CGI 2008 Conference Proceedings

Bernhard Thomaszewski · Simon Pabst · Wolfgang Straßer
WSI/GRIS, Universität Tübingen, Germany

Asynchronous Cloth Simulation

Abstract This paper presents a new method for cloth simulation, which uses asynchronous variants of both time integration and collision handling. Implicit integration methods like backward Euler and BDF-2 are very popular in computer graphics, since they allow for fast and stable animations. However, when combined with large time steps, their inherent numerical dissipation results in over-damped simulations, which lack high frequency details such as small folds and wrinkles.

In this paper, we present a computationally efficient method which does not suffer from these restrictions. The time integration component uses an asynchronous variational integrator (AVI), which allows dedicated time steps for every element. Thanks to its energy preserving nature, low-damped cloth materials can be simulated without compromising dynamic motion or suppressing important details. Our collision handling scheme combines both synchronous and asynchronous strategies and, in this way, allows focusing computation power on the important regions where collisions actually occur. We provide timings for several integration methods and show that our AVI-based scheme performs consistently better than synchronous explicit variants. Compared to implicit schemes, superior quality is obtained while remaining comparable in terms of computation times. Finally, we demonstrate the robustness of our method on a series of challenging animations.

Keywords Asynchronous Time Integration · Cloth Simulation · Collision Handling · Explicit Integrators

1 Introduction

The physically-based simulation of deformable objects has been an active research area in computer graphics for more than two decades. Since the work of Baraff et al. [BW98], A-stable (or unconditionally stable) implicit methods like backward Euler and BDF-2 have become predominant in computer graphics, since they allow fast computations when combined with large time steps [HE01]. However, this combination is known to suffer from significant numerical dissipation [CK02, OAW04, VMT05]. This manifests as over-damped simulations in which the formation of high frequency details such as small folds and wrinkles is suppressed (see Fig. 2, left). As a promising alternative, variational integrators like the symplectic Euler, the Verlet scheme or the implicit midpoint scheme exhibit excellent energy conservation properties [HLW06]. However, these methods are not widely used for computer animation because of their inherent stability limitations: the time step has to be small enough to match the stability requirements of the stiffest component of the system. This is especially unfortunate when using unstructured meshes, where a few small elements (e.g. in regions of high curvature) can drastically limit the global time step and thus computational efficiency. This situation gets even worse for collision handling, which in many implementations is performed once every time step. Although there is some room for optimisation, it is in general not advisable to carry out collision handling at a significantly lower frequency than

Fig. 1 Snapshots of an obstacle-walk animation.

WSI/GRIS, Universität Tübingen
Tel.: +49-7071-2970425
Web: www.gris.uni-tuebingen.de/∼thomasze
eMail: {thomaszewski,pabst,strasser}@gris.uni-tuebingen.de

Fig. 2 Views of a representative frame from a $4s$ simulation of a low-damped textile (5016 faces). Left to right: backward Euler (Δt 0.033s), backward Euler (Δt 0.00075s), implicit Midpoint Rule (Δt 0.001s), symplectic Euler (Δt 0.0005s), AVI (Δt_e 0.00037s). Notice the lack of detailed folds when using backward Euler with a large time step (left). For the computation time, the visual quality obtained with our method (right) is better than for backward Euler (2nd from left).

time integration, since explicit methods can, unlike implicit schemes, usually not recover from large strains or strain rates due to abrupt collision response. In summary, the computational costs associated with standard explicit cloth simulation make this approach less attractive in practice.

In this paper, we describe how an explicit and yet computationally efficient method can be designed using both asynchronous time stepping and asynchronous collision handling. The time integration component uses an explicit variant of asynchronous variational integrators [LMOW04]. Because of its symplecticity and associated energy conservation properties, low-damped cloth materials can be simulated without compromising their dynamic character (see Fig. 1) or suppressing important details such as small folds and wrinkles (see Fig. 2, left). Furthermore, the AVI scheme automatically distributes computation power by assigning a dedicated elemental time step to every element according to local stability requirements. This is especially advantageous when materials of different stiffness are used or when the mesh has differently sized elements. In order to further improve the stability and performance of the method we use element-based strain and strain rate limiting. This enables us to use softer materials and thus larger time steps without affecting the realism of the simulation.

Our collision handling scheme overcomes the above mentioned problems using a three-stage strategy, which combines both synchronous and asynchronous techniques. A global (synchronous) detection pass determines potentially colliding elements at equidistant instants in time, defined by a global detection step size. The second and computationally most intensive phase is tightly coupled to the element updates during asynchronous time integration. Here, fast geometric proximity tests are applied to the potentially colliding triangle pairs obtained in the first phase and impulse-based responses are generated when necessary. Finally, the third, global pass ensures that no collisions are missed during a global detection step. We further exploit the flexibility of AVIs to selectively reduce elemental times steps in critical situations, when sudden character motion or high-velocity collisions threaten to destabilise the animation.

An advantage of this approach is that, due to the comparably small elemental time steps, the resolution at which collision handling is carried out is much higher than for implicit integrators with large time steps. Still, the computational burden is significantly lower as when using standard explicit integration with collision handling after every global time step. As a result, stable and intersection-free simulations are obtained even for complex scenarios with multi-layer self-collisions and low inter-layer distance. Since this is achieved without excessive damping, the vivid character and lively motion of cloth is preserved (see Fig. 1). Before we start the detailed exposition of our method we briefly comment on previous and related research.

1.1 Related Work

In the early work of Terzopolous et al. [TPBF87] implicit integration was used for simulating the dynamics of elastically deformable objects. Because this was commonly believed to be too expensive, explicit integration methods like the Symplectic Euler (or Euler-Cromer) [VCMT95] or the 4th order Runge-Kutta scheme [EWS96] were preferred subsequently. In 1998 Baraff et al. [BW98] introduced a semi-implicit integration scheme for solving the linearised equations of motion. Its superior stability allowed using large time steps, which leads to fast computations. Subsequent work identified poor accuracy and unrealistically high damping as a shortcoming of semi-implicit schemes and suggested solving the full nonlinear equations [HE01], using higher order implicit methods such as BDF-2 [CK02], or extracting the damping of rotation-modes explicitly [OAW04]. Recent work also considered the Newmark scheme [BMF03, GHDS03], the implicit midpoint rule [VMT05] and other variational integrators for computer animation [KYT+06]. A different approach to variational time integration has been presented by Lew et al. [LMOW04], who derived a fully asynchronous scheme based on discrete Lagrangian mechanics.

Collision handling is a well studied field in computer graphics field and literature is abundant. We therefore refer the reader to the overview compiled by Teschner et al. [THM+05]. For cloth simulation, bounding volume hierarchies (BVH) based on e.g. k-Dops [MKE03]

are commonly used to accelerate the proximity detection. The collision response which prevents intersections is usually based on constraints, forces or, as in our case, impulses (see [BFA02]).

There are only a few works concerned with asynchronous physically-based simulation in computer graphics. Celes [Cel98] describes an approach for simulating multi-body systems in which different bodies move with different time-steps. The method described in [DGC04] is based on a similar idea but also accounts for simple inter-object collisions. To our knowledge, the method presented in this work is the first to consider the asynchronous treatment of elemental collisions for highly deformable objects.

The rest of this paper is organised as follows. The next section is a brief description of explicit AVI scheme by Lew et al. [LMOW04], which we included for the reader's convenience. The subsequent section presents our mechanical model including AVI-specific extensions. Our approach to asynchronous collision handling is detailed in section 4 and the paper concludes with Sec. 5, in which the performance of this approach is evaluated.

2 Asynchronous Time Stepping Framework

The most distinguishing aspect of AVIs is that they allow each element to have its dedicated time step and do not impose restrictions with respect to neighbouring elements. Unlike in conventional synchronous methods, with each element having a potentially different time step, the nodes of the mechanical system evolve asynchronously in time as can be seen in Fig. 3. It is therefore necessary to keep track of both elemental and nodal times. The figure also shows a further aspect of the AVI

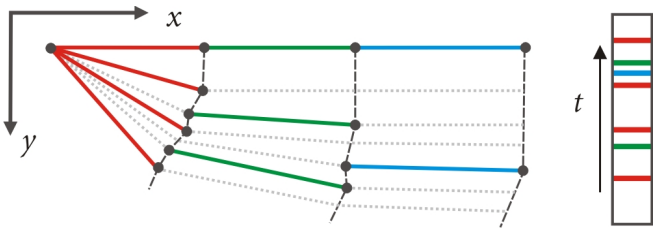

Fig. 3 A 1D-chain containing 3 elements with different step sizes is fixed at one point and subjected to gravity. *Left*: the number of times an element is updated depends on its step size. *Right*: element activations over time are indicated by their associated colours.

scheme: the time dimension is discretised with *events*, each corresponding to the activation of an element. An element becomes *active* whenever the continuous flow of time reaches a point at which, according to its local step size, the element has to be updated. Obviously, for maintaining causality, the order of elemental updates is of paramount importance. A convenient and efficient data structure to enforce this causality algorithmically is a priority queue. We provide a brief version of the update algorithm described in the original work [LMOW04], also indicating the extensions we made:

Algorithm 1 AVI Outline

```
1:  //Initialization:
2:  for i = 1 to n_elem do
3:     compute elemental time step Δt_{e,i}
4:     push element (i; Δt_{e,i}) into queue
5:  end for
6:  //Main loop
7:  while queue is not empty do
8:     (i; t_{e,i}) ⇐ queue.pop() //get top element from queue
9:     for j = 1 to 3 do
10:       x_{i,j} = x_{i,j} + v_{i,j}(t_{e,i} − t_{i,j}) //Update positions
11:       t_{i,j} = t_{e,i} //Update nodal times
12:       checkAndHandleCollisions()
13:       v_{i,j} = v_{i,j} − Δt_{e,i} (∂V_{e,i}/∂x_{i,j}) / m_{i,j} //Update velocities
14:       limitStrainAndStrainRate()
15:    end for
16:    if (t_{e,i} + Δt_{e,i}) ≤ t_end then
17:       queue.push(i, (t_{e,i} + Δt_{e,i}))
18:    end if
19: end while
```

Here, the subscript (e, i) refers to quantities related to element i and (i, j) indicates a variable related to its jth node. Moreover, $\Delta t_{e,i}$ is the elemental time step and $m_{(i,j)}$ are corresponding nodal masses. In the initialization stage (ll.2-5) a time step is computed for each element according to the Courant criterion (see Sec. 3) and the elements are inserted into the queue. During the main simulation loop the element which has to be updated next is retrieved from the top of the priority queue (l.8). The three nodal positions are then updated using the current velocities (l.10) and nodal times are set to the current element time (l.11). Subsequently, the energy stored during the previous time step is released by applying impulses to the element thus changing its nodal velocities (l.13). Finally, if the end of the simulation is not yet reached, the element is rescheduled for evaluation (l.16) using the elemental time step (which, in fact, does not need to stay constant over time). In order to create geometry output, a snapshot of the system is generated in a regular manner. This is done by extrapolating the nodal positions to the global time using the current velocities.

3 Mechanical Model and Extensions for AVI

Our simulator for deformable objects is built upon nonlinear continuum mechanics [BW97] with a Total Lagrangian formulation of linear Finite Elements for discretisation [Bat96]. Deformations of thin structures can generally be decomposed into membrane and bending components and we organise the following description accordingly.

Membrane Model The in-plane behaviour of our simulator is based on a nonlinear variant of the constant strain triangle (CST) [Bat96]. For a deformed CST element, the elastic forces (i.e. the gradients of strain energy with respect to nodal positions) needed in l.12 of Alg. 1 are obtained as follows. We first compute the deformation gradient $\mathbf{F}$ as

$$\mathbf{F}_{ij} = \mathbf{\Sigma}_k dN_{k,j} x_{k,i} \tag{1}$$

where $x_{k,i}$ are current nodal positions and $dN_{k,j}$ are the partial derivatives of the element's three linear shape functions with respect to reference coordinates. The nonlinear Green strain then follows as $E = \frac{1}{2}(F^T F - I)$, with I denoting the identity. From this we obtain the second Piola-Kirchhoff stress tensor as $S = \mathcal{C} : E$, where $\mathcal{C}$ is the elasticity tensor describing the material. Using a simple isotropic St. Venant-Kirchhoff material [BW97], we have $S = \lambda(E_{11} + E_{22})I + 2\mu E$, where λ and μ are the Lamé constants. S (and also E) is a symmetric 2×2 tensor, which is stored as a 3-vector $\mathbf{S}$ for computational convenience. The nodal elastic forces are finally obtained as

$$\mathbf{f}_k = \frac{\partial V_e}{\partial x_k} = \mathbf{F}\mathbf{B}_k^T \mathbf{S} A_0 \ , \tag{2}$$

where A_0 is the area of the undeformed triangle and $\mathbf{B}_k$ is the 2×3 strain-displacement matrix whose entries consist of the shape function derivatives for node k. The advantage of this model is that all involved quantities are easy to evaluate and, due to the non-linear strain measure, no rotations need to be explicitly extracted.

Although the explicit variational integrator excels at long term energy conservation, one will most likely want to add a small amount of dissipation for practical animations. In this case the Lagrange D'Alembert Principle has to be used, which extends the Lagrangian theory to dissipative and forced systems (see [KMOW00]). We implement energy dissipation as *viscous* stress, which in our case is a linear function of the strain rate. Since we use only low damping, the resulting viscous forces are smaller than the elastic forces and stability is not affected.

Bending Model There are several alternatives to integrate bending resistance into the AVI framework. The most accurate way is to augment the elastic potential in Eq. (2) with a thin-shell energy. In this way, both membrane and bending forces could be evaluated at the same time using a single element type. However, thin shell solutions from engineering are notoriously complex and lead to higher computational costs. As a more efficient alternative, discrete bending energies as used in computer graphics like [GHDS03], [BMF03] or [BWH$^+$06] seem attractive. For our implementation we chose to model bending directly as an external force and follow the formulation of Bridson et al., which is particularly simple. Here, a bending element is defined by two adjacent triangles (forming a hinge). Evaluating the bending forces acting on the nodes of a given triangle requires the evaluation of all hinge elements which share at least one node

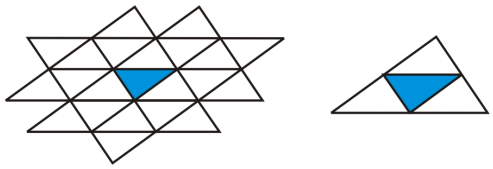

Fig. 4 Connectivity for bend force computation. *Left*: 24 hinge elements have influence on the membrane triangle (centre), defining a 22-triangle neighbourhood. *Right*: the simplified force computation considers only immediate neighbours.

with the triangle. Since elements in this neighbourhood generally have different times, their positions have to be synchronised to the current element time, first. As can be seen in Fig. 4, the bending connectivity of a triangle with regular neighbourhood extents over 22 triangles and 21 nodes. Synchronizing and evaluating forces over this large neighbourhood is an expensive process. We therefore choose a simplified bending force computation which only considers the three hinge elements corresponding to the edges of the membrane triangle. This approach is in a way similar to the bending element described e.g. in [FO01], which can be shown to capture all curvature directions. As a result, we obtain a simple and accurate way to implement bending forces at low computational costs, which integrates seamlessly with the membrane force computation.

It should be noted that we implicitly assumed that bending forces never become the limiting factor for stability, which is true for cloth, but not for general thin shell materials with high bending stiffness. In this case, separate hinge elements with their own dedicated time step should be employed.

Time Step Selection and Stability For explicit integration schemes, stability is only guaranteed when certain conditions on the time step are met. The critical time step beyond which stability is threatened can be determined using the Courant-Friedrich-Lewy (CFL) condition (see e.g. [Bat96]). For the specific case of linear-elastic materials, it is common to use an estimate of the form

$$\Delta t_{cr,i} = \alpha \ h_i \sqrt{\rho/(\lambda + 2\mu)} \ , \tag{3}$$

where h_i is related to the size of element i (its internal radius) and α is a scalar (usually $0 < \alpha < 1$). The Lamé constants λ and μ describe an isotropic material and ρ denotes the mass density. This condition requires that the elemental time step should be less than the time it takes a material wave to pass the element. Obviously, a less stiff material allows taking larger time steps. Textiles usually exhibit a highly nonlinear stress-strain relationship, which can be roughly characterised as biphasic: the stretch resistance for small deformations is only weak but becomes very stiff for large strains. Hence, using a soft material for the small deformation range is a reasonable approach if it is possible to reliably prevent large deformations, which are unphysical anyway. A computationally efficient way to do so was presented

by Provot [Pro95] who used geometric elongation correction, or *strain limiting*. This approach has proven efficient for computer animations and was adopted e.g. in [BFA02], who extended it to strain rate limiting. A different approach was recently proposed by Goldenthal et al. [GHF+07] who used implicit constraints to enforce inextensiblity.

Algorithm 1 offers a very simple way to integrate strain and strain rate limiting (see l.14). We first define two thresholds ε_c and ε_n, which, depending on the amount of deformation, indicate when to active strain (e.g. $\varepsilon_{c,lim} = 10\%$) and strain rate (e.g. $\varepsilon_{n,lim} = 7.5\%$) limiting. Having computed new positions and updated velocities for an element, we compute the current strain ε_c, the strain rate $\dot{\varepsilon}$, and the predicted next strain $\varepsilon_n = \varepsilon_c + \Delta t_e \dot{\varepsilon}$ for the edges of the element. We apply strain rate limiting impulses whenever $\varepsilon_n > \varepsilon_{n,lim}$. In the rare cases when $\varepsilon_c > \varepsilon_{c,lim}$ we additionally correct positions geometrically.

For synchronous integrators deformation limiting is usually implemented as multiple correction passes over the edges of the mesh. As pointed out e.g. in [BFA02], a problem arising in this context is the biasing caused by the fixed ordering of the edges. Although randomization can help alleviate this concern, it may also have a negative effect on convergence. Especially for unstructured meshes, bias is very unlikely to occur in the asynchronous implementation, since there is no static element ordering. Due to the relatively small elemental time steps, a single correction step after each element update is sufficient to enforce e.g. a strain limit of 10% in the first test scene, showing a low average strain of only 1.2% (see Fig. 2). In practice, this method allows taking larger average time steps and, depending on the actual scene, significant accelerations (5 times and higher) can be achieved. It should, however, be noted that this technique cannot entirely replace the physical membrane model since it dissipates energy from the system and degrades the accuracy of the simulation. It rather complements the physical model and helps to maintain stability in critical situations when energy conservation is a lesser concern anyway.

4 Asynchronous Collision Handling

In most practically interesting scenarios, cloth collides with other objects in the environment and with itself. It is therefore necessary to account for collision detection and response in the asynchronous framework. The simplest method would be to synchronise the system (in analogy to how frames are generated) and invoke a standard collision handling scheme at regular intervals. As mentioned in the introduction, stability reasons call for small intervals, which lead to high computation times. As a consequence we choose to integrate collision handling directly into the asynchronous update scheme.

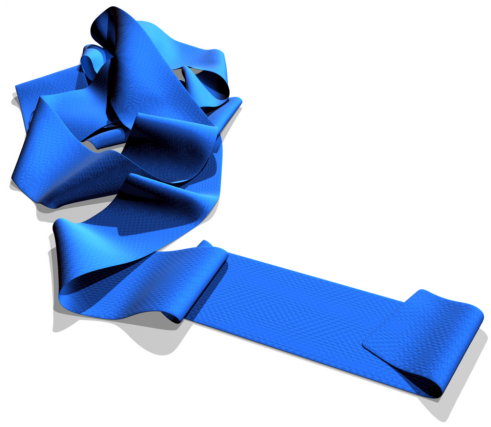

Fig. 5 Complicated multi-layer self-collisions are reliably resolved using our 3-stage collision handling pipeline. The ribbon consists of 8000 faces.

4.1 Algorithm

The asynchronous collision handling scheme builds upon techniques commonly used in standard approaches as [BFA02]. Bounding volume hierarchies are used to quickly detect potentially colliding close face pairs (cfp). These cfps are subsequently checked for actual proximity using geometric distance tests. If a cfp passes this test, it is decomposed into elementary edge-edge and vertex-triangle collisions. These elementary collisions are then used to compute stopping impulses, which prevent any imminent intersection.

The asynchronous handling scheme is organised into three stages. The pre- and post-handling stage act globally on the synchronised system state using a dedicated collision time step Δt_c (usually 0.01s). The second stage, which is coupled with the element update scheme, handles the actual collisions asynchronously. The following paragraphs explain each of these stages in more detail.

Pre-Handling Stage The task of this stage is to detect any cfps that may potentially collide in the interval $[t, t + \Delta t_c]$. Here, a conservative scheme that captures collisions as robustly as possible is preferred. To this end, we first update the BVH using the current nodal positions. We then compute candidate positions at the time $t + \Delta t_c$ using a simple explicit integration step. These candidate positions are then used to *enlarge* the BVH, i.e. to extend the bounding volumes (BVs) in such a way that they include both initial and candidate positions. This is done for both deformable and rigid moving objects. The global collision detection step yields a list of cfps valid for the entire interval $[t, t + \Delta t_c]$ and its entries are mapped to the corresponding elements of the cloth object.

Asynchronous Collision Handling Upon activation of an element its nodal positions are first updated according to algorithm 1, l.10. If the list of potentially colliding faces for this element is empty the algorithm proceeds as usual.

However, if there are close faces in the list, the computed nodal positions are reconsidered as candidate positions for the subsequently invoked local collision handling step. For every close element in the list, its nodal positions are first synchronised with the time of the current element. Subsequently, a geometric distance computation is carried out. If the distance is smaller than a user supplied threshold value, elementary collisions are created and appropriate responses are computed. In order to speed up the response calculation (involving 9 edge-edge and 6 vertex-triangle pairs) we use elementary bounding volumes for quickly ruling out pairs whose distance exceeds the threshold. Stopping impulses that prevent imminent intersections and repelling impulses that push too close elements apart are then generated for both of the faces (in the case of self or inter-cloth collisions) as described in [BFA02]. After all elements in the list have been processed, the changed velocities of the current element are used to compute its final nodal positions for this update step.

Post-Handling Stage Even with a conservative detection approach there might still be collisions left that have not been captured by the first stage or could not be resolved in the second one. For a robust collision handling scheme it is indispensable to have a fail-safe post-processing step which reliably resolves any collisions remaining at the end of interval $[t, t + \Delta t_c]$. For this purpose, we carry out an additional collision detection step which is identical to the first stage, except that the candidate positions are now known from the asynchronous simulation stage. A continuous collision detection and handling pass is then applied iteratively until all remaining collisions are resolved. Corrections made in this third stage can be considered as rather strict interventions since, unlike the responses computed in the second stage, they can constitute a rough discontinuity in the simulation. However, the second stage is usually capable of resolving almost all of the collisions, leaving no or very little work to do for the third stage.

Continuous Geometric Culling The number of cfps detected in the first stage can become very high for complicated scenarios, thus slowing down simulation. This situation can be significantly improved using the fact that usually only a small fraction of the detected cfps are actually handled subsequently. We therefore carry out an additional continuous geometric culling step after the first stage, which tests for every cfp whether the linearly extrapolated trajectories of the involved triangles (determined by their initial and candidate positions) actually intersect. If this is not the case, they are pruned from the input list for the second stage. The geometric intersection test is very fast and since it is applied only in the broad phase (i.e. once every collision time step) it does not have a significant impact on overall performance. Yet, the improvement in the ratio between detected and actually handled cfps can be significant: for our ribbon test scene (Fig. 5) we obtained an average culling rate of roughly 55% (see Fig. 6).

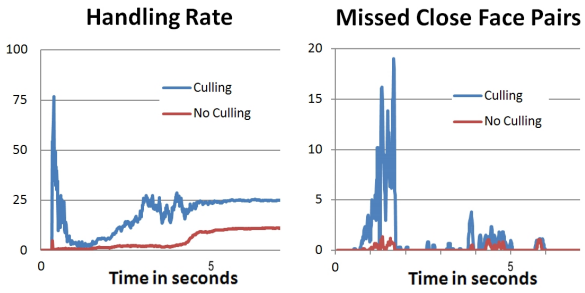

Fig. 6 Culling efficiency and incurred approximation error for the ribbon scene. *Left*: percentage of actually handled close face pairs in the asynchronous phase over time. *right*: number of close face pairs missed in the asynchronous stage, which have to be handled by the third pipeline stage.

As can be seen in the right diagram of Fig. 6 the number of cfps that are missed in the asynchronous stage is slightly higher when using culling. However, this value is still very small compared to the number of asynchronously handled collision and all of these remaining pairs can be resolved in the third stage.

Preventing Instabilities in Collision Response When sudden character motion or high-velocity impacts occur, the collision response can lead to large strains and strain rates which threaten to destabilise the simulation. Most cases can be handled using the strain and strain rate limiting described in Sec. 3. In cases of excessive deformation, which we define as $\varepsilon_n > 2{\cdot}\varepsilon_{n,lim}$, we additionally reduce the step size such that the condition holds for the new time step. Although the number of times that this correction needs to be applied is usually small, it is still an important component for stable simulation.

Discussion There are situations in which it is not enough to check only the element that is currently updated for collisions. By advancing the particular element forward in time adjacent elements are automatically affected and collisions might be introduced without being noticed. We are aware of this issue, but since a complete collision check of the element's entire neighbourhood would drastically increase computation costs, we leave this to the third stage. In practice, this decision does not have a negative effect on the robustness of our scheme. Finally, since we use large collision time steps of 0.01-0.05 seconds, the computational costs for the global pre- and post-handling stages become negligible and most of the computation time is spent on the actual asynchronous handling.

Integrator	Timestep	T_{solve}
backward Euler	0.0333	5.66
backward Euler	0.01	17.05
backward Euler	0.00075	218.4
forward Euler	0.00020	265.0
symplectic Euler	0.00025	208.0
semi-implicit MPR	0.001	247.4
AVI	0.00037	215.0

Table 1 Computation times in seconds for scene 1 in seconds. The scene consists of a simple piece of cloth swinging for 4s, consisting of 5016 faces.

Integrator	Time-step	T_{solve}	T_{coll}	T_{tot}
backward Euler	0.005	7.74	4.31	12.05
backward Euler	0.001	16.1	9.7	25.8
forward Euler	0.00001	571.5	213.3	784.8
symplectic Euler	0.00005	109.5	62.09	171.59
semi-implicit MPR	0.0001	323.9	51.98	375.88
AVI	0.0002	47.2	21.12	68.32

Table 2 Computation times in seconds for scene 2. The scene consists of an unstructured mesh of 2304 faces with non-flat rest geometry falling onto the floor.

5 Results

We evaluated our method on several test cases. In the first test we compare the performance of different explicit and implicit integration schemes with respect to both computation times and especially their visual quality. The latter is, to some extent, subjective and we therefore chose a traditional example that clearly demonstrates the differences between the methods: a piece of cloth with 5016 faces is pinned at two corners and left swinging for 4 seconds. For this example, we used a material with Young's modulus of $E = 300N/m$, a shear modulus of $G = 150N/m$ and mass density $\rho = 500g/m^2$. Moreover, strain limiting with 10% allowed strain was used. Fig. 2 shows a representative frame of this animation. Using a large time step of $dt = 0.03s$, the backward Euler integrator damps out almost all of the important features (Fig. 2, left). Decreasing the time step helps, but compared to the implicit midpoint rule, which performs slightly better, and especially the explicit integrators, the overall appearance still looks rather flat. This behaviour can be explained by the numerical dissipation inherent to the backward Euler scheme, which becomes particularly apparent when large time steps are used. The other integration schemes perform considerably better in this regard, which can be attributed to their symplectic nature and the associated energy conservation properties.

Table 1 shows the computation times for the first scene. With a time step of $dt = 0.03s$ impressive computation times can be obtained for the backward Euler. However, the result appears overly flat and lacks detail. The situation becomes better when using $dt = 0.00075s$, which leads to roughly the same computation time as for our method. Still, folds are not reproduced as detailed as with our AVI-based scheme or the symplectic Euler scheme, which yields similar visual results. Balancing visual quality against computation times, the implicit midpoint rule shows a rather deceiving performance. Despite its symplectic nature, results look still too smooth, which is due to the fact that we had to use a high damping value of 0.1 to maintain stability even for a small time step of $dt = 0.001$. We conjecture that in order to be of practical use, the variant described in [VMT05] has to be used. Note that although our AVI-based method has some overhead for managing asynchronicity, it still performs almost as well as the symplectic Euler, which shows the best overall performance in this test. This is also due to the fact that an unstructured mesh is used for simulation, which, having triangles with slightly different sizes, leads to a higher average elemental time step. This advantage is even more pronounced for meshes optimised for accurately capturing regions of high curvatures with small triangles, but which are otherwise coarse in flat parts (see Fig. 5). The second test scene uses such a mesh with non-flat rest geometry. During 4 seconds of simulation, the object falls from 50 cm height onto a flat floor, where it comes to rest. The parameters are the same as in scene 1 except for a higher bending stiffness, which lets the object keep its shape after the impact. Table 2 shows the computation times for this scene with separate timings for time stepping and collision handling. Backward Euler is again the fastest, but our method performs significantly better than the other explicit schemes and the implicit midpoint rule.

The two remaining scenes show that our method can be used for creating standard cloth simulations in combination with character animations (see Fig. 1) as well as challenging self-collision scenarios (see Fig. 5). The accompanying video also shows a comparison between the results obtained with our method and using backward Euler.

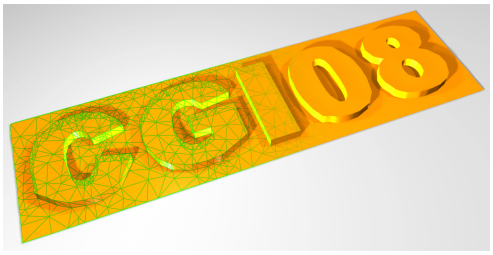

Fig. 7 Simulation of an unstructured mesh containing elements of different sizes. The ratio between the area of the largest and the smallest element is 160:1. The mesh consists of 2304 faces.

6 Conclusion

We presented an asynchronous simulation method for thin flexible objects, which enables efficient simulation of low damped cloth. Thanks to the energy conservation properties of the underlying symplectic time integration scheme, lively cloth motion is obtained, exhibit-

ing important details like small folds and wrinkles. The computational costs of our method are similar to backward Euler with a time step of $dt = 0.001$. Yet it reveals more surface details and leads to results which are less damped. Our method is particularly attractive when combined with unstructured meshes having elements of different sizes. For future work we would like to extend our simulator to account for spatially adaptive meshes since in this way computation times could be further reduced.

Acknowledgments

The first author would like to thank the DAAD for grant D/07/45108. The second author was supported by DFG grant STR465/21-1.

References

[Bat96] K.-J. Bathe. *Finite element procedures*. Prentice Hall, New Jersey, 1996.

[BFA02] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of ACM SIGGRAPH '02*, pages 594–603, 2002.

[BMF03] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2003)*, pages 28–36, 2003.

[BW97] J. Bonet and R. D. Wood. *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press, Cambridge, 1997.

[BW98] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proceedings of ACM SIGGRAPH '98*, pages 43–54, 1998.

[BWH+06] M. Bergou, M. Wardetzky, D. Harmon, D. Zorin, and E. Grinspun. A Quadratic Bending Model for Inextensible Surfaces. In *Fourth Eurographics Symposium on Geometry Processing (SGP)*, pages 227–230, Jun 2006.

[Cel98] W. Celes. Efficient asynchronous evolution of physical simulations. In *Proceedings of SIBGRAPI'98*, pages 224–231, 1998.

[CK02] K. J. Choi and H. S. Ko. Stable but responsive cloth. In *Proceedings of ACM SIGGRAPH '02*, pages 604–611, 2002.

[DGC04] J. Dequidt, L. Grisoni, and C. Chaillou. Asynchronous interactive physical simulation. Technical Report RR-5338, INRIA, October 2004.

[EWS96] B. Eberhardt, A. Weber, and W. Straßer. A fast, flexible, particle-system model for cloth draping. *IEEE Computer Graphics and Applications*, 16(5):52–59, 1996.

[FO01] F. Flores and E. Onate. A basic thin shell triangle with only translational dofs for large strain plasticity. *International Journal for Numerical Methods in Engineering*, 51:57–83, 2001.

[GHDS03] E. Grinspun, A. Hirani, M. Desbrun, and P. Schröder. Discrete shells. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2003)*, pages 62–67, 2003.

[GHF+07] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun. Efficient simulation of inextensible cloth. In *Proceedings of ACM SIGGRAPH '07*, pages 281–290, 2007.

[HE01] M. Hauth and O. Etzmuss. A high performance solver for the animation of deformable objects using advanced numerical methods. In *EG 2001 Proceedings*, pages 319–328. Blackwell Publishing, 2001.

[HLW06] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration*. Springer-Verlag, Berlin, 5th edition, 2006.

[KMOW00] C. Kane, J. Marsden, M. Ortiz, and M. West. Variational integrators and the newmark algorithm for conservative and dissipative mechanical systems. *International Journal for Numerical Methods in Engineering*, 49:1295–1325, 2000.

[KYT+06] L. Kharevych, W. Yang, Y. Tong, E. Kanso, J. E. Marsden, P. Schröder, and M. Desbrun. Geometric, variational integrators for computer animation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2006)*, pages 43–51, 2006.

[LMOW04] A. Lew, J. E. Marsden, M. Ortiz, and M. West. Variational time integrators. *International Journal for Numerical Methods in Engineering*, 60:153–212, 2004.

[MKE03] J. Mezger, S. Kimmerle, and O. Etzmuß. Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG*, 11(2):322–329, 2003.

[OAW04] S. Oh, J. Ahn, and K. Wohn. A new implicit integration method for low damped cloth simulation. In *Proceedings of GMCG 2004*, pages 115–121, 2004.

[Pro95] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface '95*, pages 147–154, 1995.

[THM+05] M. Teschner, B. Heidelberger, D. Manocha, N. Govindaraju, G. Zachmann, S. Kimmerle, J. Mezger, and A. Fuhrmann. Collision Handling in Dynamic Simulation Environments. In *Eurographics Tutorials*, pages 79–185, 2005.

[TPBF87] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Proceedings of ACM SIGGRAPH '87*, pages 205–214, 1987.

[VCMT95] P. Volino, M. Courchesne, and N. Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *Proceedings of ACM SIGGRAPH '95*, pages 137–144, 1995.

[VMT05] P. Volino and N. Magnenat-Thalmann. Implicit midpoint integration and adaptive damping for efficient cloth simulation: Collision detection and deformable objects. *Computer Animation in Virtual Worlds*, 16(3-4):163–175, 2005.