Animation of Surfaces with Applications to Cloth Modelling

Dissertation
der Fakultät für Informatik
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

vorgelegt von
Dipl.-Math. Olaf Etzmuß
aus Gehrden

Tübingen
2002

Tag der mündlichen Qualifikation:	6. Februar 2002
Dekan:	Prof. Dr. Andreas Zell
1. Berichterstatter:	Prof. Dr. Wolfgang Straßer
2. Berichterstatter:	Prof. Dr. Andreas Weber (Universität Bonn)

Zusammenfassung

Die Anforderungen an den Realismus virtueller Szenen in der Computergraphik haben sich stetig erhöht. Daher reichen geometrische Modelle in einigen Anwendungsgebieten nicht mehr aus. Textilsimulation ist eine dieser Anwendungen, in denen die Modelle durch ihr physikalisches Verhalten beschrieben werden müssen.

In der Computeranimation werden vielfach Partikelsysteme eingesetzt, um Flächen zu animieren. In dieser Dissertation werden Partikelsysteme weiterentwickelt, sodaß sie kontinuierliche Materialien modellieren und Modelle der Kontinuumsmechanik approximieren. Mit solchen Partikelsystemen können dann Materialien wie z.B. Textilien simuliert werden, ohne daß die Materialeigenschaften von der Diskretisierung abhängig sind.

Das Partikelsystem beschreibt das dynamische System durch gewöhnliche Differentialgleichungen, und die eigentliche Simulation besteht aus dem Lösen dieser Gleichungen. Animationen erfordern, daß diese Lösung schnell berechnet wird. Geeignete implizite Verfahren garantieren zwar Stabilität, aber machen die Berechnung eines Zeitschritts sehr aufwendig. Daher werden Verfahren vorgestellt, die die Kosten eines solchen Zeitschritts wesentlich reduzieren.

Ein weiteres Problem in der Animation ist die Behandlung von Kollisionen, und es wird gezeigt, daß Kollisionen sich über einen Mechanismus, der Zwangsbedingungen benutzt, realisieren lassen. Die Kollisionsantwort erfolgt so stabil und schnell, und auch Ruhekontakte lassen sich modellieren.

Weiterhin wird ein Verfahren vorgestellt, das in Regionen, in denen Kollisionen auftreten, die Diskretisierung adaptiv verfeinert. Dies ermöglicht es, realistische Resultate sogar mit sehr groben Auflösungen zu erzielen.

Die in dieser Arbeit entwickelten Konzepte wurden in einem System zur Kleidersimulation realisiert. Am Schluß dieser Arbeit wird dessen Softwarearchitektur beschrieben und Resultate vorgestellt.

Abstract

The realism requirements of virtual scenes in computer graphics have been increasing permanently. Therefore, geometric models do no longer suffice in some application areas. Textile simulation is one of these applications in which the models must be described by their physical behaviour.

In computer animation, particle systems are frequently employed in order to animate surfaces. In this thesis particle systems are advanced so that they model continuous materials and approximate models of continuum mechanics. With such particle systems materials like textiles can be simulated without the material properties depending on the discretization.

The particle system model describes the dynamics system by ordinary differential equations, and the actual simulation consists of solving these equations in time. Animations require that the solution be computed fast. Suitable implicit methods guarantee stability but make the computation of one time step very expensive. Hence, methods that reduce the costs of such time steps significantly are presented.

The treatment of collisions is another problem in animation, and it will be shown that collisions can be implemented by a constraint mechanism. The collision response takes place stably and fast. Also resting contacts can be modelled.

Furthermore, a method that refines the discretization adaptively in regions in which collisions occur is developed. This allows to achieve realistic results even with very coarse resolutions.

The concepts developed in this work have been implemented in a system for cloth modelling. At the end of this work the software architecture of this implementation is described, and results are presented.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Deformable Objects and Cloth Modelling

Physically based animation has been established as one of the major fields of computer graphics. Animation has developed efficient techniques for fluid dynamics, rigid-body animation, and modelling of deformable objects. Among deformable objects, highly flexible surfaces are particularly interesting because they have important applications. Modelling of virtual textiles and clothes is one of the most prominent applications, and the specific properties of textiles will guide the development of solutions in this thesis.

Commercially, virtual clothes are particularly interesting within two scenarios. In the first one, a virtual movie or environment is generated, and the virtual actors therein have to be dressed with virtual clothes. Simple animations imitate clothes by two-dimensional textures mapped onto the virtual characters. However, there is demand for real three-dimensional objects that give the impression of real clothes. Particularly in virtual reality environments, real-time capability of the animations is crucial, whereas there are only moderate requirements on accuracy because only a realistic impression of these clothes is relevant. Applications can be found in the generation of computer animated movies or commercials.

In the second scenario, clothes are bought on the Internet, and the customer is provided with a preview in which he sees a three-dimensional image of himself and the clothes to be able to decide on fit and appearance of the clothes. In this application more accurate physical models are necessary because a specific material has to be modelled, and the customer must be able to distinguish among materials by their physical properties and behaviour.

Without a physically based model, deformable objects in computer generated scenes have to be modelled manually in a tedious process. In animated sequences this problem becomes even more severe and a geometric approach infeasible. Physically-based models introduce the laws of mechanics into artificial scenes such that virtual curtains, tablecloths, and dresses can be computed automatically.

Clearly, the area of research in deformable models spans out from computer science to physics and mathematics as in an animation system problems from each of these fields have to be solved. Figure 1.1 gives an overview over all these tasks.

The first task is to choose or develop a physical model to represent the properties of a material as accurately as possible. Physically-based animation means that the objects move according to forces exerted on them. It can be distinguished between external and internal, elastic and viscous forces. External forces, for instance, are gravity, wind, collision impact, and friction, internal forces are elastic forces that restore the object's shape and viscous forces that reduce the internal movement of the object and dissipate its energy. The model is parametrised by material constants that specify the type of material.

Naturally a deforming object is described by a partial differential equation (PDE), the solution of which must be computed with methods of numerical analysis. The numerical solution involves the discretisation of the continuous surface by a discrete, polygonal mesh, which is also a common object description in computer graphics. Then the simulation is run by solving an initial value problem for an ordinary differential equation (ODE).

Interaction plays a major role in computer animation, and collisions with other objects are the most important kind of interaction. Hence, in animation there are no predefined boundary conditions, but the boundary conditions are imposed by collisions with obstacles in the scene and must be handled on the fly. In each time step collisions have to be detected and treated appropriately afterwards. Thus, collisions lead to two major problems: First, collisions have to be detected. Collision detection is a classical problem in computer graphics. However, most algorithms are designed to deal with rigid objects. Deformable objects require specialised algorithms that can handle a large number of colliding triangles or tetrahedrons, as the continuous objects are discretized by such.

The second problem of collisions is the response to them and there is no obvious way to do this physically and mathematically correctly and fast. In computer animation solutions are developed that approximate the collisions so that the simulation is not slowed down.

Finally, the meshes computed in each time step have to be rendered in a way that the material properties are not only obvious in the dynamics but also in their light reflectance properties.

1.2 Overview and Contributions

This thesis makes several contributions to the research in physically-based animation of surfaces. New results are presented for the physical, numerical, and collision model.

For modelling deformable objects, continuum mechanics provides a solid base and the models are well known. However, there have been several reasons not to

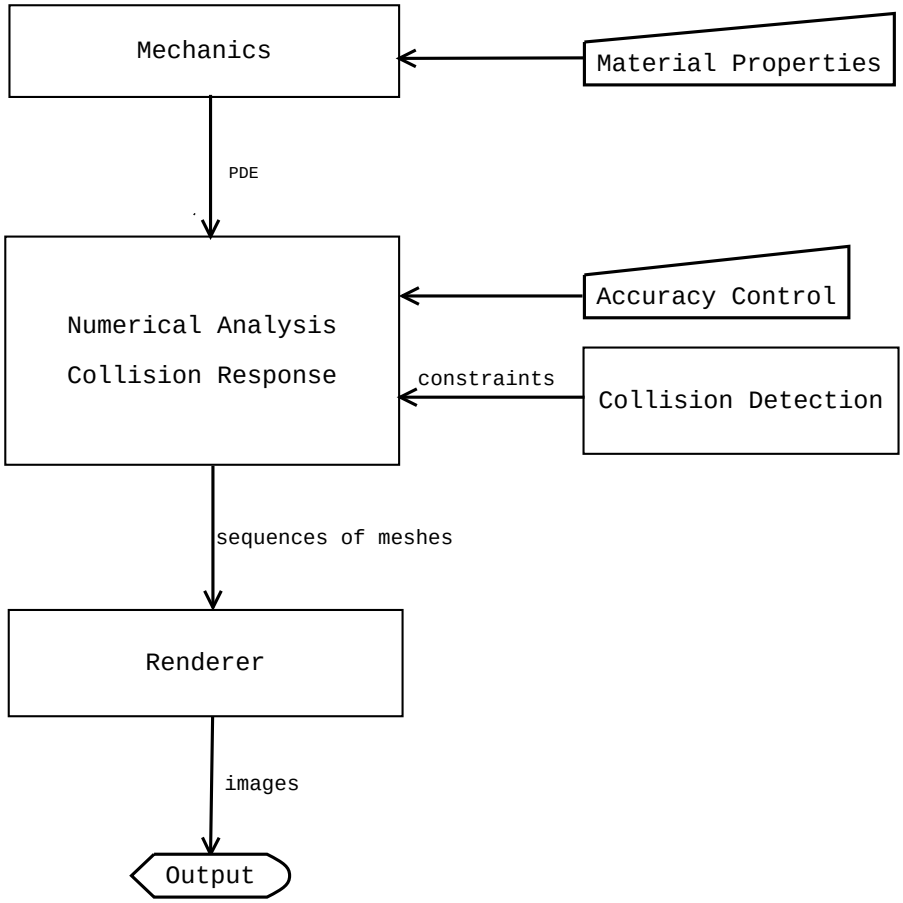


Figure 1.1: Textile modelling overview

use continuum mechanics. First, in computer animation researchers often were interested in very simple and intuitive models that were easy and fast to animate, whereas they were not interested in preserving material properties accurately. Second, linear elasticity is not always applicable in animation. Highly flexible objects need to be computed by nonlinear models, which are computationally too expensive in animation. Finally, materials like textiles were not considered continuous materials but assemblies or structures of several threads. Other researchers [BHW94, EWS96] have used particle systems as an intuitive model for these interconnected threads. They analysed the interaction between different threads in woven or knitted yarn. These structures were modelled by particle systems with the objective to map measured material properties onto the model. Unfortunately, the model cannot have so much detail to model each thread of the textile. Therefore, we prefer to view textiles as continuous materials.

In this work the notions of particle systems and continuum mechanics will be

merged to combine their respective advantages. Particle systems are employed because they allow fast simulations, and we will show that they are capable of modelling continuous deformable objects. Chapter 2 comprises an entirely new derivation of a particle systems from continuum mechanics and illuminates how particle systems work. For rectilinear meshes, we will even be able to prove and verify that such a particle system is a discretization of a partly linear continuum model.

The benefits of this new particle system are obvious. Since properties of previous particle systems remained partly unclear, the main problem in particle system modelling was to find a sound description that allowed to model material properties with some material constants independent of the resolution of the particle system. These problems are solved by the continuum mechanics based particle systems.

The physical model derived in chapter 2 is stated as an ordinary differential equation in time. The actual simulation is run by solving the ODE numerically. It turns out that a fast numerical solution requires sophisticated numerical methods. Very stiff forces, which are necessary to model textiles, present a major hurdle on the way to interactive applications. For stiff forces lead to stiff differential equations that require implicit numerical solvers. Although employing implicit solvers has become standard in cloth animation since the work by Baraff and Witkin [BW98], the efficiency problems have remained hard because this class of solvers requires a nonlinear system of equations to be solved at each time step. Hence, some specialised methods are necessary to allow for both fast and accurate solutions for the ODE of the particle system.

A thorough analysis of the system of ODEs leads us to methods that are customised to the specific properties of the derived particle system. By choosing the appropriate numerical methods, a superior performance in the execution of a time step of the solver is reached while the time step can be arbitrarily large. The full framework for solving the ODEs will be developed in chapter 3 and allows to control accuracy and speed of the simulation.

Discontinuities imposed by collisions are a major problem in physically based animation because less sophisticated collision response techniques lead either to instabilities of the simulation or require a reduction of the step size and make the computations very time consuming. Therefore, the collision response has to be integrated into the numerical integration. The collision response scheme that we will introduce in chapter 4 imposes the collision conditions by filtering and reaction constraints. This ensures the stability during the collisions, and the time step does not have to be reduced to guarantee valid results. Moreover, resting contacts are modelled naturally without increased computational work.

In colliding surface regions the resolution of the surface should be increased to model these collisions realistically. Therefore, a collision adaptive approach has been developed, which will be presented in chapter 5. It is based on a collision response for colliding particles. After this response, in the colliding areas the mesh is refined by projecting one object onto the other. The resulting mesh models the collision area accurately. This permits that very coarse meshes collide with rigid

objects and give valid results, even if the geometry of the rigid objects is complex.

Finally, the efficiency of the theoretical concepts of the animation of deformable objects depends on an appropriate software framework as a base for a flexible and efficient implementation of the animation. In chapter 6 we describe the system architecture that the software developed for this thesis is based on. A special focus has been on the capability of extending to future developments so that the software is not restricted to the models described in this thesis. Also, applications of the software system demonstrating its capabilities are shown.

1.3 Notation

In this thesis we will use the following notation for mathematical expressions:

$\langle \cdot, \cdot \rangle$	the scalar product of two vectors
s_u	first derivative of s with respect to u
s_{uv}	second mixed derivative of s with respect to u and v
$\triangle s$	Laplacian of s
grad	gradient of a vector
div	divergence of a vector or tensor
v^0	normalised vector v

Chapter 2

Two-dimensional Deformable Models

In their book on cloth modelling Donald House and David Breen state that *"Cloth is a mechanism, not a continuous material"* [BH00, p. 55]. This is certainly true. However, in computer animation we cannot afford to model each detail of this mechanism, i.e. each single thread and structure. The only way to avoid this is to represent a patch of textile as a continuous material, which allows us to use low resolution models without losing basic material properties.

Nevertheless, some discrete systems that have been developed in computer animation for the animation of clothes and other surfaces have the advantage that they allow very fast simulations. In particular, particle systems have been successfully used for rapid animations.

This chapter will provide a new derivation of particle systems that allows to model continuous objects. Also, as will be shown in the following chapter, this particle system permits very fast simulations.

We will start this chapter by giving an overview over the existing continuous and discrete models that have been suggested in literature for textiles and other surfaces. In section 2.2 we will show how a chain of masses and springs can be obtained by a discretization of the linear wave equation generalised to curves in 3D. This will be extended to surfaces in section 2.3, which develops the elastic model of a particle system for clothes. After the derivation of all elastic forces, section 2.4 gives an account of the viscous forces to model energy dissipation. Detailed implementations are given in section 2.5. Completing the survey and derivation of all forces involved, section 2.6 specifies the external forces acting on the surfaces. This is followed by a discussion about grid generation in section 2.7, which proposes how the particle system derived for rectangular meshes can be applied to general shapes. In section 2.8 we present experiments in which the implemented particle system is compared to analytical continuum mechanics solutions and the predicted approximation properties are verified. This section is followed by the final conclusions.

2.1 A Classification of Deformable Models

In literature many models for the animation of deformable objects have been proposed. In this section we will classify these models. They are either based on continuum mechanics or on some discrete mechanism. The models comprise elastic forces, which preserve the deformation energy, viscous forces, which dissipate the deformation energy, plastic forces, which change the material properties irreversibly when a certain strain is exceeded, and fracture. Most deformable object animations focus on elasticity, whereas plasticity and fracture are widely neglected or treated by specialised algorithms [OH99].

2.1.1 Continuum Models

Continuum mechanics is the standard theory to describe and model deformable objects, and the following elaborations are based on several text books [Bra97, Cia92, LL89, SK95].

The basic quantities of continuum mechanics are *strain*, which is a dimensionless deformation noted by ϵ, and *stress*, which is a force per length for surfaces or per area for volumes and is denoted by σ. These quantities are coupled by Hooke's law:

$$\sigma = C\epsilon$$

In the case of a one-dimensional spring, all these entities are scalars and C is the spring constant. The strain of this spring is its elongation per length, while the stress is the spring force. In the case of surfaces or volumes, these entities are tensors, and C comprises all material properties.

Surfaces are more complicated than a one-dimensional spring, and the description of strain is more involved. Textiles can be described as regular surfaces (in the sense of differential geometry, e.g. [DoC76]). The deformation of a regular surface embedded in $\mathbb{R}^3$ is described by Green's strain tensor with respect to a certain undeformed state. In this equilibrium state, denoted by r, the object is not deformed, and the elastic energy is zero. Let r be parametrised over a domain $U \times V$. Under forces the rest state deforms to a state $s(u, v)$. The displacement is a mapping d defined by $d(u, v) = s(u, v) - r(u, v)$ as depicted in figure 2.1.

The difference of the first fundamental forms I_s and I_r of the current state and the equilibrium state I_r of the object describes the in-plane strain and defines a nonlinear strain tensor [Kli89]

$$\tilde{G} = \frac{1}{2}(I_s - I_r) = \frac{1}{2}\begin{pmatrix} \langle s_u, s_u\rangle & \langle s_u, s_v\rangle \\ \langle s_u, s_v\rangle & \langle s_v, s_v\rangle \end{pmatrix} - \frac{1}{2}\begin{pmatrix} \langle r_u, r_u\rangle & \langle r_u, r_v\rangle \\ \langle r_u, r_v\rangle & \langle r_v, r_v\rangle \end{pmatrix}.$$

For planar surfaces, the deformation is defined uniquely by the difference of the metrics of these states. For arbitrary surfaces in 3D curvature has to be taken into account as well.

Commonly, the rest state is assumed to be the identity mapping. This yields Green's strain tensor

$$G = \frac{1}{2} \begin{pmatrix} \langle s_u, s_u \rangle - 1 & \langle s_u, s_v \rangle \\ \langle s_u, s_v \rangle & \langle s_v, s_v \rangle - 1 \end{pmatrix}.$$

Green's tensor, unfortunately, is nonlinear and yields fourth order terms in the

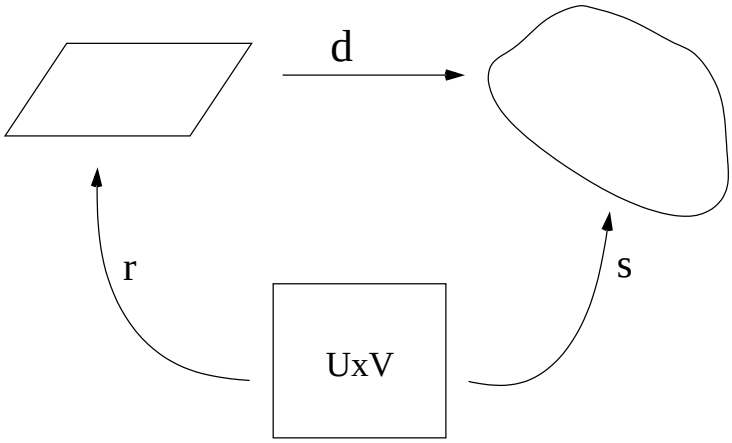


Figure 2.1: The reference configuration: the rest state is parametrised by a mapping r on a space $U \times V$. By deformation d it transforms into the deformed (strained) configuration, which is parametrised by the mapping s.

energy formulation, and these are computationally very costly and lead to various numerical problems. The implementations of such approaches have turned out to be too slow for interactive animation.

Linear elasticity theory suits a fast simulation much better. It makes use of the linear approximation of Green's tensor, called Cauchy's strain tensor. It is obtained by neglecting terms of order higher than one in the displacement d in the components of Green's tensor, here for two dimensions:

$$\begin{aligned} \langle s_u, s_u \rangle - 1 &= \langle e_1 + d_u, e_1 + d_u \rangle - 1 = 2\langle d_u, e_1 \rangle + O(d^2) \\ \langle s_v, s_v \rangle - 1 &= \langle e_2 + d_v, e_2 + d_v \rangle - 1 = 2\langle d_v, e_2 \rangle + O(d^2) \\ \langle s_u, s_v \rangle &= \langle e_1 + d_u, e_2 + d_v \rangle \\ &= \langle d_u, e_2 \rangle + \langle d_v, e_1 \rangle + O(d^2), \end{aligned} \tag{2.1}$$

where (e_1, e_2) is the Cartesian basis of $\mathbb{R}^2$. Thus, Cauchy's tensor can be written as

$$\epsilon = \begin{pmatrix} d_u^1 & \frac{1}{2}(d_v^1 + d_u^2) \\ \frac{1}{2}(d_v^1 + d_u^2) & d_v^2 \end{pmatrix},$$

where the superscripts denote the vector components. Linear elasticity is based on this linearised strain tensor and yields much simpler formulations. In particular, it results in linear partial differential equations. These linear equations are widely used in engineering and lend themselves to finite element formulations very easily.

The strain tensor of linear elasticity, as we have seen, is derived from Green's strain tensor by linearisation in the deformations, i.e. the displacement d (figure 2.1) is assumed to be small, and all terms of order two or higher in d are neglected in the strain tensor. For this reason this linear theory is not appropriate for highly flexible objects. It only applies to small displacements. It is therefore not invariant under rotations and leads to unphysical behaviour if the object or a part of it is rotated. As animated surfaces can bend strongly, the displacements become very large, although the deformations are only small.

In the following section we will see that a linearisation similar to the above that led to Cauchy's strain tensor can be used to derive a particle system for rapid simulations.

So far, we only have dealt with the description of strain. In linear elasticity, also the relation between the stress tensor σ and strain ϵ is assumed to be linear, and the dependence is given by the elastic tensor C. This is formulated by Hooke's law:

$$\sigma_{ij} = C_{ijkl}\, \epsilon_{kl} \tag{2.2}$$

C is a symmetric rank-4 tensor containing the material properties. Here, symmetry means $C_{ijkl} = C_{klij}$ as well as $C_{ijkl} = C_{jikl}$. If we use Green's tensor instead of Cauchy's tensor and a linear strain-stress relation is assumed as well, this model is called a Vernant-Kirchhoff material. It is applicable to small strain and is invariant under rotations. Unfortunately, Green's tensor is too costly for cloth animation applications with implicit time integration.

Finally, the equation of motion of a continuous elastic material is

$$\rho \frac{\partial^2 s}{\partial t^2} - \text{div}\, \sigma = f, \tag{2.3}$$

where ρ is the mass density and the divergence of the stress σ yields the force density due to the interior energy of the elastic object. f denotes an external force density (e.g. the gravity force density ρg). Equ. (2.3) is a partial differential equation (PDE) that has to be solved over the parameter domain and time. A standard procedure is to semidiscretize the system in space with finite differences or finite elements. This reduces the PDE to an ordinary differential equation (ODE) that can be solved by any suitable integration method.

Proposed Models

In spite of the described shortcomings of linear elasticity, for some applications in computer animation linear elasticity is used. Cotin et al. [CDA99], for instance,

adopt the finite element method for linear elasticity to derive a tensor-mass model for virtual surgery. In this model the global stiffness matrix is made local, such that topological changes due to cutting can be handled. Physically, this model is equivalent to linear elasticity.

As early as in 1988 Terzopoulos and Fleischer [TF88] use a nonlinear continuum formulation in order to animate surfaces. In their primal model they define the elastic energy by

$$\int \sum_{i,j=1}^{2} w_{ij}^{1}(I_s(u,v) - I_r(u,v))_{ij}^{2} dudv.$$

That is, the energy is computed by integrating over the sum of all squared components of the strain tensor. The weights w_{ij}^{1} are used to vary the material properties and correspond to elastic constants. This nonlinear model is then semi-discretized by finite differences and afterwards solved in time. As the first fundamental form is not sufficient for determining the shape of a surface, Terzopoulos and Fleischer give the second fundamental forms a similar treatment and add

$$\int \sum_{i,j=1}^{2} w_{ij}^{2}(II_s(u,v) - II_r(u,v))_{ij}^{2} dudv$$

to the elastic energy.

In this early work already advanced effects like plasticity and fracture are modelled. For instance, it incorporates slip units, the ideal plastic unit, which changes its elastic constants when a certain threshold of strain is exceeded. Fracture is modelled by setting weights to zero, when a certain stress is exceeded.

Eischen et al. [EDC96] model clothes accurately by nonlinear shell theory. In order to solve the resulting nonlinear PDE's, the system is discretized by finite elements, and Newton's method is used to compute an equilibrium solution. In this process several numerical problems like forking points occur. Obviously, this method has not been developed for animation, but it serves well as a reference solution that animation systems can be compared with.

2.1.2 Discrete Models

Since the continuous models from classical physics are either not suitable or computationally expensive, various other approaches have been proposed to model deformable objects in computer animation.

All these approaches start out with a grid or mesh of particles. Then, forces on each particle are computed depending on its position and the positions of a set of particles within its topological neighbourhood. When the function F computing

the forces has been determined, Newton's equation of motion governs the movement of the particles. The trajectory of each particle with mass m_i at position x_i is computed by

$$F(x) = m_i \cdot \frac{d^2 x_i}{dt^2}. \tag{2.4}$$

Here x denotes the vector containing all particle positions. Note that, since particle systems already represent a discretization in space, only a system of ordinary differential equations has to be solved. The systems presented in literature differ by their methods of computing the forces.

Comparing the continuous equ. (2.3) to Newton's law (2.4) shows that all continuous quantities are simply replaced with their corresponding discrete counterparts and the term F in Newton's law comprises the internal and external forces.

Uncoupled particle systems

First introduced into computer graphics by Reeves [Ree83] to model fuzzy objects, a particle system is a number of particles with some forces acting on them. Desbrun and Cani-Gascuel [DG96] employ this idea to model very inelastic objects. They adopt the notion of *Smoothed Particle Hydrodynamics* (SPH), which has been developed for computational fluid mechanics, to computer animation. In SPH a smoothing kernel gives each particle a spatial mass distribution.

Because there is no topology defining the structure of the object, the material can undergo arbitrarily large deformations. These uncoupled systems have the disadvantage that they are computationally more expensive than coupled systems, because each particle can interact with any other particle. In practice, search structures are used to locate nearby particles (Desbrun and Cani-Gascuel use voxel grids). Nevertheless, the benefit of sparse matrices, which are due to the fact that a particle only interacts with a small set of neighbours, is lost (cf. chapter 3). For surfaces, the two-dimensional structure cannot be preserved by this approach.

Mass-Spring Systems

In mass-spring systems, particle interaction is solely modelled by linear springs. Previously they were used to model deformable solids [BMG99, BC00, KCM00], for facial animation [KHS01], and textiles [Pro95, HPH96]. A first theoretical analysis of triangular mass-spring systems can be found in an article by van Gelder [VG98], who shows that it is not possible to model an isotropic material by a triangular spring mesh.

Provot [Pro95] proposes a mass-spring system for textiles and uses a rectangular mesh in which the particles are connected by structural springs to counteract tension, diagonal springs for shearing, and interleaving springs for bending as

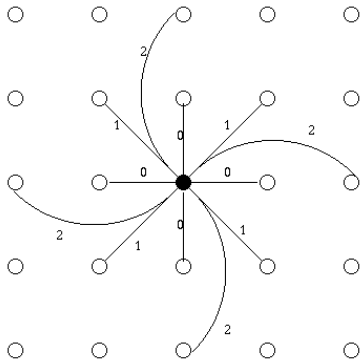


Figure 2.2: Provot's mass-spring system with (0) tension springs, (1) shear springs, and (2) bending springs, *Picture by Saraswati Venkatram*

shown in figure 2.2. Obviously, there is a strong interdependence between the different kinds of springs leading to nonlinear, uncontrolled effects. The diagonal shear springs, for instance, also lead to additional tension and transversal contraction.

Provot's mass-spring system is frequently used when a very simple model is sufficient and the conservation of material properties is not required.

Coupled Particle Systems

Breen et al. [BHW94] were the first to employ particle systems for cloth modelling. These systems extend the concept of mass-spring systems which only incorporate extensional linear spring forces by arbitrary forces that act on a certain set of adjacent particles. A particle system discretizes the object by a set of particles, and there are permanent links that connect each particle to its neighbours. Thus, the particle system has a constant topology that defines the structure of the deformable object. Hence, particle systems are represented by a mesh with this topology and the particles as its vertices. There are interior elastic energies due to tension, bending, and shearing in the object. They are computed from the distances between linked particles (lengths of the edges of the mesh) and the angles between the edges adjacent to a particle.

Breen et al. [BHW94] use potential functions for tension, bend, and shear energy to compute an equilibrium state for the surface. Eberhardt et al. [EWS96] employ similar energy functions and extend the work on particle systems for cloth animation. Their tension energy is evaluated for each particle and depends on the four neighbours of that particle in a rectangular mesh. The tension energy of a

particle at position x_0 is

$$E = \sum_{i=1}^{4} \begin{cases} \frac{1}{2} C_{t_1,i} (\|x_0 - x_i\| - l_i - h_{t_1,i})^3 & \text{if} \quad \|x_0 - x_i\| \geq l_i \\ \frac{1}{2} C_{t_2,i} (\|x_0 - x_i\| - l_i - h_{t_2,i})^5 & \text{if} \quad \|x_0 - x_i\| \leq l_i \end{cases}, \tag{2.5}$$

where l_i are the rest lengths between particles and $C_{t,i}$ and $h_{t,i}$ are material parameters. The energy is computed from a strain $(\|x_0 - x_i\| - l_i)$, and the strain-stress relation is modelled piecewise cubic or quintic (cf. section 2.1.3). If we introduce a linear strain-stress relationship by replacing the exponents with 2 and set $h_{t,i} := 0$, we get linear spring energies. The shear energy is modelled as

$$E_s = \sum_{i=1}^{4} \frac{1}{2} C_s (\phi_i - \frac{\pi}{2} - h_{s,i})^2 \tag{2.6}$$

and the bend energy as

$$E_b = \sum_{i=1}^{2} \frac{1}{2} C_b (\psi_i - \pi - h_{b,i})^2. \tag{2.7}$$

Here C_s, C_b and $h_{s,i}, h_{b,i}$ are the material constants. These energies implement hinges functioning like springs that linearly depend on the shear angle ϕ and the bend angle ψ, respectively. These are the angles formed by the incident edges as depicted in figure 2.3.

Furthermore, kinetic energy is included to compute dynamic animations of clothes by means of the Lagrange formalism. For the computation of the Lagrange function and optimisation the computer algebra system *Maple* is used [EW97].

Eischen and Bigliani [EB00] take a different approach in constructing their particle system. Instead of modelling mechanisms in a textile, they try to match their particle system with a continuous model. All forces are constructed by comparison with a linear continuous model. In particular, they use the material parameters from elasticity theory to allow comparisons with a continuous reference model. After manually adjusting tuning parameters in the experiments, good results for small deformations are obtained when the particle system is compared with the nonlinear shell theory model. Nevertheless, without a suitable discretization method the particle system cannot be expected to match the continuous model without manual tuning. Later in this chapter we will show how a spatial semi-discretization allows us to match a particle system with a continuum model directly.

Baraff and Witkin [BW98] present a fast model for the animation of clothes. They give a physical formulation in terms of constraints rather than forces or energies. A constraint is a vectorial condition $C(x)$ that is zero in a state without energy. From a constraint C an energy

$$E = \frac{1}{2} C^2(x)$$

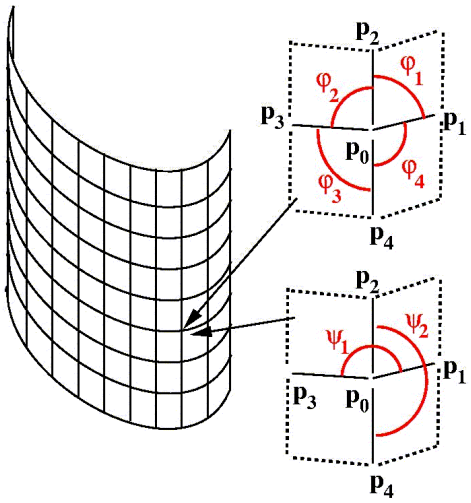


Figure 2.3: Shear and bend energy in a particle system. *Picture by Eberhardt and Weber*

and a force

$$F_{\text{elast}} = -\text{grad}\, E = -\frac{\partial C}{\partial x} \cdot C$$

can be derived. The stress-strain relation is modelled to be linear. The material constants are not obtained from experimental data. Furthermore, explicit viscous forces are modelled:

$$F_{\text{diss}} = \frac{\partial C}{\partial x} \cdot \frac{dC}{dt}$$

Baraff and Witkin use triangle meshes to allow a simple and flexible modelling of clothes. The constraints on these meshes are formulated in terms of partial derivatives along the edges. Let $s(u, v)$ be the deformable surface. The parametrisation is given by the planar, undeformed state of the triangle, in which each vertex is described by the planar coordinates (u_i, v_i). The two edges of this triangle sharing a vertex are considered as partial derivatives s_u and s_v. The partial derivatives for a deformed triangle with vertices at x_0, x_1, x_2 are computed from

$$\begin{pmatrix} x_1 - x_0 & x_2 - x_0 \end{pmatrix} = \begin{pmatrix} s_u & s_v \end{pmatrix} \begin{pmatrix} u_1 - u_0 & u_2 - u_0 \\ v_1 - v_0 & v_2 - v_0 \end{pmatrix}.$$

Although the authors do not state it, these are finite difference approximations of the directional derivatives along the edges. They are used to define constraints for stretch, bend, and shear forces. The stretch constraints are given for each triangle by

$$C(x) = A \begin{pmatrix} \|s_u\| - 1 \\ \|s_v\| - 1 \end{pmatrix}.$$

The triangle area A scales the stretch forces. These constraints lead to linear spring forces as we will see later in this chapter. The shear constraints are given by the scalar product of a pair of edges

$$C(x) = \langle s_u, s_v \rangle.$$

This is used as a small angle approximation of the shear angle $\angle(s_u, s_v)$. The bend constraint simply measures the angle of the normals of each pair of adjacent faces. As this angle can be related to the surface curvature, the bend forces are curvature dependent.

When transferred to quadrilateral meshes, the forces derived from these constraints show strong similarity to those of the previous particle systems. Although finite differences are computed, physical quantities cannot be assigned to the model, as they depend on the mesh topology. The number of constraints acting on a particle depends on the number of incident edges. Furthermore, the direction of the derivatives is arbitrary as it depends on the direction of the incident edges. Since no method is suggested to compensate for the arbitrary triangle topology, stretch, shear, and transverse contraction are mixed in an uncontrolled way, as will become clear in the remainder of this chapter. Nevertheless, the proposed particle system produces convincing cloth animations and serves well for this purpose.

Extended Particle Systems

The concept of particle systems can be generalised further. In a triangular mesh the deformation of each face is uniquely determined by the elongation of its edges. Forces acting on each of its particles can be formulated depending only on these (vectorial) elongations. This results in a particle system in which the forces on one particle do not only depend on adjacent edges but also on the opposite edges. The coefficients of these dependencies are the material constants. Volino [VMT97] describes such a model for textiles to reduce the transversal contraction in simple mass-spring systems.

Dias et al. [DGR00] use the same idea in the context of linear elasticity. The strain of a deformed triangle face is measured along its edges. From the edge elongations, the strains in the principal strain directions are computed. Then, Hooke's law (2.2) is applied to compute the principal stresses. The stresses on each edge are applied as forces to the triangle vertices such that a particle system is obtained. However, this work fails to describe a proper method for spatial discretization.

Considering that a triangular element can apply arbitrary forces onto each of its nodes, it is not different from a finite element. Gross [GB01] exploits this to compute the material coefficients in this generalised particle system. He fits a triangle of the mesh with a triangle of a finite element discretization that has been obtained from a linear continuum mechanics equation. This way, arbitrary linear elastic materials can be modelled.

Problems in Particle System Modelling

Until now there have been some open problems in particle system modelling. First, since all of them aim to describe real world materials, naturally the question arises what the fundamental differences are and how elastic energies in them have to be modelled to describe reasonably an elastic surface. Second, how do these systems scale when the discretization is changed? The discrete systems widely fail to indicate how the forces are scaled when switching to a different resolution. The properties of the modelled object should remain invariant under grid refinement.

When looking at the grids of the discrete systems and the edges between the particles, it resembles a space discretization of a PDE. The edges can be represented by difference vectors which appear to be finite difference operators, and the discrete systems suggest a comparison with finite difference discretization of a PDE.

These problems motivate the following elaborations. The objective will be to establish a link between particle systems and continuum mechanics. We will derive a particle system from a continuous formulation by discretization by means of finite differences to achieve a sound transition from one level of the discretization to another and to approximate continuum mechanics models.

2.1.3 Textile Material Properties

Unfortunately, textile materials behave highly nonlinearly, and approximations to these nonlinear properties have been an issue in computer animation. Another characteristic property of textiles is orthotropy. Orthotropic materials possess two orthogonal symmetry axes in continuum mechanics. This reduces the number of free elastic material constants (entries in the elastic tensor) to four for in-plane deformations. The axes or directions are called weft and warp directions for woven textiles. The textiles show very different physical behaviour in these directions, and this characterises the materials. Therefore, material measurements are carried out for the two directions independently.

Breen et. al [BHW94] make a first attempt to base their models on experimental data in order to map specific material properties of textiles onto their particle system. There is a standardised system of experiments called Kawabata system [Kaw80] for the measurement of textile material properties. The modelling of tension, shear, and bend energy is based on this Kawabata system, as each of these energies is measured by one specific experiment in the Kawabata standard. These experiments are carried out for the weft and warp direction.

The measured stress-strain relation is highly nonlinear in textiles and polynomials are used to approximate the Kawabata measurements. However, hysteresis effects due to energy dissipation, which are strongly visible in textiles, cannot be modelled. Hence, no energy is dissipated in that model, i.e. the model is purely elastic.

Eberhardt, Weber, and Straßer [EWS96, EW97, EW99] extend the cloth model. In particular, they focus on the integration of the observed nonlinear stress-strain relation in textiles and the plastic behaviour which is observed in the Kawabata experiments.

They approximate the measured stress-strain curves by piecewise affine functions. For each curve they store a list of two-parameter pairs $C_{\cdot,i}, h_{\cdot,i}$ as they already appear in the energy (2.5). One parameter pair describes an affine function by its slope $C_{\cdot,i}$ and offset $h_{\cdot,i}$. In the simulation one pair, i.e. one affine function, is chosen according to the current strain. Hysteresis effects then can be incorporated by not only storing one curve for each measurement but several curves each describing one branch of the measured hysteresis. Depending on the maximum and minimum strain at that node or edge, the system chooses one of the curves and can switch from one branch of the hysteresis to another. Hence, this approach models plastic effects, because by the dependence on the maximum and minimum strain in the material history the change of the parameters is not reversible.

Although the Kawabata system (a readable description can be found in an article by Greuel et al. [GWZ91]) is the only standard to measure textile properties, this standard has some shortcomings when employed for simulation. Since these measurements were developed for quality control and ensurance in the textile industry rather than for simulations, sufficient information cannot be extracted from the standard experiments. In order to simulate a material, this must be described by a complete and unique set of parameters, which is not provided by the Kawabata system. For instance, the system does not comprise an experiment to measure the transverse contraction, although such deformation is apparent when a textile is stretched. Furthermore, the viscoelastic properties cannot be fully reconstructed. For in all experiments the probe is deformed at only one constant velocity. In general, the deformation depends on the deformation velocity, and the general relation cannot be reconstructed. A repetition of the experiments at different deformation velocities would allow to approximate the frequency dependent stress-strain relation.

2.2 Elastic curves

Although our focus is on the animation of surfaces, curves are of interest because they allow to study how a chain of masses and springs, the simplest discrete model, behaves.

In this section we will show that a one-dimensional particle system, which is a chain of masses connected by linear springs, represents a discretization of linear elasticity extended to curves. Here we are going to deal with the propagation of a wave along the curve only and assume that the wave is constrained to that curve. Forces that counteract bending and torsion are not considered here but can be added

by discretizing curvature and torsion.

In linear elasticity the wave propagation along a straight line is given by the wave equation

$$k \triangle x = \rho \frac{\partial^2 x}{\partial t^2}, \tag{2.8}$$

where k is the longitudinal stiffness and ρ the mass density. Note that this coincides with the Lamé equation for the one-dimensional case. If we discretize this differential equation by means of finite differences, we obtain a mass-spring system (see Kass [Kas95]).

In the case of a curved string embedded in a three-dimensional space, the situation is more complicated. To obtain a continuum equation for the mass-spring system, we consider a chain of springs with equal length l and let the lengths approach zero.

The curve will be denoted as $x(u)$ with sampled points $x_i = x(u_i)$, and $\Delta_i x$ is the backward difference operator applied to the curve defined by $\Delta_i x = x_i - x_{i-1}$. For the following derivations we need some curvature properties, which are computed from the arc-length parametrisation of x. The arc-length is denoted as $s = s(u)$, t_i denotes the unit tangent vector (tangent vector of the arc-length parametrised curve) at u_i, n_i the normal vector and κ_i the curvature. These quantities are linked by Fresnet's equations

$$\begin{aligned} \frac{dx(s)}{ds} &= t(s), \\ \frac{dt(s)}{ds} &= \kappa(s) \cdot n(s). \end{aligned}$$

The spring force F_i acting between particles at position x_{i-1} and x_i is

$$\begin{aligned} F_i &= \frac{k}{l}(\|x_i - x_{i-1}\| - l) \cdot \frac{x_i - x_{i-1}}{\|x_i - x_{i-1}\|} \\ &= \frac{k}{l}(x_i - x_{i-1}) - k\frac{x_i - x_{i-1}}{\|x_i - x_{i-1}\|} \\ &= k\left[\frac{\Delta_i x}{l} - \frac{\Delta_i x}{\|\Delta_i x\|}\right]. \end{aligned} \tag{2.9}$$

We let l converge to 0 to obtain differential operators:

$$F_i = k\left[\frac{dx(u_i)}{du} - t_i\right]$$

This shows that the stress σ, given as a scalar field along the curve, only depends on the parameterisation of the curve:

$$\sigma(u) = \pm\|F(u)\| = k \cdot (\|\frac{dx(u)}{du}\| - 1), \tag{2.10}$$

because $\frac{dx(u)}{du}$ and $t(u)$ are collinear. From this the force density along the curve can be derived: There are two springs attached to particle i exerting a combined force $F_{i+1} - F_i$ on this particle. This force scaled by $\frac{1}{l}$ can be interpreted as discrete approximation of the force density $\frac{dF}{du}$ at $x(u_i)$:

$$\begin{aligned}\frac{dF}{du} &= \lim_{l\to 0} \frac{F_{i+1} - F_i}{l} \\ &= k \lim_{l\to 0} \left[\frac{\Delta_{i+1}(\frac{dx(u)}{du})}{l} - \frac{\Delta_{i+1}(t)}{l} \right] \\ &= k \left(\frac{d^2x}{du^2} - \frac{ds}{du}\kappa(u) \cdot n(u) \right)\end{aligned}$$

With these force densities we can write the generalised one-dimensional wave equation, which is modelled by a chain of springs, as

$$k \left[\frac{d^2x}{du^2} - \frac{ds}{du}\kappa(u) \cdot n(u) \right] = \rho \frac{\partial^2 x}{\partial t^2}. \tag{2.11}$$

If the curve is a straight line, the curvature vanishes, the forces are proportional to the second derivative (or Laplacian), and equ. (2.11) reduces to the standard wave equation (2.8). For an arbitrary curve we measure the difference between the parametrised curve and its corresponding arc-length parametrised curve. As a result we have obtained a continuous formulation that describes the behaviour of a chain of masses and springs. It generalises the wave equation (2.8) to an arbitrary curve, and the parameterisation of this curve measures the strain.

2.3 Elastic Surfaces

In this section we will extend the continuous model to surfaces. By linearisation a modified strain and stress tensor will be derived. These tensors are plugged into the equation of motion of continuous objects and yield forces for a resolution independent particle system.

2.3.1 Approximation of the Strain Tensor

Most discrete systems have the property that different kinds of forces are modelled independently, i.e. shearing is modelled without taking tension into account. In order to separate tension from shear deformation, we define a local rest state r at each point $s(u, v)$ such that its local frame (r_u, r_v) is normalised and aligned with the distorted frame (s_u, s_v):

$$s(u, v) = r(a(u), b(v)),\ a_u = \|s_u\|,\ b_v = \|s_v\| \tag{2.12}$$

a and b are the arc lengths of the isoparametric curves (note that they only need to be defined up to an integration constant). This sheared rest frame is an approximation to the orthonormal rest frame used to derive the Cauchy's strain tensor.

However, there is no obvious choice for a local orthonormal frame, which is continuous on the surface, and therefore the sheared one is used. Now, the strain tensor can be written as follows:

$$G = \frac{1}{2}\begin{pmatrix} a_u^2 - 1 & a_u b_v \langle r_u, r_v \rangle \\ a_u b_v \langle r_u, r_v \rangle & b_v^2 - 1 \end{pmatrix} \tag{2.13}$$

We are going to approximate Green's strain tensor (2.13) such that this formulation of the strain yields a particle system. The approximated strain tensor will be denoted by ϵ.

Tension We start by approximating the diagonal components G_{ii} of the strain tensor first. They are approximated such that they only represent a strain due to tension. At any point of the object we have to define a rest state with respect to which the strain tensor can be linearised. The natural basis of the tangent plane (tangent space for volumes) is given by (s_u, s_v). Normalising this basis yields (r_u, r_v), which is defined to be our local rest state. This rest state r is constructed such that the deformed state does not contain any shear strain with respect to r. For the linearisation we exploit $s(u,v) = r(u,v) + d(u,v)$ (cf. figure 2.1) and neglect higher order terms of deformation derivatives d_u and d_v analogously to the linearisations (2.1). For the diagonal terms we obtain

$$\begin{aligned} G_{11} &= \frac{1}{2}(\langle r_u + d_u, r_u + d_u \rangle - 1) \\ &= \langle r_u, d_u \rangle + \frac{1}{2}\langle d_u, d_u \rangle \\ &= \langle r_u, d_u \rangle + O(d_u^2). \end{aligned}$$

Hence, the linearised strain in the direction of r_u is given by

$$\epsilon_{11} = \langle s_u - r_u, r_u \rangle = \|s_u\| - 1, \tag{2.14}$$

because r_u and s_u are collinear. Analogously, in the direction of r_v they are given by

$$\epsilon_{22} = \|s_v\| - 1. \tag{2.15}$$

Note that ϵ_{11} and ϵ_{22} are exactly the constraints that Baraff and Witkin [BW98] use to model stretch forces.

Shearing The off-diagonal components account for shearing effects. They are given by

$$G_{12} = G_{21} = \frac{1}{2}\langle s_u, s_v \rangle = \frac{1}{2} a_u b_v \langle r_u, r_v \rangle. \tag{2.16}$$

There is no natural local rest state such that ϵ_{12} could be linearised with respect to this rest state. The basis (r_u, r_v) cannot be used because this basis describes the

sheared state. Therefore, we set $\epsilon_{12} = G_{12}$. In the next chapter we will show that this nonlinear term does not impede an efficient numerical solution.

ϵ_{12} is closely related to the terms describing shearing in other particle systems [BHW94, EWS96, BW98], in which an average of the angles formed by the edges adjacent to one particle is used to measure the shear strain. This is a finite difference approximation of $\arccos\langle r_u, r_v\rangle$.

Summarising, the strain tensor of this particle systems is

$$\epsilon = \begin{pmatrix} \|s_u\| - 1 & \frac{1}{2}\langle s_u, s_v\rangle \\ \frac{1}{2}\langle s_u, s_v\rangle & \|s_v\| - 1 \end{pmatrix}. \tag{2.17}$$

2.3.2 Stress Tensor and Material Constants

The strain computed by the strain tensor gives rise to elastic forces and leads to stress on the boundary. For surfaces, stress is the force per unit length on the object boundary. In the model considered, stress and strain are assumed to be linearly related and they are linked by the elastic tensor C and Hooke's law (2.2). In particle systems the material constants are given by the tension (spring) constants in u and v direction, k_1 and k_2, and the shear modulus μ. These constants become the components of the elastic tensor:

$$C_{iiii} = k_i, C_{ijij} = C_{jiij} = \frac{1}{2}\mu, i \neq j$$

and all other components are zero at this point. Hence the stress is given by

$$\begin{aligned} \sigma_{11} &= k_1 \cdot \epsilon_{11}, \\ \sigma_{22} &= k_2 \cdot \epsilon_{22}, \\ \sigma_{12} &= \frac{1}{2}\mu \cdot \epsilon_{12}. \end{aligned} \tag{2.18}$$

2.3.3 Derivation of a Particle System

The equation of motion (2.3) for continuous materials has already been introduced in section 2.1.1. We will discetize this PDE in order to obtain a particle system.

The term $\rho\frac{\partial^2 s}{\partial t^2}$ is discretized using a discrete mass density $\rho(x_{ij})$ that is only defined on a grid of mass points (particles). In equ. (2.3) the term $\rho\frac{\partial^2 s}{\partial t^2}$ then is replaced by $\rho\frac{\partial^2 x_{ij}}{\partial t^2}$, the vector of accelerations of the particles. This method is called mass-lumping [Bat82] and is a standard procedure in animation to simplify equations that would be too complex for animation applications otherwise (e.g. [DCA00, TF88]). Also the external force densities f are discretized to obtain an external force f_{ij} for each grid point.

In order to derive the interior forces per area (per volume in 3D) that appear in equ. (2.3), the divergence of the stress tensor has to be computed. The divergence of the tensor σ_{ij} is defined as $\sum_j \frac{\partial_i \sigma ij}{x_j} e_i$, where e_i is the i-th Cartesian basis vector. As the deformed surface has the local basis (r_u, r_v), this basis replaces the Cartesian basis in the computation of the divergence. Then the divergence is replaced by an approximation of the divergence, because the coordinate system is not orthogonal. Therefore, the approximated divergence will be denoted with the subscript γ indicating the dependence on the shear angle $\gamma = \arccos\langle r_u, r_v\rangle$:

$$\mathrm{div}_\gamma \sigma = \frac{\partial \sigma_{11}}{\partial u} r_u + \frac{\partial \sigma_{12}}{\partial u} r_u + \frac{\partial \sigma_{21}}{\partial v} r_v + \frac{\partial \sigma_{22}}{\partial v} r_v \tag{2.19}$$

This induces error terms which are (in the lowest order) linear in the shear angle if s is orthogonal in the equilibrium state. However this error only shows, when there is a superimposed longitudinal deformation in the corresponding direction. Since both shear and stretch are proportional to the applied forces, the resulting error is quadratic in the forces and thus does not take effect in the low deformation limit.

For the diagonal components, the finite difference discretization yields exactly the four spring forces (tension) acting on a single particle at position $s(u_i, v_j)$. This will be shown here for the u-direction by replacing the differential operators with difference operators step by step. For simplicity, it is assumed that all rest distances between the grid points (particles) x_{ij} are equal and denoted by l. The forces at a given point $s(u_i, v_j)$ are obtained by derivation of the stress components with respect to the transformed coordinate system:

$$\begin{aligned}
\frac{\partial \sigma_{11}}{\partial u} \cdot r_u \; &= k_1 \cdot \frac{\partial \epsilon_{11}}{\partial u}|_{(u_i,v_j)} \cdot r_u(u_i, v_j) \\
&= k_1 \cdot \frac{1}{l} \left[\left(\frac{\|x_{i+1,j} - x_{i,j}\|}{l} - 1 \right) - \left(\frac{\|x_{i,j} - x_{i-1,j}\|}{l} - 1 \right) \right] r_u + O(l) \\
&= k_1 \cdot \frac{\|x_{i+1,j} - x_{i,j}\| - l}{l^2} \cdot \frac{x_{i+1,j} - x_{i,j}}{\|x_{i+1,j} - x_{i,j}\|} \\
&\quad - k_1 \cdot \frac{\|x_{i,j} - x_{i-1,j}\| - l}{l^2} \cdot \frac{x_{i,j} - x_{i-1,j}}{\|x_{i,j} - x_{i-1,j}\|} + \; O(l)
\end{aligned} \tag{2.20}$$

In each line one continuous term is replaced with a finite difference approximation employing equations (2.14) and (2.15). Thus, we have derived linear spring forces as in the one-dimensional case (cf. equ. (2.9)).

We can conclude that the transformed basis leads to tension forces that depend on the shear angle γ by the approximated divergence div_γ. In contrast to the classical model, we do not have orthogonal coordinates in general. An example is given in figure 2.4, where r_u is aligned with the x-axis and the tension force in that direction is $F_1 + \cos(\gamma) \cdot F_2$, i.e. the force density in direction r_v also contributes to the force density in the r_u-direction.

The shear term σ_{12} is approximated by $\mu\langle s_u, s_v\rangle$. The finite difference approx-

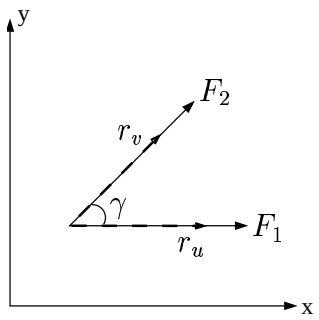


Figure 2.4: Tension forces are computed with respect to a rest state r that is sheared as the current deformed state

imation yields

$$\sigma_{12} = \frac{\mu}{4l^2}\langle (x_{i+1,j} - x_{i-1,j}), (x_{i,j+1} - x_{i,j-1})\rangle + O(l^2). \tag{2.21}$$

Analogously to the diagonal components of the stress tensor, the corresponding force terms $\frac{\partial \sigma_{12}}{\partial v} r_u$ and $\frac{\partial \sigma_{21}}{\partial u} r_v$ in equ. (2.19) are computed. The implementation of the resulting forces is presented in section 2.5.

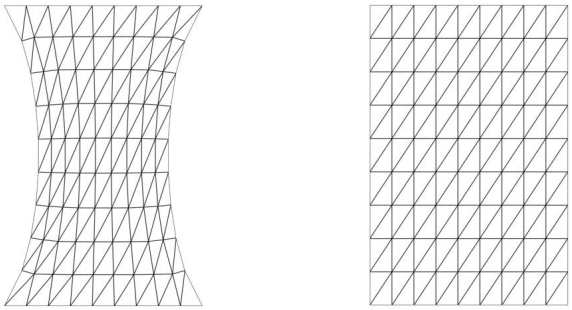


Figure 2.5: A stretched object with (left) and without (right) transverse contraction

In cloth simulation systems that are based on experimental data [BHW94, EWS96] transverse contraction has not been modelled because it is not included in the standard material specification [Kaw80]. Nevertheless, transverse contraction is apparent in textiles, and by our continuum mechanics approach it can easily be included in the model. So far our elastic tensor does not have any nonzero components $C_{iijj}, i \neq j$. Hence, only materials with Poisson ratio $\nu = 0$ are modelled (cf. figure 2.5). This shortcoming can be corrected in two ways:

1. Diagonal springs can be inserted. They are related to the strain in the direction $r_u + r_v$. Unfortunately, these terms introduce a coupling between

tension and transverse forces. Thus, the identification of spring stiffnesses with elastic moduli is not trivial.

2. Transverse contraction is modelled by deriving forces from an elastic tensor with off-diagonal entries. If the entry $C_{1122} = C_{2211}$ in the elastic tensor is not zero, we have

$$\begin{aligned} \sigma_{11} &= C_{1111}\epsilon_{11} + C_{1122}\epsilon_{22}, \\ \sigma_{22} &= C_{2222}\epsilon_{22} + C_{1122}\epsilon_{11}. \end{aligned} \tag{2.22}$$

This means we introduce additional forces $C_{1122}\frac{\partial \epsilon_{22}}{\partial u} r_u$ and $C_{1122}\frac{\partial \epsilon_{11}}{\partial v} r_v$ according to equ. (2.19) to implement transverse contraction directly.

In a triangular particle system, for instance the one by Baraff and Witkin [BW98], the physical properties do not only depend on the material constants but also on the mesh topology. From equ. (2.19) it can be seen that strongly non orthogonal angles lead to edges that combine tension and transverse contraction. As a consequence, triangular systems usually show too large transverse contraction.

2.3.4 Comparison

We have seen that we can derive forces and constraints that were used in the various discrete systems. Hence, these discrete systems represent a semi-discretization of the continuum equation (2.3), i.e. the equation is discretized only in space but not in time. The time discretization has yet to be carried out by a solver for ordinary differential equations and will be treated in chapter 3.

One might wonder why there is a scaling factor $\frac{1}{l^2}$ in equ. (2.20) and equ. (2.21), but no such factor appears in the discrete systems. In discrete systems that directly employ Newton's law (2.4) rather than the continuum version (2.3), the masses are scaled when we change the discretization, whereas in the discretization of the continuum equation (2.3) the strain is scaled by the factor $\frac{1}{l^2}$. However, both scalings are equivalent. To see this, consider a quadratic patch with N^2 particles, mass M, and constant mass density. Then the mass of a single particle is $m_i = \frac{M}{N^2}$. If we now use the resolution $s \cdot N$ in each direction, we have $(s \cdot N)^2$ particles with masses $m_i = \frac{M}{(sN)^2}$. On the other hand we have $\frac{1}{(\frac{l}{s})^2} = s^2 \frac{1}{l^2}$ in the continuous case (2.3). Thus we see that both formulations lead to the same scaling factor s^2 when we switch to a different resolution. Note that a refinement of the discretization makes the numerical treatment harder because the stiffness of the system is increased (cf. chapter 3).

The major difference between the particle systems and linear continuum mechanics is the underlying reference frame. The basis (r_u, r_v) is not orthogonal in general; when we take derivatives with respect to this basis, forces are obtained that are not perpendicular, whereas the continuum mechanics equation is not explicitly dependent on the shear angle.

2.3.5 Bending Forces

As two-dimensional objects embedded in three-dimensional space are modelled, we have to introduce bending forces into our model. All particle systems so far model bending energy depending on some curvature properties of the surface.

Breen et al. [BHW94] and Eberhardt et al. [EWS96] derive energies from the angle of two consecutive edges in each isoparametric direction (corresponding to weft and warp in textiles). This angle is a measure of the curvature of an isoparametric curve in the respective direction (Baraff and Witkin [BW98] employ the angle between normals of adjacent patches because they use triangle meshes). But the actual correspondence between angle and curvature has to be reconstructed. Terzopoulos and Fleischer [TF88] incorporate the second fundamental form into their energy term.

A surface is uniquely defined by its first and second fundamental form. While tension and shearing forces could be derived from the first fundamental form, bending forces have to be derived from the second fundamental form, which contains the curvature properties of the surface. It is given by the following matrix:

$$II_s = \begin{pmatrix} \langle s_{uu}, N\rangle & \langle s_{uv}, N\rangle, \\ \langle s_{uv}, N\rangle & \langle s_{vv}, N\rangle, \end{pmatrix} = \begin{pmatrix} e & f \\ f & g \end{pmatrix}, \tag{2.23}$$

where N is the unit surface normal. II_s is used as the curvature strain tensor (if the rest state is not planar we have to use $II_s - II_r$). Any force counteracting curvature must act in the direction of the surface normal. Interpreting the diagonal components directly as stress on the surface, i.e. force per area in the the surface normal direction, we derive force densities

$$F = B_1 e \cdot N + B_2 g \cdot N. \tag{2.24}$$

B_1 and B_2 are the elastic material constants, called bending moduli, for the respective directions. These are the forces due to the membrane energy of the surface projected onto the normal direction. The force is given by the variational derivative of the energy:

$$F = \frac{1}{2}\langle N, \partial \int B_1 s_u^2 + B_2 s_v^2 du dv\rangle = \langle N, B_1 s_{uu} + B_2 s_{vv}\rangle. \tag{2.25}$$

With the finite difference approximation of the differential operators for constant l

$$\begin{aligned} s_{uu} &= \frac{x_{i+1,j} - 2x_{i,j} + x_{i-1,j}}{l^2} + O(l) \\ s_{vv} &= \frac{x_{i,j+1} - 2x_{i,j} + x_{i,j-1}}{l^2} + O(l) \end{aligned}$$

and the usual vertex normals it is straightforward to compute the force densities in equ. (2.24). These force densities apply directly to the surface area along the

surface normal field. Hence, no divergence has to be computed. Summarising, we avoid the troublesome use of angles and implement the curvature as bending strain in a straightforward way.

Note that no forces due to the off-diagonal entry f are modelled because f is dependent on the diagonal entries e and g and the first fundamental form (Gaussian theorem, see [DoC76]). The diagonal entries e and g approximate the *normal curvature* in the directions r_u, r_v, whereas the bend angles as defined by equ. (2.7) approximate the curvature of the isoparametric curves. However, employing the curvature, which is the orthogonal sum of normal and geodesic curvature, leads to in-plane forces that counteract both shearing and bending, because the geodesic curvature can measure shear.

The membrane energy actually only describes stress due to the elongation of a membrane, but does not correspond to a physical model of curvature. But instead of the membrane energy in equ. (2.25), also the thin plate energy can be employed. This describes the bend energy of thin plates and leads to fourth order derivatives

$$\begin{aligned} F &= \frac{1}{2}\langle N, \partial \int B_1 s_{uu}^2 + B_2 s_{vv}^2 du dv \rangle \\ &= \langle N, B_1 s_{uuuu} + (B_1 + B_2) s_{uuvv} + B_2 s_{vvvv} \rangle. \end{aligned} \tag{2.26}$$

The discrete fourth order operators are computed by applying the second order operators twice.

2.4 Energy Dissipation

In the previous section an elastic model, which conserves the elastic energy, has been developed. However, in every material energy is dissipated due to internal friction, i.e. viscous forces counter the progress of deformation. Viscous forces produce hysteresis effects that appear as a closed loop if the deformation is periodic (this is an approximation because the deformation has to be assumed to last infinitely long). Figure 2.6 shows an example of such a stress-strain plot, which is obtained when the material is deformed and afterwards relaxed at a constant velocity. External viscous forces like air resistance will be described in section 2.6.

In the following we will show how explicit viscous forces can be derived from the strain rate tensor analogously to the derivation of elastic forces from the strain tensor, and we will get another set of material parameters that describe the viscous properties of the deformable object. In the second subsection we will present memory parameters as an alternative way of implementing viscosity. This approach allows a straightforward modelling of more complex viscous models. For instance, J. Groß et al. [GEHB01] employ these memory parameters to implement a model that fits measured material data to the viscoelastic model.

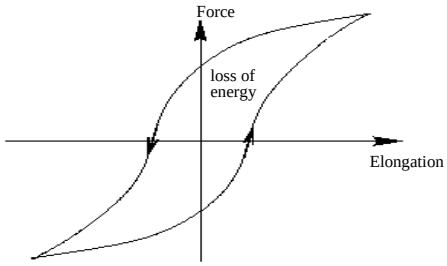


Figure 2.6: Hysteresis

2.4.1 Viscous Forces

Internal friction depends on the relative motion of parts of the deformable object, that is on the strain rate. The strain rate tensor (cf. [BW00]) is defined as the time derivative of strain,

$$\nu_{ij} = \frac{d\epsilon_{ij}}{dt}. \tag{2.27}$$

Explicitly, the following strain rates are derived from the strain components given by equations (2.14), (2.15), and (2.16):

$$\begin{aligned} \nu_{11} &= \frac{\langle \dot{s}_u, s_u \rangle}{\|s_u\|} \\ \nu_{22} &= \frac{\langle \dot{s}_v, s_v \rangle}{\|s_v\|} \\ \nu_{12} &= \langle \dot{s}_u, s_v \rangle + \langle s_u, \dot{s}_v \rangle \end{aligned} \tag{2.28}$$

After discretization, the diagonal components of the strain rate tensor can be interpreted easily as the relative velocity of the particles projected onto the edge between them:

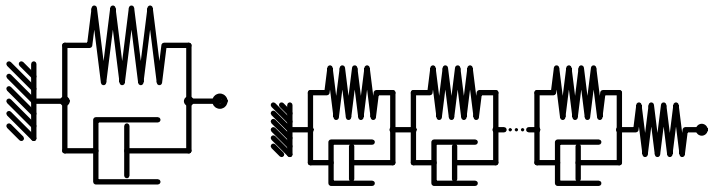
$$\begin{aligned} \nu_{11} &= \frac{\langle v_{i,j} - v_{i-1,j}, x_{i,j} - x_{i-1,j} \rangle}{\|x_{i,j} - x_{i-1,j}\|} \\ \nu_{22} &= \frac{\langle v_{i,j} - v_{i,j-1}, x_{i,j} - x_{i,j-1} \rangle}{\|x_{i,j} - x_{i,j-1}\|} \end{aligned}$$

A viscous stress can be computed analogously to equ. (2.2):

$$\sigma^v_{ij} = D_{ijkl}\, \nu_{kl}, \tag{2.29}$$

where the "viscosity tensor" D has the same properties as the elastic tensor C. The viscous stress is added to the purely elastic stress such that

$$\sigma_{ij} = C_{ijkl}\, \epsilon_{kl} + D_{ijkl}\, \nu_{kl}.$$



(a) (b)

Figure 2.7: Kelvin-Voigt elements

The elastic bend strain is computed by the projection of the second or fourth derivatives onto the surface normal. In order to derive simple damping terms, we take the time derivatives of these spatial derivatives and project them onto the surface normal:

$$\nu_1^{\text{bend}} = \langle N, \dot{s}_{uu} \rangle, \quad \nu_2^{\text{bend}} = \langle N, \dot{s}_{vv} \rangle$$

or alternatively

$$\nu_1^{\text{bend}} = \langle N, \dot{s}_{uuuu} + \dot{s}_{uuvv} \rangle, \quad \nu_2^{\text{bend}} = \langle N, \dot{s}_{vvvv} + \dot{s}_{vvuu} \rangle \tag{2.30}$$

for the membrane and thin plate energy, respectively. These strains lead to forces $\nu_{1,2}^{\text{bend}} N$ in the surface normal direction.

In most implementations the viscosity tensor D is chosen constant and proportional to the elastic tensor C. This produces a material called a Kelvin-Voigt solid, the simplest viscoelastic solid. In one dimension it can be modelled as a damping dash-pot parallel to a spring as shown in figure 2.7(a).

2.4.2 Memory Parameters

Hysteresis effects as in figure 2.6 are due to energy dissipation. When the material deforms under external forces, the stress-strain relation is described by the upper branch of the hysteresis. When it is released and contracts again, this relationship is described by the lower branch and the area between both branches is proportional to the dissipated energy. The mechanical quality Q of a material is defined by the ratio of the mean energy stored in the material during one cycle and the energy lost. Q is another material property, which is zero in purely viscous fluids.

Although the branches of the hysteresis seem to indicate a nonlinear relationship, this is not necessarily true. In linear viscoelasticity, Hooke's law holds in the frequency domain:

$$\sigma(\omega) = C(\omega) \cdot \epsilon(\omega)$$

When transferred into the time domain by the inverse Fourier Transform, we obtain

$$\sigma(t) = \int_{-\infty}^{+\infty} C(t-\tau)\epsilon(\tau)d\tau, \tag{2.31}$$

where $C(t)$ is a time dependent modulus. This modulus is the time derivative of the relaxation function $R(t)$ of the material, which is the stress response to a strain described by unit step.

Hooke's law can be exploited to implement viscosity without explicit viscous forces. Instead, in each time step an additional term computed from *Memory parameters* is added to the force terms.

In equ. (2.31) the upper integration border can be set to t, as the behaviour of the material cannot depend on the future. Moreover, the strain $\epsilon(t)$ can be assumed to vanish for $t \to -\infty$ such that partial integration of equ. (2.31) yields

$$\sigma(t) = \int_{-\infty}^{+t} R(t-\tau)\dot{\epsilon}(\tau)d\tau, \tag{2.32}$$

The relaxation function of a Kelvin-Voigt element can be computed analytically. The integral is split into a memory part and a current part:

$$\sigma(t_{i+1}) = \int_{t_i}^{t_{i+1}} R(t_{i+1}-\tau)\dot{\epsilon}(\tau)d\tau + \int_{-\infty}^{t_i} R(t_{i+1}-\tau)\dot{\epsilon}(\tau)d\tau \tag{2.33}$$

The memory term is constant and has been computed in the previous time step. Gross et al. [GEHB01] transform the first term by interpolation of $\dot{\epsilon}(\tau)$ and successive integration into a linear term in ϵ. In the simplest case, linear interpolation, this gives

$$\sigma(t_{i+1}) = \frac{1}{t_{i+1}-t_i}\left[\int_{-\infty}^{t_i} R(t_{i+1}-\tau)d\tau\right](\epsilon(t_{i+1}) - \epsilon(t_i)) + \sigma(t_i).$$

Thus, we regain a model that in each time step solely depends linearly on the strain extended by the memory term, which is constant in each time step. This term can be integrated easily into the numerical framework that will be presented in the following chapter.

After the introduction of Kelvin-Voigt elements it remains to discuss how to model real world materials with these elements. The animation applications are mostly limited to a constant elastic and viscous tensor. In applications where certain materials are to be modelled and their properties are to be preserved, however, this does not suffice, and a more sophisticated viscoelastic model is necessary.

Gross et al. [GEHB01] propose a viscoelastic model that easily can be incorporated into the particle system by means of memory parameters.

A constant Q model allows us to approximate a specified viscoelastic behaviour. Since textiles show a hysteresis widely independent of the frequency, we assume a frequency independence of Q (CQ-material). A CQ-material can be approximated by a series of Kelvin-Voigt elements as in figure 2.7(b). The compliance $S(\omega)$ (inverse of the elastic modulus) of such a Kelvin-Voigt-series is given by

$$S(\omega) = \frac{1}{k_0} + \sum_j \frac{1}{k_j} \frac{1}{1 + i\frac{\omega}{\omega_j}}, \tag{2.34}$$

where $\omega_j = k_j/d_j$ and k_j and d_j are the elastic and viscous constants of the j-th Kelvin-Voigt element, and k_0 the constant of the single spring. The elastic and viscous constants then are chosen such that $S(\omega)$ approximates the measured compliance of the material in a least squares sense. The relaxation function of such a series of Kelvin-Voigt elements is computed by superposition of single elements. With this relaxation function the memory parameters can be computed as above.

2.5 Implementation of the Internal Forces

In sections 2.3 and 2.4 the elastic and viscous internal forces have been derived. In the implementation of the cloth modelling system the elastic tension, shear, and bend forces have been included together with their viscous counterparts. In this section it will be elaborated how these forces are implemented.

Note that, again, we assume a uniform spacing with distance l for the sake of simplicity. A generalisation to non-uniform spacing can be easily achieved. F_i is the force applied to particle i. f is a temporary vector to compute forces.

As has been shown, the stretch forces are identical to linear springs, which can be implemented as follows:

Algorithm 1: Stretch forces

(1) **for each** edge $(i,j) \in$ mesh **do**
Tension forces according to equ. (2.20)
(2) Compute $f = \frac{k_{1,2}}{l^2}(\|x_i - x_j\| - l)(x_i - x_j)^0$
Add viscosity according to equ. (2.27)
(3) Add $\frac{D_{1111,2222}}{l^2}\langle v_i - v_j, (x_i - x_j)^0\rangle$ to f
(4) Add f to F_i
(5) Add $-f$ to F_j
end

In the next chapter we will explain that the computation of tension forces has to be split into two parts to solve the ODE efficiently. Therefore, the implementation within the numerical integration differs from the above algorithm.

For the shear forces the divergence of the shear stress as given by equ. (2.21) must be computed. The partial derivatives $\frac{\partial \sigma_{12}}{\partial u}$ and $\frac{\partial \sigma_{12}}{\partial v}$ are computed by the differences of σ_{12} in two adjacent grid points.

Algorithm 2: Shear stress elements

(1) **for each** particle $i \in$ mesh **do**
(2) Get left, upper, right, lower neighbours of particle i
with indices j, k, l, m, respectively
One-sided differences on the boundary
(3) **if** neighbour $n \in \{j, k, l, m\}$ does not exist
then replace with central particle $n := i$
Stress computation according to equ. (2.21)
(4) Compute $\sigma = \frac{\mu}{8l^2}\langle x_j - x_l, x_k - x_m\rangle$
Add viscosity according to equ. (2.27)
(5) Add $\frac{D_{1212}}{8l^2}$ $(\langle v_j - v_l, x_k - x_m\rangle$
$+\langle x_j - x_l, v_k - v_m\rangle)$ to σ
Compute divergence of shear stress with equ. (2.19)
(6) $f_u = \frac{\sigma}{2l}(x_l - x_j)^0$
(7) $f_v = \frac{\sigma}{2l}(x_m - x_k)^0$
(8) Add f_v to F_l
(9) Add $-f_v$ to F_j
(10) Add f_u to F_m
(11) Add $-f_u$ to F_k
end

The factor $\frac{1}{2}$ in lines (6) and (7) is needed because each edge is evaluated twice in the algorithm. Note that for nonuniform spacing the algorithm has to use a more general expression to compute the shear stress in line (4):

$$\sigma_{12} = \frac{\mu}{8}\langle \frac{x_j - x_i}{l_1} + \frac{x_i - x_l}{l_3}, \frac{x_k - x_i}{l_2} + \frac{x_i - x_m}{l_4}\rangle + O(l^2),$$

which yields additional forces on the central particle.

The algorithm for the computation of bending forces computes the projected membrane energy using second order derivatives. The second order finite difference operators are computed by a difference of first order finite differences. Thus, the computation can be carried out analogously to the computation of stretch forces in the previous algorithm followed by a projection onto the surface normals.

Algorithm 3: Membrane bend forces

Compute Laplacian
(1) **for each** edge $(i,j) \in$ mesh **do**
(2) Compute $f = \frac{B_{1,2}}{l^2}(x_i - x_j)$.
Add viscous term
(3) Add $\frac{\nu_{1,2}^{\text{bend}}}{l^2}(v_i - v_j)$ to f.
(4) Add $-f$ to R_j
(5) Add f to R_i
end
Project Laplacian onto surface normal
(6) **for each** particle i **do**
(7) Add $\langle R_i, N_i \rangle N_i$ to F_i
end

N_i denotes the vertex normal at particle i. The elastic parameters are B_1, B_2 and the viscous parameters are $\nu_{1,2}^{\text{bend}}$. In lines (1) to (5) the differences are summed up in the temporary vectors R_i. Afterwards, each vector R_i is projected onto the surface normal.

The alternative forces derived from thin plate energy are implemented likewise. The Bilaplacian operator requires two passes to compute the differences.

Algorithm 4: Thin plate bend forces

Compute first Laplacian
(1) **for each** edge $(i,j) \in$ mesh **do**
(2) Compute $f = \frac{1}{l^2}(x_i - x_j)$.
(3) Add f to R_i
(4) Add $-f$ to R_j
Viscous term
(5) Compute $f = \frac{1}{l^2}(v_i - v_j)$.
(6) Add f to T_i
(7) Add $-f$ to T_j
end
Compute second Laplacian

(8) **for each** edge $(i, j) \in$ mesh **do**
(9) Compute $f = \frac{B_{1,2}}{l^2}(R_i - R_j)$.
(10) Add f to S_i
(11) Add $-f$ to S_j
(12) Compute $f = \frac{\nu_{1,2}^{\text{bend}}}{l^2}(T_i - T_j)$.
Add viscous term
(13) Add f to S_i
(14) Add $-f$ to S_j
end
Project Laplacian onto surface normal
(15) **for each** particle i **do**
(16) Add $\langle S_i, N_i \rangle N_i$ to F_i
end

In lines (1) to (7), the first Laplacian is computed in the temporary vectors R_i and T_i. These serve as input for the computation of the second Laplacian in lines (8) to (14). Finally, in lines (15) to (16) the Bilaplacian operator is projected onto the surface normals.

Most particle systems do not incorporate transverse contraction forces. If such forces are required, they can be implemented by equ. (2.22). The following algorithm computes these forces for each particle:

Algorithm 5: Transverse contraction forces

(1) **for each** particle $i \in$ mesh **do**
(2) Get left and upper neighbours of particle i
with indices j, k, respectively
(3) **if** left neighbour does not exist
then use right neighbour
(4) **if** upper neighbour does not exist
then use lower neighbour
Stress computation according to equ. (2.22)
(5) Compute $\tau_1 = \frac{C_{1122}}{l^2}(\|x_j - x_i\| - l)$
(6) Compute $\tau_2 = \frac{C_{1122}}{l^2}(\|x_k - x_i\| - l)$
Add viscosity according to equ. (2.27)
(7) Add $\frac{D_{1122}}{l^2}\langle v_j - v_i, (x_j - x_i)^0 \rangle$ to τ_1
(8) Add $\frac{D_{1122}}{l^2}\langle v_k - v_i, (x_k - x_i)^0 \rangle$ to τ_2
Compute divergence of stress
(9) $f_u = \tau_2 (x_j - x_i)^0$
(10) $f_v = \tau_1 (x_k - x_i)^0$
(11) Add f_u to F_j
(12) Add $-f_u$ to F_i

(13) Add f_v to F_k
(14) Add $-f_v$ to F_i
end

2.6 External Forces

After we have described the internal elastic and viscous forces of the deformable object, external forces must be introduced. By external forces the deformable object can interact with its environment. These forces appear in the equation of motion (2.3) as the term f. Here we state the most important external forces, which are necessary for textile animation.

Gravity The most important potential force (density) is the gravity that causes the draping of clothes. It is given by ρg.

Collision Forces These also play a fundamental role in animation and are detailed in chapter 4.

Other forces account for energy dissipation:

Friction When a face of the deformable objects collides, friction forces counteract the movement in the tangential direction. Columb's friction (cf. [Vog97]) describes the friction of a body that slides over another body perpendicular to the surface normal N:

$$f_{\text{fric}} = -\gamma \|f_{\text{N}}\| \cdot v^0_{\text{tan}},$$

where the tangent velocity can be computed as

$$v_{\text{tan}} = v - \langle v, N \rangle N.$$

The magnitude of the friction force only depends on the normal stress f_{N} that counteracts the collision force in the direction of N. This normal stress can be easily computed by projecting the forces on the colliding particles onto N. The friction forces are applied to each colliding particle. If stiction is to be implemented, constraints have to be used to hold colliding particles in place, until a certain threshold is reached and it is switched back to sliding friction.

Wind and Air Resistance Wind is a complex phenomenon, and we restrict the wind simulation to a simple model which is designed to create wind-like effects but is by no means physically exact. A more sophisticated wind model for cloth simulation can be found in an article by Ling [Lin00]. For our purposes it suffices

to implement some wind-like effects that allow to sway the textiles. We choose an oscillating or constant wind from a constant direction:

$$f_{\text{wind}} = (a + sin(\omega t))w, \tag{2.35}$$

where w is a wind vector, ω the angular velocity, and a a constant term that allows for a constant wind. The wind acts on the surface of the textile. Hence, the force is split into a drag force into the surface normal direction and a lift force into the tangential directions:

$$f_{\text{response}} = c_{\text{drag}} \langle f_{\text{wind}}, N \rangle N + c_{\text{lift}} (f_{\text{wind}} - \langle f_{\text{wind}}, N \rangle N) \tag{2.36}$$

Note that the expression is independent of the surface area, since f_{response} actually is a force density.

Additionally, there is air resistance, which can be modelled as Stoke's friction. It depends on the body's velocity and acts in a direction opposite to the body's velocity. The air resistance therefore is given by

$$f_{air} = -c_w \sigma \langle v, N \rangle \frac{v}{\|v\|}, \tag{2.37}$$

where N the vertex normal, σ the air density, and c_w a resistance coefficient. Note that no shadowing effects are implemented. For instance, a sheet is moved by the wind if occluded by an obstacle. Accounting for such shadowing effect would be too time consuming.

2.7 Mesh Generation

It is important to note that the presented finite difference derivation only holds for regular quadrilateral meshes and is only accurate for rectangular meshes. Rectangular meshes also have the advantage that they correspond to the orthotropic textiles, and the rectangles can be aligned with the weft and warp directions. If triangle meshes are used, we loose control over the material properties and even simple materials cannot be modelled correctly.

For surfaces topologically equivalent to a disc the restriction of rectangular meshes can be overcome by trimmed surfaces. A triangle mesh approximating the curved object boundaries can be inscribed into the rectangular mesh as follows: In the planar state, which is equivalent to the parametrisation, each vertex of the triangle mesh is situated in one rectangle (neglecting the special cases that a vertex may lie on a rectangle boundary). If the rectangle is split into a left and right triangle, the vertex position is uniquely described by the rectangular face index, an entry for the left or right triangle, and its barycentric coordinates therein. This defines a local coordinate frame, and when the rectangular mesh deforms, the triangle vertex positions can be updated from the local coordinates such that the triangle mesh moves with the rectangle mesh.

An example is shown in figure 2.8, where the triangle mesh vertices are depicted as bold dots. Each of them lies in one of the underlying rectangles and can either be assigned to the left lower half or right upper half of the rectangle.

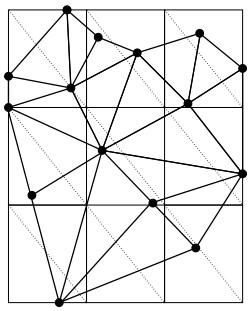


Figure 2.8: A triangle mesh inscribed in a rectangular mesh

Unfortunately, clothes are not topologically equivalent to a disc, i.e. they contain holes, and the trimmed surface solution is not applicable. However, most clothes can be assembled from quadrilateral patterns which allow a regular sampling. The faces generated by this sampling are close to rectangles such that we still get a fairly good approximation of the physical properties. An example of such a pattern is depicted in figure 2.9, which shows the front part of a dress. In chapter 6 it will be demonstrated how an avatar is clothed with this dress.

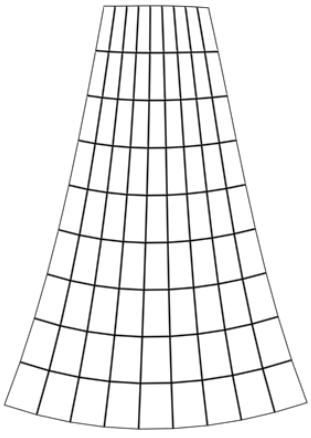


Figure 2.9: The front of a dress pattern with curvilinear coordinates

2.8 Results

Here we present some results on the accuracy of the implemented cloth modelling system. Since collisions would distort the accuracy, we only show examples in which the textile is fixed at the edges and does not collide with itself or other objects. Our sample model is a square mesh discretized by an increasing number of particles.

The simulations are run until the textile stops moving and an equilibrium state is reached. Note that for the equilibrium state the damping constants are irrelevant as all velocities are zero. This computed equilibrium solution is compared to the analytical solution of nonlinear continuous models computed by Joachim Groß [Gro01]. In all experiments the mass density is set to $1\frac{kg}{m^2}$.

Experiment 1: hanging textile In this experiment the sample is fixed along its right edge and is stretched by the external force. Here, we only verify the tension forces. The Young's modulus is set to $10^4\frac{N}{m}$, and the external force density is set to $981\frac{N}{m^2}$. In figure 2.11 (a) the error is plotted for different resolutions.

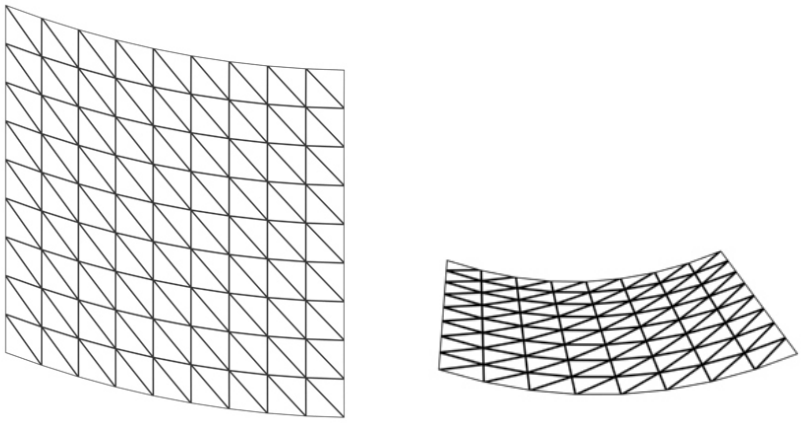
Experiment 2: sheared textile The sample is fixed along one edge, and under the influence of a body force of $10\frac{N}{m^2}$ the textile is sheared along the direction of the constrained edge as shown in figure 2.10 (a). We set the Young's modulus to $10^6\frac{N}{m}$ and the shear module to $100\frac{N}{m}$. The error plot is depicted in figure 2.11 (b).

Experiment 3: hanging textile with bend forces In this experiment the left and the right edges of the textile are constrained, the left one in all directions, the right one only vertically such that it is still free too move horizontally. Under the influence of an external force ($1.0\frac{N}{m^2}$) this free edge will approach the opposite edge until it is stopped by bend forces as shown in figure 2.10(b). In this experiment the bending forces according to equ. (2.26) are used, and B is set to $5\frac{N}{m^2}$, while the Young's modulus is set to $10^6\frac{N}{m^2}$. The results are depicted in figure 2.11(c).

In all experiments the error is computed from the difference between the analytic solution and the simulated solution at the nodes (particles) of the grid normalised by the number of nodes. This error is plotted over the resolution, i.e. the number of particles in one direction. From the results we see that the simulated solutions converge to the exact solutions.

2.9 Conclusions

Deformable models developed in computer animation such as mass-spring, coupled particle, and constraint systems fall into the same category of discrete models for deformable objects. We have shown how the models fit into the notion of continuum mechanics. A particle system has been derived from continuum mechanics



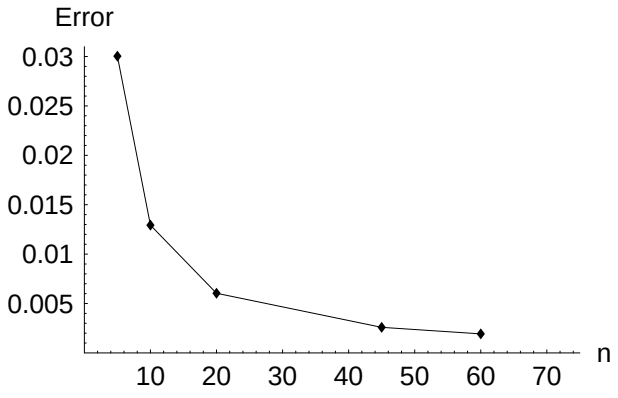
(a) (b)

Figure 2.10: A hanging and a sheared textile

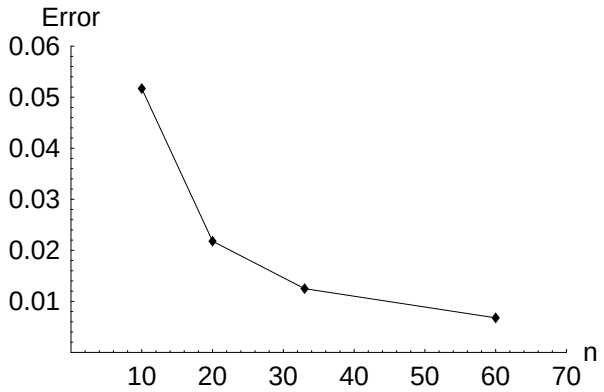
by linearisation, the choice of a local reference frame, and successive finite difference discretization. This gives these systems the advantage of rotational invariance over classical linear elasticity. Therefore, they can be applied to highly flexible deformable objects like textiles. The theoretical drawback of employing reference frames that are not orthogonal is a behaviour which depends on the local shear angles. For moderately large deformations, however, we find this theoretical difference negligible and the approximation to be sufficiently accurate for practical applications in computer graphics. After we have established a link between continuous and discrete systems, we can model materials using the material constants as in continuum mechanics to model continuous materials.

In previous discrete systems presented in computer animation, transverse contraction was not included or controlled. In this chapter we showed how to implement these forces.

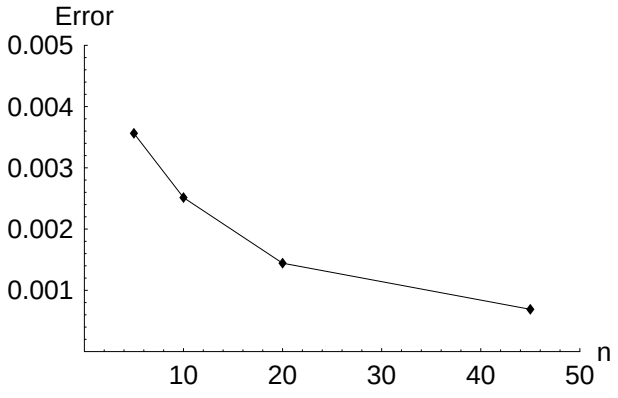
The derived particle system provides a simple model that is well suited for fast animations because it lends itself to numerical treatment easily. Surfaces embedded in three-dimensional space require bending energy that is modelled by means of the surface normal curvature. Viscosity can be derived from the strain rate tensor as elastic forces are derived from the strain tensor.



(a)



(b)



(c)

Figure 2.11: The error plots for the (a) tension, (b) shear, and (c) bend experiment. The error converges to zero, when the resolution is increased.

Chapter 3

An Efficient Framework for Numerical Solutions

In the previous chapter a physical model and a spatial discretization has been derived. This model is given as an ODE, and it remains to solve this ODE in time. In animation we would like to make quick, large time steps to run simulations rapidly. Since the ODE to be solved is stiff, explicit integration methods require small time steps, whereas in implicit integration time steps are usually very expensive and slow.

In this chapter a framework is developed that speeds up time steps in implicit methods and allows very rapid simulations. We use methods from numerical analysis and adopt them to our needs. By specialising numerical methods, their performance can be improved significantly compared to general purpose methods. The solution will be controlled by an error tolerance that can be set to obtain either a rapid animation or an accurate simulation.

The structure of this chapter is as follows: First, some basic numerical background is presented, then an implicit-explicit Euler integration scheme for particle systems is developed, which subsequently is extended and improved by higher order methods and fully implicit schemes.

3.1 Numerical Foundations

Before the framework will be developed, some standard methods from numerical analysis are described. More details can be found in many available text books [SB00, DH91, HW96, SW92].

3.1.1 Numerical Solutions of Initial Value Problems

Since the numerical solution of ordinary differential equations (ODE) plays a central role in the animation of deformable objects, we present here the general concepts of a numerical ODE solver.

A *k-step method* is given by a function Φ and a scheme to compute the numerical approximation y_{n+k} to the exact solution $y(t_{n+k})$:

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h\Phi(t_n, f_n, \ldots, t_{n+k}, f_{n+k}), \tag{3.1}$$
$$f_{n+j} := f(t_{n+j}, y_{n+j}),$$

where h is the (time) step size and y_i is the numerical solution at time t_i. The coefficient α_k is required to be nonzero. A k-step method is *convergent* if and only if

$$\lim_{h\to 0} \max_{0\le l\le N(h)} \|y_l - y(t_l)\| = 0, \tag{3.2}$$

where $N(h)$ is the number of computed points for step size h and y(t) the exact solution.

A method is called *implicit*, if and only if Φ depends on f_{n+k}. For an implicit method in each step a system of equations has to be solved, because the unknown next point y_{n+k} of the numerical solution is needed to evaluate f_{n+k}.

A k-step method is called *linear*, if Φ is linear and equ. (3.1) can be written as

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f_{n+j}. \tag{3.3}$$

Single-step methods are k-step methods with $k = 1$. Most schemes are derived by a collocation approach. For given nodes t_i and values y_i the interpolating polynomial is constructed and integrated. The coefficients of the resulting polynomials become the weights in the integration scheme.

The linear single-step methods are the explicit and implicit Euler methods, which are often used for the sake of simplicity:

$$y_{n+1} - y_n = hf_n, \tag{3.4}$$
$$y_{n+1} - y_n = hf_{n+1}. \tag{3.5}$$

The main issue of this chapter is the stiffness in the ODEs that arise from textile animation. Therefore we are going to introduce the concept of numerical stiffness, although there is no formal definition.

Numerical stiffness is very much related to physical stiffness. A stiff Hookean spring contracts very rapidly, when released after elongation. A stiff linear elastic continuous body behaves likewise after the elongation in one direction. Textiles typically show this physical stiffness. They are very hard to stretch and contract rapidly after an elongation. Physically, stiffness means that the deformation waves have very high frequencies and propagate very rapidly through the material.

We can linearise the solution function that describes such deformation in time and develop it into its eigenbasis to obtain a linear combination of exponentials:

$$Y(t) = \sum_i A_i \exp(-2\pi f_i i t) + O(t^2)$$

with very high eigenfrequencies f_i, corresponding to very large negative eigenvalues $\omega_i = 2\pi f_i$. Physically stiff materials as well as fine spatial discretization usually lead to the occurrence of large frequencies, i.e. eigenvalues with very large real modulus.

Dahlquist's test equation evaluates the behaviour of integration methods with the one-dimensional linear ODE with a variable eigenvalue λ:

$$y' = \lambda y, \qquad y(t_0) = y_0, \qquad \lambda \in \mathbb{C}. \tag{3.6}$$

The correct solution is given by the exponential $y = \exp(\lambda t) y_0$. If the ODE is stable (the stability of an ODE must not be confused with the stability of a numerical solution), i.e. only has eigenvalues with a non-positive real part ($\mathrm{Re}\,\lambda \leq 0$), the solution is bounded. If explicit integration schemes are applied to equ. (3.6), the product of the eigenvalue λ and the step size h has to be small for the numerical solution to be bounded, whereas there is no such restriction for suitable implicit methods.
Those methods that yield bounded solutions for all step sizes and eigenvalues with non-positive real part, when applied to Dahlquist's test equation, are called *A-stable*, and we will only consider A-stable methods as suitable solvers.

Also from the physical point of view it can be explained why in general explicit solvers fail in the case of stiff ODEs. The velocity c of the deformation waves travelling in a solid object depends on the elastic modulus k and the mass density ρ:

$$c = \sqrt{\frac{k}{\rho}}$$

Hence, in stiff materials, which are characterised by a large Young's modulus k, the waves propagate very fast through the object. In one integration step, however, an explicit solver can only transmit forces from one particle to its direct neighbours in the discretization. In order to sample the deformation waves correctly, the step size has to be reduced to synchronise the sampling rate of the solver and the propagation of waves. The propagation speed is used frequently in animation [DDCB01, DG96] to control the step size of an explicit solver (Courant's Condition). In contrary, an implicit solver can propagate deformations and forces through all the system during one time step by solving a system of equations.

3.1.2 Accelerated Root Finding

If we employ implicit solvers, we have to solve a system of equations in each step. We will aim to keep this system linear such that we do not have to solve nonlinear equations.

The fastest methods to solve linear systems are multigrid methods and the conjugate gradient (*cg*) method. The latter requires the matrix to be symmetric and positive definite. In this application we will meet these conditions, and, because

it is more flexible and constraint enforcement can be built in neatly, we will use a preconditioned *cg* method to solve the linear systems. The number of iterations to compute a solution depends on the condition number $\kappa(A)$. In order to reduce the error by a factor ϵ in the energy norm induced by A so that

$$\|x - x_k\|_A \leq \epsilon \|x - x_0\|_A,$$

k *cg* iterations must be taken, where k is the smallest integer number k that fulfils

$$k \geq \frac{1}{2}\sqrt{\kappa(A)} \cdot \ln(\frac{2}{\epsilon}). \tag{3.7}$$

This condition number can be improved by preconditioning of matrix.

Iterative methods usually exploit the sparsity of the system matrix. Since each particle only interacts with a small number of neighbours, only very few entries in the system matrix are not zero. A sparse matrix data structure exploits this fact and only iterates over the nonzero elements in every matrix operation. Therefore, the complexity of a matrix vector multiplication is $O(N)$, where N is the number of particles. An iterative linear solver benefits from the sparsity, and the costs of each iteration depend only linearly on the system dimension.

In general, however, the equations are nonlinear and harder to solve. Stiff problems require the nonlinear system to be solved by Newton's method. Newton's method converges quadratically in a neighbourhood of the solution. When an ODE is solved, the solution from the previous solver step can be assumed to be close to the next solution and provides the starting value for the iteration. Unfortunately, Newton's method involves the computation of the Jacobian and the solution of a linear system in each iteration with a system of a large dimension (in our application usually between 1000 and 20000). At first glance, this seems infeasible for interactive animations. However, modified techniques can simplify and speed-up the root finding. The first important technique is the *inexact Newton method*, which solves each linear system only up to a certain tolerance with an iterative linear solver, for instance the *cg* method. Furthermore, we already have a good starting value from the previous iteration. Therefore, if the convergence rate is fast, usually only few iterations of the linear solver have to be made in each Newton iteration. The second modification is called *simplified Newton method*, which constructs an approximated Jacobian only once and applies it in all iterations. This saves a considerable amount of computation time for the Jacobian.

3.2 Comparison with Alternative Approaches

In order to overcome the stiffness problem in textile simulation, various approaches have been taken and published.

Early work in textile modelling employs standard explicit solvers like Runge-Kutta methods and has to use extremely small time steps. Thus, the simulations of

Breen et al. [BHW94] takes days or weeks. Eberhardt et al. [EWS96] reduce the simulation times by optimising the evaluation of the force function by applying a computer algebra system.

Hauth [Hau99] recognises the necessity to employ implicit solvers for animation. He uses various ODE solver types to speed up the simulations in the system developed by Eberhardt et al. [EWS96]. A number of advanced implicit solvers are adopted to the animation and the time steps can be increased significantly. However, no interactive rates can be achieved. This is mainly due to the fact that the computation of the matrices, analytically or numerically, is still expensive, and also the matrices are not symmetric. Hence, rather expensive iterative solvers like GMRES must be employed to solve the linear system within the Newton's method. In each time step, many evaluations of the MAPLE generated force functions are necessary, and a single solver step is still computationally expensive.

Until Baraff and Witkin presented their cloth animation [BW98], implicit integration had often been considered too time consuming and expensive for the purposes of animations. But Baraff and Witkin simplify the necessary computations dramatically. They use the implicit Euler method and consequently simplify all steps to compute the solution. They formulate nonlinear constraints but only use their linear approximation to obtain a linear system of equations. Then, Baraff and Witkin are able to solve the system efficiently with a *cg* method.

Their system linearises the equations in each time step and only carries out one Newton iteration in order to solve the nonlinear system in the implicit solver. This helps to build a fast system. However, because the nonlinear part is not integrated, an error bound and a certain accuracy cannot be guaranteed, and the movement of the deformable object might be too slow. Another factor that helps to speed up the simulations is the introduction of explicit damping forces. These expedite the solution of the linear systems and are widely ignored by other models.

Volino and Magnenat-Thalmann [VMT00b] use similar methods to solve the ODE. But instead of a matrix representation of the derivatives, they apply numerical differentiation, and the matrices are replaced with operators that describe the system.

Desbrun et al. [DSB99] also use the implicit Euler method. They employ Provot's mass-spring system [Pro95] in order to exploit the properties of the spring force terms. Instead of linearising the whole system, they split it into a linear and a nonlinear part, set the nonlinear part constant, and solve the arising linear system. Desbrun et al. do not aim at solving the equation completely as they do not integrate the nonlinear term correctly. To compensate for that, a correction force is introduced to preserve the combined angular momentum of the system. Since in this work the matrix remains constant for constant material parameters and step size, the inverse matrix is precomputed and the linear system is solved by a matrix multiplication with the inverse matrix. Unfortunately, the inverse matrix is dense and the matrix-vector multiplication can be expected to be more costly than

the solution of the linear system with the sparse original matrix. Furthermore, the usage of a precomputed inverse prohibits a change of the step size h and changes of the material parameters.

Based on Desbrun's work Kang et al.[KCC+00] make some further simplification to avoid solving the linear system. In order to update the solution vector in one step they divide each row by its diagonal entry of the matrix of the linear system. But there is no indication that this method gets somewhere close to the solution of the linear system.

In several mass-spring systems [Pro95, DSB99, KCC+00], another popular idea is exploited. Instead of using a very large spring stiffness that would be appropriate to model clothes, smaller values are used to reduce the stiffness and thus simplify the solution of the ODE. Then, after each time step the system is post-processed if the springs are elongated too much. In this process, iteratively all particle positions are modified such that a certain maximum elongation is not exceeded. This is motivated by the nonlinear properties of textile materials that become stiffer with increasing tension. Such a post-processing is justified for simple mass-spring systems that do not model specific material properties anyway. In more accurate systems, however, physical soundness cannot be guaranteed by this modification of the numerical solution. Moreover, the result depends on the order in which the spring elongations are corrected.

Summarising, all approaches suffer from one problem or another. Explicit schemes have stability problems, all implicit solution schemes that have been proposed are only of order one and hence provide very low accuracy. This results in slow movements and loss of detail, for instance wrinkles. Other schemes even do not solve the system but give some more or less reasonable approximation. In the implicit schemes the objective should be to minimise the number of entries in the system matrix. The approach that will be taken in this chapter does so. Further objectives are to guarantee an error tolerance and to be scalable according to accuracy and speed requirements.

3.3 An IMEX Scheme for Particle Systems

In order to reduce the computational work, we try to lower the costs of the time step in the implicit integration. The basic idea is to apply an implicit time step only to the stiff terms in the ODE, while other terms can be integrated by a fast explicit solver. Such schemes that combine an implicit and an explicit solver are called IMEX schemes.

3.3.1 Implicit-Explicit Schemes

Consider an ODE of the form

$$y'(t) = p\big(t, y(t)\big) + q\big(t, y(t)\big), \tag{3.8}$$

where q incorporates the stiff part of the system and p the nonstiff remainder. Then an implicit integrator is applied to q, and an explicit integrator is applied to p. If q contains all stiff components, the stability properties of the implicit method can be preserved, while the integration of p is simplified. A thorough analysis for combining various methods can be found in the work done by Ascher et al. [ARW95, ARS97].

For a constant step size h we obtain the IMEX formula that combines two linear multi-step methods (compare to equ. (3.3)):

$$y_{n+k} + \sum_{j=0}^{k-1} \alpha_j y_{n+j} = h \sum_{j=0}^{k-1} \beta_j p_{n+j} + \sum_{j=0}^{k} \gamma_j q_{n+j}, \tag{3.9}$$

where the β_j and γ_j are the coefficients of the explicit and the implicit method, respectively. We set $\gamma := \gamma_{n+k}$ and $\Delta y^{(i)} := y^{(i+1)} - y^{(i)}$ to simplify the notation. Equ. (3.9) is restated as root finding problem:

$$G(y_{n+k}) = y_{n+k} - h\gamma q_{n+k} + c_n, \tag{3.10}$$
$$c_n = \sum_{j=0}^{k-1} (\alpha_{n+j} y_{n+j} - h\beta_{n+j} p_{n+j} - \gamma_{n+j} q_{n+j}).$$

In general, this is solved by a Newton's method. In the i-th Newton iteration the linear system

$$(I - h\gamma A)(\Delta y^{(i)}) = G(y^{(i)}), \tag{3.11}$$

has to be solved. Here $A = \frac{\partial}{\partial y} q\big(t_{n+k}, y(t_{n+k})\big)$ is the Jacobian of q and contains the stiff eigenvalues of the system. This method can save a considerable amount of computational work because only a part of the Jacobian has to be computed.

However, the IMEX methods are mostly employed for ODEs in which q is linear with respect to y, i.e.

$$q(t, y) = A(t)y.$$

In this case the system reduces to the linear system

$$(I - h\gamma A(t)) y_{n+k} = G. \tag{3.12}$$

Therefore, the objective is to split the ODE of the particle system into a stiff linear term and a second nonstiff term.

3.3.2 Reduction of the Dimension in Newton's Law

After the semi-discretization of the PDE in chapter 2, the continuous equation of motion (2.3) has been reduced to Newton's law (2.4). This is an ODE of second order, i.e. has the structure

$$x'' = f(t, x, x').$$

For N particles this system has the dimension $3N$. We employ the standard procedure to transform the second order ODE to a first order ODE of dimension $6N$ and introduce the velocity vector v as an the auxiliary variable, which yields the system

$$\begin{bmatrix} x \\ v \end{bmatrix}' = \begin{bmatrix} v \\ f(t,x,v) \end{bmatrix}. \tag{3.13}$$

The Jacobian of the right-hand side is

$$\frac{\partial}{\partial [x,v]^T} \begin{bmatrix} v \\ f(t,x,v) \end{bmatrix} = \begin{bmatrix} 0 & I \\ A_1 & A_2 \end{bmatrix}$$

with $A_1 = \frac{\partial}{\partial x} f\big(t,x,v\big)$ and $A_2 = \frac{\partial}{\partial v} f\big(t,x,v\big)$.

Using this block matrix, the Newton iteration step (3.11) is applied to the ODE (3.13):

$$\begin{bmatrix} I & -h\gamma I \\ -h\gamma A_1 & I - h\gamma A_2 \end{bmatrix} \begin{bmatrix} \Delta x_{n+k}^{(i)} \\ \Delta v_{n+k}^{(i)} \end{bmatrix} = \begin{bmatrix} G_1(x^{(i)}, v^{(i)}) \\ G_2(x^{(i)}, v^{(i)}) \end{bmatrix} \tag{3.14}$$

The upper row $\Delta x_{n+k}^{(i)} = G_1(x^{(i)}, v^{(i)}) + h\gamma \Delta v_{n+k}^{(i)}$ can be substituted into the lower row to yield

$$\big(I - (h\gamma)^2 A_1 - h\gamma A_2\big)\Delta v_{n+k}^{(i)} = G_2(x^{(i)}, v^{(i)}) - h\gamma A_1 G_1(x^{(i)}, v^{(i)}). \tag{3.15}$$

This way the dimension of the system is reduced to the original dimension $3N$ of the second order ODE. In each iteration we only solve for the velocities v and update the positions from the computed velocities with the upper row of equ. (3.14) afterwards. Therefore, G_1 is zero in every iteration and vanishes in equ. (3.15):

$$\big(I - (h\gamma)^2 A_1 - h\gamma A_2\big)\Delta v_{n+k}^{(i)} = G_2(x^{(i)}, v^{(i)}). \tag{3.16}$$

This is the final system that has to be solved in each Newton iteration. If the the function f can be split into a linear stiff and a nonstiff term, only the linear system (3.12) has to be applied to the ODE (3.13) and we obtain

$$\big(I - (h\gamma)^2 A_1 - h\gamma A_2\big) v_{n+k} = G_2. \tag{3.17}$$

3.3.3 Construction of a Split ODE

In this section we reduce the complexity of one time step by reducing the nonlinear system to a linear system using an IMEX method. To keep the derivation simple, here only the explicit and implicit Euler methods are used. Later on, higher order methods will be introduced also.

We note that the stiffness in particle systems that model textiles is due to the linear springs that have been derived from the diagonal strain tensor components in equ. (2.20). Therefore it is sufficient to solve only for these springs implicitly and use the explicit scheme for the remaining forces.

In this section we will denote the particle indices of the positions and velocities by superscripts to distinguish them from the time indices that are written as subscripts. Forces for linear springs between two particles at x^i and x^j are given by

$$F^{ij}(x) = \frac{k_{ij}}{l_{ij}^2}(\|x^j - x^i\| - l_{ij})\frac{x^j - x^i}{\|x^j - x^i\|},$$

where k_{ij} is the elastic modulus of this spring and l_{ij} its rest length. Desbrun et al. [DSB99] split these forces into a linear part

$$F_1^{ij}(x) = \frac{k_{ij}}{l_{ij}^2}(x^j - x^i)$$

and a non-linear part

$$F_2^{ij}(x) = -\frac{k_{ij}}{l_{ij}}\frac{x^j - x^i}{\|x^j - x^i\|}.$$

Here we include f_1 in q, while f_2 is treated by the explicit method as part of p. This is possible because f_2 has a constant absolute value and its Jacobian does not contribute significantly. The solution of the IMEX-scheme remains stable under all circumstances and for all feasible parameters, although f_2 is treated explicitly.

Additionally, we need damping forces to account for energy dissipation. We simplify the expression for the strain rate in equ. (2.28) by omitting the projection onto the edge in order alleviate the implicit integration. These are modelled for each pair of adjacent particles by

$$F_d^{ij}(x) = \frac{d_{ij}}{l_{ij}^2}(v^j - v^i),$$

where d_{ij} is the damping coefficient. These dissipative forces are stiff as well as linear, hence they are incorporated in q. Thus, we get linear forces

$$F_{\text{lin}}^{i,j} = \frac{k_{ij}}{l_{ij}^2}(x^j - x^i) + \frac{d_{ij}}{l_{ij}^2}(v^j - v^i) \tag{3.18}$$

for each spring acting on particle i. Defining matrices

$$(K)_{ij} = \begin{cases} -\sum_{r,r\neq i}\frac{k_{ir}}{l_{ir}^2} & \text{if } i = j, \\ \frac{k_{ij}}{l_{ij}^2} & \text{if } i \neq j, \end{cases}$$

and, analogously,

$$(D)_{ij} = \begin{cases} -\sum_{r,r\neq i}\frac{d_{ir}}{l_{ir}^2} & \text{if } i = j, \\ \frac{d_{ij}}{l_{ij}^2} & \text{if } i \neq j, \end{cases}$$

the linear forces can be written

$$f_{\text{lin}}(x, v) = Kx + Dv. \tag{3.19}$$

These matrices represent the discretized Laplacian operator, which appears as the stiff part of the forces due to diagonal stress elements in equ. (2.20).

The explicit forces contained in p can be arbitrary nonstiff forces. Besides F_2, we integrate the other elastic and viscous forces due to bending and shearing, which are typically several magnitudes smaller compared to tension forces, with the explicit method. External forces due to air resistance, wind, and gravity are not stiff and treated explicitly as well. Hence we include all nonlinear parts in f_{expl}:

$$f_{expl} = \left(\sum_i F_2^{ij} \right)_{j,j=1,..,N} + f_{\text{shear}} + f_{\text{bend}} + f_{\text{external}},$$

where the sum contains all forces F_2^{ij} due to the springs adjacent to a particle j. We conclude this section by stating the split ODE:

$$\rho \frac{d^2x}{dt^2} = f_{\text{lin}} + f_{\text{expl}}. \tag{3.20}$$

3.3.4 An IMEX Euler Solver for Particle Systems

Here we describe how an IMEX first order method is applied to the described particle system. It uses the implicit and explicit form of Euler's method:

$$y_{l+1} = y_l + hp(y_l) + hq(y_{l+1}) \tag{3.21}$$

First, we follow section 3.3.2 and reduce the system from $6N$ coordinates to $3N$ coordinates by applying the integration scheme to equ. (3.17). The implicit Euler method for $\frac{dx}{dt} = v$ yields

$$x_{l+1} = x_l + hv_{l+1}. \tag{3.22}$$

Equ. (3.21) is applied to the split ODE (3.20), and x_{l+1} is substituted with equ. (3.22):

$$\begin{aligned} Iv_{l+1} - h\frac{1}{\rho}(Kx_{l+1} + Dv_{l+1}) &= v_l + h\frac{1}{\rho} f_{\text{expl}}(x_l, v_l) \\ \Leftrightarrow Av_{l+1} &= v_l + h\frac{1}{\rho} f_{\text{expl}}(x_l, v_l) + h\frac{1}{\rho} Kx_l, \end{aligned} \tag{3.23}$$

with a symmetric, positive definite system matrix

$$A = I - h^2 \frac{1}{\rho} K - h\frac{1}{\rho} D. \tag{3.24}$$

This linear system can be solved efficiently by the *cg* method. With sparse matrices in general this is even less expensive than multiplying with an inverse, which can have $(3N)^2$ nonzero elements, because the inverse of a sparse matrix is not sparse in general.

A major advantage of the system matrix is that it only changes when either the step size changes or the material parameters change. Thus, we often save time on the recomputation of the matrix. A recomputation of the matrix is necessary in two cases. First, if adaptive step sizing is used, a change of the time step h requires a recomputation. Second, if a nonlinear stress-strain relation is approximated by a piecewise linear function, the material parameters k_{ij} and d_{ij} may change in each time step.

For the *cg* method a starting value has to be computed such that it predicts the new solution. The predictor extrapolates the new solution vector from the most recent values of the solution. Considering that a single-step integration method is employed, we also only use the last value and compute the predictor assuming constant velocity:

$$v_{n+1} \approx v_n \text{ and } x_{n+1} \approx x_n + h \cdot v_{n+1} \tag{3.25}$$

3.4 Extensions of the IMEX scheme

Although the developed first order IMEX scheme already computes animations quite rapidly, it can be improved further.

Higher order methods will be evaluated with respect to their suitability for the application. They produce more accurate results and can also reduce the computational work. Therefore, an implicit-explicit integration scheme based on a backward-differentiation-formula (BDF) is constructed. Afterwards, we derive an updating scheme for the IMEX methods that leads to fully implicit methods. The integration schemes are compared, and examples are given to demonstrate the efficiency of the integration methods.

3.4.1 Second Order Solvers

So far we have restricted our considerations to the first order Euler methods. The implicit Euler method is A-stable, and arbitrarily large time steps can be taken while stability is guaranteed. However, comparing them to a second order method using step size h, the Euler method has to be used with step size of order h^2 to achieve the same accuracy. Implicit methods in general add artificial damping to the numerical solution compared to the exact solution, whereas explicit methods amplify the oscillations. Hence, the lower the accuracy of an implicit method is the more it seems damped and smooth. Visually, the simulated model moves too slowly or looses detail like wrinkles if too large time steps are taken. In order to alleviate these effects without a reduction of the time step, we examine higher order methods that are A-stable and allow a fast computation of one time step.

Single-step methods do not use history points and compute the solution for the next point only from the previous one. Therefore they are better suited to handle discontinuities, which are common in animation because of the occurring

collisions. However, higher order single-step methods consist of several nested stages. In the explicit case this does not pose a problem. If the method is implicit, however, each stage requires a system of equations to be solved and therefore one step would be too expensive for animation. An exception is the implicit midpoint rule, which is the optimal A-stable single-step method of order 2 or higher that only involves one internal stage:

$$y_{n+1} = y_n + hf\Big(\frac{y_{n+1} + y_n}{2}\Big). \tag{3.26}$$

The midpoint rule constructs a linear function interpolating f at the midpoint $\frac{1}{2}(y_{n+1} + y_n)$ and is the only method of order 2 that is constructed from an order 1 collocation polynomial.

As a stable, fast to compute single-step scheme, the implicit midpoint rule appears to be a good choice of an integration scheme. Unfortunately, it has a flaw which impedes the performance of the midpoint rule. The midpoint rule is not L-stable. A-stability only requires the numerical solution to be bounded. Yet this solution could still oscillate around the correct solution. This does not happen, if the method is L-stable, which guarantees that high frequencies are damped out quickly. [1] This has two consequences for the animation with the implicit midpoint rule. First, the midpoint rule might produce unphysical artifacts. Second, high frequencies that are not damped out of the system reduce the performance, i.e. the iterative solution of the implicit equations converges more slowly.

The implicit midpoint rule can be combined with the explicit midpoint rule

$$y_{n+1} = y_n + hf(y_n + \frac{1}{2}hf(y_n)),$$

which is a second order Runge-Kutta method, to yield a second order IMEX method [ARS97].

The other important class of methods are linear multi-step methods. Comparing them to single-step methods, advantages and disadvantages are exchanged. They are computationally inexpensive, because they only have one stage, i.e. one system of equations that has to be solved per step. On the other hand they involve history points.

The BDF–methods (**b**ackward **d**ifferentiation **f**ormulas) are the most suitable multi-step methods for stiff problems. They are defined by a scheme

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h\beta_k f_{n+k}. \tag{3.27}$$

These integration schemes are based on the construction of a polynomial interpolating the points $y_n, \ldots, y_{n+k}$. BDF-methods were the first to be developed to

[1] The definition of L-stability is given in terms of the stability function, which has not been introduced here, cf. [HE01].

deal with stiff equations, and the variant of order two, BDF(2), is A-stable. Furthermore, they are L-stable. For these reasons, they are among the most popular methods for stiff problems and also seem like a good choice for the ODE of the particle system. For $k+1$ points these methods possess order $k+1$, and only one nonlinear system has to be solved, whereas s nested systems have to be solved for an s-stage single-step Runge-Kutta method.

Unfortunately, collisions are discontinuities of the tangent field of the trajectories. This conflicts with the differentiable collocation polynomials that are constructed for the BDF–methods over several steps. When a collision occurs within the support of such a polynomial, the interpolation cannot be expected to yield valid results. We choose to use BDF(2) from this class as a good compromise between higher order (2 in this case) and good performance with collisions.

From the BDF methods, an IMEX scheme can be easily constructed by extending it with an explicit method extrapolating the nonlinear terms. These schemes are referred to as semi-implicit BDF (SBDF). The Euler-IMEX scheme derived in section 3.3 is the order one variant. Ascher et. al [ARW95] derive the following SBDF scheme of order two:

$$\frac{3}{2}y_1 - 2y_0 + \frac{1}{2}y_{-1} = h[q(y_1) + 2p(y_0) - p(y_{-1})] \tag{3.28}$$

Now we have three candidate IMEX schemes for surface animation. The Euler method, midpoint rule and SBDF(2). All of them comprise an A-stable implicit integrator, they are fast to compute, and their performance in the animation system will be measured by experiments.

3.4.2 Assembly of an SBDF(2) Scheme

Since the BDF formulas appear to be promising, we will construct a complete SBDF(2) integration scheme in this section. Basically, we have to provide the implicit equation for one time step and an iterative solver for the arising linear system including a prediction for its initial value. The handling of collisions is excluded here and will be covered by the next chapter.

The Implicit Equations

The SBDF(2) scheme is given by

$$\frac{3}{2}y_1 - 2y_0 + \frac{1}{2}y_{-1} = hq(y_1) + h(2p(y_0) - p(y_{-1})),$$

where we use the indices $1, 0$, and -1 for the current time step. For adaptive time stepping this formula must be modified to incorporate the step size h_{prev} of the previous time step as well:

$$y_1 - c_1 y_0 + c_2 y_{-1} = hc_3 q(y_1) + hc_3(2p(y_0) - p(y_{-1}))$$

with the coefficients

$$c_1 = \frac{(1+\omega)^2}{1+2\omega}, c_2 = \frac{\omega^2}{1+2\omega}, \text{and } c_3 = \frac{1+\omega}{1+2\omega},$$

where $\omega = \frac{h}{h_{\text{prev}}}$. Following section 3.3.2, we apply equ. (3.14) to SBDF(2):

$$\begin{aligned} x_1 - c_1 x_0 + c_2 x_{-1} &= h c_3 v_1 \\ v_1 - c_1 v_0 + c_2 v_{-1} &= h c_3 q(y_1) + h c_3 (2p(y_0) - p(y_{-1})) \end{aligned}$$

We substitute x_1 in the second equation and set $\tilde{p} := hc_3(2p(y_0) - p(y_{-1}))$ to obtain the SBDF(2) version of equ. (3.17):

$$v_1 - c_1 v_0 + c_2 v_{-1} = h c_3 q\big(h c_3 v_1 + c_1 x_0 - c_2 x_{-1}, v_1\big) + \tilde{p} \tag{3.29}$$

Again, the force function f is split into a linear part and a nonlinear part as in equ. (3.20), and we set $p := \frac{1}{\rho} f_{\text{expl}}$ and $q := \frac{1}{\rho} f_{\text{lin}}$.

$$f(x,v) = \frac{1}{\rho} f_{\text{lin}}(x,v) + \frac{1}{\rho} f_{\text{expl}}(x,v) = \frac{1}{\rho}(Kx + Dv + f_{\text{expl}}(x,v))$$

Then we have $\tilde{p} = \frac{h}{\rho} c_3 \left[2 f_{\text{expl}}(x_0, v_0) - f_{\text{expl}}(x_{-1}, v_{-1})\right]$ and substitute q in equ. (3.29),

$$\begin{aligned} v_1 - \frac{h^2}{\rho} c_3^2 K v_1 - \frac{h}{\rho} c_3 D v_1 - \frac{h}{\rho} c_3 c_1 K x_0 + \frac{h}{\rho} c_2 c_3 K x_{-1} & \\ - c_1 v_0 + c_2 v_{-1} - \tilde{p} &= \\ (I - \frac{h^2}{\rho} c_3^2 K - \frac{h}{\rho} c_3 D) v_1 - \frac{h}{\rho} c_1 c_3 K x_0 + \frac{h}{\rho} c_2 c_3 K x_{-1} & \\ - c_1 v_0 + c_2 v_{-1} - \tilde{p} &= 0. \end{aligned}$$

Thus, eventually the linear system of equations

$$A v_1 = \frac{h}{\rho} c_1 c_3 K x_0 - \frac{h}{\rho} c_2 c_3 K x_{-1} + c_1 v_0 - c_2 v_{-1} + \tilde{p} \tag{3.30}$$

with the unknown v_1 and matrix

$$A = I - \frac{h^2}{\rho} c_3^2 K - \frac{h}{\rho} c_3 D \tag{3.31}$$

is obtained.

Iterative Solution of the Linear System

In each IMEX step the linear system is solved inexactly, i.e. only as many iterations are taken as necessary for the required accuracy of the ODE solution. The linear system

$$Av = b$$

is stated in equ. (3.30) and solves for the new velocity v.

The matrix is very similar to the matrix derived for the Euler scheme and has the same properties, namely it is diagonally dominant and symmetric. This allows us to employ the *cg* method. From equ. (3.31) we can see that the condition number of matrix A depends on the factors $h^2 \frac{k_{ij}}{\rho l_{ij}^2}$ and $h \frac{d_{ij}}{\rho l_{ij}^2}$. If these factors are small, the matrix is close to the unit matrix (condition number one). If these values grow, the diagonal dominance vanishes in the limit. Therefore, if the stiffness increases, more *cg* iterations have to be taken because the the number of iterations depends on the condition number of the matrix according to equ. (3.7).

Hence, it is even more important to use an appropriate preconditioner in order to improve the convergence rate. Basically, the objective is to replace the system matrix with some approximation of the unit matrix to improve its condition number.

In the implementation an incomplete cholesky preconditioner is applied. An incomplete cholesky decomposition of A is a cholesky composition of A in which all entries are thrown away that are zero in A, i.e. the decomposition matrices only have entries where the sparse matrix A has entries as well:

$$A \approx \tilde{L}^t \tilde{L}$$

With this matrix the linear system is transformed to

$$\tilde{L}^{-1} A \tilde{L}^{-t} \tilde{L}^t v = \tilde{L}^{-1} b.$$

Then, we solve solve a new system

$$\widetilde{A} \tilde{v} = \widetilde{b}$$

with $\widetilde{A} = \tilde{L}^{-t} A \tilde{L}^{-t}$, $\tilde{v} = \tilde{L}^t v$, and $\widetilde{b} = \tilde{L}^{-1} b$. With the exact cholesky decomposition $A = LL^t$, the new matrix can be written as

$$\widetilde{A} = \tilde{L}^{-1} L L^t \tilde{L}^{-t},$$

and approximates the unit matrix according to the objective.

The start value for the *cg* iteration is computed by predicting the new values under the assumption of a constant velocity analogously to the prediction in the Euler method in equ. (3.25):

$$v_{n+1} \approx v_n \text{ and } x_{n+1} \approx h c_3 v_{n+1} + c_1 x_n - c_2 x_{n-1} \tag{3.32}$$

3.4.3 Derivation of a Fully Implicit Scheme

In this section we analyse the structure of the linear system in the IMEX methods and reconsider the implicit-explicit Euler scheme. The matrix of the linear system (3.23) does not have any entries that couple the different coordinate directions, i.e.

we could also solve three independent systems. Changes in one direction cannot be compensated in another direction in the same integration step, because all coupling terms are contained in f_{expl}. They appear only in the single computation of f_{expl} on the right-hand side of the linear system (3.23).

Thus, the accuracy may be improved by updating these coupling terms within one time step. Hence, one time step of the IMEX scheme is carried out to compute a solution vector $y^{(i)}$, then the explicit part of the IMEX scheme is updated with the computed values $y^{(i)}$ and the IMEX scheme step is repeated. This is iterated until convergence is reached. The solution of this iteration becomes the new value y_{l+1}.

Applying this scheme to the IMEX Euler method, we carry out the following steps:

1. Set $v^{(0)} := v_l$ and $x^{(0)} := x_l + hv^{(0)}$.
2. Do one IMEX step by solving the linear system (3.23).
3. Update positions: $x^{(i)} = x_l + hv^{(i)}$.
4. Update $f_{\text{expl}}(x, v)$ with new values $x^{(i)}, v^{(i)}$.
5. Goto 2. and repeat the IMEX step until convergence.
6. Set $x_{l+1} := x^{(i)}$ and $v_{l+1} := v^{(i)}$.

Thus, the linear system is wrapped by a fixed point iteration for the nonlinear terms. In order to explain the convergence of this iteration method, we consider the equation that is obtained by changing the the arguments of f_{expl} from x_l, v_l to x_{l+1}, v_{l+1} in equ. (3.23):

$$Av_{l+1} = v_l + \frac{h}{\rho} f_{\text{expl}}(x_{l+1}, v_{l+1}) + h\frac{1}{\rho} K x_l \tag{3.33}$$

Thus, the explicit Euler scheme is replaced with the implicit Euler scheme, i.e. the IMEX method is turned into a full implicit Euler method. This nonlinear equation can be solved by a simplified Newton's method with an approximated Jacobian. The system matrix A is the Jacobian of the linear term f_{lin} and contains all major stiff components of the system. Thus, this matrix can be used to approximate the complete Jacobian. Now the nonlinear equation (3.33) , which we write as

$$G(x_{l+1}, v_{l+1}) = 0,$$

is to be solved. The simplified Newton's method is applied to this nonlinear equation using A as the approximated Jacobian:

$$\begin{aligned} G(x^{(i)}, v^{(i)}) + A(v^{(i+1)} - v^{(i)}) &= 0 \\ \Leftrightarrow & \\ Av^{(i)} - \frac{h}{\rho} f_{\text{expl}}(x^{(i)}, v^{(i)}) - v_l - \frac{h}{\rho} K x_l + A(v^{(i+1)} - v^{(i)}) &= 0 \\ \Leftrightarrow & \\ \frac{h}{\rho} f_{\text{expl}}(x^{(i)}, v^{(i)}) + v_l + \frac{h}{\rho} K x_l &= Av^{(i+1)} \end{aligned} \tag{3.34}$$

This is equivalent to an IMEX step with $(x_l, v_l) = (x^{(i)}, v^{(i)})$ and $(x_{l+1}, v_{l+1}) = (x^{(i+1)}, v^{(i+1)})$. That means one IMEX step within the fixed point iteration is equivalent to a Newton iteration, and the nonlinear equations of the fully implicit Euler method are solved.

The derived scheme has two major advantages over the full Newton's method. First, A is inexpensive to compute and only changes when either the material constants or the step size change. Second, we reduce the number of entries in the Jacobian to approximately a third of the entries in the sparsity pattern of the full Jacobian. Hence, an iteration of the linear solver only requires a third of the original time. Obviously this is a major speed-up for the integration.

As already mentioned, the matrices K and D represent the discretized Laplacian operator Δ. It is not surprising that they show up here, because the Laplacian appears in the wave equation (2.8) and corresponds also to the first term in the generalised wave equation (2.11) in the one-dimensional case. This means that within each fixed point iteration the linear elasticity problem described by the Laplacian operator is solved, while the reference frame and the nonstiff forces remain constant. The linear system only propagates the linear waves along the isoparametric lines during one iteration. In the next fixed point iteration the nonlinear term in equ. (2.11) is updated, and the linear problem is solved again and so on.

The outer fixed point iteration can also be used to control the step size of the ODE solver. If the convergence of the fixed point method is poor, the time step h is reduced such that the solution of the previous time step is a better start value for current time step and needs fewer iterations.

In our implementation a lower and upper bound on the number of iterations can be specified. If the number of iterations is too large, the time step is increased by a specified factor; if it is too small, the time step is increased. This way the step size is always chosen such that the iteration works effectively. The overall architecture of this updating scheme for the IMEX method is depicted in figure 3.1.

A fixed point iteration not only can extend the IMEX Euler method to the fully implicit scheme. The same mechanism can also lead from SBDF(2) to the BDF(2)

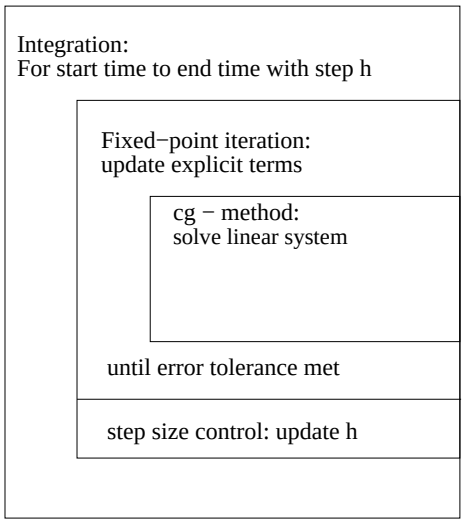


Figure 3.1: Architecture of an implicit integration scheme for animation.

scheme. The BDF(2) scheme only differs from the implicit-explicit scheme by the nonlinear term.

To extend SBDF(2) to BDF(2), we only have to integrate the explicitly evaluated term f_{expl} into the implicitly treated term q by simply setting

$$\tilde{p} := \frac{h}{\rho} c_3 f_{\text{expl}}(x_1, v_1)$$

in equ. (3.30) with matrix A as defined in equ. (3.31). Again, the indices -1, 0, and 1 are used for simplicity. Then, in each time step the equation

$$Av_1 = \frac{h}{\rho} K(c_3 c_1 x_0 - c_3 c_2 x_{-1}) + c_1 v_0 - c_2 v_{-1} + \frac{h}{\rho} c_3 f_{\text{expl}}(x_1, v_1)$$

has to be solved. The unknown v_1 is replaced with the iterates $v^{(i)}$, and in the i-th fixed point iteration the linear system

$$Av^{(i+1)} = \frac{h}{\rho} K(c_3 c_1 x_0 - c_3 c_2 x_{-1}) + c_1 v_0 - c_2 v_{-1} + \frac{h}{\rho} c_3 f_{\text{expl}}(x^{(i)}, v^{(i)}) \quad (3.35)$$

is solved for $v^{(i+1)}$ with the BDF(2) matrix from equ. (3.31). After solving this system, also the position vector is updated in order to reevaluate f_{expl}:

$$x^{(i+1)} = hc_3 v^{(i+1)} + c_1 x_0 - c_2 x_{-1}. \quad (3.36)$$

This procedure, again, can be shown to be equivalent to a simplified Newton's method with approximated Jacobian A. Note that, if only one fixed point iteration is carried out, we obtain a slightly modified SBDF scheme which combines BDF(2) with the explicit Euler integration. Such scheme only has the convergence order one.

The BDF(2) integration is implemented by the following algorithm:

Algorithm 6: BDF2-Solver

time stepping loop
(1) **while** $t < t_{end}$ **do**
(2) $t = t + h$
(3) **if** K,D or h modified
then recompute $A = I - c_3^2 \frac{1}{\rho} h^2 K - c_3 \frac{1}{\rho} h D$
Compute the constant terms
(4) $\widetilde{b} = \frac{h}{\rho} K (c_3 c_1 x_0 - c_3 c_2 x_{-1}) + c_1 v_0 - c_2 v_{-1}$
(5) Initialise the values $x^{(0)}, v^{(0)}$ by extrapolation
fixed point loop
(6) **do**
(7) $b = \widetilde{b} + \tilde{p}(x^{(i)}, v^{(i)})$
(8) Solve $Av^{(i+1)} = b$ such that $\|b - Av^{(i+1)}\| \leq \eta \|b\|$.
(9) Compute $x^{(i+1)}$ using (3.36)
(11) $i = i + 1$
while (no. of *cg* iterations > 0)
end do

Here η is the residual tolerance for the *cg* method. The outer iteration method stops, when the *cg* method exits with zero iterations. That means that the same convergence test is used for the linear and the fixed point iteration. In order to get more control, it is possible to decouple both convergence tests and introduce an extra tolerance parameter for the outer fixed point iteration.

It is also possible to derive a fixed point iteration to implement the implicit midpoint rule analogously to the BDF(2) rule.

3.4.4 Properties of the System and Results

Before presenting experimental results the influence of some parameters and methods is discussed.

Although stiffness is not a problem for an implicit solver in theory, in practice it makes the solution of the involved equations harder, because the number of iterations in each step grows depending on the factors $h^2 \frac{k_{ij}}{\rho l_{ij}^2}$ (nondimensional). This factor will be denoted as numerical stiffness. The performance is reduced, if

- the step size is increased,
- the discretization is refined and l_{ij} reduced,
- the mass density is reduced,

- the physical stiffness k_{ij} is increased.

Also, increasing the damping (viscosity) smoothes the solution function and alleviates the solution of the equations. In this case, the trajectories do not change very much from one step to the next, and fewer iterations are needed in the iterative root finding.

k	d	#updates	*#cg*
$10^4 \frac{N}{m}$	160 $\frac{Ns}{m}$	803	1160
$10^4 \frac{N}{m}$	1600 $\frac{Ns}{m}$	198	538
$10^2 \frac{N}{m}$	160 $\frac{Ns}{m}$	377	566
$10 \frac{N}{m}$	160 $\frac{Ns}{m}$	52	52

Table 3.1: Performance dependence on material parameters

In order to demonstrate the dependence of numerical work on the material properties, an experiment is carried out with varying Young's modulus k, which dominates the stiffness of the material, and damping coefficient d. The draping of a tablecloth over a table is simulated with BDF(2). In the experiment the number of *cg* iterations and fixed point updates is counted. In table 3.1 it can be seen that increasing the Young's modulus leads to a larger number of iterations because higher frequencies have to be sampled. On the other hand, a larger damping coefficient smoothes the trajectories and fewer iterations are necessary.

Figure 3.2: A textile blowing in the wind: Smoothing effects of Euler (lower) compared to BDF(2) (upper)

We have predicted that higher order methods produce more details in the animation. An experiment to demonstrate the increased quality of simulations computed by BDF(2) is set up as follows: A textile sheet with the upper edge fixed blows in the wind. In figure 3.2 the results from the computations with Euler and BDF(2) methods at the same point in time are depicted. Both methods use the same time step size. In the Euler sample the curves are filtered out and the textile swings less than it should - as predicted by the presented theory. Furthermore, BDF(2) is even faster (11.6s) than Euler (14.74s).

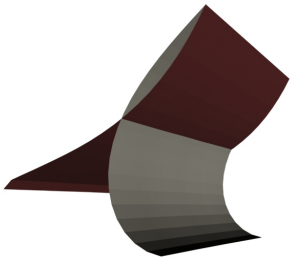


Figure 3.3: A textile fixed along the upper edge draping under gravity. Dark textile: result of BDF(2). Bright textile: result of SBDF(2)

Significant computational work can be saved by the IMEX schemes compared to the corresponding full implicit schemes. Yet, in some cases it is possible that the solution computed by the IMEX scheme is far from the correct solution. Such a case is depicted in figure 3.3. Two solutions are computed for a piece of textile, which is fixed along its upper edge and drapes due to gravity. The first one is computed with the BDF(2) scheme and the second one with the semi-implicit SBDF(2), which only solves one linear system per time step. For a time step of 0.01s, the numerical stiffness is $5 \cdot 10^3$, which is quite realistic for interesting applications. Obviously, the simplified solution is somewhat far from the accurate one. However, in many simulations the IMEX and the purely implicit solutions are hard to distinguish, and the inexpensive IMEX methods are preferable.

In order to compare the performance of the different integration methods, we carry out two experiments. In the first one a tablecloth consisting of 400 particles is draped on a table at a constant time step of 0.01s. Young's modulus is set to $10^4 \frac{\mathrm{N}}{\mathrm{m}}$, the damping constant to $50 \frac{\mathrm{Ns}}{\mathrm{m}}$, the mass density to $1 \mathrm{kg/m}^2$, and the tablecloth is discretized such that the particles are $0.04m$ apart. Thus, the numerical stiffness is approximately $5.6 \cdot 10^4$. The results for 1s of simulation time are printed in the upper rows of table 3.2. Three fully implicit methods, Euler, implicit midpoint, and BDF(2), and two IMEX schemes, Euler IMEX and SBDF(2), are compared. The IMEX schemes count one update per time step. Note that we only measure the solver time but no time measurements are given for collision detection.

In the second experiment, a dress with 1310 particles is simulated. Figure 3.4 shows this dress, which drapes over the body of the avatar from the original position in which it has been set up. Then, five seconds are simulated. Young's modulus is $300 \frac{\mathrm{N}}{\mathrm{m}}$, the damping factor $6 \frac{\mathrm{Ns}}{\mathrm{m}}$, and the mass density $1 \frac{\mathrm{kg}}{\mathrm{m}^2}$. The results show that SBDF(2) and BDF(2) do not only improve the accuracy but also reduce the computational work. Furthermore, although the IMEX methods only solve one linear system per time step, the combined number of *cg* iterations is not always significantly higher than in the implicit schemes.

Solver	#updates	#cg	solver time
tablecloth 1s			
Euler IMEX	100	745	1.69s
implicit Euler	206	1038	3.06s
SBDF(2)	100	304	1,57s
BDF(2)	122	857	2,57s
implicit Midpoint	494	813	4.39s
Dress 5s			
Euler IMEX	500	17198	152.7s
impl. Euler	1654	21424	204.9s
SBDF(2)	500	15360	169.5s
BDF(2)	1431	19307	194,4s
impl. Midpoint	2078	15877	211.3s

Table 3.2: Performance of integration methods on an Intel P3/660MHz

3.5 Conclusions

We have built a numerical framework based on an deliberate choice of integration methods as well as a consequent reduction of computational work.

Among the integration methods, particularly the BDF(2) solver produces results fast with a high quality. IMEX schemes have been employed to reduce the nonlinear systems to linear ones in each time step. Moreover, the computational work has been reduced by a reduction of the number of matrix computations as well as a minimisation of the matrix occupancy.

The techniques derived for the IMEX schemes are extended to purely implicit methods by a fixed point iteration. With these techniques the large nonlinear systems arising from implicit integrators are solved rapidly.

The numerical solution can be adopted to the time and accuracy requirements by setting the *cg* tolerance. Also, an upper limit on the number of iterations per time step can be imposed. The step size is chosen adaptively to optimise the ratio of step size and costs per time step.

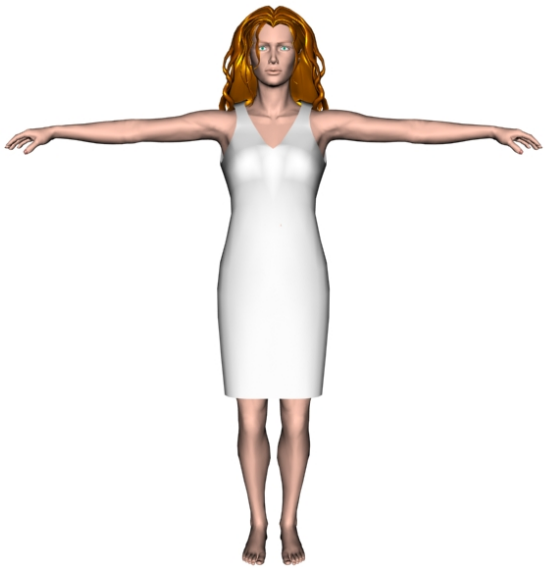


Figure 3.4: Woman wearing a simple dress

Chapter 4

Collision Response

The incorporation of collisions distinguishes animation from classical problems in numerical analysis. The effects cannot be modelled a priori in the differential equation, since the collision reaction depends on the collisions that are detected in each time step at run-time. Therefore, the ODE solver has to incorporate a collision response that is able to respond to detected collisions immediately. This requires the ODE to be modified in each time step.

In this chapter we will first review common techniques for collision response. Then, the application of these techniques in computer animation literature is reviewed. Afterwards, *Reaction Constraints* and a purely constraint based method will be presented. Both are similar in that they filter infeasible directions in all state vectors.

4.1 Physical and Geometric Response

In literature, frequently it is distinguished between geometric and physical collision response. The deformable object is described by a position and velocity field which are governed by the forces exerted on the object. On the one hand, collision forces can be applied in each step of the ODE solver, and this method is referred to as physical response. On the other hand, if we constrain or modify positions, velocities, and accelerations, the number of degrees of freedom in the system is reduced, and this is called geometric correction.

The main problem of constructing collision forces is that collisions are discontinuities of the force field. As the ODE solvers are constructed to work with smooth functions, these discontinuities have to be modelled with care.

A microscopic physical model would not describe the collisions as singularities but as the response to a steep, deflective potential field around the collision objects. However, for a realistic model the gradient of the force field must be very large leading to large stiffness. In physics, these potentials are typically modelled by a term r^n, where r is the distance between the colliding objects, and the exponent

n is 10 or even larger. Although our stable integration methods are able to cope with arbitrary stiffness, the solution of the systems requires more iterations when the stiffness is increased. Hence, physically correct potential fields are too costly.

Furthermore, the collisions of textiles are mostly inelastic. Therefore, the collision model has to include a damping component to prevent the textile from bouncing off the obstacle, and a collision force should consist of an elastic and a viscous component.

Instead of using very steep potential fields, the range of the field can be enlarged such that objects are retarded already at a larger distance from the obstacle. This way, the collisions are smoothed and damped and allow easier handling. But it has to be ensured that the the result be still realistic.

In computer animation, mostly the collisions are modelled as singularities by a modification of the positions and velocities. However, the straightforward repositioning leads to a severe loss of numerical performance, because the modelling of singularities induces extremely high frequencies and the stiffness is increased. Moreover, a repositioning approach is hardly able to model resting contacts.

Using constraints is much more suitable. Constraining positions and particles is equivalent to adding algebraic equations to the system. Therefore, a differential algebraic system has to be solved with varying constraints (algebraic equations) in each step. Here, the collision forces are given implicitly, i.e. the predefined positions and velocities result in constraint forces, which can be computed after each step.

The constraints imposed by the collisions are not holonomous, i. e. they are stated by inequalities rather than equalities. Hence, they have to be switched on and off, when the collision occurs and when the contact is detached, respectively. This switching causes discontinuities the solver has to overcome. Singular collisions can be tackled by finding the point of switching and restarting the solution of the ODE with new initial values computed from the collision response (this is a main issue in rigid body dynamics [ESF98]). However, in the animation of deformable objects we can have a continuous switching in a contact region as the textile moves along the surface of the virtual character. This would lead to a permanent back stepping, because in each time step that is taken a collision occurs. This would bring the simulation to a standstill. Therefore, an approximation is unavoidable, and for each particle the constraints are switched on and off only before a time step and have to persist during that time interval.

4.2 Previous Work

All approaches to model deformable objects have to include some way to model collisions. Terzopoulos and Fleischer [TF88] maintain a purely continuous formulation by using a continuous, deflective force field around collision objects. They

suggest a repulsive force around an object that is defined implicitly by $g(u,t) = 0$:

$$f(u,t) = -c \cdot \text{ grad } g(u) \exp(-\frac{1}{\epsilon}\langle g(u), N(u,t)\rangle),$$

where $N(u,t)$ is the object normal, and c and ϵ determine the shape of the potential.

Most other researchers choose to modify the positions and velocities directly. Eberhardt et al. [EWS96], for instance, implement a reflection model by relocating the positions and velocities after each time step.

Also Provot [Pro97] models a resting contact by estimating the impact force $\tilde{F}$ and setting the velocity $\tilde{v}$ of a colliding particle by exploiting the relation

$$\tilde{F} = \frac{\tilde{v} - v}{h},$$

where v is the velocity before the impact. This way, the velocity is set to counteract the estimated collision force during the following time step.

Baraff and Witkin [BW98] employ a mechanism of constraint enforcement and repositioning to respond to collisions, and we will exploit their ideas in the following. The constraint mechanism is embedded in the implicit solver and does preserve a good performance under collisions. The constraints are used to control the velocities. Additionally, the positions are changed by modifying the velocity-position relation. This means that the equation $\frac{dx}{dt} = v$ is changed to $\frac{dx}{dt} = v + c$, where c is the relocation offset. This modification is incorporated into the implicit equation that is solved in the time step in which the collision occurs. For that, the offset c is included in the substitution leading to equ. (3.15). The integrator can handle these modifications more smoothly than with a simple relocation.

Volino and Magnenat-Thalmann [VMT00a] pick up this idea and extend it to modify positions as well as velocities and accelerations. In a three-phases collision response they change the accelerations of faces far from the collision object, closer faces are deflected by a modification of velocity, and the very close or colliding ones change their positions.

4.3 Reaction Constraints

The collision response described here uses the *Reaction Constraints* introduced by Platt and Barr [PB88]. Reaction constraints are a mechanism to enforce constraints in a dynamic system.

When a collision occurs, a particle is at an infeasible position. In the collision response we can correct this by moving the particle to a target position $\overline{x}$ and target velocity $\overline{v}$. For instance, a linear correction force

$$F = k(\overline{x} - x) + d(\overline{v} - v)$$

moves the the particle gradually to the target position, if no other forces apply. If the collisions have a designated collision direction D, for instance given by the

object normal or line of shortest distance, the collision force should only act in that direction:

$$F = k\langle \overline{x} - x, D\rangle D + d\langle \overline{v} - v, D\rangle D$$

To ensure that no other forces are exerted, the force term for that particle is filtered before. That means the velocities and positions are projected onto the space of feasible directions:

$$F^+ = F^- - \langle F^-, D\rangle D + k\langle \overline{x} - x, D\rangle D + d\langle \overline{v} - v, D\rangle D, \tag{4.1}$$

where F^- is the force acting on the particle before collision response and F^+ the force after collision response. The force in the constrained direction is modified such that only the collision force can act in this direction.

If the forces are updated with ΔF, only this update has to be projected:

$$F^+ = F^- + \Delta F - \langle \Delta F, D\rangle D \tag{4.2}$$

The reaction constraints are switched off if

$$\langle F^-, D\rangle \ > 0,$$

i.e. if the force before the response drags the particle away from the collision. As the collision occurs and resolves, the reaction constraints are switched on and off. The switching induces discontinuities into the system. However, if we use damped correction forces, the numerical integration runs smoothly over these switching points.

The reaction constraints can be easily integrated into an explicit integration or an implicit integration that makes use of numerical differentiation, because such integration only requires the evaluation of the force terms.

However, it is not obvious how the reaction constraints can be integrated into the framework of chapter 3. Therefore, we turn our attention to a different way of implementing constraints.

4.4 Constraint Based Collision Response

Since the numerical framework that we have presented in the previous chapter uses an analytically computed matrix to solve the linear system, the reaction constraints cannot be directly embedded into this framework. We will exploit the constraint enforcement by Baraff and Witkin [BW98] and will see that this mechanism is very similar to reaction constraints.

Given the ODE, positions, and velocities are directly linked by the equation $\frac{dx}{dt} = v$. For instance, in the implicit Euler step this is discretized by $v_1 = \frac{1}{h}(x_1 - x_0)$. Hence, the position and velocity of a particle cannot be constrained at the same time, unless the ODE is modified. Baraff and Witkin introduce such a modification to set positions and velocities. Unfortunately, such modifications are not admissible

for multi-step methods. However, if the collisions are responded to early, there is no need for position correction, and it suffices to constrain velocities.

The constraints are implemented like the reaction constraints. The state vector is projected onto the feasible subspace after each matrix operation. This can be written by another matrix multiplication with the singular projector matrix $P := I - \sum_i c_i c_i^T$, where c is the state vector of the normalised constrained directions, which contains zeros for unconstrained directions. Now we can state the problem as

$$PAv = Pb. \tag{4.3}$$

Here the multiplication on the right projects the forces in b onto the feasible directions, the multiplication on the left guarantees that the solution vector remains in the reduced space.

Although this system is no longer positive definite, the *cg* method still finds the unique solution. The algorithm to solve as given by Baraff and Witkin implements the described method :

Algorithm 7: Constrained cg method

(1) $b = b(x^{l)}, v^{(l)})$
(2) $r = P(b - Av)$
(3) **while** $\|r\| \geq \epsilon\|b\|$
(4) $i = i + 1$
(5) **solve** $M^{-1}z = r$
(6) $z = Pz$
(7) **if** $i = 0$
$\rho = \langle r, z\rangle$
else
$\beta = \frac{\rho}{\rho_1}$
(8) $p = z + \beta p$
(9) $q = Ap$
(10) $q = Pq$
(11) $\alpha = \frac{\rho}{\langle p,q\rangle}$
(12) $v = v + \alpha p$
(13) $r = r - \alpha q$
(14) $\rho_1 = \rho$
end

In this algorithm the projection on the right-hand and left-hand side of equ. (4.3) can be found in lines (2) and (10), respectively. An additional projection is required by the preconditioner M in line (6).

If we compare this *cg* method to the reaction constraints, we notice that the projection in line (10) of the *cg* method applies the same filter that appears in the update formula (4.2) for the reaction constraints. But in the filtered *cg* method

there is no explicit collision response force as in equ. (4.1). Instead, presetting the velocities before to correct the collisions suffices to move the particles to their target positions.

After deriving the mechanism to enforce constraints, we have to employ these constraints to model inelastic collisions. Basically, for each collided particle a target position must be specified. The collision response is described for the case of collisions with a static rigid object. Nevertheless, the response mechanism can be extended to deal with collisions with moving objects and self-collisions.

First, we define a collision area around an object. This area is the set of all points around an object that have a distance smaller than d from the object. The distance d can be specified according to the needs of the collision response for the particular animation. All objects within the collision area are marked as infeasible and collision response is applied to them.

Consider a particle within the collision area. We want the particle to move out of the collision area but not to bounce off. For that purpose we define the collision direction D either as the normal of the collision object or as the line of shortest distance. The direction D will be constrained. If the velocity is set such that in the next step the particle moves out of the collision area, the constraint will be released, and the particle is likely to collide again. This results in a bouncing on the surface. In order to avoid this effect, the velocity is set such that the particle only moves gradually to the boundary of the collision area. We define the target position $\overline{x}$ to be the intersection point of the direction D and the border of the collision area. Then we set

$$\pi(v_{l+1}) = \omega \cdot \frac{1}{h}\pi(\overline{x} - x_{l+1}), \tag{4.4}$$

where π is the projection onto the oriented direction D, and $\omega, 0 < \omega \leq 1$, is a relaxation constant that determines how fast the particle moves to position $\overline{x}$. For $\omega \geq 1$, the particle moves out of the collision area and is released in the following time step, if a single-step method is employed. In multi-step methods, the new position is also affected by the previous velocity.

Figure 4.1 shows a particle that enters the collision area, where the collision is detected. The target point is on the boundary of the collision area and will be reached by moving along the collision direction D.

Since we constrain only one direction of the particle velocity, the particle is still free to move according to the forces acting on it in the other directions, including friction forces. Even if the particle penetrates an object, the velocity is constrained in the collision direction such that the particle is driven back to the surface and can move freely on the surface.

Collision constraints must be released when forces drag the particle away from the collision object. This can be detected by evaluating the residuum in the constraint directions [BW98]. In the filtered *cg* method, the residuum is not required to vanish in the constrained directions, but it is proportional to the exerted constraining

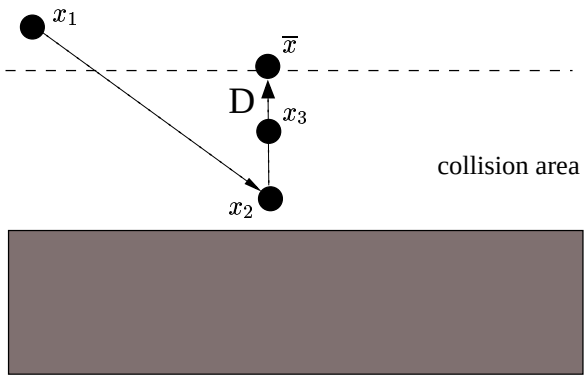


Figure 4.1: Response by constraints

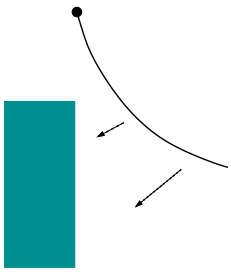


Figure 4.2: A textile patch attached at the upper corners swinging against a box

forces. Hence, it is checked whether these constraining forces drag the particle away from the colliding object. If so, the constraints are released.

4.5 Results and Conclusions

In order to verify that the collision enforcement mechanism does not impede the performance of the numerical solver, the following experiment is carried out: A tablecloth as depicted in figure 4.2 with 400 particles is fixed at two corners and swings against an upright standing box and collides. The required numerical effort in terms of the number of *cg* iterations is plotted in figure 4.3 together with the number of colliding face pairs.

When the textile hits the box the first time, an increase of the number of iterations is unavoidable because the system has to adopt to the impact. Since the impact is strong, there is also a strong response such that the textile moves out of the collision area, collides again, and eventually comes to rest on the box. Here hardly any additional work is necessary. While the number of collisions is very large, the solver has adapted to the contact and runs smoothly with a low number

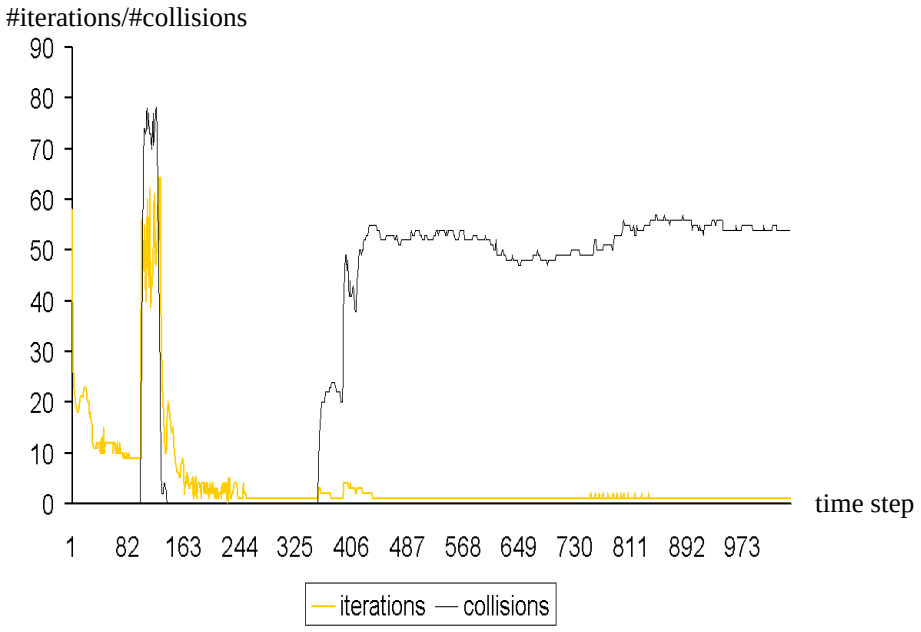


Figure 4.3: Correlation of numerical effort and collisions

of iterations.

This shows that the constrained based collision response mechanism has the required properties for modelling inelastic collisions and resting contact. In particular, the direct integration in the numerical architecture makes it superior to other mechanisms. As collisions are modelled without modification of particle positions, it is more robust and better suited for multi-step methods than methods that change both positions and velocities simultaneously.

Chapter 5

Collision Adaptive Particle Systems

Adaptivity is a common concept to avoid a uniformly very fine discretization in numerical analysis and geometric modelling. Adaptive concepts allow to achieve error tolerances with less computational work. Until here, only adaptivity in time has been exploited by an adaptive selection of the time step size. Meanwhile, the spatial discretization and topology has remained constant, because a constant topology gives us the advantage that data structures, for instance for sparse matrices, do not have to be modified because the system size is constant. But it might be beneficial to modify the spatial discretization during the simulation as well.

This chapter develops a concept for adaptivity in regions that take part in collisions without changing the topology of the system. The key will be the introduction of virtual particles.

We observe that in regions without collisions of the particle system often a coarse discretization is sufficient for an accurate animation. Wherever collisions occur, however, the deformable objects have to fit the object surface they collide with, and the resolution is required to be fine. These observations have led us to develop collision adaptive particle systems that generate new particles where collisions occur.

In this chapter we will first introduce the concept of collision adaptive particle systems. Then the computation of the positions of virtual particles in an adaptive particle system is laid out. Finally, we will outline the modification of forces by the use of virtual particles. This chapter is concluded by some results achieved with the presented method. The collision adaptive method applies to collisions of the deformable surface with rigid objects in its environment. The adaptively inserted particles will be noted as virtual particles, while the original particles are called base particles.

5.1 Adaptivity and Collisions

Adaptivity is commonly used in the numerical solution of PDE's. The grid of the finite element or finite difference method is refined in those areas where the complexity is high. Usually an error measure is used to determine where further refinement is necessary to guarantee a certain error tolerance.

In the animation literature, there are two recent approaches that introduce spatial adaptivity into the simulation. Hutchinson et al. [HPH96] develop a multiresolution particle system that is adaptive to high curvature, i.e. a higher resolution is used where the angles in the system deviate strongly from their rest angles. This approach suffers from its incapability to scale coherently to a different resolution. Also the usage of explicit solvers might result in different results for different resolutions. Hence, the material is likely to show different behaviour after a refinement.

Debunne et al. [DDCB01] develop a multiresolution model for three-dimensional deformable models described by a continuous formulation. The object is modelled by several non-nested meshes, each mesh describing the object at a different resolution. The local error is measured by the norm of the Laplacian of the displacement field. If this error exceeds a certain tolerance, locally the simulation switches to a finer level.

Here we will take a different approach to spatial adaptivity and make the refinement solely dependent on the collisions.

The collision detection can be either solely based on particles or on an accurate computation of face/face distances. In this chapter we will use ray-tracing to detect the collision of particles with other objects only. This method is detailed in the work by Eberhardt et al. [EHH00]. Collisions are detected by intersecting a ray from the old particle position to the new particle position with all objects in the environment. When an intersection is reported, the intersection point and the collision normal are returned (figure 5.1). Consequently, the collision response is restricted to colliding particles.

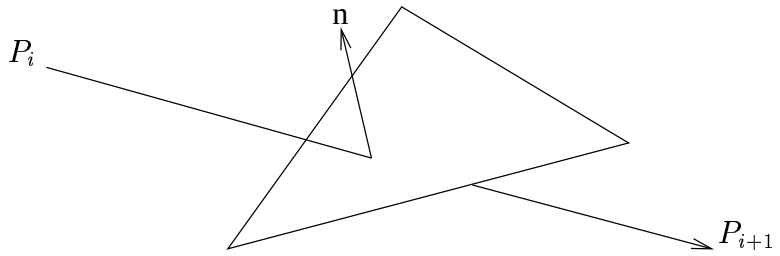


Figure 5.1: Collision detection by ray-tracing. The intersection point and the collision normal n are computed.

If a deformable object collides with another object in its environment and both objects are given as meshes, the occurring collisions can be classified as follows:

1. face/particle collision (figure 5.2, upper)

2. edge/edge collision (figure 5.2, lower)

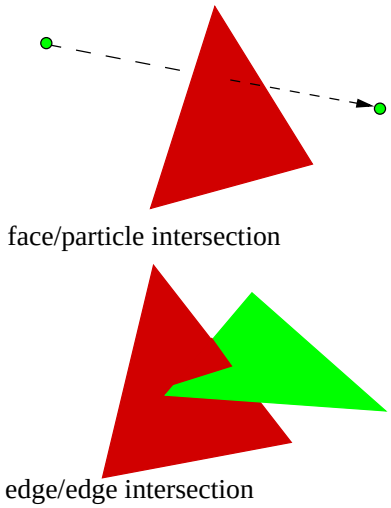


Figure 5.2: Types of collisions

Of course, the ray-tracing collision detection can only detect the first case. Therefore, only such collisions that occur when a particle of the deformable object penetrates the environment (face/particle collision with an animated penetrating particle) will be treated without the adaptive refinement that will be introduced.

Often a minimal offset distance between the particles of the textile and the environment is employed that prevents collisions that are not handled correctly by the system. When less particles are to be used, the minimal distance has to be increased, and the shape of the environment object covered by a coarse textile mesh is not correctly represented. Since our method works on top of a conventional collision detection and response method, it cannot be compared with one of these methods but refines the detection and response. This method extends the coarse collision response of the underlying conventional method such that all types of collisions are treated correctly, and it allows correct simulations requiring only a small number of particles.

The adaptively inserted particles give rise to a new class of particles which are only governed by the collision and not directly by the differential equation. Therefore, these particles will be called virtual particles. Using this adaptivity, we are able to simulate textiles with a very coarse discretization and such reduce the dimension of the ODE significantly. The method is independent of the actual implementation of the particle system.

5.2 Adding Virtual Particles

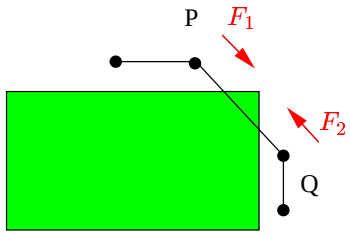


Figure 5.3: A collision of an edge in 2D-space

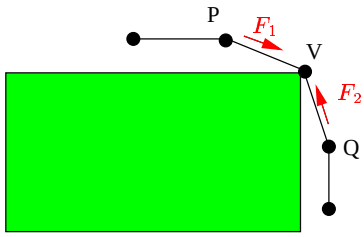


Figure 5.4: Adding a virtual particle to correct penetration

We start by considering the collision in 2D depicted in figure 5.3. The particles of the textile are guaranteed not to penetrate the object by the methods of collision response as described in the previous chapter. An edge collision, however, causes a penetration. We can observe that this failure is due to an insufficient resolution of the deformable model. This can be fixed by adding a new particle to the system (figure 5.4). In general, the edge can intersect the environment arbitrarily often and several particles have to be inserted in the edge as it will be described in the next section.

We could now add new particles to the permanent mesh that describes the deformable object. But this would lead to an increase of the number of particles in each time step, and the particles that were included at time t_n may not be needed at time t_{n+1}, or they are needed at a different position according to the collision at this time.

Hence, we do not compute trajectories from the differential equation for these particles, but we compute the positions of these virtual particles from the collision at each time step. They are only valid for this single time step. Thus the positions of these virtual particles are governed by the current position of the textile and the environment.

Having inserted the virtual particles at a certain time, they can be used to display the current frame, such that the user cannot observe any penetration of the

animated object.

Furthermore, we observe that the length of the modified edge in figure 5.4 has increased due to the inserted point and the model does not match the original physical description, i.e. there is a stretch of the textile without an increase of tension energy. When we use a coarse mesh even cases like in figure 5.5 may occur, where the textile drapes on the box in the environment penetrating the box without any interaction. The textile would eventually fall through the box because there are no forces to prevent this.

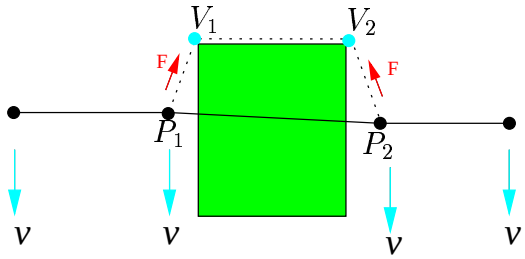


Figure 5.5: No interaction results from collision, unless virtual particles are used

Hence, physical accuracy is improved by integrating the virtual particles into the model. Virtual particles apply forces that act on their neighbours. In the situation of figure 5.4 the forces acting on P are now computed using the topology of the textile mesh in which the neighbour of P is point V instead of Q. In figure 5.5 the textile now is held by the virtual particles at the corners of the box.

In each time step a new topology of the mesh is generated and forces (or energies) are computed for this topology. Note that no forces that act on virtual particles must be computed, because these particles do not move on a trajectory of a differential equation. But the virtual particles exert forces that drag the adjacent particles back together to counteract the stretch due to the collisions.

5.3 Computing Virtual Particles

This section describes how, starting from a triangular textile mesh, all virtual particles can be computed. Since we require a quadrilateral mesh for the physical simulation, this is converted to a triangle mesh by splitting each quadrilateral into two triangles. The method assumes that all base particles are outside the environment mesh, i.e. our method cooperates with a conventional collision detection and response method that prevents the base particles from penetrating the environment. Only after this conventional collision response, virtual particles are computed for each face.

The objects in the environment are required to be given as a mesh (this is not a serious limitation as in practice all geometric objects are given as meshes or can be converted to one).

We aim to guarantee that all edges of the adaptively refined textile mesh be outside the environment mesh. This is achieved by a projection method in which all edges or segments of edges of the environment are projected onto the textile. Those particles and segments of edges of the environment that have penetrated the textile are inserted in the topology of the original textile mesh.

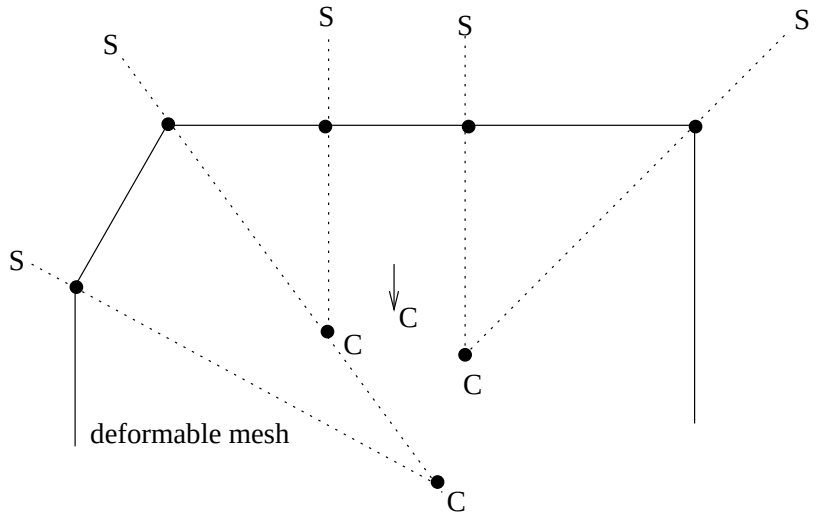


Figure 5.6: Planes S partition the volume above the faces and define centres of projection C_{f_i}

In the following the projection method will be described. The projection P that maps the edges of the environment mesh onto the textile mesh is given procedurally by defining the restriction of P to one textile mesh face f noted as P_f. All face normals n_f are assumed to be normalised.

As a first step we define a centre of projection for each face f: We define a bounding volume by three planes S_i, where S_i is a plane separating f from the adjacent textile face f_i. S_i is given by the normal

$$n_{\mathrm{av}_i} = \frac{n_f \times n_{f_i}}{\|n_f \times n_{f_i}\|},$$

which is defined to be perpendicular to both face normals, and the vertex P_i:

$$S_i : \langle n_{\mathrm{av}_i}, (X - P_i) \rangle = 0$$

Let C_f be the intersection points of these three separating planes. The restriction of P to face f is a central projection given by its centre of projection C_f. The projections are defined such that the space above the faces is partitioned by the bounding volumes. Figure 5.6 illustrates this projection for the two-dimensional space.

If the normals n_{av_i} are linearly dependent, then C_f is a point at infinity and the central projection degenerates to a parallel projection. In either case the projection P_f is defined by a point in projective space represented by its homogeneous coordinates.

We have defined a projection that maps an edge e onto the plane spanned by f. Since we want to restrict the central projection to a single mesh face f and also only parts of the textile mesh that actually penetrate the environment are to be considered, some clipping steps have to be taken.

Each edge $e = \overline{XY}$ of the environment edge is projected onto f with vertices P_1, P_2, P_3 as follows:

1. Compute the heights of X and Y above f: $h_X = \langle n_f, X - P_1 \rangle, h_Y = \langle n_f, Y - P_1 \rangle$. Only points of an edge that have positive height are to be considered. Points with negative height do not penetrate. If a segment of an edge has negative height, it is clipped off and a new end point is computed by linear interpolation. If $h_X < 0$ and $h_Y < 0$, the edge does not penetrate at all and is skipped.

2. Given C_f, project e onto the plane spanned by f.

3. Intersect the projected edge with the edges of face f. Clip off segments that are outside the face f. The end points of the segment contained in f are noted as $\overline{V}, \overline{W}$ (all projected points will be denoted using an overline).

   Four cases can occur (figure 5.7): If no segment of the edge is contained in f, the edge is skipped (5.7(a)). In 5.7(b) the edge is clipped twice. In 5.7(c) only one new end point is computed, whereas in 5.7(d) e lies completely in f. The mapping of two edges e_1 and e_2 is illustrated in figure 5.8.

4. If, say, $\overline{V}$ is not an end point of the projected edge e , $V = P^{-1}(\overline{V})$ is yet to be computed. Let α, β be the barycentric coordinates of V with respect to X, Y. The ratio, which is defined as

   $$\text{ratio}(X, V, Y) = \frac{\alpha}{\beta},$$

   is invariant under affine maps and V can be computed by linear interpolation. In this case of a central projection, however, V cannot be computed by linear interpolation because ratios are not invariant under projective maps, whereas the cross ratio is invariant:

   $$\frac{\text{ratio}(X, Y, V)}{\text{ratio}(X, Y, A)} = \frac{\text{ratio}(\overline{X}, \overline{Y}, \overline{V})}{\text{ratio}(\overline{X}, \overline{Y}, \overline{A})}$$

   This can be exploited to compute V if we have an auxiliary point A and its image $\overline{P}$. Hence, we project the auxiliary point $A = \frac{1}{2}(X + Y)$ onto face f

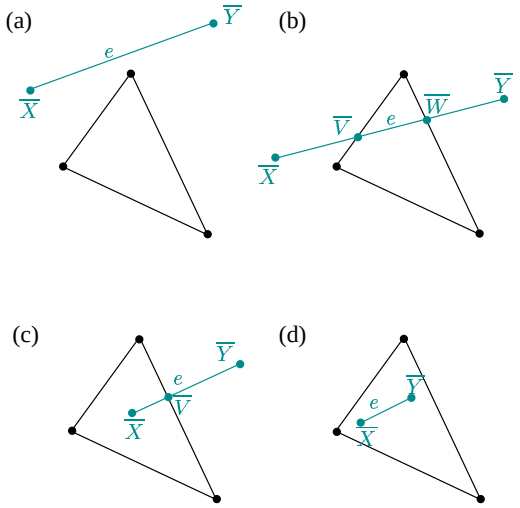


Figure 5.7: Four possible positions of the edge e with respect to the mesh face f

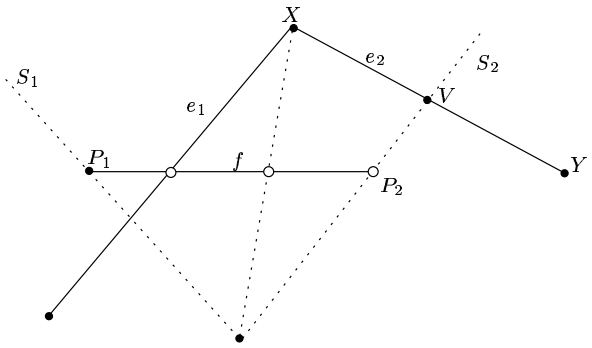


Figure 5.8: Projection P in a two-dimensional cross section

to get $\overline{A}$ and solve the above equation for V.

5. The rays from the end points of an edge to their images under projection P must not intersect other objects. For instance, this case occurs if the textile has not penetrated, but is on the opposite side of the environment object. Then a ray from the opposite side of the environment object would penetrate an environment face facing the deformable object.

   In the situation of figure 5.9, edge e would be handled as an edge penetrating face f, although the textile face is on the other side of the solid environment object. This situation is detected by the rays intersecting the lower face of the solid.

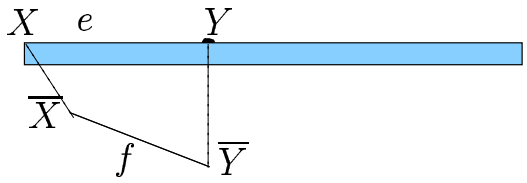


Figure 5.9: Detection of non penetrating edges

The points $\overline{V_i}$ are either on the boundaries of f corresponding to an edge/edge collision or inside f corresponding to a particle/face collision (environment particle penetrating the textile).

Each face f of the textile becomes a sub-mesh that contains the original three vertices (base particles) as well as all newly computed virtual particles and the edges connecting them. At this stage the geometry of this sub-mesh is already known, because we know the positions of P_1, P_2, P_3 and all virtual particles V_i. The topology is yet to be computed by a constrained Delaunay triangulation, and for this we will use $\overline{V_i}$.

First the new boundary topology is obtained by inserting the new vertices that lie on the boundary of f in the face boundary in the correct order. One component of their barycentrics with respect to P_1, P_2, P_3 is zero and therefore the barycentrics can be used to sort the vertices that lie on the boundary.

The line segments in the face as well as the face boundary are constraints to the topology of the sub-mesh. We perform a constrained 2D Delaunay triangulation of P_1, P_2, P_3 and all $\overline{V_i}$ that lie in f in the plane of f (in our implementation we use a very stable triangulator by Shewchuk [She96]). Then the topology computed in 2D is combined with the 3D geometry to yield the sub-mesh, i.e. we replace $\overline{V_i}$ with V_i.

Finally all sub-meshes (faces) are merged to one single mesh that contains all base and virtual particles and all edges connecting them.

An example of the projection is given in figure 5.10. Those edges above the triangle penetrate the textile. Edge e_3 only partly penetrates the face, and only the penetrating segment of e_3 is considered. Vertex V_1 is projected onto $\overline{V}_1$ and becomes an interior point. The edge/edge collisions give new vertices $\overline{Q}_1$ and $\overline{Q}_2$, while $\overline{Q}_3$ is the point of e_1 where e_3 is clipped off.

A simple example of a collision is depicted in figure 5.11. The four particle system has edge/edge collisions with the upper right edge of the box. Three virtual particles have been inserted into the mesh of the textile due to these edge/edge collisions.

Using acceleration structures is necessary, because the projection method has a

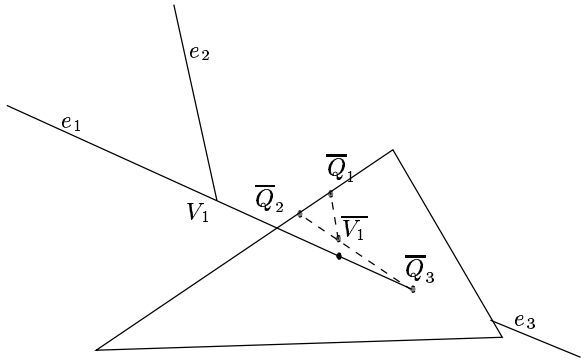


Figure 5.10: Edges projected onto a face

complexity of $O(\#\text{edges}_{\text{env}} \cdot \#\text{faces}_{\text{textile}})$ as all edges are projected onto all faces.

In order to reduce the number of edge/face pairs for which a projection has to be carried out, the same algorithms as used for collision detection for deformable objects can be employed. These algorithms have been widely discussed in literature (e.g. Volino et al. [VMT94], Eberhardt et al. [EHH00]). In our implementation we are using a grid based acceleration. The faces of the deformable object are sorted into a grid. Then for each edge those grid cells that the edge is contained in are selected. Thus, the projection only has to be performed for those faces of the textile that are in these grid cells.

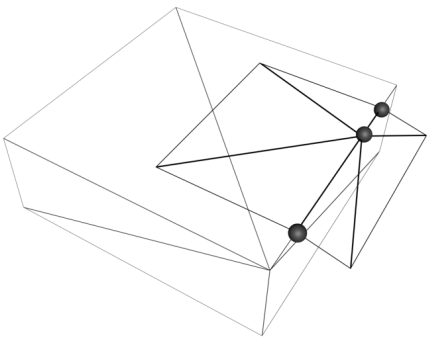


Figure 5.11: A four particle mesh falling onto a box

The complexity of the triangulation step is $O(\#\text{faces}_{\text{textile}})$, because the triangulation is local and its cost is limited by the maximum number of particles to be inserted into one face.

5.4 Physically Interacting Virtual Particles

As mentioned above virtual particles are not only used for visualisation but also exert forces acting on their neighbours. Otherwise the textile would appear unnaturally stretched when virtual textiles are inserted. In this section we therefore describe how the virtual particles can interact with their neighbours in a regular quadrilateral mesh.

If the triangle mesh computed in the previous section were used for the physical simulation, the number of neighbours of a particle would change in each time step and the data structure would have to adopt to a new dimension each time. This is avoided by the following technique: We find it sufficient to employ the virtual particles on the face boundaries only for the physical simulation and ignore such virtual particles that lie inside a mesh face. These virtual particles on the edges exert forces on their neighbours like base particles do. However, no forces are exerted on the virtual particles themselves. This might seem to contradict the principle of *actio equals reactio*; however, the forces exerted by the virtual particle are considered as collision forces exerted by the rigid collision object, the mass of which is too large to react to collisions with the textile. By this principle, we do not have to deal with a changing topology, and locally we only have to change positions and velocities.

The rest distances and rest angles where the system is at an equilibrium state must be adopted as well to fit the new topology and geometry.

We will explain this looking at the following example: Assume that particle P_0 has four neighbours P_1, P_2, P_3, P_4, and two virtual particles V_1 and V_2 are computed on edge $\overline{P_0, P_1}$ as shown in figure 5.12. The forces acting on P_0 are then computed by using V_1 instead of P_1 as a direct neighbour of P_0.

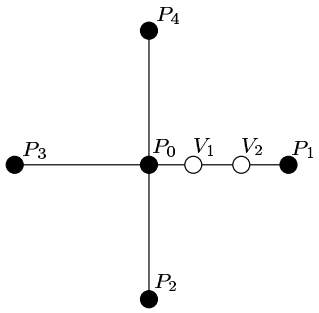


Figure 5.12: Local topology at P_0

Hence, in the implementation we only have to replace the position and velocity of P_1 with V_1. The velocity of the virtual particles is needed to compute damping forces and is set equal to the velocity of the point in the environment it corresponds to, because it is assumed that this point in the textile will stay attached to that point of the environment. For a static environment all the velocities of virtual points are

set to zero. Likewise virtual particles can replace the base particles P_2, P_3, and P_4, too.

Note that V_1 and V_2 are not actually T-vertices that are usually harmful in the discretization of dynamic systems. The problem occurs if only forces act on one side of the particle and the other one is free. In this method, however, no forces whatsoever act on these particles.

New rest lengths of the springs controlling the tension must be computed. If an edge is split into several segments, the sum of the rest lengths of these segments must be equal to the rest length of the original edge to ensure area preservation. Hence, the rest length r_j of the j-th segment is computed by

$$\frac{l_j}{\sum l_i} = \frac{r_j}{r},$$

where r is the rest length assigned to that edge and l_i is the current length of the i-th segment. This way every time the evaluation function is called each particle is checked for virtual neighbours. If virtual neighbours are found for a base particle they replace the original neighbours, and forces guide the particle into the correct direction and restore the correct edge lengths. Hence, the insertion of virtual particles is a trivial replacement of vertex coordinates.

One might also want to use virtual particles to compute other forces, for instance bend and shear forces. Unfortunately, this would decouple the system because these forces cannot be transmitted from one base particle to the next if there are virtual particles in between.

Note that the insertion of virtual particles does not change the material properties but only refines the discretization. We can imagine that we first split a spring between two particles into two. Using new rest lengths as described above, this does not change the material properties. Then the virtual particle is moved to accommodate the collision such that the modelling becomes more accurate, while the simulation maintains the correct material parameters.

When discussing the effects of inserted particles on the numerical stability, we have to keep in mind that the forces exerted by the virtual particles are equivalent to springs attached to the rigid collision object. These springs function similarly to conventional collision detection and have similar effects as described in chapter 4.

5.5 Results

The collision adaptive techniques are embedded in the particle system system described by Eberhardt et al. [EWS96]. The ODE is solved by the implicit Euler method.

Figure 5.13 shows a cloth modelled by only two triangle patches falling over a box. The penetration in picture 5.13(a) is caused by three edge/edge collisions. Three virtual particles on the boundary are inserted as in figure 5.11 so that this new mesh does not intersect the box (figure 5.13(b)).

Example	#particles	projection	topology	combined
round table	100	38.5ms	38ms	76.5ms
square table	400	24ms	147.5ms	171.5ms
ball	100	150.5ms	83.5ms	234ms

Table 5.1: Execution times for the computation of virtual particles averaged over 1s of simulation time

In figure 5.14 (a) the mesh of a tablecloth that falls over a round table can be seen. Although it only contains 100 particles, the circular shape of the table can be modelled accurately. The final result is shown by picture 5.14 (b).

The third example shows a cloth falling over a ball. This example demonstrates what can be achieved by adaptive simulations. The mesh of the ball is quite complex and consists of 1227 edges. When no adaptively inserted particles are used, a coarse mesh over a ball looks as shown in figure 5.14 (f). The particles on top of the ball are still visible, but the faces have sunk into the ball. Figures 5.14 (c) and (d) show two refinements at different time steps of the simulation. The fine mesh of the ball is merged into the coarse mesh of the textile. Finally, figure 5.14 (e) shows the refined mesh of the textile covering the top of the sphere completely.

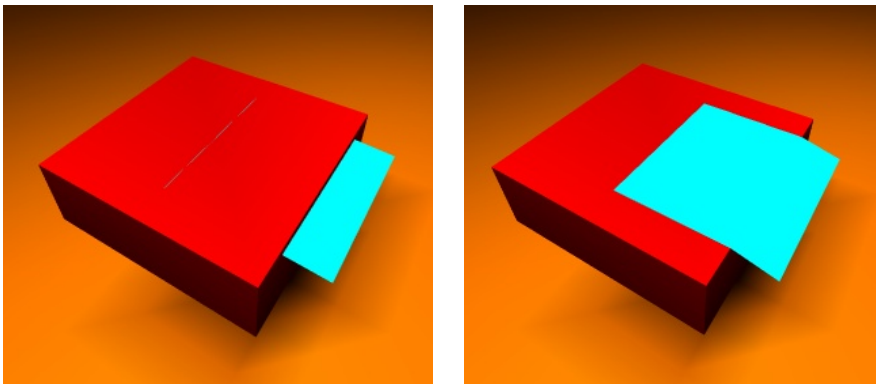
Figure 5.15 shows a comparison between a textile with 2500 particles falling over a table without adaptivity and a mesh with 400 base particles and adaptively inserted virtual particles. The 400 particle mesh, which is obviously much faster to compute, represents a good approximation of the high resolution mesh.

The additional computational costs of the adaptivity are due to the computation of virtual particles as described in section 5.3. In the simulation process we only have to fill in the new coordinates and new rest lengths. In table 5.1 the execution times for the three examples in the figures 5.15 and 5.14 are given. The values are averaged over one second of simulation time on an R10000/180MHz processor. The third column gives the times for the execution of the projection step, the fourth column gives the times for the computation of the topology, and the last column shows the combined values.

Note that for the physical simulation as described in section 5.4 we need not compute the new topology, since only virtual particles on the boundary are used. The overall performance depends on the underlying physical system and the numerical solvers used.

5.6 Conclusion

In this chapter we have presented a method that handles all types of collisions that occur in animation with particle systems. It applies to any conventional particle system that is based on regular quadrilateral meshes. With a very low number of



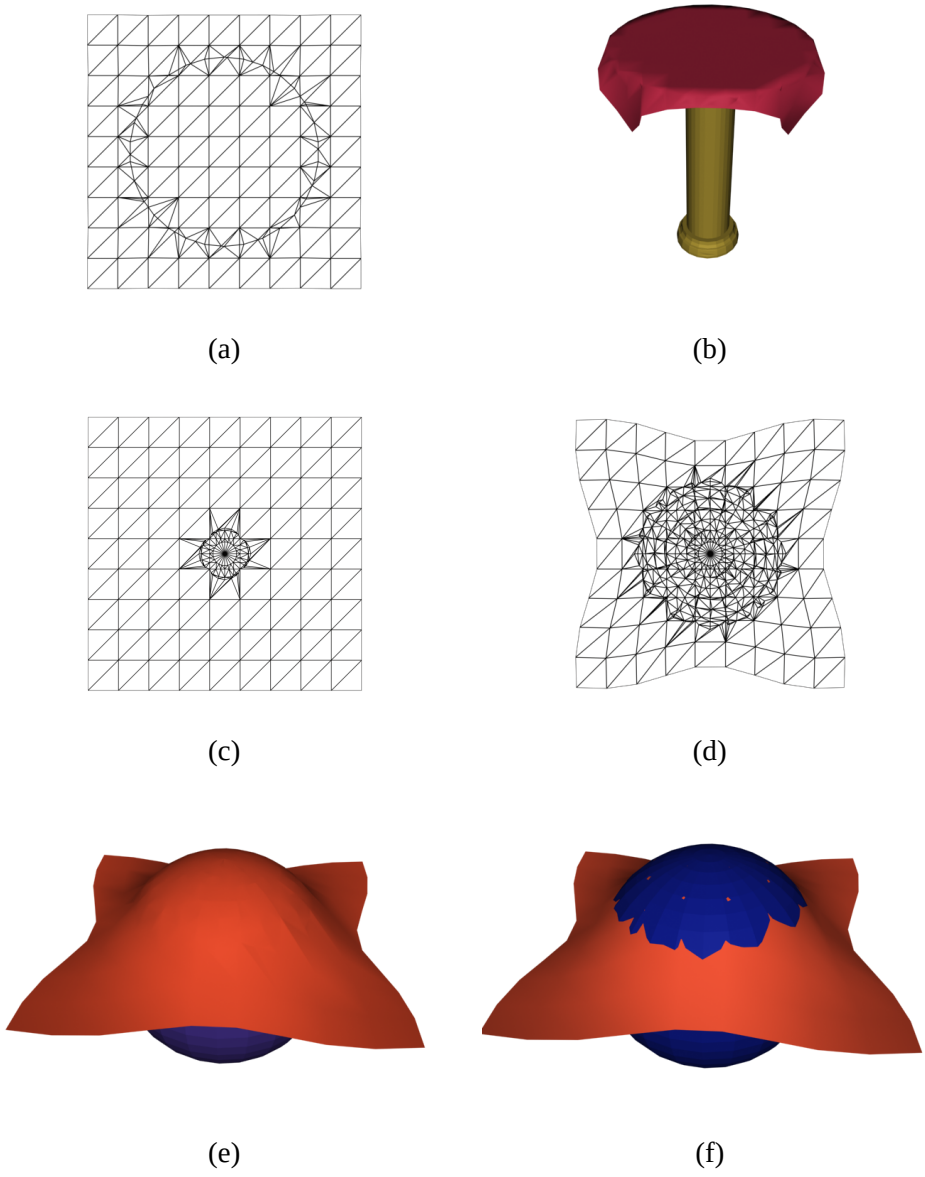
(a) (b)

Figure 5.13: 4-particle mesh draping on a box, (a) without virtual particles, (b) with virtual particles

base particles results showing a high quality are obtained. That means the numerical solver only has to solve low-dimensional systems and the collision detection only has to deal with small meshes.

Unfortunately, there is the drawback of the additional computational costs due to the mesh projection. The computation of virtual particles is rather expensive. The costs of the proposed projection algorithm might compensate all advantages gained by the smaller number of particles. More specialised acceleration methods and algorithmic improvements could further reduce the costs of the projection.

The wrinkles of the simulated textiles partly look coarse, because the adaptivity only applies to the collision regions. In order to alleviate this problem, the collision adaptive method can be combined with other methods that produce realistic wrinkles. For instance, Hadap et al. [HBVT99] map textures onto the mesh to give the impression of wrinkles.



(a) (b)

(c) (d)

(e) (f)

Figure 5.14: Examples with 100 particles: refined particle systems draping over a round table and a ball. Figure (f) shows the same mesh as figure (e) without refinement.

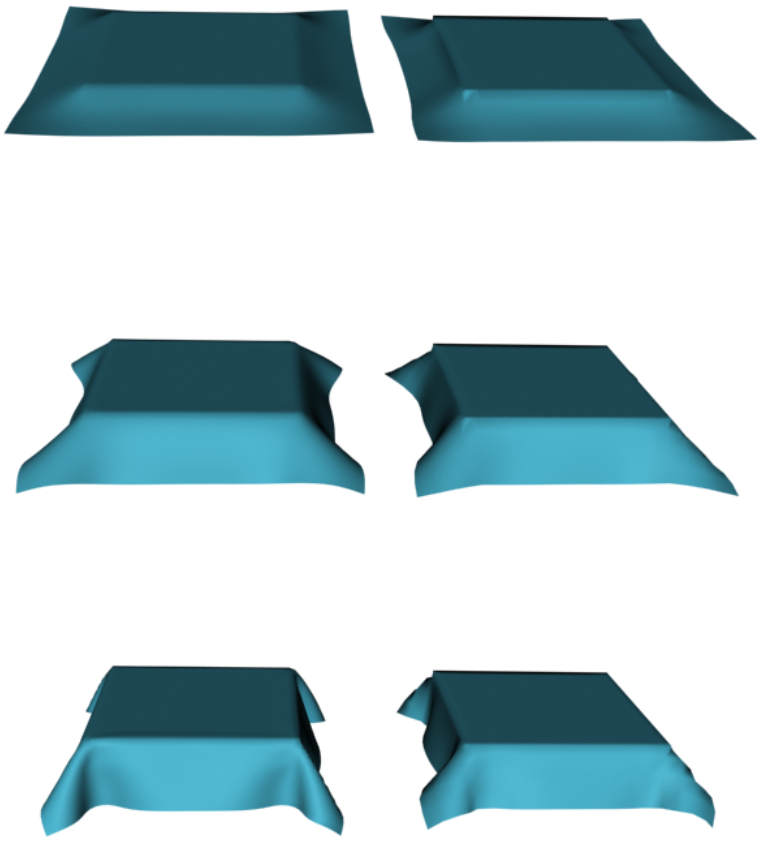


Figure 5.15: Right: 2500 particles without adaptivity, left: fast simulation with only 400 particles and adaptivity

Chapter 6

A Software Architecture for Particle Systems

The techniques described previously in this thesis have been implemented in a cloth simulation software written in C++. In the design, flexibility and efficiency were the major objectives. The design should be easily adaptable to future requirements and extensions, for instance additional forces. Although in this thesis only quadrilateral meshes are studied, the architecture should be flexible enough to handle triangular meshes as well. Clearly, an object oriented design helps to meet these requirements.

Efficiency is as important as flexibility, and we want to ensure that the high number of computations of forces can be carried out fast. This requires the minimisation of overhead and the data to compute these forces to be readily available.

This chapter first describes the supporting software libraries, then the classes that store the forces, and finally the class structure of the simulator kernel. The developed system has been applied to several scenarios of cloth animation, and the results will be shown in the final section.

6.1 Supporting Data Structures

The system design includes two external libraries. A numerical class provides data structures and algorithms for the linear algebra operations in the numerical integration. A mesh class handles all geometry objects in the animated scene and provides an interface for input and output.

6.1.1 Matrix Library

The matrix and vector operations in the numerical framework must be supported by efficient data structures and algorithms. In particular, it is crucial to exploit the sparsity of the matrices. In the implementation we use the *Matrix Template Library* (MTL) [SL99, Not] developed at the University of Notredame. It applies

generic programming to achieve as much flexibility as possible without sacrificing the efficiency.

Most importantly, we use the *MTL* data structures for sparse symmetric matrices. These matrices store the matrix components compressed by the omission of zero entries. They save memory space but also allow to carry out the matrix-vector multiplication and the routine to solve a system with a triangular matrix in linear time, because in these operations only the nonzero elements are accessed. These routines are the most costly operations in the *cg* method.

6.1.2 Mesh Data Structure

All objects of the scene are read as polygonal meshes. For the computation of forces, the topology information has to be extracted from the mesh data structure. However, this will be done only once in a preprocessing step and is not time critical. The design of the mesh data structure therefore is governed by the requirements of the collision detection algorithm, which is not treated in this thesis. During runtime the simulation only employs the mesh data structure to compute the surface normals.

We choose a face based mesh data structure for the sake of simplicity and efficiency. As we assume that the topology does not change and not a lot of navigation in the mesh is necessary, it is competitive with any half-edge data structure. The access routines are simple enough to allow an optimisation of the performance. Each vertex stores its adjacent faces. Hence, the vertex normals can be computed from the adjacent faces in constant time.

6.2 A Cloth Simulation Framework

In this section we outline the class structure of the system. First, we present the classes that contain all information to compute the internal forces in an object. Afterwards, the classes of the simulator kernel are described.

6.2.1 Stress Components

In order to compute the forces rapidly, fast access to the data is crucial. Therefore, all geometrical, topological, and physical data is retrieved from the mesh data structure in the initialisation and stored in objects of the class *StressComponent.* For each kind of stress, i.e. tension, bend, and shear, a stress class is provided. The initialisation allocates an object for each node or edge, respectively. The class hierarchy is depicted in figure 6.1 in UML notation. The symbols +, -, and # indicate the attributes *public*, *private*, and *protected*, respectively.

Class StressComponent The base class contains data that is shared by all kinds of stress. These are in particular an elasticity and a viscosity parameter.

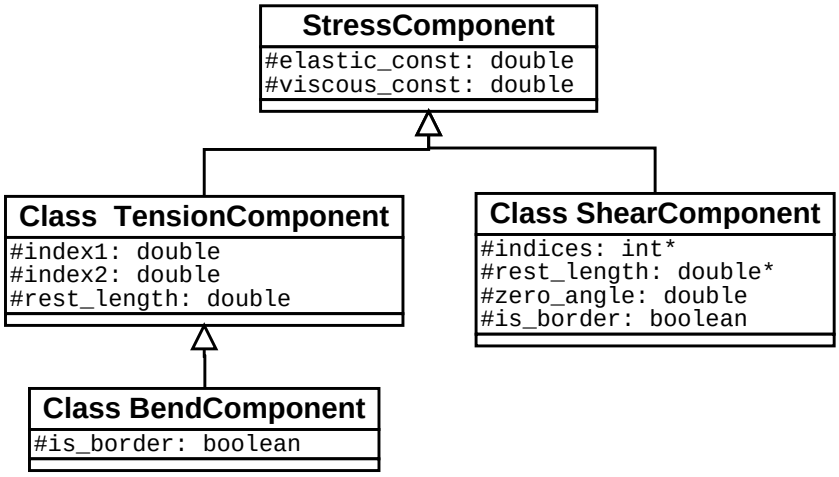


Figure 6.1: Stress component classes

Class TensionComponent There is one instance for each edge. Each instance stores the indices of its two adjacent nodes and the rest length between them.

Class ShearComponent For the computation of shear forces one instance is stored for each node. To compute the shear strain at a node i, its four direct neighbours have to be known in the specified order. Therefore, an instance stores five indices together with four rest lengths. If the central particle is on the border, the missing neighbours are replaced with the index of the central particle. If the mesh is not orthogonal, a value for the scalar product in the equilibrium state as computed in equ. (2.16) can be stored as well.

Class BendComponent The bend stress is very similar to the tension stress as it uses the second partial derivatives. Therefore, it makes use of the same data as class *TensionComponent* except for the border information, which is needed because on the border no bend forces in the normal direction of that border are computed.

The simulation stores a vector of stress components, and in each iteration the forces can be computed efficiently by running over all stress components in that vector and evaluating the forces.

6.2.2 The Kernel Classes

The core of the simulator is situated in two kernel classes: *ParticleSystem* and *Cloth*. Other classes provide information about the materials and the geometry of the objects in the scene. The class structure is shown in figure 6.2.

Class ParticleSystem This class represents an abstract simulation. It contains data structures to control the numerical solver and to store positions and velocities. Also, it contains data structures for constraint handling. The method *solveODE* provides a loop that simulates a specific time interval and calls the *solve* method of the application to carry out one time step. *fixParticle* sets one or more permanent constraints for each particle. Also, a velocity can be prescribed for each constrained direction. This allows to attach a part of the surface somewhere or to prescribe a specific path. Other application classes than *Cloth* could be derived from *ParticleSystem* as well and use its functionality.

Class Cloth This class actually implements the specific surface animation. It contains several ODE solvers: IMEX Euler, implicit Euler, SBDF(2), BDF(2), and the implicit midpoint rule. All of these solvers make use of the *cg* method, which is wrapped by the fixed point iteration, and the stress components. Each solver computes the entries of the sparse system matrix *matrix_pattern.* The tension forces are fully integrated into the solvers as the matrix of the linear system is set up using data from *StressComponent.* All other forces are explicitly computed from the classes *ExternalForces* and the other kinds of *StressComponent.* After each time step, this step size can be modified if the number of fixed point iterations is out of the specified range. The method *solve* provides the interface to the outside and carries out one time step.

Class CollisionDetection The class *CollisionDetection* stands for a container of an arbitrary collision detection algorithm. The implementation of the cloth simulation makes use of an object hierarchy for the detection of proximities [Mez01]. The deformable objects are decomposed into regions, and from these regions a hierarchy is built. This hierarchy is kept in a search tree, and for each node of the tree a bounding volume is stored for rapid collision detection between nodes. Additionally, heuristics are used to reduce the number of collision tests.

All other classes provide the required data for the simulation:

Class ClothMaterial An object of this class specifies one specific material with all its physical parameters. In the construction of an object a material file is read to initialise the data. *Cloth* uses the *ClothMaterial* objects to initialise the *StressComponent* objects with the physical parameters.

Class Environment The class *Environment* provides all scene data. The meshes of the deformable and rigid objects are contained in objects of this class. All bodies in the scene are represented by meshes and read by the class *Collision* for the detection. Queries can be sent to this class to retrieve the velocities and positions of any object at any time in the scene. If the scene is dynamic, i.e. one or more rigid

objects move, for each of these objects a sequence of key-frames is stored. When such a dynamic object is queried at a certain time, the key-frames are interpolated.

This class also provides the functionality to write the frames of the animation to an output device.

Class ExternalForces This class implements various external forces that may act on a deformable object. It computes all external forces described in section 2.6, namely gravity, friction, air resistance, and wind.

6.3 Results and Applications

Finally, we show some results that have been computed with this animation system.

Dress on a Walking Avatar This is an example of clothes in a dynamic scene. The dress has a loose fit and can move rather freely on the body. In the pictures 6.3 the dress is discretized by 1310 particles. The time step size in these animated scenes is constrained by the collisions rather than by the integration method because, if the time step is large, a particle can move a large distance during this step and can penetrate another object without being detected as colliding in time.

Sweater and Trousers A walking man wears a sweater and trousers in figures 6.4 and 6.5. The sweater consists of 5759 particles and the trousers of 4998 particles. Joints like shoulders, knees, and elbows require a particularly high resolution, because a coarse mesh is hardly flexible enough to be wrapped around a bending joint. Note that the trousers are not held in place by constraints. Only the stiffness of the material prevents the trousers from slipping over the pelvis.

Sewing by Constraints The implemented constraints not only can be employed for collision response, they also can be used for the assembly of clothes. An example is shown in in figure 6.6. The dress is assembled from a front and back part, which has already been presented as an example of curvilinear meshing in figure 2.9. These parts are placed in the front and in the back of the avatar in figure 6.6(a). The left and right boundaries of these parts are constrained to move on a path such that the boundaries coincide after a certain time while the gravity is switched off. Then the meshes of the front and back part are merged. At this point, there is a very high tension in the dress, which can be seen in figure 6.6(b). Afterwards the constraints are switched off and the dress relaxes gradually in pictures 6.6(c) and (d).

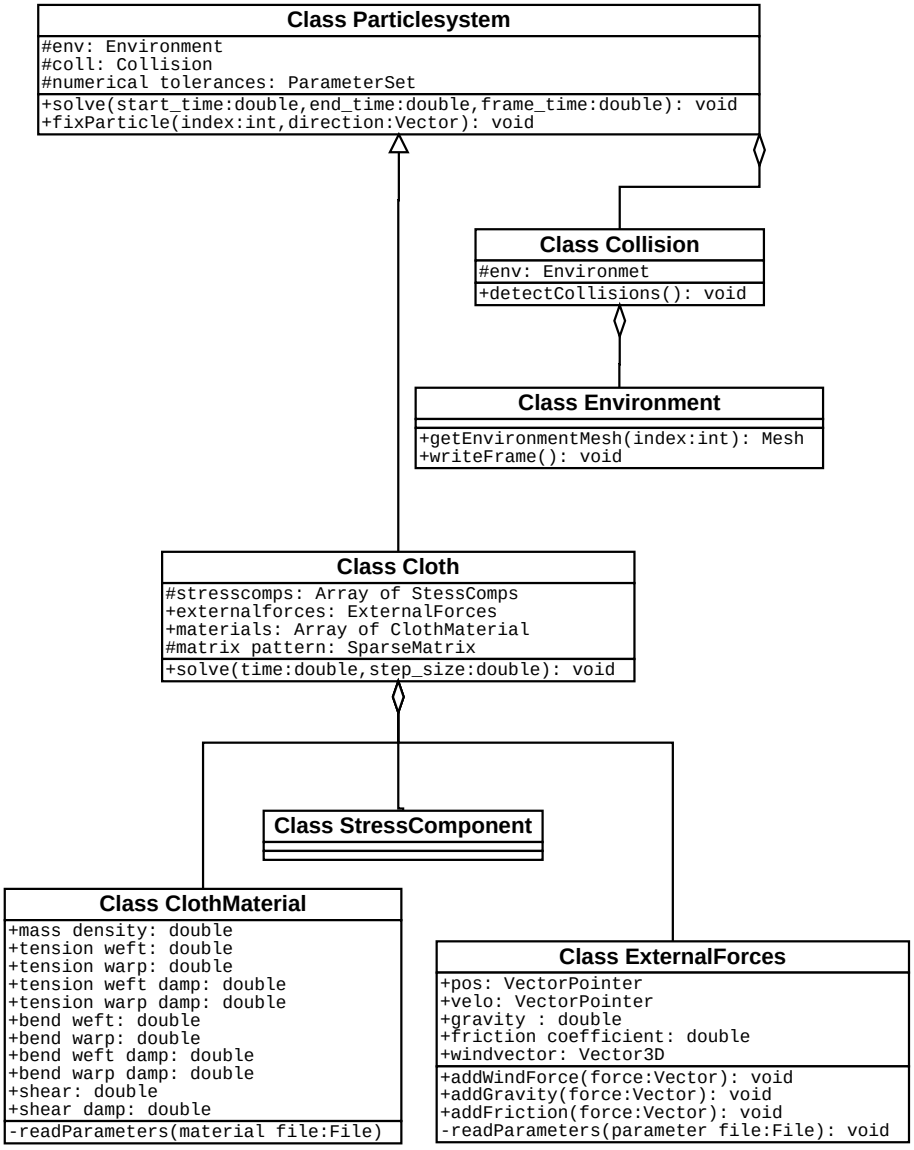
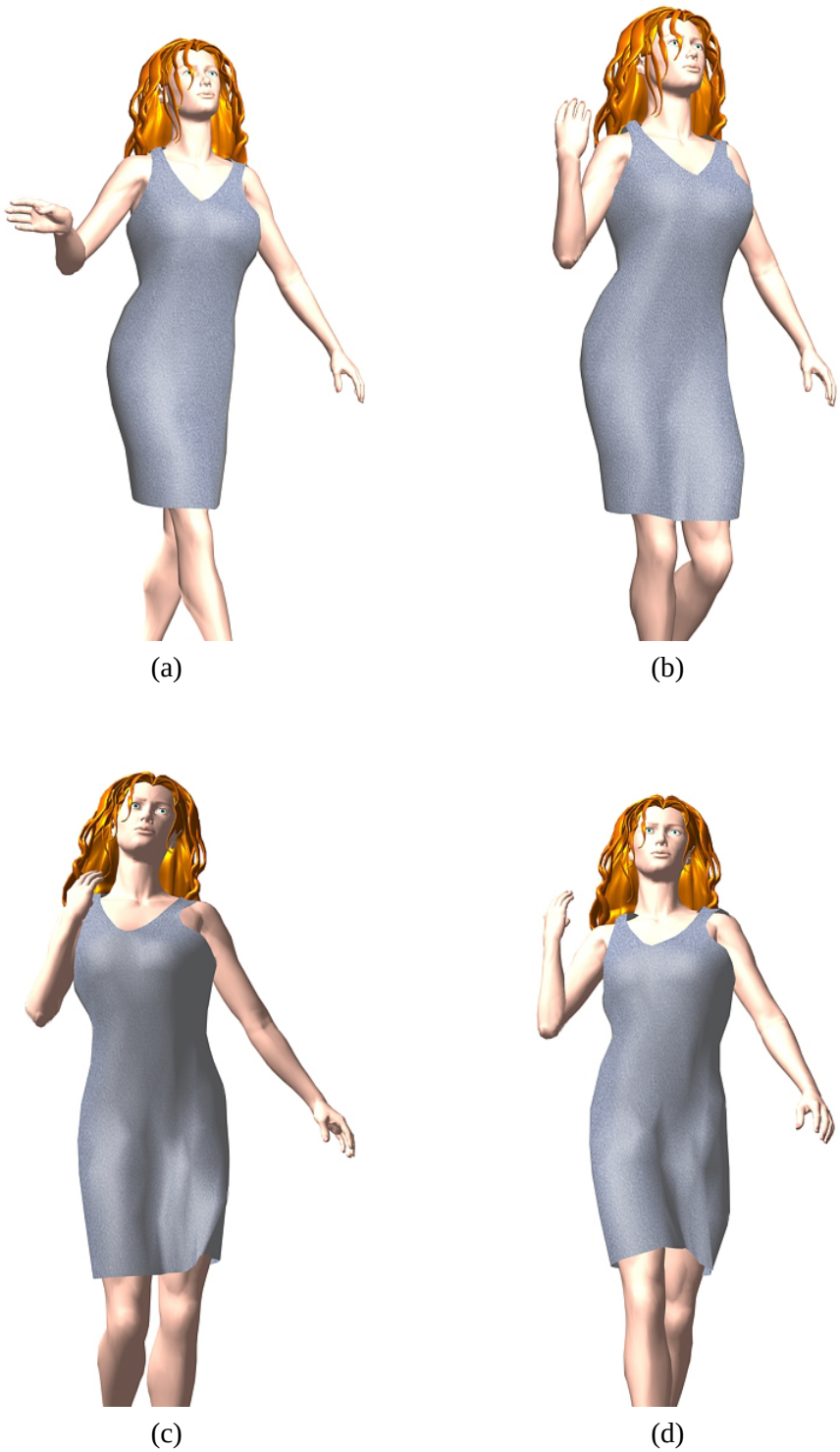


Figure 6.2: Class architecture overview



(a) (b)

(c) (d)

Figure 6.3: Walking woman with a dress

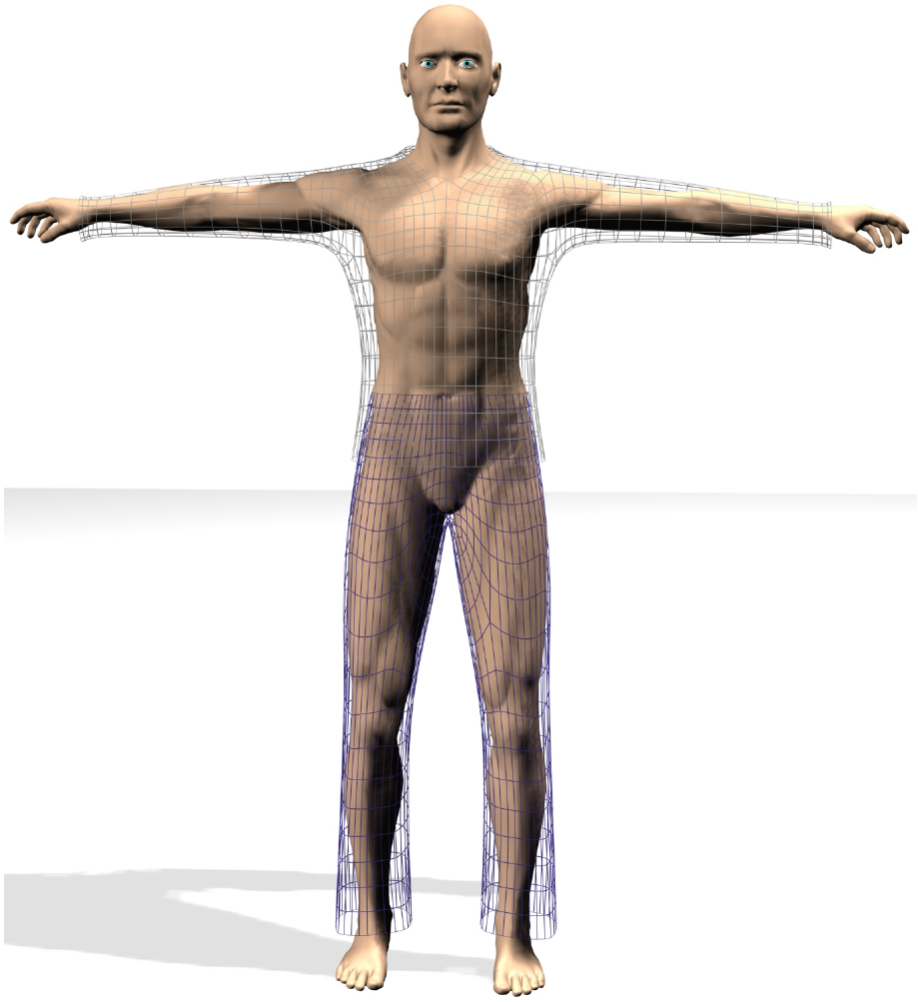
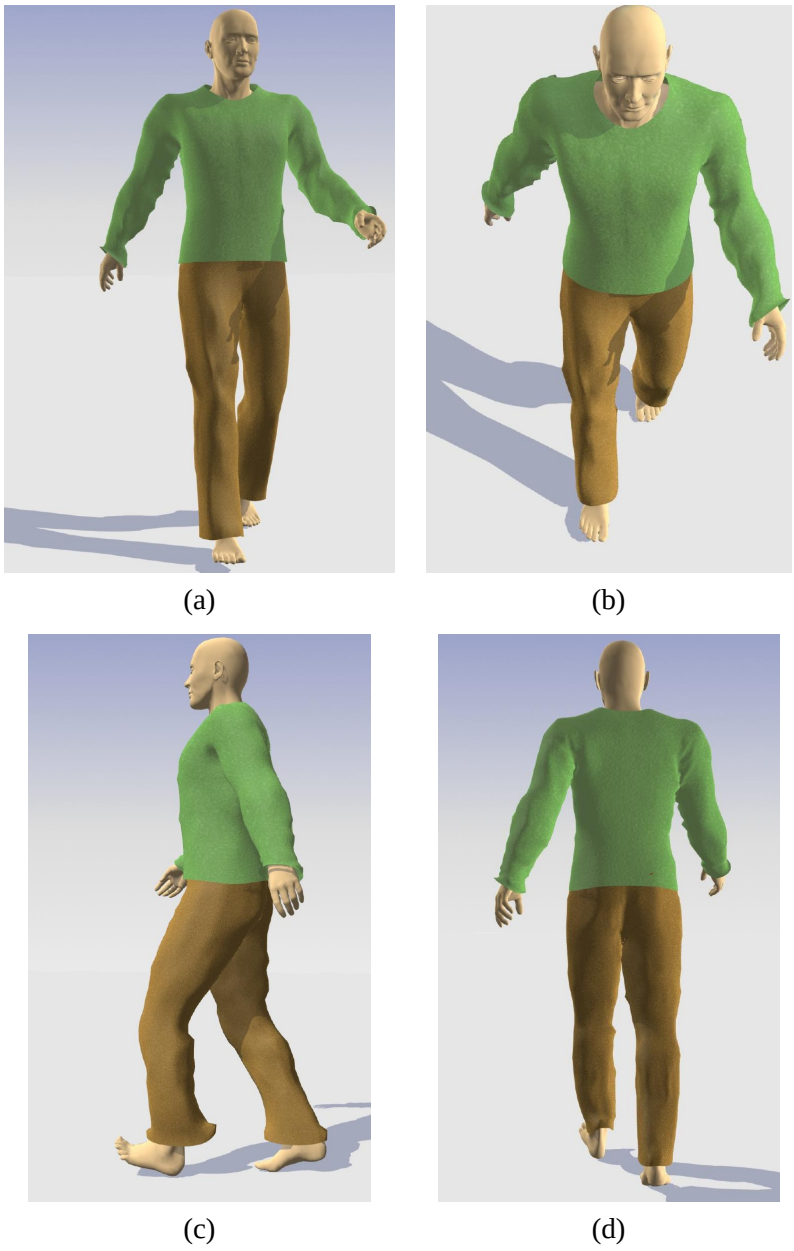


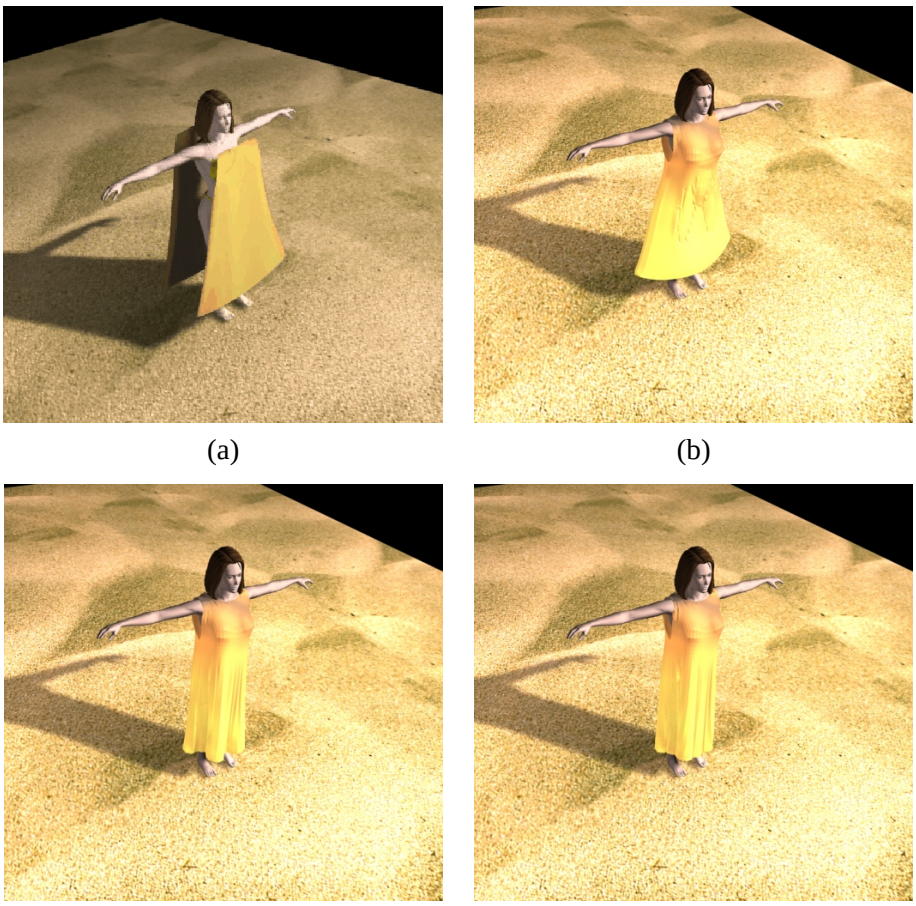
Figure 6.4: Man with sweater and trousers



(a) (b)

(c) (d)

Figure 6.5: Walking man with sweater and trousers



(a) (b)

(c) (d)

Figure 6.6: Relaxation after dressing by constraints

Bibliography

[ARS97] Uri M. Ascher, Steven J. Ruuth, and Raymond J. Spiteri. Implicit–explicit Runge–Kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics: Transactions of IMACS*, 25(2–3):151–167, 1997.

[ARW95] Uri M. Ascher, Steven J. Ruuth, and Brian T.R. Wetton. Implicit-explicit methods for time-dependent partial differential equations. *SIAM J. Numer. Anal.*, 32(3):797–823, 1995.

[Bat82] K. L. Bathe. *Finite Element Methods.* Prentice Hall, 1982.

[BC00] David Bourguignon and Marie-Paule Cani. Controlling Anisotropy in Mass-Spring Systems. In N. Thalman-Magnenat, D. Thalman, and B. Arnaldi, editors, *Proceedings of the Eurographics Workshop on Computer Animation and Simulation (EGCAS-00)*, pages 113–124, Wien-NY, 2000. Springer.

[BH00] David E. Breen and Donald H. House, editors. *Cloth Modeling and Animation.* A K Peters, 2000.

[BHW94] David E. Breen, Donald H. House, and Michael J. Wozny. Predicting the drape of woven cloth using interacting particles. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94*, pages 365–372. ACM SIGGRAPH, July 1994.

[BMG99] Daniel Bielser, Volker A. Maiwald, and Markus H. Gross. Interactive Cuts through 3-Dimensional Soft Tissue. *Computer Graphics Forum*, 18(3):31–38, September 1999.

[Bra97] Dieter Braess. *Finite Elemente.* Springer, 1997.

[BW98] David Baraff and Andrew Witkin. Large Steps in Cloth Simulation. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, 1998.

[BW00] Javier Bonet and Richard D. Wood. *Nonlinear continuum mechanics for finite element analysis.* Cambridge University Press, 2000.

[CDA99] Stéphane Cotin, Hervé Delingette, and Nicholas Ayache. Real-Time Elastic Deformations of Soft Tissues for Surgery Simulation. *IEEE Transactions on Visualization and Computer Graphics*, 5 (1):62–73, 1999.

[Cia92] Philippe G. Ciarlet. *Mathematical Elasticity. Vol. I.* North-Holland Publishing Co., Amsterdam, 1992.

[DCA00] Herve Delingette, Stephane Cotin, and Nicholas Ayache. A Hybrid Elastic Model allowing Real-Time Cutting, Deformations and Force-Feedback for Surgery Training and Simulation. In *Computer Animation Conference*, 2000.

[DDCB01] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic Real-Time Deformations using Space and Time Adaptive Sampling. In *Computer Graphics (SIGGRAPH 01 Proceedings)*, 2001.

[DG96] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles : A new paradigm for animating highly deformable bodies. In R. Boulic and G. Hegron, editors, *Computer Animation and Simulation '96*. Springer-Verlag, 1996.

[DGR00] Jose Miguel S. Diaz, Manuel N. Gamito, and Jose M. Reborado. A Discretized Linear Elastic Model for Cloth Buckling and Drape. *Textile Research Journal*, 70:285–297, 2000.

[DH91] Peter Deuflhard and Andreas Hohmann. *Numerische Mathematik*. de Gruyter, 1991.

[DoC76] Manfredo P. DoCarmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1976.

[DSB99] Mathieu Desbrun, Peter Schröder, and Alan Barr. Interactive Animation of Structured Deformable Objects. In *Graphics Interface*, pages 1–8, June 1999.

[EB00] Jeffrey W. Eischen and Roberto Bigliani. Continuum versus particle representation. In Breen and House [BH00], chapter 4, pages 79–122.

[EDC96] Jeffrey W. Eischen, Shigan Deng, and Timothy G. Clapp. Finite-Element Modeling and Control of Flexible Fabric Parts. *IEEE Computer Graphics and Applications*, 16(5):71–80, 1996.

[EEH00] Bernhard Eberhardt, Olaf Etzmuß, and Michael Hauth. Implicit-Explicit Schemes for Fast Animation with Particle Systems. In N. Thalman-Magnenat, D. Thalman, and B. Arnaldi, editors, *Proceedings of the Eurographics Workshop on Computer Animation and Simulation (EGCAS-00)*, pages 137–154. Springer, 2000.

[EEHS00] Olaf Etzmuß, Bernhard Eberhardt, Michael Hauth, and Wolfgang Straßer. Collision Adaptive Particle Systems. *Proceedings Pacific Graphics 2000*, 2000.

[EG01] Olaf Etzmuß and Joachim Gross. Deriving a Particle System from Continuum Mechanics. Technical Report WSI–GRIS–6–01, Wilhelm-Schickard-Institut für Informatik/GRIS, Universität Tübingen, 2001.

[EH00] Olaf Etzmuß and Michael Hauth. Fast Animation of Deformable Objects with Particle Systems. In *Proceedings Graphiktag*, 2000.

[EHH00] B. Eberhardt, J. Hahn, and T. Hüttner. Raytracing and Collision Detection for Cloth Modelling. Technical Report WSI-2000-10, Eberhard-Karls-Universität Tübingen, April 2000.

[EKK+01] Olaf Etzmuß, Michael Keckeisen, Stefan Kimmerle, Johannes Mezger, Michael Hauth, and Markus Wacker. A Cloth Modelling System for Animated Characters. In *Proceedings Graphiktag*, 2001.

[ESF98] Edda Eich-Soellner and Claus Führer. *Numerical Methods in Multibody Dynamics*. B.G. Teubner Stuttgart, 1998.

[EW97] B. Eberhardt and A. Weber. Modelling the Draping Behaviour of Woven Cloth. *Maple Tech. Journal*, 4(2):25–31, 1997.

[EW99] B. Eberhardt and A. Weber. A Particle System Approach to Knitted Textiles. *Computers & Graphics*, 23(4):599–606, 1999.

[EWS96] B. Eberhardt, A. Weber, and W. Straßer. A Fast, Flexible, Particle-System Model for Cloth Draping. *IEEE Computer Graphics and Applications*, 16(5):52–60, September 1996.

[GB01] J. Groß and G. Bueß. VR models for surgical training with realistic biophysical properties. In H. U. Lemke, M. W. Vannier, K. Inamura, A. G. Farman, and K. Doi, editors, *Proceedings CARS*. Springer, 2001.

[GEHB01] Joachim Groß, Olaf Etzmuß, Michael Hauth, and Gerhard Bueß. Modelling viscoelasticity in soft tissues. In *Int. Workshop on Deformable Modeling and Soft Tissue Simulation*, 2001.

[Gro01] Joachim Groß. Personal communications, 2001.

[GWZ91] Matthias Greuel, Felicitas Weisse, and Udo Zastrow. Der Griff eines Gewebes - subjektive Beurteilung und objektive Messung. *B+W/B+M*, Mai 1991.

[Hau99] Michael Hauth. Numerische Verfahren zur Simulation von Textilien, 1999. Diplomarbeit.

[HBVT99] Sunil Hadap, Endre Bangerter, Pascal Volino, and Nadia Magnenat Thalmann. Animating wrinkles on clothes. In *Visualization 99 Conference Proceedings*, Annual Conference Series, pages 175–182. IEEE, 1999.

[HE01] Michael Hauth and Olaf Etzmuß. A High Performance Solver for the Animation of Deformable Objects using Advanced Numerical Methods. In *Proc. Eurographics 2001*, 2001.

[HPH96] Dave Hutchinson, Martin Preston, and Terry Hewitt. Adaptive Refinement for Mass/Spring Simulations. In *7th Eurographics Workshop on Animation and Simulation*, pages 31–45. Eurographics, Springer, 1996.

[HW96] E. Hairer and G. Wanner. Solving Ordinary Differential Equations II. Springer-Verlag, Berlin, 1996.

[Kas95] Michael Kass. An Introduction To Physically Based Modeling, Chapter: Introduction to Continuum Dynamics for Computer Graphics. In *Siggraph 95 Course Notes*, 1995.

[Kaw80] S. Kawabata. *The Standardization and Analysis of Hand Evaluation.* The Textile Machinery Society of Japan, Osaka, 1980.

[KCC+00] Young-Min Kang, Jeong-Hyeon Choi, Hwan-Gue Cho, Do-Hoon Lee, and Chan-Jong Park. Real-time Animation Technique for Flexible and Thin Objects. In *WSCG*, pages 322–329, February 2000.

[KCM00] U. Kühnapfel, H. K. Cakmak, and H. Maass. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers & Graphics*, 24:671–682, 2000.

[KHS01] K. Kähler, J. Haber, and H.-P. Seidel. Geometry-based Muscle Modeling for Facial Animation. In *Proc. Graphics Interface*, pages 37–46, 2001.

[Kli89] E. Klingbeil. *Tensorrechnung für Ingenieure*. BI Wissenschaftsverlag, 1989.

[Lin00] Li Ling. Aerodynamic effects. In Breen and House [BH00], chapter 7, pages 175–196.

[LL89] L. D. Landau and E. M. Lifschitz. *Elastizitätstheorie.* Akademischer Verlag, 1989.

[Mez01] Johannes Mezger. Effiziente Kollisionsdetektion in der Simulation von Textilien, 2001. Diplomarbeit, WSI/GRIS, Universität Tübingen.

[Not] University of Notre Dame. Matrix Template Library. Avaiable via ftp from `http://www.lsc.nd.edu/research/mtl`.

[OH99] J. O'Brien and J. Hodgins. Graphical modeling and animation of brittle fracture. In *Computer Graphics (SIGGRAPH 99 Proceedings)*, pages 137–146, 1999.

[PB88] John C. Platt and Alan H. Barr. Constraint Methods for Flexible Models. In *SIGGRAPH '88 Conference Proceedings*, pages 279–288, 1988.

[Pro95] Xavier Provot. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior. In *Graphics Interface '95*, pages 147–154, 1995.

[Pro97] Xavier Provot. Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garments. In *Graphics Interface '97*, pages 177–189, 1997.

[Ree83] W. T. Reeves. Particle systems — A technique for modeling a class of fuzzy objects. *Computer Graphics*, 17(3):359–376, July 1983.

[SB00] Josef Stoer and Roland Bulirsch. *Numerische Mathematik*, volume 2. Springer, 2000.

[She96] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148, pages 203–222. Springer-Verlag, 1996.

[SK95] Hans Stephani and Gerhard Kluge. *Theoretische Mechanik*. Spektrum Akademischer Verlag, 1995.

[SL99] J. G. Siek and A. Lumsdaine. The Matrix Template Library: Generic Components for High-Performance Scientific Computing. *Computing in Science and Engeneering*, pages 70–78, November 1999.

[SW92] Rober Schaback and Helmut Werner. *Numerische Mathematik*. Springer, 1992.

[TF88] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4:306–331, 1988.

[VG98] Allen Van Gelder. Approximate Simulation of Elastic Membranes by Triangle Meshes. *Journal of Graphics Tools*, 3:21–42, 1998.

[VMT94] P. Volino and N. Magnenat-Thalmann. Efficient Self-Collision Detection on Smoothly Discretized Surface Animations using Geometrical Shape Regularity. In *EuroGraphics*, volume 13, pages 155–166, 1994.

[VMT97] P. Volino and N. Magnenat-Thalmann. Developing simulation techniques for an interactive clothing system. In *International Conference on Virtual Systems and Multimedia '97*, pages 109–118, 1997.

[VMT00a] P. Volino and N. Magnenat-Thalmann. Accurate Collision Response on Polygonal Meshes. In *Computer Animation Conference*, 2000.

[VMT00b] P. Volino and N. Magnenat-Thalmann. Implementing fast Cloth Simulation with Collision Response. In *Computer Graphics International Proceedings*, 2000.

[Vog97] Helmut Vogel. *Gerthsen Physik*. Springer, 1997.

Lebens- und Bildungsgang

11. Dez. 1971	geboren in Gehrden (Han.)
1978 - 1984	Grundschule und Orientierungsstufe Gehrden
1984 - 1991	Matthias-Claudius-Gymnasium Gehrden mit Abschluß Abitur
1991 - 1992	Grundwehrdienst Bundeswehr
1992 - 1998	Studium im Fach Mathematik mit Studienrichtung Informatik an der Universität Hannover und Abschluß als Dipl.-Math.
1994	Auszeichnung des Vordiploms mit dem Preis der Universität Hannover für herausragende Leistungen
1994 - 1995	Studium an der University of Bristol (GB)
1997 - 1998	Diplomarbeit an der Arizona State University (USA)
seit 1999	Wissenschaftlicher Mitarbeiter am Lehrstuhl für Graphisch Interaktive Systeme am Wilhelm-Schickard-Institut für Informatik der Eberhard-Karls-Universität Tübingen (Prof. Straßer)