)$$

Figure 3(b) shows the result of such an operation. Note also that we first test the sign of the above dot product: if this dot product is negative, the mass point is actually *out* of the object, and no projection is required. This tangent plane projection algorithm avoids any of the visual jittering of the closest point projection algorithm.

One may notice that our collision response algorithm may allow the mass points to penetrate the object within a surface voxel. We overcome this limitation by offsetting the surface outwards by a small amount, creating what we call an ε shell. This shell serves to both alleviate any problems of surface penetration as well as ease rendering, as we no longer need to deal with coplanar cloth and surface polygons.

As a final addition to our collision response algorithm, we add an approximation of friction to enhance realism. Each time a mass point is put back onto the surface of an object, we attenuate the tangential component of the velocity to simulate friction on the surface. We also significantly attenuate the normal component (multiplying by a coefficient of restitution) to mimic a loss of energy due to the impact. These additions create visually realistic cloth effects, completing our collision response algorithm.

4.2.3 Discussion

Our collision detection and response system has several interesting features that warrant further mention. First, as we discretize each object separately, we are able to have multiple, dynamic objects in the scene. This individual discretization also allows for a separate discretization rate for each object; complex shapes may require a large number of voxels while simpler objects need fewer voxels. This technique does have some drawbacks, however. As the number of objects grows, so does the number of collisions that must be tested. We can reduce this increase by grouping like moving objects into a single, compound object before discretization. We then only need to test for intersection with this compound object, as opposed to each individual object. This compound object technique also allows us to group all static objects in the scene into a single scene object, thereby alleviating multiple collision tests. Also, these static objects do not require the transformation into object space yielding even greater efficiency. As these voxel representations can be memory intensive, we can also use a limited depth octree [17] to reduce the memory burden.

Currently, we make no provision to ensure that the cloth does not pass through small, pointy objects (tunneling). Nevertheless, in our context of clothing virtual humans, this has not been a problem as the objects' shapes and the frame to frame coherence of the mass positions guarantee that no tunneling occurs. Also, the above algorithm does not explicitly handle cloth to cloth interactions. However, existing techniques such as [24] and [28] can be integrated into our system. These techniques then handle the cloth to cloth interactions while the above algorithm handles cloth to object interactions.

Although more accurate collision algorithms exist, the requirement of real time user interaction in a complex, dynamic environment usually prohibits their use. Our algorithm offers a fast, consistent update rate for scenes involving multiple, complex, dynamic objects. As the algorithm is both easy to implement and computationally efficient, it has immediate application in interactive simulation and virtual reality.

5 Animation algorithm

We sum up the whole process in the following self-explanatory pseudo-code in Fig. 4.

```
Precompute $W = (\mathbf{I}_n - \frac{dt^2}{m} H)^{-1}$
At each time step $dt$
    //Reset the barycenter
    $\mathbf{x}_G = \mathbf{0}$
    //Compute internal forces $\mathbf{F}_i$
    //due to springs and artificial viscosity.
    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
            $\mathbf{F}_i += k_{ij} \, (||\mathbf{x}_i - \mathbf{x}_j|| - l_0^{ij}) \frac{\mathbf{x}_j - \mathbf{x}_i}{||\mathbf{x}_j - \mathbf{x}_i||}$
            $\mathbf{F}_i += k_{ij} \, dt \, (\mathbf{v}_j - \mathbf{v}_i)$
    //Final value of barycenter
    $\mathbf{x}_G /= n$
    $\delta T = \mathbf{0}$
    // Integrate the approximation (predictor, see [6])
    For each mass point $i$
        $\mathbf{F}_i^{filtered} = \sum_j \mathbf{F}_j W_{ij}$
        $\delta T += \mathbf{F}_i^{filtered} \wedge \mathbf{x}_i$
        $\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \left[\mathbf{F}_i^{filtered} + \mathbf{g}\right] \frac{dt}{m}$
        $\mathbf{x}_i^{new} = \mathbf{x}_i + \mathbf{v}_i^{n+1} \, dt$
    // Post correction of angular momentum (corrector, see [6])
    For each mass point $i$
        $\mathbf{R}_i^{\mathrm{correc}} = (\mathbf{x}_G - \mathbf{x}_i) \wedge \delta T$
        $\mathbf{x}_i^{new} += \mathbf{R}_i^{\mathrm{correc}} \frac{dt^2}{m}$
    // Add external secondary forces such as wind
    For each mass point $i$
        $\mathbf{x}_i^{new} += \mathbf{F}_i^{\mathrm{drag}} \frac{dt^2}{m}$
    // Now, use the inverse kinematics (see section 3.3)
    $nbIter = 0$
    do
        Post-step inverse dynamics (as in [23] for instance)
        While doing so, detect collision. (see Section 4.2.1)
        If collision, bring back onto the surface. (Equation 6)
        $nbIter = nbIter + 1$
    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 4. *Pseudo-code of our algorithm.*

6 Results

All the algorithms described in this paper have been implemented and tested in a virtual environment, using our version of the Responsive Workbench [15]. All the examples that we describe below work in real-time, at 50 Hz (dt=0.02), within a stereo display environment. We used a single-processor Infinite Reality Onyx 2. Interactive frame rates can also be achieved on Pentium based PCs (PII 333MHz) driving monocular displays. Additionally, significant improvements can be achieved if parts of the algorithm are parallelized.

This set of examples has been made to demonstrate the feasibility and the stability of our method. A video, demonstrating real-time sessions in a VR environment using our cloth technique can be found at *http://www.cs.caltech.edu/~mmeyer/Research/Cloth*. Portions of the video are also available in the IEEE Virtual Reality 2000 video proceeding [18]. We now describe some of the examples shown on this video.

The naked scarf

The first animation shows the basic algorithm. The user manipulates a sort of scarf, with rudimentary obstacles (namely, a sphere and a box). Even in this simplistic environment, the motion we obtained is realistic. The scarf can be thrown away, put on an obstacle, dragged along the floor, etc. We note again that this demo is unconditionally stable, regardless of how the user may try to break the algorithm.

Addition of wind

To the previous example, we now add wind. At first, the wind has a constant velocity. This velocity can be tuned, but for obvious stability reasons, cannot go over a maximum magnitude. We also introduce gusts of wind to improve the realism. If the scarf gets behind an obstacle, it doesn't flap anymore as the obstacle makes a barrier against the wind. Even with our simple drag force, we obtain realistic wind flapping.

Complex obstacles

To prove the efficiency of our collision detection and response system, we also show that the scarf can interact with complex and numerous objects, like a mannequin head or a set of furniture (see Figures 5 and 6(a)). Thanks to our voxel-based algorithm, we can still perform all the computations of this demo at 50Hz.

Interaction with a river

Using exactly the same algorithm, we make a compelling example where the scarf, when put into the flowing water, gets dragged along with the current (see Figure 6(c)). The flowing water is made out of an animated texture, translated at a constant speed. This can provide a perfect base for a fishing game.

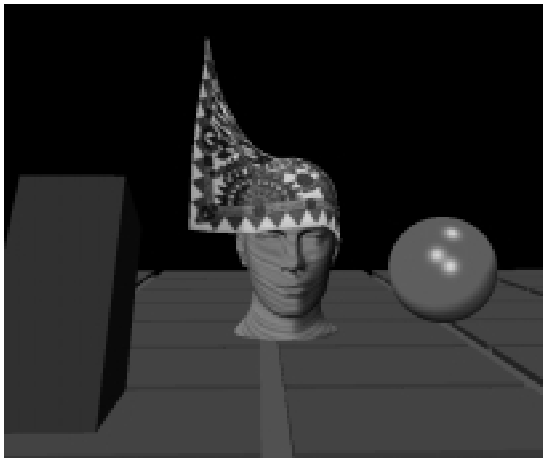

Figure 5. *Picture captured during live sessions in a VR environment: A silk scarf is moved around in a scene with complex obstacles in real-time.*

Variations: Ellipsoid Man and the flag pole

Finally, we present other interesting effects, such as the effect of the wind on the cape of a (rather simplistic) superhero (see Figure 6(b)). Now, we control the direction of the wind in real-time with a stylus (a 3D mouse with 6DOF). We also present the animation of a flag pole (see Figure 6(d)), with a gusting wind interactively controlled by the user.

Creating a Cloth Skirt

As a final example, we developed an interactive clothing design system in the vein of [29]. Our system supports cutting and seaming allowing complex garmets to be created from simple panels. In addition, we handle complex constraints, collision, and irregular discretizations. Using this system, users are able to interactively clothe characters and test the fit in a variety of situations.

In one instance (see Figure 1), the user interactively outfitted a female character with a skirt. The user first wrapped the skirt material around the virtual character by interactively modifying and constraining 3 points. An elastic waistband was then simulated by interactively reducing the rest lengths of the waist springs. To determine how this skirt would react to different situations, a wind force was simulated. From this simulation, it was easy to see that the open slit in the skirt allowed the skirt to drift upwards in the wind. A seam was deemed necessary and interactively created by the user (by selecting corresponding seam points with the mouse). Inspecting the new skirt design in the wind, the user found a much more pleasing result.

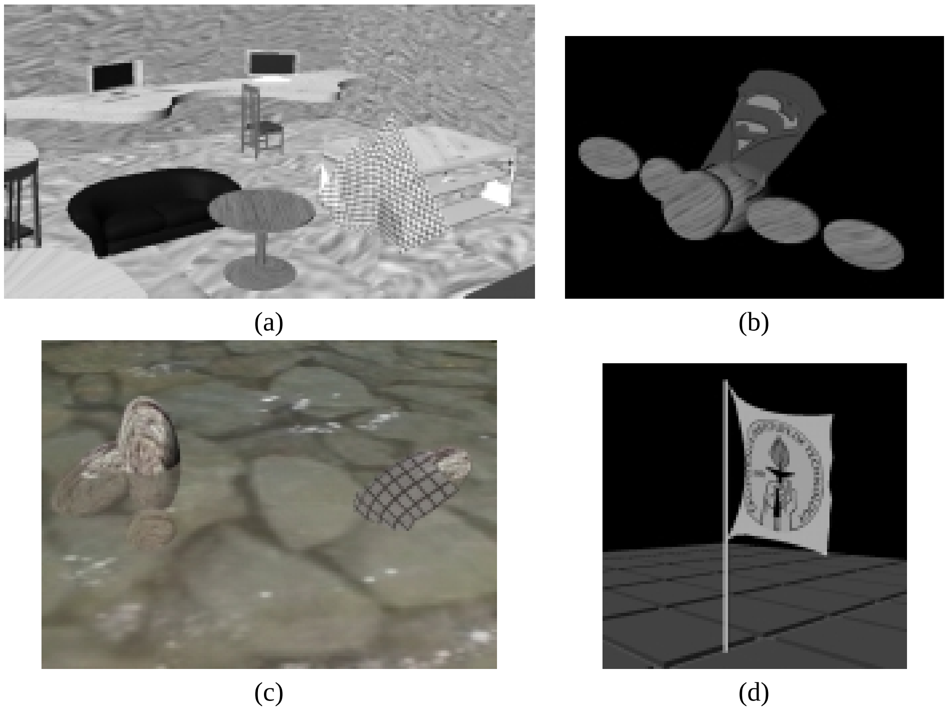

Figure 6. *(a) Complex obstacles— an office. (b) A caped crusader. (c) Interaction with a river. (d) A flag, flapping.*

7 Conclusion

In this paper, we have explored the current possibilities of real-time animation of cloth and other simple deformable objects. With our implicit-based integration scheme for stiff internal forces, mixed with a conventional explicit scheme for additional external forces, we obtain interesting results in any VR environment. Stability (and therefore, robustness) is achieved whatever the user decides to do with the simulated object(s). Collision detection and response is handled by a simple algorithm gauranteeing fast, consistent update rates for complex, dynamic environments. Multiple constraints are also handled and drag forces due to surrounding fluids are taken into account, resulting in a rich class of behaviors with a guaranteed frame rate.

As a conclusion, it seems that both hardware and algorithms are mature enough to animate nontrivial phenomena in real-time, guided by user interactions. Several avenues can now be explored, like interactive clothing design systems and virtual surgery simulators, and the constant improvement of hardware efficiency will only make things easier.

References

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

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

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

[4] G. Debunne, M. Desbrun, M.-P. Cani, and A. Barr. Adaptive simulation of soft bodies in real-time. In *Computer Animation 2000*, Annual Conference Series, May 2000.

[5] M. Desbrun, M. Meyer, P. Schröder, and A. 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] M. Desbrun, P. Schröder, and A. H. Barr. Interactive Animation of Structured Deformable Objects. In *Graphics Interface'99 proceedings*, pages 1–8, June 1999.

[7] B. Eberhardt, A. Weber, and W. Strasser. A fast, flexible, particle-system model for cloth draping. In *Computer Graphics in Textiles and Apparel*, pages 52–59. IEEE Computer Graphics and Applications, Sept. 1996.

[8] F. Faure. Interactive solid animation using linearized displacement constraints. *9^{th} Eurographics Workshop on Computer Animation and Simulation*, Sept. 1998.

[9] B. Froehlich, H. Tramberend, A. Beers, M. Agrawala, and D. Baraff. Physically-based manipulation on the responsive workbench. In *IEEE VR 2000*, March 2000.

[10] J.-D. Gascuel and M.-P. Gascuel. Displacement constraints for interactive modeling and animation of articulated structures. *The Visual Computer*, 10(4):191–204, Mar. 1994.

An early version of this paper appeared in the *Third Eurographics Workshop on Animation and Simulation*, Cambridge, UK, Sept 92.

[11] R. Grzeszczuk, D. Terzopoulos, and G. Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In M. Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 9–20. ACM SIGGRAPH, Addison Wesley, July 1998.

[12] D. James and D. Pai. Accurate real time deformable objects. In A. Rockwood, editor, *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series, pages 65–72. ACM SIGGRAPH, Addison Wesley, Aug. 1999.

[13] A. 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, Nov. 1996.

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

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

[16] 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, Sept. 1991.

[17] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. In A. Rockwood, editor, *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series, pages 401–408. ACM SIGGRAPH, Addison Wesley, Aug. 1999.

[18] M. Meyer, G. Debunne, M. Desbrun, and A. Barr. Interactive animation of cloth-like objetcs in virtual reality. In *IEEE Virtual Reality'2000 Video Proceedings*, 2000. New Brunswick, NJ.

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

[20] C. V. 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.

[21] A. Pentland and J. 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).

[22] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C, second edition*. Cambridge University Press, New York, USA, 1992.

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

[24] X. Provot. Collision and self-collision handling in cloth model dedicated to design garments. In *Graphics Interface'97 proceedings*, pages 177–189, 1997.

[25] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Computer Graphics*, 21(4):205–214, July 1987. Proceedings of SIGGRAPH'87 (Anaheim, California).

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

[27] D. Terzopoulos and A. Witkin. Physically based model with rigid and deformable components. *IEEE Computer Graphics and Applications*, pages 41–51, Dec. 1988.

[28] P. Volino, M. Courchesne, and N. Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 137–144. ACM SIGGRAPH, Addison Wesley, 1995.

[29] P. Volino and N. Magnenat-Thalmann. Developing simulation techniques for an interactive clothing system. In *VSMM 97 Proceedings*, pages 109–118. Virtual Systems and Multimedia, 1997.

[30] J. Wejchert and D. Haumann. Animation aerodynamics. In R. J. Beach, editor, *SIGGRAPH 91 Conference Proceedings*, Annual Conference Series, pages 19–22. ACM SIGGRAPH, Addison Wesley, 1991.

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

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