Simulating inextensible cloth using locking-free triangle meshes

Jan Bender[1], Raphael Diziol[2], Daniel Bayer[2]

[1]Graduate School CE, TU Darmstadt, Germany
[2]Karlsruhe Institute of Technology, Germany

Abstract

This paper presents an efficient method for the dynamic simulation of inextensible cloth. The triangle mesh for our cloth model is simulated using an impulse-based approach which is able to solve hard constraints. Using hard distance constraints on the edges of the triangle mesh removes too many degrees of freedom, resulting in a rigid motion. This is known as the locking problem which is typically solved by using rectangular meshes in existing impulse-based simulations. We solve this problem by using a nonconforming representation for the simulation model which unfortunately results in a discontinuous mesh. Therefore, we couple the original conforming mesh with the nonconforming elements and use it for collision handling and visualization.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

The simulation of inextensible surfaces is an important research topic in computer graphics. For example, cloth is often treated as an elastic material due to performance reasons. Elastic materials can be efficiently simulated using mass-spring systems. However, many textiles do not stretch significantly under their own weight. Therefore, an inextensible cloth is required for a realistic simulation.

In this paper we present an impulse-based approach for the simulation of inextensible cloth. Impulse-based simulation has already been used for such models. In [BB08] the models consist of particles linked by hard distance constraints in a regular rectangular structure. Triangle meshes are not supported yet by impulse-based methods due to locking problems. We solve this problem by using a nonconforming mesh based on the idea of English and Bridson [EB08]. Therefore, the simulation method presented in this paper can also handle triangle mesh models.

2. Related work

The dynamic simulation of cloth has a long history in computer graphics (see [MtV05] for a survey over 20 years of cloth simulation). In [TPBF87] the first general physical model to simulate elastic non-rigid curves, surfaces, and solids was presented. In this paper a semi-implicit integration scheme is used in order to perform a stable simulation with stiff materials. In the following years many publications focused on pure explicit integration methods as for example [BHW94] where cloth is treated as elastic material. The use of explicit methods was mainly justified by the lower computational costs compared to (semi-)implicit methods. The simulation of inextensible cloth using spring models leads to stiff differential equations [HES03]. Solving the differential equation with an explicit integration method leads to instabilities in the simulation and requires small time steps. Baraff and Witkin [BW98] solved this problem by using an implicit integration scheme which is able to handle large time steps. Instead of purely relying on stiff springs, Provot [Pro95] proposed to correct the particle positions directly if their extensions exceed more than 10%, eliminating the need for an implicit inte-

gration scheme. Since such corrections may introduce self-penetrations, Bridson et al. [BFA02] used impulses in order to limit the strain and the strain rate.

Instead of computing spring forces to satisfy the constraints approximately other simulation methods try to compute an accurate solution. Goldenthal et al. [GHF*07] used constrained Lagrangian mechanics and presented a novel fast projection method for the simulation of inextensible cloth. Bender and Bayer [BB08] proposed a parallel simulation method based on an impulse-based approach [Ben07] which enables strain limitation. A continuum-based strain limiting approach was presented by Thomaszewski et al. [TPS09]. In [MC10] only a coarse mesh is simulated to reduce stretchiness and a higher resolution mesh is attached in order to get highly detailed wrinkles. In order to solve positions constraints, Müller et al. [MHHR06] used direct position corrections computed with a Gauss-Seidel type iteration method. The disadvantage of this approach is that velocity based constraints cannot be handled.

No matter which of the above described simulation methods is used, locking can occur in case of inextensible cloth [LTJ07]. The main target of this work is to solve this problem for the impulse-based simulation. The locking problem is not only important when simulating cloth but also for elastically deformable objects. For example, when simulating a deformable model using a linear finite element method with tetrahedral elements, the model generally locks in case of a high Poisson's ratio. Irving et al. [ISF07] showed how to still conserve the volume with a locking-free formulation by generating a divergence-free velocity field based on the one-ring of tetrahedrons. Instead of using tetrahedral elements, Kaufmann et al. [KMBG09] showed that the problem does not occur when using a Galerkin FEM method which is based on nonconforming or discontinuous shape functions. Following English and Bridson [EB08], we also use nonconforming elements to resolve locking in case of inextensible cloth.

3. Simulation method

3.1. Time integration

The cloth model used for the simulation is a mesh of particles. Each particle has a mass m, a position $\mathbf{x}$ and a velocity $\mathbf{v}$. In our simulation we integrate the velocities and positions in the following way:

$$\begin{aligned} \mathbf{v}(t+h) &= \mathbf{v}(t)+\frac{1}{m}\mathbf{F}h \\ \mathbf{x}(t+h) &= \mathbf{x}(t)+\mathbf{v}(t)h+\frac{1}{2m}\mathbf{F}h^2 \end{aligned} \qquad (1)$$

where $\mathbf{F}$ are forces acting on the particles and h is the time step size. Any other integration method can be used but the same method must also be used for the preview in section 3.2.

3.2. Constraints

For the simulation of inextensible cloth we use distance constraints in order to keep the distance between two particles constant. Such a constraint consists of two parts. The first part constrains the motion of the linked vertices by the following implicit function:

$$C_{\text{pos}}(\mathbf{x}_i,\mathbf{x}_j) = |\mathbf{x}_i-\mathbf{x}_j| - d_{ij}^0 = 0$$

where $d_{ij}^0 > 0$ is the initial distance between particle i and j. The second part is the time derivative of C_{pos} and requires that the relative velocity of the particles in direction of the constraint is zero. Therefore, we define the function:

$$C_{\text{vel}}(\mathbf{x}_i,\mathbf{x}_j,\mathbf{v}_i,\mathbf{v}_j) = (\mathbf{v}_j-\mathbf{v}_i)\cdot\frac{\mathbf{x}_j-\mathbf{x}_i}{|\mathbf{x}_j-\mathbf{x}_i|} = 0.$$

In the following we will call the first part *position constraint* and the second part *velocity constraint*.

We use an impulse-based approach for the simulation of the constraints. The simulation is performed by the following steps:

1. Solve all position constraints by computing impulses using a preview of the particle positions.
2. Perform an integration step as described in section 3.1.
3. Determine impulses to solve the velocity constraints.

A constraint is satisfied by computing a pair of impulses $\mathbf{p}$ and $-\mathbf{p}$ for the corresponding particles. The velocity of a particle can be directly changed by an impulse but its position only changes during the integration step. Therefore, a preview is used to solve the position constraints.

In this work the preview of a particle position is done by evaluating equation 1. Since we only change the velocities of the particles by impulses, this preview can also be done by any other integration method (e.g. by the Euler method) as long as the same method is used for the preview and the integration step.

After computing the preview positions $\mathbf{x}_i(t+h)$ for all particles, the constraint functions are evaluated in order to get the expected position errors $C_{\text{pos}}(\mathbf{x}_i(t+h),\mathbf{x}_j(t+h))$. If we assume that the relative motion of two linked particles is linear, the magnitude of the velocity change that eliminates the corresponding position error during an integration step is

$$\Delta v_{\text{pos}} = \frac{C_{\text{pos}}(\mathbf{x}_i(t+h),\mathbf{x}_j(t+h))}{h}. \qquad (2)$$

In general the relative motion is not linear but in practice it is almost linear during the small time steps used

for the simulation. Therefore, equation 2 is at least a good approximation of the required velocity change.

Since an impulse changes the velocity of a particle directly, the velocity change required for satisfying a velocity constraint is simply

$$\Delta v_{\text{vel}} = C_{\text{vel}}(\mathbf{x}_i(t), \mathbf{x}_j(t), \mathbf{v}_i(t), \mathbf{v}_j(t)).$$

Now that we know the required velocity changes, we compute corresponding impulses for the particles. The relative velocities of two dynamic particles i and j change by

$$\frac{1}{m_i}\mathbf{p} - \frac{1}{m_j}(-\mathbf{p}) = \left(\frac{1}{m_i} + \frac{1}{m_j}\right)\mathbf{p} = \Delta\mathbf{v}$$

when a pair of impulses $\mathbf{p}$ and $-\mathbf{p}$ is applied to the particles. To differentiate between static and dynamic particles we define

$$k_i = \begin{cases} \frac{1}{m_i} & \text{if particle } i \text{ is dynamic} \\ 0 & \text{otherwise.} \end{cases}$$

The correction impulse $\mathbf{p}$ for a distance constraint is then determined by:

$$\begin{aligned} \mathbf{p} &= \frac{1}{k_i + k_j}\Delta v_{\text{pos}} \cdot \frac{\partial C_{\text{pos}}}{\partial \mathbf{x}_i} \\ \mathbf{p} &= \frac{1}{k_i + k_j}\Delta v_{\text{vel}} \cdot \frac{\partial C_{\text{vel}}}{\partial \mathbf{v}_i}. \end{aligned}$$

The direction of the impulse is the gradient of the constraint C at time t which results from D'Alembert's principle. Notice the difference to position-based dynamics [MHHR06] where the direction is the gradient at time $t+h$. The impulse can be computed if the two conditions $k_i + k_j \neq 0$ and $|\mathbf{x}_j - \mathbf{x}_i| \neq 0$ are fulfilled. The first condition is generally satisfied because a distance constraint between two static particles makes no sense. Since the position constraint is fulfilled after each simulation step, $|\mathbf{x}_j - \mathbf{x}_i| = 0$ is only possible if $d_{ij}^0 = 0$ which is excluded by the definition of the position constraint. Also note that the second condition is not necessarily satisfied for position-based dynamics.

The impulses of two distance constraints depend on each other if they have a common particle. Therefore, we determine all impulses at once by solving a system of linear equations which takes the dependencies into account. The matrix of this system reflects the mesh structure of the simulation model. We have a diagonal element for each distance constraint and an off-diagonal element for each pair of constraints with a common particle. Hence, the resulting matrix is sparse. For the sign of an off-diagonal element it is important if the common particle is the first or second one of the corresponding constraints. Therefore, we differentiate between the following cases for two constraints i and j:

$$K_{i,j} = \begin{cases} k_{i_1} & \text{if } i_1 = j_1 \wedge i_2 \neq j_2 \\ k_{i_2} & \text{if } i_2 = j_2 \wedge i_1 \neq j_1 \\ -k_{i_1} & \text{if } i_1 = j_2 \wedge i_2 \neq j_1 \\ -k_{i_2} & \text{if } i_2 = j_1 \wedge i_1 \neq j_2 \\ k_{i_1} + k_{i_2} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

where i_1 and i_2 are the indices of the first and second particle of constraint i.

In order to compute an impulse for a constraint i regarding all dependencies, we must project the impulses of the dependent constraints into the space of constraint i. This is done by using the projection matrix

$$\mathbf{P}_i = \left(\frac{\mathbf{x}_{i_2} - \mathbf{x}_{i_1}}{|\mathbf{x}_{i_2} - \mathbf{x}_{i_1}|}\right)^T \in \mathbb{R}^{1\times 3}.$$

In the following we define the system of linear equations for the impulse magnitudes:

$$\mathbf{A}\tilde{\mathbf{p}} = \Delta\mathbf{v}.$$

The dimension of this system equals the number of distance constraints in the model. An element $A_{i,j}$ of the matrix is the value $K_{i,j}$ combined with the corresponding projection matrices:

$$A_{i,j} = K_{i,j}\mathbf{P}_i\mathbf{P}_j^T.$$

Therefore, the matrix $\mathbf{A}$ is symmetric. For the diagonal elements no projection is required since $\mathbf{P}_i\mathbf{P}_i^T = 1$. The vectors $\Delta\mathbf{v}$ and $\tilde{\mathbf{p}}$ contain all required velocity changes and the magnitudes of their corresponding impulses:

$$\begin{aligned} \Delta\mathbf{v} &= (\Delta v_1, \ldots, \Delta v_n)^T \\ \tilde{\mathbf{p}} &= (\tilde{p}_1, \ldots, \tilde{p}_n)^T. \end{aligned}$$

Finally, we multiply the projection matrices with the impulse magnitudes in order to get three-dimensional impulses:

$$\mathbf{p}_i = \tilde{p}_i\mathbf{P}_i^T.$$

The impulses solve the position constraints for $\Delta\mathbf{v}_{\text{pos}}$ on the right hand side of the system and the velocity constraints for $\Delta\mathbf{v}_{\text{vel}}$.

In contrast to $\Delta\mathbf{v}_{\text{vel}}$ the vector $\Delta\mathbf{v}_{\text{pos}}$ was only a good approximation. Therefore, we solve the system multiple times in the case of position constraints until a certain accuracy is reached. In practice, we need one or two iterations for accurate results. The velocity constraints are satisfied immediately after applying the impulses while the position constraints are fulfilled after the next integration step because the desired position changes are caused indirectly by manipulating the velocities. The most time consuming part

of solving the system is the factorization of the matrix. The matrix for position and velocity constraints is the same and it is constant at time t. Hence, the matrix generation and factorization must be performed only once per time step. The factorization can then be used for solving position and velocity constraints at time t. Therefore, the simulation does not slow down much even if several iterations are necessary in order to correct the positions. In this work we used PARDISO [SG02, SG04] for solving which is optimized for sparse systems.

3.3. Simulation model

The distance constraints described in the last section are hard constraints. If we use a triangle mesh for the simulation and define such a constraint for each edge, the triangles remain rigid and just a trivial bending is possible. Wrinkles in the triangle mesh are only possible along edges which build a straight line through the whole mesh. Therefore, we have a locking problem with this kind of model. To solve this problem we use nonconforming elements as proposed in [EB08].

Instead of positioning the particles of the simulation model at the vertices of the mesh, we put them on the midpoints of the edges (see figure 1). In this way the number of variables is increased since two adjacent triangles have now only one common particle instead of two. These particles are then linked by distance constraints. The resulting model consisting of nonconforming elements has more degrees of freedom which solves the locking problem.

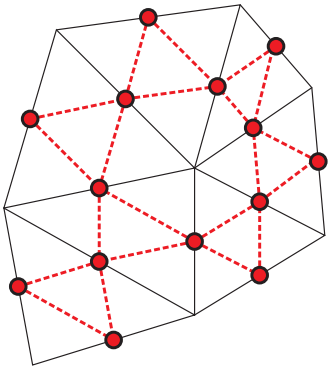


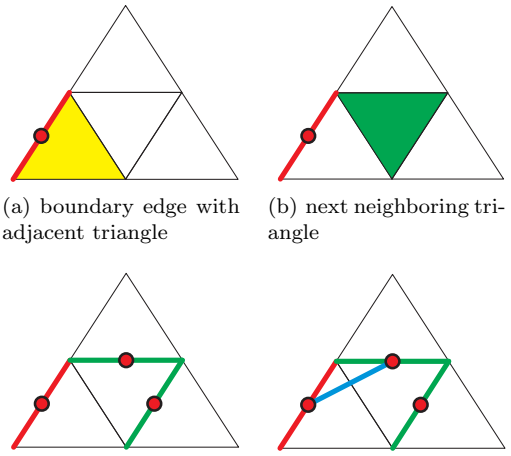
Figure 1: *The particles of the nonconforming mesh (red) are located at the midpoints of the original edges (black). Distance constraints are defined between these particles.*

The masses of the particles depend on the triangulation of the simulated mesh. If we assume that the area density ρ of the model is constant, the mass of a particle can be determined by:

$$m = \rho \cdot a$$

where a is the area represented by the particle. The total mass of the original mesh and the new midpoint mesh must be equal. Therefore, the particles of the new mesh must have a mass which corresponds to their area in the original mesh. Each particle lies on an edge of the original mesh. An edge is part of one (on the boundary) or two triangles. The area represented by a particle is therefore determined by summing up the areas of these adjacent triangles and dividing the result by three since we have three particles per triangle.

The model described so far has too many degrees of freedom at the boundary. Boundary triangles of the nonconforming mesh have only one or two common particles with interior triangles. Therefore, they can rotate freely around these common particles without regarding the orientations of adjacent triangles. This problem is solved by introducing additionally distance constraints.



(a) boundary edge with adjacent triangle

(b) next neighboring triangle

(c) candidate particles for boundary constraint

(d) boundary constraint

Figure 2: *Definition of a boundary constraint*

In order to define the boundary constraints, we first determine all boundary edges of the original mesh. These edges have only one adjacent triangle (see figure 2(a)). This boundary triangle has at most two neighboring triangles which are determined in the next step (see figure 2(b)). A neighboring triangle has two edges which are not common with the boundary triangle. The particles at the midpoints of these edges can be used for the definition of a boundary constraint (see figure 2(c)). Finally, we just use the particle with the smallest distance to the particle on the boundary edge

(see figure 2(d)). The result for the mesh of figure 1 is shown in figure 3.

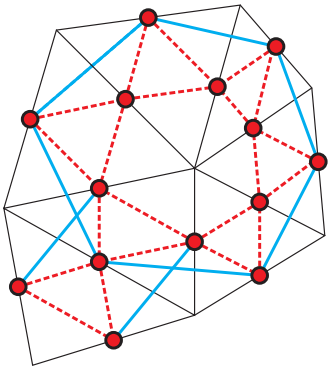


Figure 3: *Additional boundary constraints for a nonconforming mesh*

The problem of using a nonconforming mesh for simulation is that we get discontinuities in our simulation model apart from the midpoint particles. Therefore, the collision detection and the visualization of the model is not performed directly. Instead we couple the original mesh with the nonconforming elements and adapt the vertex positions.

The vertex positions of the conforming mesh are determined by interpolating the vertex positions of the original mesh. For a triangle of the nonconforming mesh, we can compute the vertex positions of its corresponding original triangle by

$$\begin{aligned}\bar{\mathbf{x}}_1^{\mathrm{c}} &= \mathbf{x}_2^{\mathrm{nc}} + \mathbf{x}_3^{\mathrm{nc}} - \mathbf{x}_1^{\mathrm{nc}} \\ \bar{\mathbf{x}}_2^{\mathrm{c}} &= \mathbf{x}_3^{\mathrm{nc}} + \mathbf{x}_1^{\mathrm{nc}} - \mathbf{x}_2^{\mathrm{nc}} \\ \bar{\mathbf{x}}_3^{\mathrm{c}} &= \mathbf{x}_1^{\mathrm{nc}} + \mathbf{x}_2^{\mathrm{nc}} - \mathbf{x}_3^{\mathrm{nc}}\end{aligned}$$

where $\mathbf{x}_i^{\mathrm{nc}}$ is the midpoint of the edge opposite to the conforming point i. The final positions $\mathbf{x}^{\mathrm{c}}$ of the conforming vertices are determined by averaging their corresponding values $\bar{\mathbf{x}}^{\mathrm{c}}$. We use an averaging matrix $\mathbf{B}$ to describe the transfer of positions and velocities between the nonconforming and the conforming mesh:

$$\begin{aligned}\mathbf{x}^{\mathrm{c}} &= \mathbf{B}\mathbf{x}^{\mathrm{nc}} \\ \mathbf{v}^{\mathrm{c}} &= \mathbf{B}\mathbf{v}^{\mathrm{nc}}\end{aligned}$$

The ghost conforming mesh is used for visualization and collision handling. We use the collision handling described in [BWK03] in order to obtain a state without intersections for the conforming mesh. The resulting mesh is used for rendering. After collision handling we must transfer the position change $\Delta\mathbf{x}^{\mathrm{c}}$ of the conforming mesh to the nonconforming vertices. The final positions $\mathbf{x}_f^{\mathrm{nc}}$ are determined using a Lagrange multiplier form as described in [EB08]:

$$\mathbf{x}_f^{\mathrm{nc}} = \mathbf{x}^{\mathrm{nc}} + \mathbf{B}^T \lambda.$$

The vector λ is determined by solving the symmetric, positive definite linear system

$$\mathbf{B}\mathbf{B}^T \lambda = \Delta\mathbf{x}^{\mathrm{c}}.$$

This guarantees that the interpolation of the final positions cause exactly the desired position change in the conforming mesh:

$$\mathbf{B}\mathbf{x}_f^{\mathrm{nc}} = \mathbf{x}^{\mathrm{c}} + \Delta\mathbf{x}^{\mathrm{c}}.$$

Since the matrix $\mathbf{B}\mathbf{B}^T$ is constant, its factorization can be precomputed. Therefore, the computation of the vector λ does not require much computation time during the simulation. Finally, we update the velocities of the nonconforming mesh by

$$\mathbf{v}_f^{\mathrm{nc}} = \mathbf{v}^{\mathrm{nc}} + \frac{\mathbf{x}_f^{\mathrm{nc}} - \mathbf{x}^{\mathrm{nc}}}{h}.$$

Positions and velocities of the nonconforming mesh must only be updated if the collision handling causes a change of the conforming mesh.

Since we use a nonconforming mesh for the simulation, we also have to compute the bending forces for this mesh. Therefore, an adapted version of the bending model described by Wardetzky et al. [WBH*07] can be used as proposed in [EB08].

4. Results

In this section we present results of the presented method. All simulations were performed on a PC with two Intel X5650 processors with 2.66 GHz. The systems of linear equations are solved using PARDISO. We performed the impulse-based simulation with a maximum tolerance of 10^{-6} m for the position constraints.

Figure 4 shows the model of a table cloth. The original mesh consists of 4722 triangles and 7143 edges. The nonconforming mesh has 7143 particles, 14166 distance constraints and 240 boundary constraints.

In the second simulation (see figure 5) we dropped a 100×100 mesh on a sphere. The model consists of 10000 particles and 59396 distance constraints. English and Bridson ran a similar simulation at 9.52 seconds/step on an Athlon 64 3500+.

The timings for a simulation step with a step size of 1 ms are shown in table 1. The main part of the total computation time is required for the factorization of the matrix. The computation time for solving the position constraints is a bit higher than the one for the velocity constraints. The reason for this is that

Scene	# constraints	factorization	position constraints	velocity constraints	total
Table cloth	14406	73.6 ms	7.2 ms	6.6 ms	109.5 ms
Sphere	59396	335.6 ms	31.1 ms	26.1 ms	572.4 ms

Table 1: *Simulation timings for table cloth and sphere scene. The table shows the time required for the creation and factorization of the matrix, the solution of all position and velocity constraints and the total computation time including collision handling.*

Figure 4: *Table cloth simulated with a nonconforming mesh*

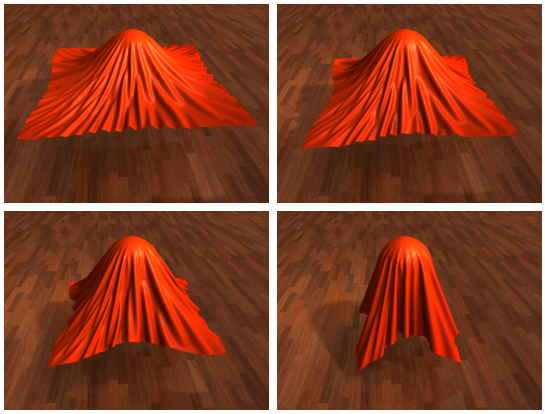


Figure 5: 100×100 *mesh dropped on a rigid sphere*

the solver required between one and two iterations for an accurate solution of the position constraints while the velocity constraints can be solved exactly without an iteration process.

5. Conclusion

We presented an impulse-based simulation method for inextensible surface models. In contrast to mass-spring models, hard distance constraints are used to prevent the surfaces from stretching. Former impulse-based methods were limited to regular rectangular mesh models, since the distance constraints cause locking problems when using a conforming triangle mesh for the simulation. We overcome this limitation by using a nonconforming mesh defined in the midpoints of the original mesh. Since this introduces discontinuities, the original conforming mesh is coupled in order to perform collision handling and visualization.

References

[BB08] Bender J., Bayer D.: Parallel simulation of inextensible cloth. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (Grenoble (France), Nov. 2008), pp. 47–56. 1, 2

[Ben07] Bender J.: Impulse-based dynamic simulation in linear time. *Computer Animation and Virtual Worlds 18*, 4-5 (2007), 225–233. 2

[BFA02] Bridson R., Fedkiw R., Anderson J.: Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. 21* (2002), 594–603. 2

[BHW94] Breen D. E., House D. H., Wozny M. J.: Predicting the drape of woven cloth using interacting particles. In *Proceedings of Computer graphics and interactive techniques* (New York, NY, USA, 1994), SIGGRAPH '94, ACM, pp. 365–372. 1

[BW98] Baraff D., Witkin A.: Large steps in cloth simulation. In *Proceedings of Computer graphics and interactive techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 43–54. 1

[BWK03] Baraff D., Witkin A., Kass M.: Untangling cloth. *ACM Transactions on Graphics 22*, 3 (2003), 862–870. 5

[EB08] English E., Bridson R.: Animating developable surfaces using nonconforming elements. *ACM Trans. Graph. 27* (August 2008), 66:1–66:5. 1, 2, 4, 5

[GHF*07] Goldenthal R., Harmon D., Fattal R., Bercovier M., Grinspun E.: Efficient simulation of inextensible cloth. *ACM Transactions on Graphics 26*, 3 (2007), 49. 2

[HES03] Hauth M., Etzmuss O., Strasser W.: Analysis of numerical methods for the simulation of deformable models. *The Visual Computer 19*, 7-8 (2003), 581–600. 1

[ISF07] Irving G., Schroeder C., Fedkiw R.: Volume conserving finite element simulations of deformable models. In *ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM. 2

[KMBG09] Kaufmann P., Martin S., Botsch M., Gross M.: Flexible simulation of deformable models using discontinuous Galerkin FEM. *Graph. Models 71* (July 2009), 153–167. 2

[LTJ07] Liu Y.-J., Tang K., Joneja A.: Modeling dynamic developable meshes by the hamilton principle. *Comput. Aided Des. 39* (September 2007), 719–731. 2

[MC10] Müller M., Chentanez N.: Wrinkle meshes. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2010), SCA '10, Eurographics Association, pp. 85–92. 2

[MHHR06] Müller M., Heidelberger B., Hennix M., Ratcliff J.: Position based dynamics. In *Workshop in Virtual Reality Interactions and Physical Simulations (VRIPHYS 2006)* (Madrid, Nov. 2006). 2, 3

[MtV05] Magnenat-thalmann N., Volino P.: From early draping to haute couture models: 20 years of research. *The Visual Computer 21* (2005), 506–519. 1

[Pro95] Provot X.: Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *In Graphics Interface* (1995), Davis W. A., Prusinkiewicz P., (Eds.), Canadian Human-Computer Communications Society, pp. 147–154. 1

[SG02] Schenk O., Gärtner K.: Two-level dynamic scheduling in pardiso: improved scalability on shared memory multiprocessing systems. *Parallel Comput. 28*, 2 (2002), 187–197. 4

[SG04] Schenk O., Gärtner K.: Solving unsymmetric sparse systems of linear equations with pardiso. *Future Gener. Comput. Syst. 20*, 3 (2004), 475–487. 4

[TPBF87] Terzopoulos D., Platt J., Barr A., Fleischer K.: Elastically deformable models. In *Proceedings of Computer graphics and interactive techniques* (New York, NY, USA, 1987), SIGGRAPH '87, ACM, pp. 205–214. 1

[TPS09] Thomaszewski B., Pabst S., Strasser W.: Continuum-based strain limiting. *Comput. Graph. Forum 28*, 2 (2009), 569–576. 2

[WBH*07] Wardetzky M., Bergou M., Harmon D., Zorin D., Grinspun E.: Discrete quadratic curvature energies. *Comput. Aided Geom. Des. 24* (November 2007), 499–518. 5