Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2016)
Ladislav Kavan and Chris Wojtan (Editors)

Asynchronous Implicit Backward Euler Integration

Danyong Zhao Yijing Li Jernej Barbič

University of Southern California, USA

Abstract

In standard deformable object simulation in computer animation, all the mesh elements or vertices are timestepped synchronously, i.e., under the same timestep. Previous asynchronous methods have been largely limited to explicit integration. We demonstrate how to perform spatially-varying timesteps for the widely popular implicit backward Euler integrator. Spatially-varying timesteps are useful when the object exhibits spatially-varying material properties such as Young's modulus or mass density. In synchronous simulation, a region with a high stiffness (or low mass density) will force a small timestep for the entire mesh, at a great computational cost, or else, the motion in the stiff (or low mass density) region will be artificially damped and inaccurate. Our method can assign smaller timesteps to stiffer (or lighter) regions, which makes it possible to properly resolve (sample) the high-frequency deformable dynamics arising from the stiff (or light) materials, resulting in greater accuracy and less artificial damping. Because soft (or heavy) regions can continue using a large timestep, our method provides a significantly higher accuracy under a fixed computational budget.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Animation—Finite Element Method-Implicit Backward Euler

1. Introduction

Simulation of three-dimensional solid deformable objects has been widely used in many areas such as engineering, mechanics, robotics, computer graphics and virtual reality. Almost two decades ago, Baraff and Witkin introduced the implicit backward Euler integrator [BW98] to the graphics community. Due to its simplicity and stability under large timesteps, this integrator is very popular and widely used in computer animation. Previous implicit backward Euler methods, however, assume that the timestep employed for the simulation is constant at all the mesh elements. A constant timestep is inefficient when the mesh contains spatially-varying material parameters: one either has to choose a *spatially-global* timestep that resolves the dynamic deformations in the stiff (or low mass density) region (incurring a high computational cost), or one uses a spatially-global larger timestep, which in turn artificially damps the motion in stiffer (or low mass density) regions, effectively making them rigid. Note that *numerical stiffness* is proportional to the physical stiffness of the material (Young's modulus), and inversely proportional to the object's mass density. In this paper, we give a method where the timestep can vary spatially across the mesh (called *asynchronous integration*, or also multi-rate or subcycling integration). Our method strikes a compromise, by employing a small timestep in the stiff (or low mass density) regions and a large timestep in the soft (or high mass density) regions. It resolves deformable dynamics everywhere in the mesh, and greatly decreases artificial damping and computational cost, all the while still benefiting from stability of implicit integration. To the best of our knowledge, we are the first work in computer graphics that timesteps internal elastic forces of a three-dimensional deformable object using an asynchronous implicit method. We demonstrate our results using linear elasticity simulated using the Finite Element Method (FEM), wherein we resolve large deformations using geometric warping. We also give an extension to nonlinear simulation using co-rotational linear FEM.

Asynchronous implicit integration involves cross-timestep coupling of the vertex displacements and velocities, and is thus challenging to perform efficiently. For example, suppose we divide the vertices of a mesh into two regions, with the first region integrated at graphics rates, but the second region having a 10x smaller timestep. Then, for implicit integration, the positions and velocities at these smaller timesteps in the second region are coupled to the positions and velocities of the first region at the end of the timestep. In this paper, we demonstrate how to properly couple these positions and velocities, and solve the resulting system efficiently. We assume that the region with the smaller timestep is relatively small compared to the entire mesh, and that the interface between the slow and the fast region is small. We first demonstrate how to perform simulations with two timestep regions, where the large timestep is an integer multiple of the small timestep. We then extend our method to more than two regions, each with a separate timestep, under the assumption that the ratio between the timestep

DOI: 10.2312/sca.20161217

of the main region and the smaller timesteps is an integer, and that these smaller timestep regions do not interact.

2. Related Work

Asynchronous integration has been studied in many fields including mechanics and engineering [BYM79, Bel81, SW98], molecular dynamics simulation [KPD97], circuit design [vEvSB90], medical simulation [KPC09] and haptics [AH98, cT00]. Most methods use explicit asynchronous integration, for example, an early method of [Bel81] and more recently [Dan03, CH09]. Explicit asynchronous integration is simple and easily couples the regions with different timesteps. However, very small timesteps are often needed to guarantee the accuracy and stabilize the simulation. This limits the computational speed of explicit integration. Debunne et al. [DDCB01] accelerated the simulation by building an adaptive framework both in space and time. The explicit variational integrator in [LMOW04, KL07] allowed each element to have its own timestep. Regions with a single timestep were integrated implicitly. However, its cross-timestep coupling is still explicit. In haptics, Koçak and colleagues [KPC09] used an explicit asynchronous method to save computation costs and give a fast local contact response. An explicit asynchronous scheme is also used in [HVS*09, TPS08] to resolve collisions.

Implicit asynchronous integration is generally more complex than its explicit counterpart. Sand and Skelboe [SS92] studied the stability of multi-rate backward Euler with waveform relaxation. This method is designed for use in circuit simulation where the magnitude of one subsystem is much larger than the remaining subsystems. Thus, different timesteps can be used in different subsystems to maintain small local truncation error. They studied a method which is implicit in each individual subsystem but explicit in the coupling between subsystems, as well as implicit coupling. Their implicit coupling solves all DOFs in one large timestep by using the result from the former method as initial guess and then applying relaxation sweeps to refine the result iteratively. They discussed several interpolation schemes on the interface. However, situation in deformable object simulation is different from circuit simulation. Usually, we do not have such large magnitude differences in different regions of one object. Besides, such relaxation sweep is relatively slow, and the convergence is not guaranteed. We instead solve the system directly without iteration. Gravouil and colleagues [GC01] give a multi-timestep algorithm that allows coupling between arbitrary Newmark schemes, including explicit and implicit variations. This work is later extended to α-schemes [GCB15]. They divide elements into regions with different timesteps and use Lagrange multipliers to enforce cross-timestep coupling. For linear problems, their method has similar time complexity as ours. Our method is simpler because we use implicit backward Euler directly, without needing Lagrange multipliers. Constantinescu and colleagues [CS10] constructed a multi-rate scheme that jointly integrates the state in the large timestep region and the first step of the small timestep region. They then fix the motion in the large timestep region, and integrate the small timestep region in isolation. This method is simple and fast, but does not properly perform the cross-timestep coupling.

Since the pioneering work of [TPBF87] and [TF88] on deformable object simulation, many methods have been proposed to improve simulation efficiency. Among them, implicit backward Euler integration [BW98] was introduced to allow large timesteps. This method has the drawback of adding numerical damping when the timestep is too large [MSW14]. In computer graphics, simulations with different timesteps in different parts of the mesh have been investigated in [DDCB01, TPS08, SKZF11, SD06]. Variational integrators using both explicit and implicit methods have been proposed by [FSH11] and [SG09]. Schroeder et al. [SKZF11] partitioned the simulation mesh into a semi-implicit outer layer and implicit interior. The outer layer is integrated at a higher rate than the interior to produce less damped motion and respond timely to collision and contact. The interior is integrated at graphical frame rates using stable implicit integration. When a graphical frame begins, the outer layer is integrated semi-implicitly (explicitly in elastic force, but implicitly in damping) for a few timesteps, leaving the interior behind in time. At the end of the graphical frame, both regions are coupled in an implicit solve to get the shape of the entire object for one graphical frame. The implicit elastic forces from the interior are absent when integrating the outer layer, as if the interior is empty. The method needs to filter forces on the outer layer, adding rigid body force components to maintain a solid shape. Although the method avoids the large solve on all DOFs in one large timestep, the force filtering is not physical, decreasing accuracy, and the cross-timestep coupling is not implicit. We compare to their method in Figure 9.

Different from previous methods, we perform the coupling by performing an implicit solve on all the timesteps and both regions in one graphical frame. We include the fast region velocity changes from multiple small-timestep steps, and the velocity change of the slow region in one graphical frame, to perform implicit Euler integration across multiple small-timestep steps. In this way, the slow region forces are correctly interpolated within the graphical frame, for use in small timesteps in the fast region.

Our asynchronous method works well on linear FEM [ZTZ05]. It is because the linearization of the internal force is exact, so that during the cross-timestep coupling, all the stiffness matrices on different small-timestep steps are correct. However, linear FEM is only accurate under small rotations. Modal warping [CK05] was proposed to remedy the visible artifacts in linear modal analysis. Later, Huang et al. [HTZ*11] introduced rotation-strain coordinate warping as a better geometric post-process [LHdG*14]. We adopt this approach to improve our linear simulation results. In Section 3.4, we extend our work to nonlinear simulation using co-rotational linear FEM.

3. Algorithm

In this section, we explain our asynchronous implicit method, starting from implicit backward Euler. We first discuss a two-timestep system, followed by a generalization to more timesteps. Then we discuss an extension to co-rotational large deformation FEM simulations.

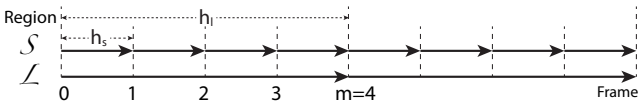

Figure 1: **Illustration of the multi-timestep scheme** used in our method, for the timestep ratio $m = 4$. The timestep h^L in region **L** is 4× larger as the timestep h^S in region **S**.

3.1. Implicit Backward Euler

We begin by describing the implicit backward Euler method. Its update rule is

$$u_{k+1} = hv_{k+1} + u_k, \tag{1}$$

$$v_{k+1} = hM^{-1} f(u_{k+1}, v_{k+1}) + v_k, \tag{2}$$

where $u_k \in \mathbb{R}^{3n}$ and $v_k \in \mathbb{R}^{3n}$ are the displacement and velocity of an object with n vertices at step k, $h = t_{k+1} - t_k$ is the timestep, $M \in \mathbb{R}^{3n\times 3n}$ is the mass matrix, and $f(u,v,t) \in \mathbb{R}^{3n}$ is the net force on all the vertices. The net force $f(u_{k+1}, v_{k+1})$ is

$$f(u_{k+1}, v_{k+1}) = f_{ext} - f_{int}(u_{k+1}) - Dv_{k+1}, \tag{3}$$

where f_{ext} is the external force applied at step $k+1$, $f_{int}(u)$ is the internal elastic force under displacements u, and $D \in \mathbb{R}^{3n\times 3n}$ is the damping matrix. We use Rayleigh damping in all of our examples. Here, we assume linear FEM simulations. Therefore, f_{int} is linear and we have

$$f_{int}(u_{k+1}) = f_{int}(u_k) + K(u_{k+1} - u_k), \tag{4}$$

where $K \in \mathbb{R}^{3n\times 3n}$ is the stiffness matrix. So Equation 2 becomes:

$$(M + hD + h^2K)\Delta v = h(f_{ext} - f_{int}(u_k) - Dv_k - hKv_k), \tag{5}$$

where $\Delta v = v_{k+1} - v_k$. To decrease computation costs, matrix K is evaluated at u_k, and Equation 5 is solved only once per timestep.

3.2. Two-Timestep System

We first explain the algorithm for implicitly integrating two regions with different timesteps. Integrators with more than two timesteps can be extended from the two-timestep version and will be discussed later.

3.2.1. System Formation

The input of the algorithm is an object whose vertices are divided into regions **L** and **S**. There will be a few elements that form an interface between **L** and **S**. The elements that have vertices both from **L** and **S** form an "interface" layer, and are the cause of interaction between the two regions. Similarly to [SKZF11], we enforce that the large timestep is an integer multiple of the small timestep,

$$h^L = mh^S, \tag{6}$$

where h^L, h^S are the timesteps used in **L**,**S**, respectively, and $m \in \mathbb{Z}^+$. In the remaining sections of the paper, we use subscripts to denote evaluation time and superscripts to denote the region. Without loss of generality, we start at time 0, assuming the displacements and velocities for both regions are known at step 0. Next, **S** is evaluated at time indices $1, 2, \ldots, m$, whereas **L** is only evaluated at time index m. We can rearrange the order of the vertices and partition Equation 2 for timestep size of h^L into

$$\begin{bmatrix} M^{SS} & M^{SL} \\ M^{LS} & M^{LL} \end{bmatrix} \begin{bmatrix} \Delta v^S_{0,m} \\ \Delta v^L_{0,m} \end{bmatrix} = h^L \begin{bmatrix} f^S_m \\ f^L_m \end{bmatrix}. \tag{7}$$

Here, $\Delta v^S_{0,m}$ and $\Delta v^L_{0,m}$ are the velocity changes on **S** and **L**, respectively, from step 0 to step m. Similarly, f^S_m and f^L_m are the forces on **S** and **L**, respectively, evaluated at step m. The mass matrix M is partitioned into four sub-matrices M^{ij}, and we have $M^{SL} = (M^{LS})^T$. Note that if mass lumping is used, M^{SL} and M^{LS} will be empty, but this does not significantly improve the efficiency because the stiffness blocks are still present in the system solve.

The equation for f can also be partitioned in the same manner, resulting in

$$\begin{bmatrix} f^S_m \\ f^L_m \end{bmatrix} = \begin{bmatrix} f^S_{ext,m} \\ f^L_{ext,m} \end{bmatrix} - \begin{bmatrix} D^{SS} & D^{SL} \\ D^{LS} & D^{LL} \end{bmatrix} \begin{bmatrix} v^S_m \\ v^L_m \end{bmatrix} - \begin{bmatrix} f^S_{int,m} \\ f^L_{int,m} \end{bmatrix}, \tag{8}$$

and similarly for the linearization of f_{int}:

$$\begin{bmatrix} f^S_{int,m} \\ f^L_{int,m} \end{bmatrix} = \begin{bmatrix} f^S_{int,0} \\ f^L_{int,0} \end{bmatrix} + \begin{bmatrix} K^{SS} & K^{SL} \\ K^{LS} & K^{LL} \end{bmatrix} \begin{bmatrix} u^S_m - u^S_0 \\ u^L_m - u^L_0 \end{bmatrix}. \tag{9}$$

If we look at the DOFs in **L** alone in Equations 7, 8 and 9, their motion at step m is determined by

$$M^{LL}\Delta v^L_{0,m} + M^{LS}\Delta v^s_{0,m} = h^L f^L_m, \tag{10}$$

$$f^L_m = f^L_{ext,m} - \begin{bmatrix} D^{LS} & D^{LL} \end{bmatrix} \begin{bmatrix} v^S_m \\ v^L_m \end{bmatrix} - f^L_{int,0} - \begin{bmatrix} K^{LS} & K^{LL} \end{bmatrix} \begin{bmatrix} u^S_m - u^S_0 \\ u^L_m - u^L_0 \end{bmatrix}. \tag{11}$$

Equations 7, 8 and 9 do not have any asynchronous mechanism yet. Now we will discuss how to substep **S** asynchronously. We integrate the motion of **S** at a smaller timestep than **L**. The motion at each step $r+1$ on **S**, $0 \le r < m$, is solved by

$$M^{SS}\Delta v^S_{r,r+1} + M^{SL}\Delta v^L_{r,r+1} = h^S f^S_{r+1}, \tag{12}$$

$$f^S_{r+1} = f^S_{ext,r+1} - \begin{bmatrix} D^{SS} & D^{SL} \end{bmatrix} \begin{bmatrix} v^S_{r+1} \\ v^L_{r+1} \end{bmatrix} - f^S_{int,r} - \begin{bmatrix} K^{SS} & K^{SL} \end{bmatrix} \begin{bmatrix} u^S_{r+1} - u^S_r \\ u^L_{r+1} - u^L_r \end{bmatrix}. \tag{13}$$

Region **L** is never evaluated at intermediate steps r, for $0 < r < m$. To acquire a correct interpolation for u^L_r and v^L_r, we have to use u^L_m and v^L_m. Assuming the acceleration of **L** is constant between steps 0 and m, we have

$$v^L_r = \frac{m-r}{m} v^L_0 + \frac{r}{m} v^L_m. \tag{14}$$

From the equations above, we find that v^S_1 depends on v^L_1, which depends on v^L_m. We also find that v^L_m depends on v^S_m, and v^S_m obviously depends on v^S_1. This cross-timestep dependency is challenging to handle in nonlinear simulations because there is no way to predict how internal force and tangential stiffness matrix will change across multiple steps. However, by using a linear force model, the

unknowns can be solved together as follows:

$$M^{SS}\Delta v^S_{0,1} + M^{SL}\Delta v^L_{0,1} = h^S f^S_1, \tag{15}$$
$$M^{SS}\Delta v^S_{1,2} + M^{SL}\Delta v^L_{1,2} = h^S f^S_2, \tag{16}$$
$$M^{SS}\Delta v^S_{2,3} + M^{SL}\Delta v^L_{2,3} = h^S f^S_3, \tag{17}$$
$$\dots$$
$$M^{SS}\Delta v^S_{m-1,m} + M^{SL}\Delta v^L_{m-1,m} = h^S f^S_m, \tag{18}$$
$$M^{LL}\Delta v^L_{0,m} + M^{LS}\Delta v^S_{0,m} = h^L f^L_m. \tag{19}$$

For $0 \le r < m$, we can replace $\Delta v^S_{r,r+1}$ with

$$\Delta v^S_{r,r+1} = \Delta v^S_{0,r+1} - \Delta v^S_{0,r}. \tag{20}$$

According to the implicit backward Euler update rule, we have

$$u^S_{r+1} = u^S_r + h^S v^S_{r+1} = u^S_0 + h^S \sum_{i=0}^{r} v^S_{i+1}, \text{ for } 0 \le r < m, \tag{21}$$
$$u^L_m = u^L_0 + h^L v^L_m. \tag{22}$$

If we substitute Equations 20, 21 and 22 into Equations 15 to 19, we can arrange Equations 15 to 19 into a matrix form:

$$\begin{bmatrix} A^S & & & & & & & B_1 \\ C & A^S & & & & 0 & & B_2 \\ G & C & A^S & & & & & B_3 \\ G & G & C & A^S & & & & B_4 \\ & & & & \dots & & & \dots \\ G & G & G & G & \dots & A^S & & B_{m-1} \\ G & G & G & G & \dots & C & A^S & B_m \\ E & E & E & E & \dots & E & F & A^L \end{bmatrix} \begin{bmatrix} \Delta v^S_{0,1} \\ \Delta v^S_{0,2} \\ \Delta v^S_{0,3} \\ \Delta v^S_{0,4} \\ \dots \\ \Delta v^S_{0,m-1} \\ \Delta v^S_{0,m} \\ \Delta v^L_{0,m} \end{bmatrix} = \begin{bmatrix} b^S_1 \\ b^S_2 \\ b^S_3 \\ b^S_4 \\ \dots \\ b^S_{m-1} \\ b^S_m \\ b^L_m \end{bmatrix}, \tag{23}$$

where

$$A^S = M^{SS} + h^s D^{SS} + (h^S)^2 K^{SS}, \tag{24}$$
$$A^L = M^{LL} + h^L D^{LL} + (h^L)^2 K^{LL}, \tag{25}$$
$$B_r = \frac{1}{m}(M^{SL} + r h^s D^{SL} + r^2 (h^S)^2 K^{SL}), \tag{26}$$
$$C = (h^S)^2 K^{SS} - M^{SS}, \quad G = (h^S)^2 K^{SS}, \tag{27}$$
$$E = \frac{1}{m}(h^L)^2 K^{LS}, \quad F = M^{LS} + h^L D^{LS} + E. \tag{28}$$

On the right hand side, we have

$$b^S_r = h^S(f^S_{ext,r} - f^S_{int,0} - (\begin{bmatrix} D^{SS} & D^{SL} \end{bmatrix} + r h^S \begin{bmatrix} K^{SS} & K^{SL} \end{bmatrix}) \begin{bmatrix} v^S_0 \\ v^L_0 \end{bmatrix}), \tag{29}$$

$$b^L_m = h^L(f^L_{ext,m} - f^L_{int,0} - (\begin{bmatrix} D^{LS} & D^{LL} \end{bmatrix} + h^L \begin{bmatrix} K^{LS} & K^{LL} \end{bmatrix}) \begin{bmatrix} v^S_0 \\ v^L_0 \end{bmatrix}). \tag{30}$$

3.2.2. System Solve

The linear system in Equation 23 has size $3m|\mathbf{S}| + 3|\mathbf{L}|$. Theoretically, one could factor the entire system before simulation since we are using linear FEM. However, a large m would greatly increase the system size. The complexity to solve using this pre-factored unsymmetric matrix directly is $O((m|\mathbf{S}| + |\mathbf{L}|)^2)$. To solve the entire system efficiently, we exploit its regular structure. Our algorithm (described below) has running time $O(m|\mathbf{S}|^2 + |\mathbf{L}|^2)$, i.e., a reduction from m^2 to m. Our experiments confirmed that the solver time increases about quadratically with the naive method, whereas our method is much faster (Figure 3).

We partition Equation 23 as

$$\begin{bmatrix} H^{SS} & H^{SL} \\ H^{LS} & H^{LL} \end{bmatrix} \begin{bmatrix} \bar{\Delta} v^S_m \\ \Delta v^L_{0,m} \end{bmatrix} = \begin{bmatrix} \bar{b}^S_m \\ b^L_m \end{bmatrix}. \tag{31}$$

Here, $H^{LL} = A^L$, $H^{SS} \in \mathbb{R}^{3m|\mathbf{S}| \times 3m|\mathbf{S}|}$ is a lower triangular block matrix consisting of A^S, C and G, and

$$\bar{\Delta} v^S_m = \begin{bmatrix} (\Delta v^S_{0,1})^T & (\Delta v^S_{0,2})^T & \dots & (\Delta v^S_{0,m})^T \end{bmatrix}^T, \tag{32}$$
$$\bar{b}^S_m = \begin{bmatrix} (b^S_1)^T & (b^S_2)^T & \dots & (b^S_m)^T \end{bmatrix}^T. \tag{33}$$

We compute Schur's complement of H^{ss}, leading to

$$H^{LL*} \Delta v^L_{0,m} = b^{L*}_m, \tag{34}$$

where

$$H^{LL*} = H^{LL} - H^{LS}(H^{SS})^{-1} H^{SL}, \tag{35}$$
$$b^{L*}_m = b^L_m - H^{LS}(H^{SS})^{-1} \bar{b}^S_m, \tag{36}$$

and $\bar{\Delta} v^S_m$ can be solved via

$$\bar{\Delta} v^S_m = (H^{SS})^{-1}(\bar{b}^S_m - H^{SL} \Delta v^L_{0,m}). \tag{37}$$

Matrix H^{LL*} can be factored once and reused at every timestep. We note that H^{LL*} has more non-zero entries than H^{LL}. The number of non-zero entries added is dependent on the interface complexity between $\mathbf{S}$ and $\mathbf{L}$. This can be proved by noting that H^{LS} and H^{SL} have non-zero entries only in the rows and columns corresponding to the interface DOFs. If the two regions are well partitioned, the number of interface DOFs is small compared to the entire mesh, and H^{LL*} is still sparse.

Now the problem reduces to an efficient solve with the matrix H^{SS}, needed to form b^{L*}_m and compute $\bar{\Delta} v^S_m$ at runtime. By exploiting its regular structure, we have designed a forward substitution procedure (Algorithm 1), consisting of solving a $3|\mathbf{S}| \times 3|\mathbf{S}|$ linear system m times. Due to linearity, these small system matrices are kept the same in every step. We factor them only once using Cholesky decomposition. In our examples, we use PARDISO [SG04] to perform all the pre-factorization and the runtime linear system solves.

Although H^{LL*} is structurally symmetric and generally unsymmetric, it is in practice almost symmetric. The unsymmetric part is added through the additional non-zero entries, whose number is often small compared to the total number of DOFs in the object. If precision is not strictly required, one can pretend that H^{LL*} is symmetric, and use a symmetric solver to obtain potentially even faster computational speeds. We have compared the quality and speed of a symmetric solver to a structurally symmetric unsymmetric solver. We have found that there is almost no difference in the results (Figure 2), and the two solvers are very similar in speed (Figure 3).

Algorithm 1 solve $H^{SS}\bar{x}_m^S = \bar{b}_m^S$

procedure COMPUTE_SMALL_REGION($A^S, C, G, \bar{b}_m^S$)
  $[(b_1^S)^T\ (b_2^S)^T\ \ldots\ (b_m^S)^T]^T \leftarrow \bar{b}_m^S$
  $x_1^S \leftarrow (A^S)^{-1} b_1^S$
  $prev \leftarrow x_1^S$ ▷ The value of x_S in the previous substep
  $sum \leftarrow x_1^S$ ▷ The sum of x_S in all previous substeps
  for $i \leftarrow 2$ to m **do**
    $x_i^S \leftarrow (A^S)^{-1}(b_i^S - G \cdot sum - (C-G) \cdot prev)$
    $prev \leftarrow x_i^S$
    $sum \leftarrow sum + x_i^S$
  end for
  $\bar{x}_m^S \leftarrow [(x_1^S)^T\ (x_2^S)^T\ \ldots\ (x_m^S)^T]^T$
end procedure

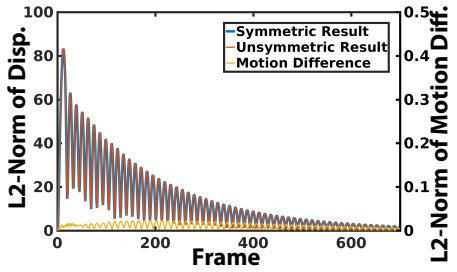

Figure 2: **The motion difference between the symmetric and unsymmetric solver**, giving the L^2-norm of object displacements computed using the symmetric solver (blue line) and the structurally symmetric unsymmetric solver (red line). The L^2-norm of motion difference (yellow line) for the two results is also shown. Because the motion difference is too small to use the same scale as the other two curves, we add another (right) vertical axis for the motion difference curve. The other two curves use the left vertical axis. The comparison is done using our Asian dragon example.

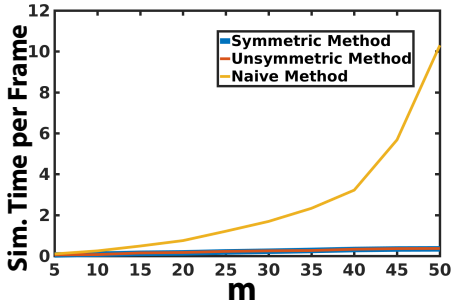

Figure 3: **Comparison of the speed** of a symmetric solver of our method, a structurally symmetric unsymmetric solver of our method, and the naive method mentioned in Section 3.2.2. The naive method is the slowest, while the the other two are almost the same in speed. The comparison is done using our Asian dragon example.

3.3. More Timesteps

Our two-timestep algorithm can be easily extended to more regions if the following conditions are satisfied:

- The vertices are divided into $n+1$ regions: $\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_n$ and $\mathbf{L}$.
- $\mathbf{S}_i$ and $\mathbf{S}_j$ are not neighbors (no in-between interface) for any i and j.
- The timestep h^L for $\mathbf{L}$ is a multiple of the timestep h^{S_i} for any $\mathbf{S}_i$ region, $h^L = m_i h^{S_i}, m_i \in \mathbb{Z}^+, 1 \le i \le n$.

Under these conditions, we can condense the DOFs in all $\mathbf{S}_i$ and solve for $\mathbf{L}$. Since there is no direct interaction among $\mathbf{S}_i$, we can condense each $\mathbf{S}_i$ into $\mathbf{L}$ one by one. The full system matrix is

$$\begin{bmatrix} H^{S_1S_1} & & & & H^{1L} \\ & H^{S_2S_2} & & & H^{2L} \\ & & \ldots & & \ldots \\ & & & H^{S_nS_n} & H^{nL} \\ H^{LS_1} & H^{LS_2} & \ldots & H^{LS_n} & H^{LL} \end{bmatrix} \begin{bmatrix} \bar{\Delta v}_{m_1}^{S_1} \\ \bar{\Delta v}_{m_2}^{S_2} \\ \ldots \\ \bar{\Delta v}_{m_n}^{S_n} \\ \Delta v_{0,m}^L \end{bmatrix} = \begin{bmatrix} \bar{b}_{m_1}^{S_1} \\ \bar{b}_{m_2}^{S_2} \\ \ldots \\ \bar{b}_{m_n}^{S_n} \\ b_m^L \end{bmatrix}, \quad (38)$$

where $H^{S_iS_i}, H^{S_iL}, H^{LS_i}, \bar{\Delta v}_{m_i}^{S_i}, \bar{b}_{m_i}^{S_i}$ are as $H^{SS}, H^{SL}, H^{LS}, \bar{\Delta v}_m^S, \bar{b}_m^S$ in the two-timestep algorithm, respectively, for $\mathbf{S}_i$. We have $\Delta v_{0,m}^L = v_m^L - v_0^L$, and $v_m^L = v_{m_i}^L$, for any i. The condensed system is

$$H^{LL*} \Delta v_{0,m}^L = b_m^{L*}, \quad (39)$$

where

$$H^{LL*} = H^{LL} - \sum_{i=1}^{n} H^{LS_i} (H^{S_iS_i})^{-1} H^{S_iL}, \quad (40)$$

$$b_m^{L*} = b_m^L - \sum_{i=1}^{n} H^{LS_i} (H^{S_iS_i})^{-1} \bar{b}_{m_i}^{S_i}, \quad (41)$$

$$\bar{\Delta v}_{m_i}^{S_i} = (H^{S_iS_i})^{-1} (\bar{b}_{m_i}^{S_i} - H^{S_iL} \Delta v_{0,m}^L). \quad (42)$$

It is difficult to design a more general multi-timestep algorithm. The block matrix in Equation 38 can be denser if there exist i, j such that $\mathbf{S}_i$ and $\mathbf{S}_j$ are neighbors. We leave this for future work.

3.4. Nonlinear Asynchronous System

With nonlinear two-timestep dynamics, f_{int} is no longer linear. We use the Taylor-series expansion of Equation 4 at u_k,

$$f_{int}(u_{k+1}) = f_{int}(u_k) + K(u_k)(u_{k+1} - u_k), \quad (43)$$

where $K(u)$ is the tangential stiffness matrix at u. Generally, $K(u)$ is not constant with u, and therefore the K^{ij} components in each row of the block matrix in Equation 23 are no longer the same. To reduce computational cost, we assume that the internal force varies nonlinearly across graphics frames, but is kept linear during the steps 0 to m. Therefore, we only need to evaluate $K(u)$ at steps $0, m, 2m, \ldots$. Under such a construction, our asynchronous method updates its nonlinear internal forces and stiffness matrices at every graphical frame. Another difference with the linear algorithm is that we have to update and factor the system matrix H^{LL*} in Equation 34, each time a large timestep is to be solved. According to Equation 35, we must solve $(H^{SS})^{-1} H^{SL}$ to get H^{LL*}. Non-zero columns in H^{SL} correspond to the DOFs on the interface between the two regions. So we only need to solve the H^{SS} system, once per each interface DOF, to get $(H^{SS})^{-1} H^{SL}$. These non-zero columns

can be solved in parallel. Our complete nonlinear two-timestep algorithm is summarized in Algorithm 2. It is straightforward to extend it to a nonlinear algorithm with more timesteps.

Algorithm 2 nonlinear two-timestep integration

procedure DO_ASYNCHRONOUS_TIMESTEP(M, f_{int}, K, u_0, v_0)
  Build $H^{LL}, H^{LS}, H^{SL}, H^{SS}$
  Compute $(H^{SS})^{-1}H^{SL}$ in parallel ▷ factor A^S first for efficient solve
  Compute H^{LL*} using Equation 35
  Compute b_m^{LL*} using Equation 36
  Solve $\Delta v_{0,m}^L$ from $H^{LL*}\Delta v_{0,m}^L = b_m^{L*}$
  Solve $\bar{\Delta} v_m^S$ using Equation 37
  Get u_m and v_m from $\Delta v_{0,m}^L$ and $\bar{\Delta} v_m^S$
end procedure

4. Examples

We have tested our method on several examples. We use the linear FEM and corotational linear FEM [MG04] implementation from the Vega simulation library [SSB13]. All linear systems are solved using PARDISO [SG04]. All examples were computed on a 3.33GHz Intel Core i7 CPU X 980 processor with 23.5Gb memory. We only allocate four threads when running all our experiments. Parallelization is achieved with internal force computation, matrix multiplication and threading in PARDISO. All timings are listed in Table 1. We compared our linear asynchronous method with standard single-rate backward Euler on the dinosaur, flower and Asian dragon examples (Figures 4, 5, 6). In the comparisons, we use a timestep for the single-rate backward Euler so that the two methods have the same computation cost in one graphic frame. In the dragon example, we compare the effect of different sizes of $\mathbf{S}$, and test our nonlinear asynchronous method. Finally, we make a comparison between our method and Schroeder's method [SKZF11].

We add geometric warping [HTZ*11] to remove linear artifacts on all linear examples. The computation cost for warping is always less than integration. The flower is the most difficult case for warping since it has many flat and narrow components. Even in the flower example, the warping cost for one frame is about 60% of the single-rate simulation cost on one step. On the other models, it is 40% typically. If multiple simulation steps are required per graphical frame and collisions are not an issue, the relative warping cost will be much lower because warping is only needed for graphical frames.

4.1. Dinosaur

We first demonstrate our method on the dinosaur example (Figure 4). We use three regions, each with a separate timestep. We put the head and neck into $\mathbf{S}_1$ and the tail into $\mathbf{S}_2$. The rest is in $\mathbf{L}$. We use linear FEM and warping. Our method produces high-frequency motions on the neck and the tail. The single-rate method cannot achieve the same quality given the same computation time. Our method only costs triple the computation time of a single-rate simulation with a timestep of h^L, but timesteps the second and third region at a timestep that is 8× and 15× smaller than the first

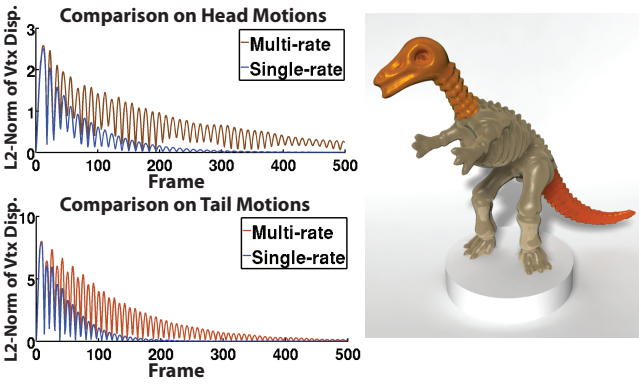

Figure 4: **Asynchronous integration of the dinosaur.** The region $\mathbf{S}_1$ (brown) includes the head and the neck. The region $\mathbf{S}_2$ (red) includes the tail. We plot the L^2-norm of a vertex displacement on the head (left) and the L^2-norm of a vertex displacement on the tail (right). We pull both the head and the tail at frame 0. For comparison, we also show the result of a single-rate backward Euler method with the same computation cost. Our method generates less damped motions on $\mathbf{S}_1$ and $\mathbf{S}_2$, at the same computation cost.

region, respectively, producing much less artificial damping and higher quality motion. One could increase the quality of a single-rate method by decreasing its timestep; however, this increases the cost of the single-rate method as it now needs to perform more timesteps per graphics frame. Actually, our experiments show that when the single-rate method's timestep is decreased so that it produces comparable quality as our method, the computational cost of the single-rate method is 3× higher than our method.

4.2. Flower

We also produce the flower example (Figure 5) where the petals are put into $\mathbf{S}_1$. We apply an external force on the stem. To improve the motion of the leaves that are close to the location where the force is applied, we also put the upper three leaves and a part of the stem they are connected to, into $\mathbf{S}_2$. The rest is in $\mathbf{L}$. We use linear FEM and warping. We achieve better motion than the single-rate method at equal computation cost (see Figure 5).

4.3. Asian Dragon

Our third example is the Asian dragon (Figure 6) where the horns and the tail are in $\mathbf{S}$. We pull the model on the head and the tail at the beginning of the simulation. We use linear FEM and warping. We obtain a better motion than the single-rate method with the same computational cost. For example, our result exhibits larger and longer vibrations of the horns and the tail.

4.4. Dragon

We test how the size of $\mathbf{S}$ affects simulation on the dragon example (Figure 7). We pull the lower jaw of the dragon and compare different $\mathbf{S}$ sizes. Linear FEM and warping is used here. The initial $\mathbf{S}$ only

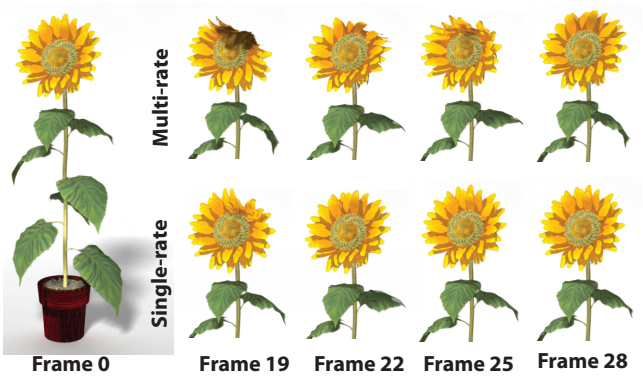

Figure 5: **Asynchronous integration of the flower.** The region $\mathbf{S}_1$ includes the petals. The region $\mathbf{S}_2$ includes the upper three leaves and the nearby parts of the stem. The flower is pulled at the beginning of the simulation. Our method produces larger displacement (frame 19) after the force is released and continues to give larger motion (frame 22, 25) than the single-rate method with the same computation cost.

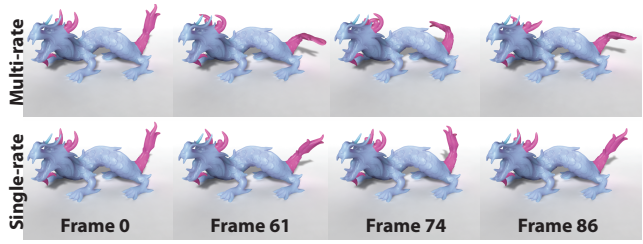

Figure 6: **Asynchronous integration of the Asian dragon.** The region **S** is colored red in the figure, including the horns and the tail. We pull the head and the tail at the beginning of the simulation. We get better vibrations than the single-rate method with the same computation cost.

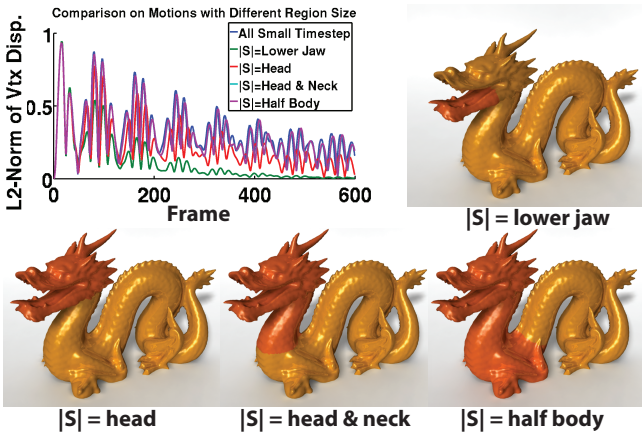

Figure 7: **Effect of different S region sizes on the motion.** We first select the lower jaw of the dragon as the smallest **S** region, and compute the motion. We then enlarge the region and compute the motion, adding in turn the head, the neck, and finally half of the body. Top-left: the graph showing the L^2-norm of a vertex displacement on the lower jaw for each **S** at every frame. Other images: the dragon models with different **S** being pulled and tested.

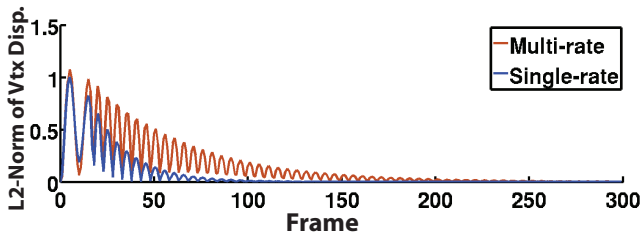

Figure 8: **Nonlinear implicit asynchronous simulation:** comparison between single-rate and our multi-rate method. Dragon example. Our method exhibits significantly less artificial damping. Animation is shown in the supplementary video.

includes the lower jaw. In the subsequent tests, **S** includes the entire head, then also the neck, and finally the front half of the dragon. The result shows that increasing $|\mathbf{S}|$ makes the motion closer to the result of a single-rate backward Euler integration with the smaller timestep used on **S**, at the cost of longer simulation time. We also note that when **S** is almost the same as the entire object, our linear asynchronous method is only a little slower than the single-rate h^S-timestep backward Euler. Therefore, the computation cost of our method will in most cases be less than the single-rate h^S-timestep ground truth, unless the user puts too many vertices into **S**.

We have also created a nonlinear example of the dragon, using co-rotational linear FEM. It can be seen that our method produces richer motion than a single-rate result with the same computation cost (Figure 8). The lower jaw is pulled. We use $m = 10$ in our method, and four simulation steps per graphical frame in single-rate integration. We use corotational liner FEM for nonlinear simulation. We have also tested our nonlinear algorithm on the other models. However, we did not observe a performance advantage because the number of interface DOFs is too large. The dragon has 96 DOFs on the interface.

We have also compared our method with Schroeder's method (dragon; Figure 9). We set the timestep ratio m to 150 for both methods. We choose such a large value of m because Schroeder's method is semi-implicit (explicit on internal forces) on **S**, and as such requires small timesteps to explicitly integrate stiff materials. We put the lower jaw into **S** and pull it down. In Schroeder's method, the coupling between the small timestep region and large timestep region is too weak. This is caused by the fact that for most timesteps, the small timestep region is integrated with no interaction from the large timestep region. In comparison, our method generates plausible and stable coupling between the two regions.

We also tested an example that involves collisions, using both linear and non-linear FEM. We compared our method to the single-rate method with the same computation cost. We made the dragon's lower jaw stiffer than the rest of the dragon, and dropped the dragon to the ground, using penalty forces to handle contacts. For both

	all #vtx	#tets	$\|\mathbf{S}\|$	h^L	m	sim. /g. fr.	warp. /g. fr.	single h^L/g. fr.	single h^S/g. fr.
dinosaur	5,413	20,150	916 / 824	0.01	8 / 15	0.066 sec	0.014 sec	0.023 sec	0.18 / 0.34 sec
flower	2,980	9,382	855 / 1,236	0.02	12 / 10	0.056 sec	0.011 sec	0.019 sec	0.21 / 0.18 sec
Asian dragon	9,961	34,441	1,146	0.03	10	0.058 sec	0.025 sec	0.046 sec	0.46 sec
dragon 1	2,717	8,252	156	0.03	10	0.0079 sec	0.0056 sec	0.0078 sec	0.078 sec
dragon 2	2,717	8,252	574	0.03	10	0.016 sec	0.0057 sec	0.0078 sec	0.078 sec
dragon 3	2,717	8,252	814	0.03	10	0.022 sec	0.0056 sec	0.0078 sec	0.078 sec
dragon 4	2,717	8,252	1,403	0.03	10	0.029 sec	0.0056 sec	0.0078 sec	0.078 sec
dragon 5	2,717	8,252	1,904	0.03	10	0.053 sec	0.0056 sec	0.0078 sec	0.078 sec
dragon 6	2,717	8,252	2,671	0.03	10	0.085 sec	0.0056 sec	0.0078 sec	0.078 sec

Table 1: **Performance for all the examples.** From left to right: model name, number of all vertices, number of all tetrahedra, number of vertices in region $\mathbf{S}$, h^L, timestep ratio m, multi-rate simulation time per graphical frame (g. fr.), warping time per graphical frame, h^L-timestep single-rate simulation time per graphical frame, and h^S-timestep single-rate simulation time per graphical frame (which includes m simulation steps). In the dinosaur example, two numbers are reported in $|\mathbf{S}|$ since it uses three timestep regions. The tail region has 916 vertices and time ratio of 8; the head and neck region has 824 vertices and time ratio of 15. In the flower example, the petal region has 855 vertices and time ratio of 12; the region including upper three leaves has 1236 vertices and time ratio of 10. Six dragon timings are reported. Each one corresponds to a different size of $\mathbf{S}$.

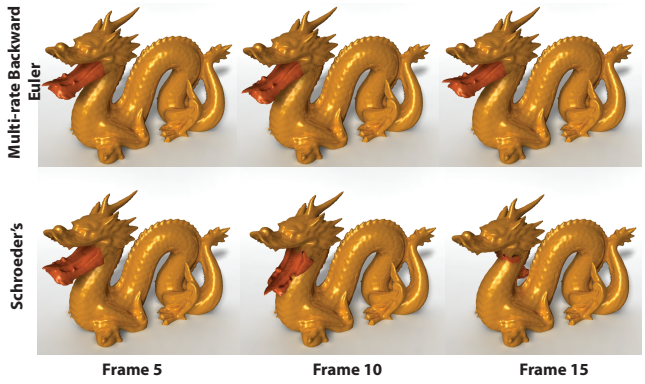

Figure 9: **Comparison to Schroeder's method** reveals that the interaction between the regions in Schroeder's method is weak, which results in much softer behavior on the interface of the two regions. First row: our method; Second row: Schroeder's method. Collision is turned off. Linear FEM and warping are used.

linear and non-linear FEM, it can be seen that when the dragon hits the ground, our method produces richer motion of the lower jaw than the single-rate method with the same computation cost. We observed that the visual difference to single-rate method is larger with linear FEM. The motions can be seen in our video.

5. Conclusion

We have designed an asynchronous implicit backward Euler for linear and nonlinear FEM to reduce artificial damping in stiff/light regions while decreasing computation cost. We believe that our method can provide artists with more control on the trade-off between accuracy and efficiency. We assume a small interface between asynchronous regions to decrease the system solve time, a common assumption for asynchronous and mixed integrators. We use linear interpolation for velocities (Equation 14) at the interface. Other interpolation schemes could be explored. We update motions only at the boundary of the largest timestep. Therefore, collision can only be checked and handled at this rate. Regions are chosen before simulation by users; we do not vary them at runtime. An automatic and adaptive grouping method would be helpful. We hope that our work will inspire more investigation on asynchronous and mixed integrators in computer graphics.

Acknowledgements

This research was sponsored in part by the National Science Foundation (CAREER-1055035, IIS-1422869), the Sloan Foundation, the Okawa Foundation, and USC Annenberg Graduate Fellowships to Danyong Zhao and Yijing Li.

References

[AH98] ASTLEY O., HAYWARD V.: Multirate Haptic Simulation Achieved by Coupling Finite Element Meshes Through Norton Equivalents. In *Proc. of the IEEE Int. Conf. on Robotics and Automation* (1998). 2

[Bel81] BELYTSCHKO T.: Partitioned and adaptive algorithms for explicit time integration. In *Nonlinear Finite Element Analysis in Structural Mechanics*. Springer, 1981, pp. 572–584. 2

[BW98] BARAFF D., WITKIN A. P.: Large Steps in Cloth Simulation. In *Proc. of ACM SIGGRAPH 98* (July 1998), pp. 43–54. 1, 2

[BYM79] BELYTSCHKO T., YEN H.-J., MULLEN R.: Mixed methods for time integration. *Computer Methods in Applied Mechanics and Engineering 17* (1979), 259–275. 2

[CH09] CASADEI F., HALLEUX J.-P.: Binary spatial partitioning of the central-difference time integration scheme for explicit fast transient dynamics. *International Journal for Numerical Methods in Engineering 78*, 12 (2009), 1436–1473. 2

[CK05] CHOI M. G., KO H.-S.: Modal Warping: Real-Time Simulation of Large Rotational Deformation and Manipulation. *IEEE Trans. on Vis. and Comp. Graphics 11*, 1 (2005), 91–101. 2

[CS10] CONSTANTINESCU E. M., SANDU A.: On extrapolated multirate methods. In *Progress in Industrial Mathematics at ECMI 2008*. Springer, 2010, pp. 341–347. 2

[cT00] ÇAVUŞOĞLU M. C., TENDICK F.: Multirate Simulation for High Fidelity Haptic Interaction with Deformable Objects in Virtual Environments. In *Proc. of the IEEE Int. Conf. on Robotics and Automation* (San Francisco, USA, 2000). 2

[Dan03] DANIEL W.: A partial velocity approach to subcycling structural dynamics. *Computer Methods in Applied Mechanics and Engineering 192*, 3 (2003), 375–394. 2

[DDCB01] DEBUNNE G., DESBRUN M., CANI M.-P., BARR A. H.: Dynamic Real-Time Deformations Using Space & Time Adaptive Sampling. In *Proc. of ACM SIGGRAPH 2001* (August 2001), pp. 31–36. 2

[FSH11] FIERZ B., SPILLMANN J., HARDERS M.: Element-wise mixed implicit-explicit integration for stable dynamic simulation of deformable objects. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2011), ACM, pp. 257–266. 2

[GC01] GRAVOUIL A., COMBESCURE A.: Multi-time-step explicit–implicit method for non-linear structural dynamics. *International Journal for Numerical Methods in Engineering 50*, 1 (2001), 199–225. 2

[GCB15] GRAVOUIL A., COMBESCURE A., BRUN M.: Heterogeneous asynchronous time integrators for computational structural dynamics. *International Journal for Numerical Methods in Engineering 102*, 3-4 (2015), 202–232. 2

[HTZ*11] HUANG J., TONG Y., ZHOU K., BAO H., DESBRUN M.: Interactive shape interpolation through controllable dynamic deformation. *IEEE Trans. on Visualization and Computer Graphics 17*, 7 (2011), 983–992. 2, 6

[HVS*09] HARMON D., VOUGA E., SMITH B., TAMSTORF R., GRINSPUN E.: Asynchronous Contact Mechanics. *ACM Transactions on Graphics 28*, 3 (2009), 87:1–87:12. 2

[KL07] KALE K. G., LEW A. J.: Parallel asynchronous variational integrators. *International Journal for Numerical Methods in Engineering 70*, 3 (2007), 291–321. 2

[KPC09] KOÇAK U., PALMERIUS K. L., COOPER M.: Dynamic deformation using adaptable, linked asynchronous fem regions. In *Proceedings of the 25th Spring Conference on Computer Graphics* (2009), ACM, pp. 197–204. 2

[KPD97] KOPF A., PAUL W., DÜNWEG B.: Multiple time step integrators and momentum conservation. *Computer physics communications 101*, 1 (1997), 1–8. 2

[LHdG*14] LI S., HUANG J., DE GOES F., JIN X., BAO H., DESBRUN M.: Space-time editing of elastic motion through material optimization and reduction. *ACM Trans. on Graphics (SIGGRAPH 2014) 33*, 4 (2014), 108:1–108:10. 2

[LMOW04] LEW A., MARSDEN J., ORTIZ M., WEST M.: Variational time integrators. *International Journal for Numerical Methods in Engineering 60*, 1 (2004), 153–212. 2

[MG04] MÜLLER M., GROSS M.: Interactive Virtual Materials. In *Proc. of Graphics Interface 2004* (2004), pp. 239–246. 6

[MSW14] MICHELS D. L., SOBOTTKA G. A., WEBER A. G.: Exponential integrators for stiff elastodynamic problems. *ACM Transactions on Graphics (TOG) 33*, 1 (2014), 7. 2

[SD06] STERN A., DESBRUN M.: Discrete geometric mechanics for variational time integrators. In *ACM SIGGRAPH 2006 Courses* (2006), ACM, pp. 75–80. 2

[SG04] SCHENK O., GÄRTNER K.: Solving unsymmetric sparse systems of linear equations with pardiso. *Future Generation Computer Systems 20*, 3 (2004), 475–487. 4, 6

[SG09] STERN A., GRINSPUN E.: Implicit-explicit variational integration of highly oscillatory problems. *Multiscale Modeling & Simulation 7*, 4 (2009), 1779–1794. 2

[SKZF11] SCHROEDER C. A., KWATRA N., ZHENG W., FEDKIW R.: Asynchronous evolution for fully-implicit and semi-implicit time integration. *Comput. Graph. Forum 30*, 7 (2011), 1983–1992. 2, 3, 6

[SS92] SAND J., SKELBOE S.: Stability of backward euler multirate methods and convergence of waveform relaxation. *BIT Numerical Mathematics 32*, 2 (1992), 350–366. 2

[SSB13] SIN F. S., SCHROEDER D., BARBIČ J.: Vega: Non-linear fem deformable object simulator. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 36–48. 6

[SW98] SMOLINSKI P., WU Y.-S.: Stability of explicit subcycling time integration with linear interpolation for first-order finite element semidiscretizations. *Computer methods in applied mechanics and engineering 151*, 3 (1998), 311–324. 2

[TF88] TERZOPOULOS D., FLEISCHER K.: Deformable models. *The Visual Computer 4* (1988), 306–331. 2

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically Deformable Models. *Computer Graphics (Proc. of ACM SIGGRAPH 87)* (July 1987), 205–214. 2

[TPS08] THOMASZEWSKI B., PABST S., STRASSER W.: Asynchronous cloth simulation. In *Computer Graphics International* (2008). 2

[vEvSB90] VAN EIJNDHOVEN J. T., VAN STIPHOUT M., BUURMAN H.: Multirate integration in a direct simulation method. In *Proceedings of the conference on European design automation* (1990), IEEE Computer Society Press, pp. 306–311. 2

[ZTZ05] ZIENKIEWICZ O., TAYLOR R., ZHU J.: The finite element method: its basis and fundamentals. 2005, 2005. 2