Volume xx (200y), Number z, pp. 1–11

Asynchronous Evolution for Fully-Implicit and Semi-Implicit Time Integration

Craig Schroeder[1], Nipun Kwatra[1], Wen Zheng[1], Ron Fedkiw[1,2]

[1]Stanford University
[2]Industrial Light+Magic

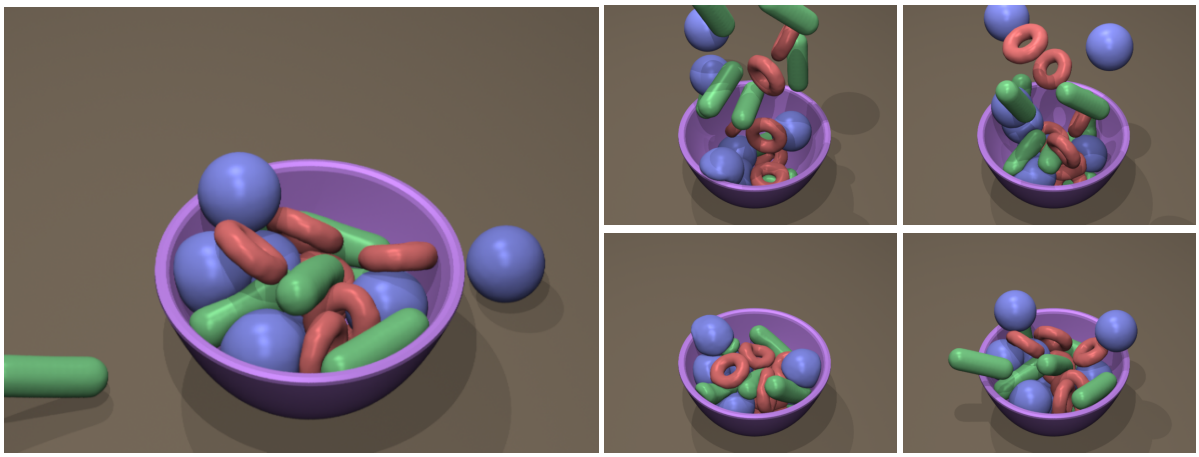


Figure 1: *Twenty-four deformable objects fall into a bowl. Each body is evolved asynchronously with* $\varepsilon = .5$.

Abstract

We propose a series of techniques for hybridizing implicit and semi-implicit time integration methods in a manner that retains much of the speed of the implicit method without sacrificing all of the higher quality vibrations one obtains with methods that handle elastic forces explicitly. We propose our scheme in the context of asynchronous methods, where different parts of the mesh are evolved at different time steps. Whereas traditional asynchronous methods evolve each element independently, we partition all of our elements into two groups: one group evolved at the frame rate using a fully implicit scheme, and another group which takes a number of substeps per frame using a scheme that is implicit on damping forces and explicit on the elastic forces. This allows for a straightforward coupling between the implicit and semi-implicit methods at frame boundaries for added stability. As has been stressed by various authors, asynchronous schemes take some of the pressure off of mesh generation, allowing time evolution to remain efficient even in the face of sliver elements. Finally, we propose a force distributing projection method which allows one to redistribute the forces felt on boundaries between implicit and semi-implicit regions of the mesh in a manner that yields improved visual fidelity.

Categories and Subject Descriptors (according to ACM CCS): I.6.8 [Simulation and Modeling]: Types of Simulation —Animation

1. Introduction

Physically based simulation of deformable bodies has a long history in computer graphics. Since its early days, physical

simulation has generally been limited by the small time step sizes required for stability and the resulting computational intensity of achieving realistic simulation. The early work of [BW98] changed this by popularizing implicit integration and the conjugate gradient method within the community. This made it possible to take much larger time steps, making cloth simulation on reasonable mesh sizes practical. With implicit integration techniques, stringent stability restrictions were largely eliminated, and time steps of arbitrary sizes could, in principle, be taken. In fact, one could envision taking time steps at the frame rate, which provides the greatest opportunity for efficiency. This approach has already received some attention, such as its recent successful application to rigid bodies [SSF09].

Taking large time steps, however, has some significant shortcomings. Large time steps tend to introduce large amounts of *artificial* damping into the simulation, and they complicate the problem of providing a robust collision response. The problem of artificial damping from large time steps could in principle be overcome with with the use of symplectic integrators [], which can limit energy loss through the preservation of invariants. In practice, however, symplectic integrators require the solution of a nonlinear system. Unfortunately, this system might not converge if large time steps are taken, thus limiting its suitability to the problem at hand.

The other complication, collision between bodies, tends to be localized within a simulation. The rest of the simulation mesh is not limited by the small time steps required to provide a robust collision response. This suggests that a technique based on asynchronous integration, where different parts of the mesh are permitted to take different time step sizes, may be a promising compromise.

Asynchronous time integration has traditionally proven to be most useful when the underlying method is fully explicit, where the size of the time step is stability limited (as opposed to accuracy limited) by both physical properties and mesh quality. Although some attention has been paid to asynchronous schemes with implicit regions in other domains [GC01,GC03], previous asynchronous time integration techniques have tended to focus on explicit schemes, since it has mainly been seen as a way of alleviating time step restrictions due to numerical stability. We refer the reader to a few standard papers on the topic [Dan03,LMOW04,FDL08, HVS*09]. The main idea of asynchronous integration is to evolve time forward, calculating forces in isolation on an element-by-element basis, where each element is evaluated at some future time based on its stability time step restriction. To make the determination of which element should be evaluated next more efficient, one typically uses a priority queue. Fully explicit methods and even semi-implicit methods, where elastic forces are treated explicitly, such as that of [BFA02, BMF03], which requires a time step restriction for elastic but not damping forces, both benefit from this sort of time evolution model, since not all elements are subject to taking time steps at the frequency of the worst element in the mesh. Larger, well-conditioned elements are allowed to take fewer time steps per frame.

When using a fully implicit method such as that of [BW98], one may take as big of a time step as visual fidelity will allow (accuracy limited), without worrying about numerical stability. Therefore, an asynchronous priority queue based on stability criteria no longer applies. However, this notion of visual fidelity does essentially bound the size of the time step. Although one might shrink the time step to increase the accuracy, in turn increasing visual fidelity, this leads to diminishing gains, because the implicit time steps are more expensive than explicit ones. Therefore, making the implicit scheme computationally efficient usually means losing some accuracy. In a typical cloth simulation, loss of accuracy would lead to a loss of folds and wrinkles, see for example [CK02] for a discussion of how buckling instabilities can be overdamped by numerical inaccuracies in implicit methods. We also refer the interested reader to [TPS08] for discussion of asynchronous integration in the context of cloth simulation. Note that the amount at which one need to reduce the time step size to get sufficient accuracy depends a lot on the possible nonlinearity of the system. In this sense, cloth usually requires smaller time steps than solid objects. Whereas the buckling instability makes things a bit more complicated, we focus on the more direct problem of the loss of high frequency motion due to large time steps. We propose using asynchronous time integration in order to alleviate this difficulty, evolving a portion of the mesh at higher frequency in order to capture elastic vibrational modes while still treating the bulk of the mesh implicitly for the sake of computational efficiency. We also note that computation efficiency of physics based simulations is of general interest to the graphics community, as can be seen by the interest that the problem of efficiency has received recently [MTS07, Dru08, WST09].

Although [Dan03] does not address implicit asynchronous time integration, they do advocate its use (as we do) for the sake of capturing high-frequency detail, even though it is not needed for stability concerns. We further stress a second and very important application of asynchronous time integration for any method, which is collisions. Large time steps can be problematic for any collision method and even worse for self-collisions as in [BFA02], whereas smaller time steps ameliorate a large number of difficulties. Thus, regardless of whether time integration is treated explicitly, fully-implicitly, or semi-implicitly (where elastic forces are treated explicitly and damping forces are treated implicitly), it can be quite important to treat elements undergoing collisions with smaller time steps.

Fully explicit schemes suffer from extremely small time step restrictions from damping forces. These can be alleviated in a straightforward and efficient manner with a semi-

	Semi-implicit	Fully-implicit	Asynchronous
Efficiency	Small time steps	Large time steps	Small time steps only where needed
Accuracy	Preserves high-frequency details	Large damping, loss of details	Preserves important details

Table 1: *Comparison of the semi-implicit method, the fully implicit method and the asynchronous method.*

implicit method which is explicit for nonlinear elastic forces and implicit for linear and damping forces. We choose the method of [SSIF09] (a small perturbation of [BMF03]) as our method of higher quality but smaller time step integration scheme. This method produces high quality results at the cost of adding computation time. We note that any other semi-implicit or implicit scheme which treats the damping forces implicitly could also be used. We use the method of [SLF08] for our fully-implicit scheme, as it is very similar to our semi-implicit scheme. As one would expect, greater numerical stability can be obtained by synchronizing the linear systems for the damping in the semi-implicit scheme with the linear system for the fully-implicit scheme. Therefore, we propose a bi-synchronous integration strategy, where the fully implicit scheme takes large time steps, while the semi-implicit scheme takes many smaller time steps which exactly equal one time step of the fully implicit scheme. To stress efficiency and also to address problems that arise more generally when transitioning to simulating with larger time step sizes, we have decided to do our numerical experimentation and to present our examples with our implicit scheme taking time steps at the frame rate. By doing this we are using the implicit method in a way that uses the least computational resources but also suffers greatly in in visual fidelity. This highlights the need for an method integrated with smaller time steps, which can capture the high frequency missing details, making our proposed line of research more challenging but also more effective. To help readers understand the contribution, a brief comparison between different methods has been shown in Table 1.

We show examples that highlight basic collisions and self-collisions, complex geometry, and poor mesh quality. For each of these, we report on the results using the semi-implicit method along with its higher computational costs but improved accuracy, the implicit method highlighting its efficiency at the sake of extreme damping, and our newly-proposed bi-synchronous integration strategy. We highlight one of the drawbacks of asynchronous integration, which is that larger elements have to wait longer for their forces to be felt, while smaller elements can push the mesh around freely while ignoring the forces from larger elements. There are ways of ameliorating this to some extent using extrapolation of forces and other techniques, and we propose a scheme that uses a projection method in the conjugate gradient solver in order to distribute the forces from the nodes being evolved at higher frequency to the nodes being evolved at lower frequency. Whereas this method does touch all particles at the frequency of the explicit or semi-implicitly evolved parts of the mesh, most particles are only touched with a simple projection, saving on force evaluations and making the method more efficient. In the limit, (which we do not use) this is a type of rigidification of the fully-implicitly evolved nodes into a kinematically deforming body - its frame is still fully two-way coupled and free to move unlike a traditional kinematic body. We use this method away from the limit where it helps to increase visual fidelity without adverse rigidification artifacts.

2. Related Work

Explicit asynchronous time integration has interested researchers for over three decades, beginning with [BM76, BYM79, Bel81], and continues to draw interest from the community (see e.g. [CH08, Dan03] and the references therein). Asynchronous integrators have also been the subject of stability and accuracy analysis (see e.g. [BS93, LMOW04, FDL08]). Asynchronous integration in the presence of collisions has also received recent attention (see e.g. [TPS08, HVS*09]).

There has also been work on explicit-implicit asynchronous evolution (see e.g. [GC01, GC03]). Unlike the context of explicit asynchronous integration, where stability considerations permit performance improvements for asynchronous evolution, the benefits sought from mixed explicit-implicit approaches are different. In the context of explicit-implicit integration, localized, highly nonlinear forces enable efficiency gains to be achieved, since heavily nonlinear forces can be quite expensive to treat implicitly.

3. Synchronous Methods

3.1. Semi-Implicit Scheme

Our semi-implicit time integration scheme is based on that of [SSIF09], which uses separate position and velocity updates to evolve bodies forward in time and takes the following basic form.

- $\mathbf{v}^{n+\frac{1}{2}} = \mathbf{v}^n + \frac{\Delta t}{2}\mathbf{a}(t^{n+\frac{1}{2}}, \mathbf{x}^n, \mathbf{v}^{n+\frac{1}{2}})$
- $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^{n+\frac{1}{2}}$
- $\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t\, \mathbf{a}(t^{n+1}, \mathbf{x}^{n+1}, \mathbf{v}^{n+1})$

The first step uses explicit evaluation of the elastic forces given the current positions $\mathbf{x}^n$ along with an implicit backward Euler update of the velocity component of the damping forces. The second step uses this temporary velocity to

evolve the positions forward in time before the final step recomputes the velocity update, this time using the new positions $\mathbf{x}^{n+1}$.

3.2. Fully-Implicit Scheme

Our fully-implicit scheme is based on that proposed in [SLF08], which is given by the scheme

- $\mathbf{v}^{n+\frac{1}{2}} = \mathbf{v}^n + \frac{\Delta t}{2}\mathbf{a}(t^{n+\frac{1}{2}}, \mathbf{x}^n + \frac{\Delta t}{2}\mathbf{v}^{n+\frac{1}{2}}, \mathbf{v}^{n+\frac{1}{2}})$
- $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^{n+\frac{1}{2}}$
- $\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t\, \mathbf{a}(t^{n+1}, \mathbf{x}^{n+1}, \mathbf{v}^{n+1})$

The only difference between this and our semi-implicit scheme is that $\mathbf{x}^n$ is replaced by $\mathbf{x}^n + \frac{\Delta t}{2}\mathbf{v}^{n+\frac{1}{2}}$, making that update fully implicit. While this scheme is appealing since it is only a small modification to the semi-implicit scheme above, it differs from that proposed in [SLF08] in that the final solution for velocity uses a backward Euler step instead of a trapezoidal rule step. We note that using trapezoidal rule for the second solution is not a good option for simulations with large time step sizes, since it tends to cause period-two oscillations. Ironically, replacing their trapezoid rule with a backward Euler solution, a modification that would normally be expected to enhance stability, makes the scheme only conditionally stable. This can be corrected by modifying the second solution to contain a predictive set of positions given by $\mathbf{x}^{n+2} = \mathbf{x}^{n+1} + \Delta t \mathbf{v}^{n+1}$, making it unconditionally stable. The precise scheme we use is

- $\mathbf{v}^{n+\frac{1}{2}} = \mathbf{v}^n + \frac{\Delta t}{2}\mathbf{a}(t^{n+\frac{1}{2}}, \mathbf{x}^n + \frac{\Delta t}{2}\mathbf{v}^{n+\frac{1}{2}}, \mathbf{v}^{n+\frac{1}{2}})$
- $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^{n+\frac{1}{2}}$
- $\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t\, \mathbf{a}(t^{n+1}, \mathbf{x}^{n+1} + \Delta t \mathbf{v}^{n+1}, \mathbf{v}^{n+1})$

4. Asynchronous Methods

We derived and experimented with many implicit asynchronous schemes, and we present the best one along with some of the more promising alternative approaches we considered along the way. We also discuss advantages, limitations, and useful properties of each approach.

We divide our forces into two sets. The first set is evolved semi-implicitly at a smaller time step using the semi-implicit evolution described above. The second set is evolved fully-implicitly at the frame rate. In each case, the last semi-implicit step of each frame is fully coupled to the fully-implicit time step for that frame. (see Figure 2). The asynchronous scheme, as well as the the semi-implicit and fully-implicit schemes it is uses, has time time steps that take the general form

- Compute $\mathbf{v}^\star$
- Use $\mathbf{v}^\star$ to update positions to $\mathbf{x}^{n+1}$
- Compute $\mathbf{v}^{n+1}$

We consider the first two steps to be the position update and the third step to be the velocity update.

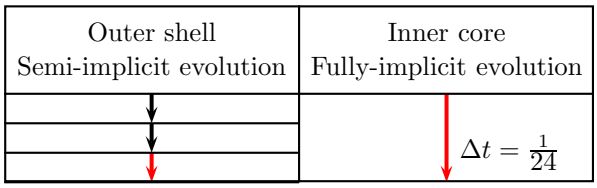


Figure 2: *Only the outer shell is updated at fine time steps. The inner core only gets updated at frame boundaries.*

4.1. Time Step and Force Scaling

Each fully-coupled step involves two backward Euler linear systems where forces are being evolved over different time step sizes. We begin the treatment of this by examining more carefully the backward Euler update for a synchronized step. Following the derivation of [BW98] for a fully implicit backward Euler update we arrive at the linear system that must be solved,

$$\left(\mathbf{I} - \Delta t \mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{v}} - \Delta t^2 \mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)\Delta \mathbf{v} = \Delta t \mathbf{M}^{-1}\left(\mathbf{f}_0 + \Delta t \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{v}_0\right). \tag{1}$$

With a coupled system of this form, it is easy to introduce additional forces to the system–these are simply added to $\mathbf{f}$ alongside the existing forces of the system. The modification that we would like to make, however, is to change the Δt over which some of the forces are evolved. Substituting $\Delta t = \alpha \Delta \hat{t}$ into (1) yields

$$\left(\mathbf{I} - \Delta \hat{t} \mathbf{M}^{-1}\frac{\partial \hat{\mathbf{f}}}{\partial \mathbf{v}} - \Delta \hat{t}^2 \mathbf{M}^{-1}\frac{\partial \hat{\mathbf{f}}}{\partial \mathbf{x}}\right)\Delta \mathbf{v} = \Delta \hat{t} \mathbf{M}^{-1}\left(\hat{\mathbf{f}}_0 + \Delta \hat{t} \frac{\partial \hat{\mathbf{f}}}{\partial \mathbf{x}}\mathbf{v}_0\right), \tag{2}$$

where $\hat{\mathbf{f}}_0 = \alpha \mathbf{f}_0$, $\frac{\partial \hat{\mathbf{f}}}{\partial \mathbf{v}} = \alpha \frac{\partial \mathbf{f}}{\partial \mathbf{v}}$, and $\frac{\partial \hat{\mathbf{f}}}{\partial \mathbf{x}} = \alpha^2 \frac{\partial \mathbf{f}}{\partial \mathbf{x}}$. In particular, an update with a different time step size can be reinterpreted as scaled force. By reinterpreting a scaled time step size as a scaled force, we can couple two forces with different time step sizes by simply adding scaled forces over the same time step size. We note that this does not itself yield a useful asynchronous time integration scheme, and a consistent means of coupling the entire time integration is still required.

It is also important to note that backward Euler depends only on positions and velocities at the *end* of the time step, where the two sets of forces will live at the same time. The velocity update already has the property that all forces involved in the backward Euler solution have their velocities living at the same time, namely the end of the current frame. This makes the velocity update relatively easy to couple, and it suffices to utilize the force scaling observation. For this reason, we concentrate on the position updates below.

4.2. Position Update and Freefall Considerations

It is important for the efficiency of a practical implicit asynchronous scheme that objects should be able to fall from rest under gravity without causing spurious internal forces. Otherwise, the spurious forces will dramatically slow down the

solver and cause servere instability. One way for spurious forces to occur is for particles to fall at slightly different rates under gravity and experience elastic restorative forces, and the second is for particles to achieve different velocities in freefall and experience internal damping. Each of these would result in a significant slow down due to extra conjugate gradient iterations during freefall. The force scaling applies the correct amount of gravity in the velocity update, so that spurious damping forces are avoided. The position update, however, requires more consideration.

We consider two approaches for avoiding positional errors in evolving gravity asynchronously. The first approach we discuss involves making sure that the position update is second order accurate, so that gravity is always evolved correctly to floating point precision. We would tend to consider an approach following these general lines to be the most desirable. In practice, we were unable to overcome the poorer conditioning that our attempts at this approach produced. Instead, we present this approach as an example of how a suitable second order asynchronous position update can be achieved. That is, this approach works, but we were unable to make it efficient enough.

The second general approach we discuss is to abandon the idea of applying gravity asynchronously. This is clearly not a satisfying route for asynchronous evolution to follow going forward. Rather, we note that applying gravity does not take a significant amount of time, and this compromise makes the resulting time integration scheme more efficient. We leave a more satisfying resolution of this problem to future work.

4.2.1. Second Order Positions

The first approach to a second order position update begins with the observation that the position update

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \left(\frac{\mathbf{v}^n + \mathbf{v}^{n+1}}{2} \right) \tag{3}$$

is second order accurate. Let $\mathbf{f}_E$ and $\mathbf{f}_I$ be forces evaluated at the time step size of semi-implicit and fully-implicit integrators, respectively. If we assume both sets of forces lack position and velocity dependence and p semi-implicit steps are taken for each fully-implicit step, then $\mathbf{v}^{n+1}$ can be written as

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \mathbf{M}^{-1} \sum_{i=1}^{p} \Delta t_i \mathbf{f}_{Ei} + \Delta t \mathbf{M}^{-1} \mathbf{f}_I. \tag{4}$$

Combining (3) and (4) we obtain

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^n + \frac{\Delta t}{2} \mathbf{M}^{-1} \sum_{i=1}^{p} \Delta t_i \mathbf{f}_{Ei} + \frac{\Delta t^2}{2} \mathbf{M}^{-1} \mathbf{f}_I. \tag{5}$$

To determine approximately how much a force that is applied only every few time steps needs to be scaled for positions to be evolved correctly, we make the assumption that $\mathbf{f}_I = 0$, $\mathbf{f}_{Ei} = \mathbf{f}_E$, and $p\Delta t_i = \Delta t$, then (see Appendix for detailed derivation)

$$\mathbf{x}^{n+1} = \mathbf{x}^{n+1-\frac{1}{p}} + \frac{\Delta t}{p} \mathbf{v}^{n-1+\frac{1}{p}} + \frac{\Delta t^2}{2p^2} \mathbf{M}^{-1} \mathbf{f}_E$$

That is, in the last (p^{th}) time step, one would apply the force $\mathbf{f}_E$ to the current $\hat{\mathbf{x}} = \mathbf{x}^{n+1-\frac{1}{p}}$ and $\hat{\mathbf{v}} = \mathbf{v}^{n-1+\frac{1}{p}}$ using the current substep size. If $\mathbf{f}_{Ei}$ is not constant or $\mathbf{f}_I \neq 0$, taking a single large step and taking many smaller steps are no longer equivalent. At this point, we make an approximation and apply the explicit forces to the evolved state $\hat{\mathbf{x}}$ and $\hat{\mathbf{v}}$. Adding back our implicit forces and dropping the assumption that the $\mathbf{f}_{Ei}$ is constant we obtain

$$\begin{aligned} \mathbf{x}^{n+1} =& \hat{\mathbf{x}} + \Delta t_p \hat{\mathbf{v}} + \frac{\Delta t_p^2}{2} \mathbf{M}^{-1} \mathbf{f}_{Ep} + \frac{\Delta t^2}{2} \mathbf{M}^{-1} \mathbf{f}_I \\ =& \hat{\mathbf{x}} + \Delta t_p \hat{\mathbf{v}} + \frac{\Delta t_p^2}{2} \mathbf{M}^{-1} (\mathbf{f}_{Ep} + \alpha^2 \mathbf{f}_I) \end{aligned} \tag{6}$$

where $\alpha = (\Delta t / \Delta t_p)^2$ is the correct scale for position update.

Note that (6) leads to a scaling factor of α^2 for the implicit elastic term rather than α as predicted by (2). That these two do not match is the primary difficulty in choosing a position update. Earlier in Section 4.1, we determined that a scaling factor of α on forces is required to apply the correct amount of force during the velocity update, but it does not give us second order accurate positions.

A scheme produced using this position update has two drawbacks. The first is that this α^2 scaling of the forces will produce erroneous velocities when used in the first velocity solution to calculate the velocities used in the position update (as pointed out in Section 4.1). These erroneous velocities will be subject to other forces such as damping and thus not yield a correct steady state. The second drawback is that the backward Euler evaluation is for the full Δt, resulting in a worse-conditioned and thus slower linear solution (we found the first linear system to require about twice the number of iterations as it would have had one used $\frac{\Delta t}{2}$ as in the semi-implicit scheme).

4.2.2. Avoiding Spurious Forces

The position update described above can be modified to avoid the mismatch in scales. In [BW98, SSIF09, SSF08], forces are recomputed after the conjugate gradients solution. In these papers, the results of the conjugate gradients solution are used to explicitly recompute the forces that were used in that solution. We utilize the same technique so that the proper scaling of α can be used inside the conjugate gradients solution while a scaling of α^2 can still be used in the subsequent position update. The conjugate gradient solver produces a solution $\mathbf{v}_\star$ to the following equation

$$\mathbf{v}_\star = \mathbf{v}^n + \Delta t \mathbf{M}^{-1} \mathbf{f}_E(\mathbf{x}^n, \mathbf{v}_\star) + \alpha \Delta t \mathbf{M}^{-1} \mathbf{f}_I(\mathbf{x}^n + \alpha \Delta t \mathbf{v}_\star, \mathbf{v}_\star). \tag{7}$$

Instead of using this velocity in the position update, we use $\mathbf{v}_\star$ to explicitly re-evaluate $\mathbf{f}_E$ and $\mathbf{f}_I$ and then scale $\mathbf{f}_I$ by α^2 to obtain the following velocity

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t \mathbf{M}^{-1} \mathbf{f}_E(\mathbf{x}^n, \mathbf{v}_\star) + \alpha^2 \Delta t \mathbf{M}^{-1} \mathbf{f}_I(\mathbf{x}^n + \alpha \Delta t \mathbf{v}_\star, \mathbf{v}_\star). \quad (8)$$

The full position update is (7), (8), and (3). This avoids the problem of spurious forces in the solution in a straightforward way. It does not address the extra computational cost of solving the linear system across the full time step.

4.3. Asynchronous Integration Scheme

We can derive a more efficient scheme based on the idea of evolving the velocity half way through the current time step as in the semi-implicit case and then using it to update positions. All positions and velocities are evolved forward in time even in the absence of forces on them so that positions can all be updated with the step size Δt. This amounts to extrapolating the behavior of the nodes that are evolved implicitly forward through the time step. In the coupled time steps that synchronize the two methods, semi-implicit forces need to be evolved for time $\frac{\Delta t}{2}$ so that the positions can be evolved in a leap-frog fashion for a time step Δt. The fully-implicit forces are evolved to the same point in time, which since they have been ignored for the entire frame means updating them for a time equal to $\Delta t_{frame} - \frac{\Delta t}{2}$. This synchronizes the velocities at a time that is $\frac{\Delta t}{2}$ before the end of the frame. This gives us the scaling factor $\alpha = \frac{2\Delta t_{frame} - \Delta t}{\Delta t}$ for use in scaling forces. The computation of the half time velocities used in the position update becomes

$$\mathbf{v}^{n+\frac{1}{2}} = \mathbf{v}^n + \frac{\Delta t}{2}(\mathbf{a}_E(\mathbf{x}^n, \mathbf{v}^{n+\frac{1}{2}}) + \alpha \mathbf{a}_I(\mathbf{x}^n + \alpha \Delta t \mathbf{v}^{n+\frac{1}{2}}, \mathbf{v}^{n+\frac{1}{2}}))$$

The main drawback of this scheme is that gravity must be applied to all particles at each step to avoid spurious elastic forces during freefall. This scheme is what we used in our examples section. The coupled time steps for this scheme are

- $\mathbf{v}^{n+\frac{1}{2}} = \mathbf{v}^n + \frac{\Delta t}{2}(\mathbf{a}_E(\mathbf{x}^n, \mathbf{v}^{n+\frac{1}{2}}) + \alpha \mathbf{a}_I(\mathbf{x}^n + \alpha \Delta t \mathbf{v}^{n+\frac{1}{2}}, \mathbf{v}^{n+\frac{1}{2}}))$
- $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^{n+\frac{1}{2}}$
- $\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t \hat{\mathbf{a}}(\mathbf{x}^{n+1} + \Delta t \mathbf{v}^{n+1}, \mathbf{v}^{n+1})$

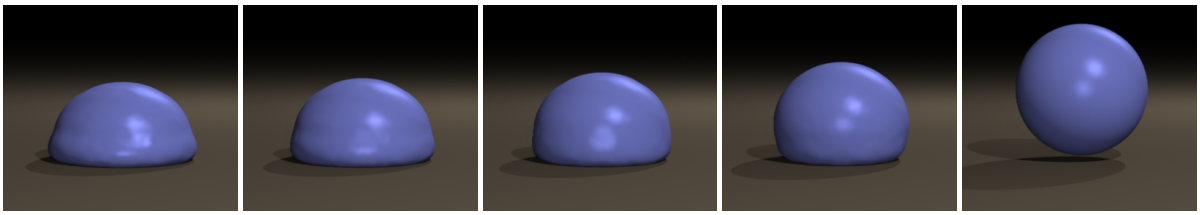
where $\hat{\mathbf{a}}$ indicates that the force scaling from Section 4.1 is applied to fully implicit forces.

5. Force Distribution

The asynchronous scheme ignores the fully-implicit forces during most of the simulation and attempts to compensate for them at frame boundaries. Consider a setup where the outer layer of tetrahedra have semi-implicit forces to resolve collisions, and the interior has fully-implicit forces for efficiency. During the semi-implicit steps, the sphere appears to be hollow, held up only by the strength of the thin outer layer of forces. Collisions cause the tetrahedra in contact with the ground to be pushed into the interior, and the top of the sphere collapses into the sphere. To overcome this limitation of asynchronous integration, we propose a novel means of distributing forces across regions of objects whose forces are not being applied in the current time step. Note that the force distribution operator only distributes the semi-implicit forces through the fully implicit regions when the fully implicit forces are not being applied. The rigid components within the semi-implicit regions are not affected. During coupled steps, the fully implicit forces are applied as usual, and the fully-implicit and semi-implicit forces interact normally. Thus, while this approach modifies the effective constitutive model, it does not override it entirely.

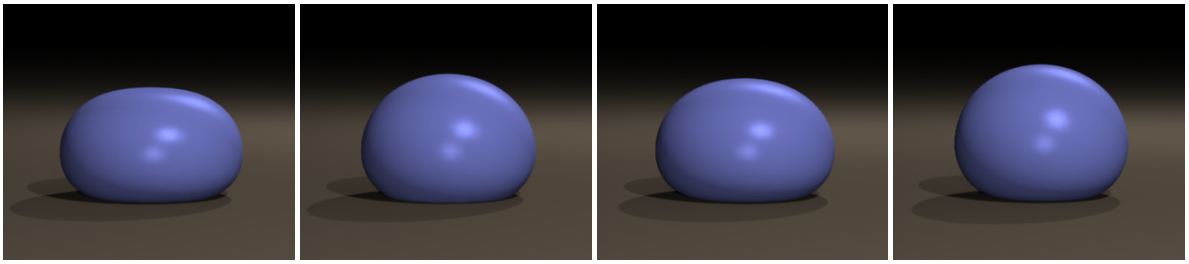
We begin by assuming that some subset of the particles are part of a rigid body with mass $\hat{m} = \sum_i m_i$ and center of mass $\mathbf{c} = \hat{m}^{-1} \sum_i m_i \mathbf{x}_i$. The particles can be described relative to the center of mass by $\mathbf{r}_i = \mathbf{x}_i - \mathbf{c}$, and the rigid body's inertia tensor can be written as $\hat{\mathbf{I}} = \sum_i m_i \mathbf{r}_i^* \mathbf{r}_i^{*T}$, where $\mathbf{r}^*$ represents the cross product matrix of $\mathbf{r}$. The total force on the rigid body is $\hat{\mathbf{f}} = \sum_i \mathbf{f}_i$, and the net torque is $\hat{\tau} = \sum_i \mathbf{r}_i^* \mathbf{f}_i$, where $\mathbf{f}_i$ represent forces in individual particles. The force and torque cause velocity changes per unit time of $\Delta \hat{\mathbf{v}} = \hat{m}^{-1} \hat{\mathbf{f}}$ and $\Delta \hat{\omega} = \hat{\mathbf{I}}^{-1} \hat{\tau}$, and the corresponding velocity changes per unit time at a particular particle is $\Delta \mathbf{v}_i = \Delta \hat{\mathbf{v}} + \mathbf{r}^{*T} \Delta \hat{\omega}$. Finally, we can compute forces that would need to have been applied at these particles in the absence of the rigid body to produce the same result, which is $\bar{\mathbf{f}}_i = m_i \Delta \mathbf{v}_i$. The operator that takes $\mathbf{f}_i$ and returns $\bar{\mathbf{f}}_i$ (obtained by combining all of these equations) is a linear operator with very nice properties. It is a rank-six mass-symmetric projection operator that preserves net force and net torque, which is required for linear and angular momentum conservation. The six degrees of freedom not projected out correspond to the three translational and three rotational degrees of freedom of the rigid body.

This operator satisfies all of the properties necessary to be used as a projection in a conjugate gradient solver. As shown in the far right image in Figure 3 where the sphere has 21,528 tetrahedra, this method constrains the particles subject to fully implicit forces to behave as a rigid body, except for the temporary drift afforded by the fully coupled steps. This provides an alternative method for simulating deformable bodies with rigid cores as proposed in [GOM*06]. However, our aim is to simulate fully deformable bodies, not those with rigid cores, and thus we modify this method as follows. The first modification is that we apply the projections to the forces only, and not the velocities, which allows the particles to retain relative motion. The second modification is that we do not fully project the forces, as that would remove all of the non-bulk force and instead only partially project. This can be accomplished by interpolating between the original forces and the projected forces using a scalar ε, where $\varepsilon = 0$ corresponds to doing nothing at all, and $\varepsilon = 1$ corresponds to using only the projected forces. Note that a more formal approach to force distribution would be to model the whole interior of the body by a single coarse element of by a small



(a) $\varepsilon = 0$ (b) $\varepsilon = .25$ (c) $\varepsilon = .5$ (d) $\varepsilon = .75$ (e) $\varepsilon = 1$

Figure 3: *An elastic sphere dropped on the ground using asynchronous time integration with varying* ε*. One obtains less deformation as* ε *increases. When* $\varepsilon = 1$ *the sphere behaves almost as a rigid body.*



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

Figure 4: *An elastic sphere dropped on the ground using (a) semi-implicit time integration, (b) fully implicit time integration, (c) fully implicit time integration with small time step sizes, and (d) accurate solution with high resolution mesh for comparison.*

number of coarse elements. The placement and constitutive properties of these elements can then be computed using any dimensionality reduction technique such as PCA. We, however, take a simpler, albeit perhaps ad hoc, approach of interpolating between a constitutive model representing a hollow inside ($\varepsilon = 0$) and a constitutive model representing a rigid core ($\varepsilon = 1$). In that sense ε can be thought of as a parameter affecting the *hardness* of the inside. We did not find it difficult to set a value for ε which results in simulations close to the actual model. Moreover, this approach is efficient to implement and provides an intuitive control over the simulation behavior. We show the effects of varying the parameter ε in Figure 3. Reference simulations with fully implicit and semi-implicit time integration are shown in Figure 4. We have found the dependence of simulation behavior on ε to be tolerable away from the boundary values, and we simply use $\varepsilon = \frac{1}{2}$. This corresponds to removing half of the non-bulk forces within the fully-implicit regions of the body. We also note that for $\varepsilon \neq 0$ and $\varepsilon \neq 1$, this does not result in a projection matrix (it lacks idempotence); nevertheless, we have encountered no problems incorporating it into the conjugate gradient solution as though it were a projection.

6. Collisions

We process collisions and contact as was done in [SSF08], but we have made an improvement to make it more suitable for use at the frame rate, i.e. 24 frames per second. In [SSF08], collision and contact fit into the time integration scheme as follows:

- Update velocities from $\mathbf{v}^n$ to $\mathbf{v}^{n+\frac{1}{2}}$
- Update positions from $\mathbf{x}^n$ to $\mathbf{x}^{n+1}$
- Process collisions, applied to $\mathbf{v}^n$, $\mathbf{v}^{n+\frac{1}{2}}$, $\mathbf{x}^{n+1}$
- Process contact, applied to $\mathbf{v}^{n+\frac{1}{2}}$, $\mathbf{x}^{n+1}$
- Process deformable-deformable collisions, applied to $\mathbf{x}^{n+1}$
- Update velocities from $\mathbf{v}^n$ to $\mathbf{v}^{n+1}$
- Process contact, applied to $\mathbf{v}^{n+1}$

The algorithm distinguishes between contact and collisions, since [SSF08] does two-way deformable rigid coupling, and separate contact and collision steps are important when simulating rigid bodies. Collisions and contacts are processed in Gauss Seidel fashion between pairs of bodies, with each particle of a deforming body treated as a rigid body with zero inverse inertia tensor (particles do not respond to torques). Within the scope of this paper, however, the contact and collision steps process particles against static rigid bodies. Moreover, since the first contact step does the same thing as the collision step it follows, it could be ignored. Deformable-deformable collisions are separated from all other types of collisions, since we use the cloth collision algorithm of [BFA02], which operates by maintaining a collision-free state and, though Gauss Seidel in nature, does

not lend itself easily to Gauss Seidel processing with other types of collisions.

The first six steps can be thought of as choosing the new positions, and the last two steps can be thought of as choosing the new velocities. Because the collision algorithm of [SSF08] is designed to function properly in the context of two-way coupling, they are not able to take the much simpler collision approach used in [SSIF09] to handle collisions with static rigid bodies. The time integration of [SSIF09] is able to directly fix both positions and velocities in a single step, and within the relatively narrow scope of this paper we could have chosen to do the same. Instead, we stay within the confines of the more general collision algorithm and address its limitations.

Collisions with rigid bodies are processed by removing the normal velocity from the particle and applying friction. The collision processing will force the normal relative velocity between the particle and static bodies to be zero, so that the relative normal separation between them will not change during the time step. This results in a gap between objects that are otherwise in contact. That is, a particle falling to the ground will stop just above the ground, separated from it by a distance of up to $\Delta t \mathbf{v}_{rel}$, where $\mathbf{v}_{rel}$ is the relative normal velocity. Since the particle is now at rest and above the ground (and thus not in contact with it), it will begin to fall under gravity until it hits the ground a second time and is stopped closer to the surface but still above it. This process allows the particle to approach the ground closer, until it is within a distance of $\frac{1}{2}g\Delta t^2$ from the ground. Within this distance, it will fall into the ground from rest in a single time step and behave as though it were in contact. With time steps as large as a frame, this initial collision gap can produce noticeable artifacts, since objects travel a noticeable distance in a single time step.

We therefore propose a modification to the way collisions are processed. The idea is to instead reduce the normal velocity so that positions will lie at the surface of the rigid body after the time step and apply friction based on this impulse. This will produce a velocity whose normal component is still approaching. That is, the particle will still appear to be colliding after the position update, but the particle's position will be on the surface as desired. After the collision step, three more steps will directly involve these modified velocities and must each be considered. First, the velocities will be processed for contact immediately after the collision step, and the modification to the collision response must be repeated here so that it is not undone. The second step that sees the modified velocity is the implicit velocity update. The third step to see the modified velocities is the final contact step, which must be handled carefully. This contact step must process the same pairs as collisions processed and should zero out relative normal velocity rather than targeting a position. After the time step has ended, the particles will lie on the collision body surface with zero relative velocity as desired. While this approach is rather subtle, it has the advantage of being applicable to the two-way coupled context of the original algorithm rather than being limited to the narrower context of this paper.

For collisions between two deformable objects, we use the collision algorithm of [BFA02]. Although this algorithm guarantees no interpenetration, it does not work well when taking time steps as large as the frame rate. Since, our adaptive asynchronous time integration takes small times steps on surface where collision occurs, it is quite useful for accurate resolution of self collisions. See Figures 5 (middle) and 6 (left middle) for examples of failed collision resolution at large time steps. Note that one can also resolve this issue by carefully designing their collision response methods like the implicit contact handling method in [**?**].

7. Examples

In all our examples the surface elements are evolved semi-implicitly, while the internal elements are evolved fully implicitly. Figures 6 shows that we are able to handle more complicated geometry, even with self collisions. Observe that the fully implicit simulation in Figure 6 has collision artifacts due to the frame rate collisions resolution, whereas the asynchronous and semi-implicit simulations lack such artifacts. We also show the extreme case $\varepsilon = 1$, which causes the armadillo to behave almost like a rigid body wrapped in a thin deformable mesh. It is still able to bend because we do not project velocities and because the coupled steps allow relative motion of the interior particles. The effects of spatial refinement can be seen in Figure 7. Timings are shown in Table 2. In addition, the semi-implicit iteration is approximately half cheaper than the fully-implicit one.

We use the meshing algorithm of [MBTF03], which generates well-conditioned elements in the interior of the mesh. The time step size in an explicit method is determined by the time required for information to travel the length of the smallest feature in the mesh at the speed of sound in the material. This restriction is relaxed significantly for semi-implicit methods and avoided entirely for fully-implicit methods. Because of the quality and adaptivity of the mesh, the smallest elements, and therefore the shortest edges, occur on the mesh's surface. Since in all of our asynchronous examples we evolve the surface elements semi-implicitly,

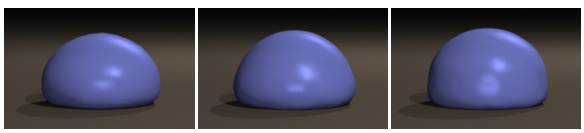


Figure 7: *One sphere falling using meshes of various resolutions: (left) 600 tetrahedra, (middle) 6,120 tetrahedra, (right) 21,528 tetrahedra.*

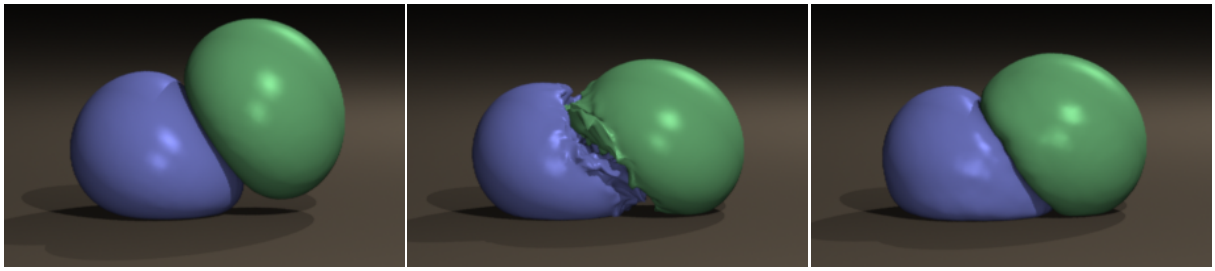


Figure 5: *Two elastic spheres colliding using (left) semi-implicit time integration, (middle) fully implicit time integration and (right) asynchronous time integration with* $\varepsilon = .5$*. Note that the fully implicit simulation has collision artifacts due to frame rate collision resolution, whereas the artifacts are significantly reduced in the asynchronous simulation and resolved entirely in the semi-implicit simulation.*

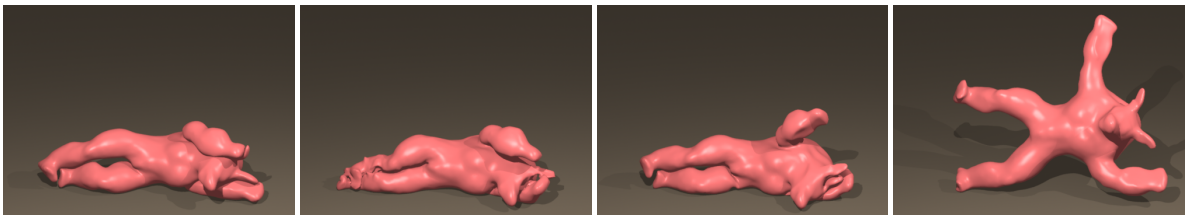


Figure 6: *An elastic armadillo falls on the ground using (left) semi-implicit time integration, (left middle) fully implicit time integration, (right middle) asynchronous time integration with* $\varepsilon = .5$ *and (right) asynchronous time integration with* $\varepsilon = 1$*. Note that the fully implicit simulation has collision artifacts (near legs and ears) due to frame rate collision resolution, whereas the asynchronous and semi-implicit simulations lack such artifacts. The extreme case of* $\varepsilon = 1$*, causes the armadillo to behave almost like a rigid body wrapped in a thin deformable mesh. It is still able to bend because we do not project velocities and because the coupled steps allow relative motion of the interior particles.*

we do not benefit much in terms of time step size during their evolution. In fact, the added cost of the worse conditioned linear solver impedes the computational efficiency of the asynchronous time integration scheme. We note however that a single bad element in the interior of the mesh can impose a severe time step restriction on the semi-implicit scheme. This can be seen in the case of Figure 8, where our asynchronous evolution yields a speed up of 8.5 times compared with the semi-implicit evolution due to the presence of a single bad spring element inside the object, whose length is 0.001 times that of the shortest length spring in the original mesh. The difference in the two results originates from difference in choices of numerical schemes. Since the fully-implicit scheme takes large time steps, we do not expect its result to match the semi-implicit scheme, which takes small time steps.

The asynchronous scheme is able to handle larger and more complicated scenes as well. Figure 1 shows a simulation with 24 deformable bodies dropped into a bowl. The bodies collide with each other, the bowl, and in the case of the tori themselves. The bodies are evolved asynchronously with the interiors fully implicit taking one step per $24Hz$ frame. The exterior is semi-implicit and takes smaller time steps. We use $\varepsilon = .5$ for all of the bodies. The simulation averaged 33 seconds per frame and 35 substeps per frame for the semi-implicit exterior. The simulation contains 94,224 degrees of freedom (134,480 tetrahedra).

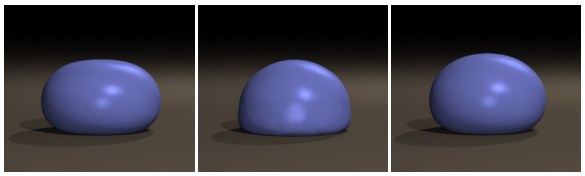


Figure 8: *An elastic sphere with a single bad element is dropped on the ground using (left) semi-implicit time integration, (middle) asynchronous time integration with* $\varepsilon = .5$*, and (right) fully-implicit with runtime matching the asynchronous scheme. We obtain an factor of 8.5 improvement in computational efficiency over the semi-implicit scheme when using the asynchronous method in this example.*

Test name	semi-implicit (seconds/frame)	fully-implicit (seconds/frame)	asynchronous (seconds/frame)				
			$\varepsilon = 0$	$\varepsilon = .25$	$\varepsilon = .5$	$\varepsilon = .75$	$\varepsilon = 1$
1 sphere	0.87	0.47	1.2	1.18	1.16	1.08	0.84
2 spheres	3.31	1.47	4.29	4.38	4.31	4.05	3.09
Armadillo	3.55	1.03	3.58	3.69	3.51	3.45	3.37
1 sphere with a single bad element	13.24	0.8	1.61	1.91	1.54	1.26	1.26

Table 2: *Wall clock times comparing the semi-implicit method, the fully implicit method and the asynchronous method.*

8. Conclusion and Future Work

We present an implicit asynchronous time integration scheme that evolves one set of forces semi-implicitly at a small time step size and another set of forces fully implicitly with a larger time step size to allow better collision resolution and visual fidelity. We use a novel force distribution method to avoid interior collapse due to asynchronous evolution. Our time integration scheme works well with collisions and self collisions, and it can achieve an order of magnitude speed improvement for meshes that contain bad elements.

Although we only consider two regions, a semi-implicit region and a fully-implicit, one is not limited to only two regions. In fact, there could be many semi-implicit regions, each one categorized to flow at some finer time step, or one might also consider using an explicit, fully asynchronous method within the semi-implicit region so that each element is applied in the most efficient fashion. Another area we have not adequately explored is different means of choosing regions for different time steps.

We note that we did not achieve the efficiency gains that we were expecting. One of the main reasons for this is that taking time steps of twice the size does not accelerate the runtime by a factor of two. Although the time steps are larger, the cost of each time step increases. This is because the matrix upon which conjugate gradient is acting has the form $\mathbf{A} = \mathbf{I} - \Delta t \mathbf{M}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{v}} - \Delta t^2 \mathbf{M}^{-1} \frac{\partial \mathbf{f}}{\partial \mathbf{x}}$. As $\Delta t \to 0$, $\mathbf{A} \to \mathbf{I}$, which greatly improves the conditioning of the problem and accelerates the rate of convergence of the conjugate gradient algorithm. In particular, the effect on conditioning becomes a very significant factor when Δt is on the order of the frame rate. This is why the fully implicit simulations, running at the frame rate, are only a small factor more efficient than the semi-implicit simulations, which run with over ten times as many time steps. This in turn severely limits the gains that asynchronous evolution is able to provide. One avenue for improving upon our asynchronous scheme is to address the conditioning problems caused by the fully implicit forces applied over the large Δt.

The force distribution scheme was necessary because the mesh tends to collapse or experience other undesirable behaviors when being evolved with the interior forces disabled. It seems likely that the asynchronous time integration scheme itself could be modified in a way that avoids this limitation without the need for adding additional forces. We leave this for future work.

While our results leave room for improvement, we feel that this is a promising avenue for future research. We also feel that many of the ideas and observations we have made along the way may be helpful to others that are interested in pursuing asynchronous integration.

Acknowledgement

Research supported in part by a Packard Foundation Fellowship, an Okawa Foundation Research Grant, ONR N0014-06-1-0393, ONR N00014-05-1-0479 for a computing cluster, NIH U54-GM072970, and NSF CCF-0541148. C.S. was supported in part by a Stanford Graduate Fellowship.

References

[Bel81] BELYTSCHKO T.: Partitioned and adaptive algorithms for explicit time integration. *Nonlinear Finite Element Anal. in Struct. Mech.* (1981), 572–584.

[BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. 21*, 3 (2002), 594–603.

[BM76] BELYTSCHKO T., MULLEN R.: Mesh partitions of explicit-implicit time integration. *Formulations and Computational Algorithms in Finite Element Analysis, K. Bathe, J. Oden and W. Wunderlich, eds., MIT Press, Cambridge, Mass* (1976), 673–690.

[BMF03] BRIDSON R., MARINO S., FEDKIW R.: Simulation of clothing with folds and wrinkles. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (2003), pp. 28–36.

[BS93] BIESIADECKI J., SKEEL R.: Dangers of multiple time step methods. *J. of Comp. Phys. 109* (1993), 318–328.

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *ACM SIGGRAPH 98* (1998), ACM Press/ACM SIGGRAPH, pp. 43–54.

[BYM79] BELYTSCHKO T., YEN H., MULLEN R.: Mixed methods for time integration. *Comp. Meth. in Appl. Mech. and Engng. 17*, 18 (1979), 259–275.

[CH08] CASADEI F., HALLEUX J.: Binary spatial partitioning of the central-difference time integration scheme for explicit fast transient dynamics. *International Journal Numerical Methods in Engineering* (2008).

[CK02] CHOI K.-J., KO H.-S.: Stable but responsive cloth. *ACM Trans. Graph. (SIGGRAPH Proc.) 21* (2002), 604–611.

[Dan03] DANIEL W.: A partial velocity approach to subcycling structural dynamics. *Comp. Meth. Appl. Mech. Engng. 192* (2003), 375–94.

[Dru08] DRUMWRIGHT E.: A fast and stable penalty method for rigid body simulation. *IEEE Trans. Viz. Comput. Graph. 14*, 1 (2008), 231–240.

[FDL08] FONG W., DARVE E., LEW A.: Stability of asynchronous variational integrators. *J. Comput. Phys. 227* (2008), 8367–94.

[GC01] GRAVOUIL A., COMBESCURE A.: Multi-time-step explicit-implicit method for non-linear structural dynamics. *Intl. J. for Numer. Meth. in Engng. 50*, 1 (2001), 199–225.

[GC03] GRAVOUIL A., COMBESCURE A.: Multi-time-step and two-scale domain decomposition method for non-linear structural dynamics. *Intl. J. for Numer. Meth. in Engng. 58*, 10 (2003), 1545–1569.

[GOM*06] GALOPPO N., OTADUY M. A., MECKLENBURG P., GROSS M., LIN M. C.: Fast simulation of deformable models in contact using dynamic deformation textures. In *Proc. of the ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (2006), pp. 73–82.

[HVS*09] HARMON D., VOUGA E., SMITH B., TAMSTORF R., GRINSPUN E.: Asynchronous contact mechanics. In *ACM SIGGRAPH 2009* (2009), ACM Press/ACM SIGGRAPH, pp. 1–12.

[LMOW04] LEW A., MARSDEN J., ORTIZ M., WEST M.: Variational time integrators. *Int. J. Num. Meth. Eng. 60* (2004), 153–212.

[MBTF03] MOLINO N., BRIDSON R., TERAN J., FEDKIW R.: A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *12th Int. Meshing Roundtable* (2003), pp. 103–114.

[MTS07] MOUSTAKAS K., TZOVARAS D., STRINTZIS M.: SQ-Map: Efficient layered collision detection and haptic rendering. *IEEE Trans. Viz. Comput. Graph. 13*, 1 (2007), 80–93.

[SLF08] SELLE A., LENTINE M., FEDKIW R.: A mass spring model for hair simulation. *ACM Transactions on Graphics 27*, 3 (Aug. 2008), 64.1–64.11.

[SSF08] SHINAR T., SCHROEDER C., FEDKIW R.: Two-way coupling of rigid and deformable bodies. In *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2008), pp. 95–103.

[SSF09] SU J., SCHROEDER C., FEDKIW R.: Energy stability and fracture for frame rate rigid body simulations. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (2009), pp. 155–164.

[SSIF09] SELLE A., SU J., IRVING G., FEDKIW R.: Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *IEEE Trans. on Vis. and Comput. Graph.* (2009), 339–350.

[TPS08] THOMASZEWSKI B., PABST S., STRASSER W.: Asynchronous cloth simulation. In *CGI Proc. 2008* (2008).

[WST09] WICKE M., STANTON M., TREUILLE A.: Modular bases for fluid dynamics. *ACM Trans. Graph. 28*, 3 (2009), 1–8.

Appendix

Derivation of Second Order Positions

Recall that the second order position update looks like (5).

If we let $\mathbf{f}_I = 0$, $\mathbf{f}_{Ei} = \mathbf{f}_E$, and $p\Delta t_i = \Delta t$, then

$$\begin{aligned}
\mathbf{v}^{n+\frac{k}{p}} =& \mathbf{v}^{n+\frac{k-1}{p}} + \frac{\Delta t}{p}\mathbf{M}^{-1}\mathbf{f}_E \\
=& \mathbf{v}^n + \frac{k\Delta t}{p}\mathbf{M}^{-1}\mathbf{f}_E \\
\mathbf{x}^{n+\frac{k}{p}} =& \mathbf{x}^{n+\frac{k-1}{p}} + \frac{\Delta t}{2p}\left(\mathbf{v}^{n+\frac{k-1}{p}} + \mathbf{v}^{n+\frac{k}{p}}\right) \\
=& \mathbf{x}^{n+\frac{k-1}{p}} + \frac{\Delta t}{p}\mathbf{v}^n + \frac{\Delta t^2}{p^2}\left(k - \frac{1}{2}\right)\mathbf{M}^{-1}\mathbf{f}_E \\
=& \mathbf{x}^n + \frac{k\Delta t}{p}\mathbf{v}^n + \frac{k^2\Delta t^2}{2p^2}\mathbf{M}^{-1}\mathbf{f}_E \\
\mathbf{x}^{n+1} =& \mathbf{x}^n + \Delta t\mathbf{v}^n + \frac{\Delta t}{2}\mathbf{M}^{-1}\sum_{i=1}^{p}\Delta t_i\mathbf{f}_{Ei} \\
=& \mathbf{x}^n + \Delta t\mathbf{v}^n + \frac{\Delta t^2}{2}\mathbf{M}^{-1}\mathbf{f}_E \\
=& \mathbf{x}^{n+1-\frac{1}{p}} + \frac{\Delta t}{p}\mathbf{v}^n + \frac{(2p-1)\Delta t^2}{2p^2}\mathbf{M}^{-1}\mathbf{f}_E \\
=& \mathbf{x}^{n+1-\frac{1}{p}} + \frac{\Delta t}{p}\mathbf{v}^{n-1+\frac{1}{p}} + \frac{\Delta t^2}{2p^2}\mathbf{M}^{-1}\mathbf{f}_E
\end{aligned}$$