EUROGRAPHICS 2002

Tutorial T3: Cloth Animation and Rendering

Michael Hauth, WSI/GRIS, University of Tübingen
Olaf Etzmuss, WSI/GRIS, University of Tübingen
Bernd Eberhardt, HDM Stuttgart, University of Applied Sciences
Reinhard Klein, AG “Computer Graphik”, University of Bonn
Ralf Sarlette, AG “Computer Graphik”, University of Bonn
Mirko Sattler, AG “Computer Graphik”, University of Bonn
Katja Daubert, Max-Planck-Institut für Informatik, Saarbrücken
Jan Kautz, Max-Planck-Institut für Informatik, Saarbrücken

Published by
The Eurographics Association
ISSN 1017-4565

The European Association for Computer Graphics
23rd Annual Conference

EUROGRAPHICS 2002

Saarbrücken, Germany
September 2–6, 2002

EUROGRAPHICS
THE EUROPEAN ASSOCIATION
FOR COMPUTER GRAPHICS

Organized by

Max-Planck-Institut
für Informatik
Saarbrücken, Germany

Universität des Saarlandes
Germany

International Programme Committee Chairs
George Drettakis (France)
Hans-Peter Seidel (Germany)

Conference Co-Chairs
Frits Post (The Netherlands)
Dietmar Saupe (Germany)

Honorary Conference Co-Chairs
Jose Encarnação (Germany)
Wolfgang Straßer (Germany)

Tutorial Chairs
Sabine Coquillart (France)
Heinrich Müller (Germany)

STAR Report Chairs
Dieter Fellner (Germany)
Roberto Scopignio (Italy)

Lab Presentation Chairs
Günther Greiner (Germany)
Werner Purgathofer (Austria)

Industrial Seminar Chairs
Thomas Ertl (Germany)
Bernd Kehrer (Germany)

Günter Enderle Award Committee Chair
François Sillion (France)

Conference Game Chair
Nigel W. John (UK)

John Lansdown Award Chair
Huw Jones (UK)

Conference Director
Christoph Storb (Germany)

Short/Poster Presentation Chairs
Isabel Navazo (Spain)
Philipp Slusallek (Germany)

Local Organization
Annette Scheel (Germany)
Hartmut Schirmacher (Germany)

Cloth Animation and Rendering

Michael Hauth[†], Olaf Etzmuss[†], Bernd Eberhardt[‡], Reinhard Klein[¶], Ralf Sarlette[¶], Mirko Sattler[¶], Katja Daubert[§], Jan Kautz[§],

[†]WSI/GRIS, University of Tübingen
[‡]HDM Stuttgart, University of Applied Sciences
[¶]AG "Computer Graphik", University of Bonn
[§]Max Planck Institute for Computer Sciences, Saarbrücken

Abstract

The area of physically-based modeling is situated in the intersection of computer science, mathematics, and physics. The animation of cloth is a particularly interesting application of physically-based modeling, because it aims at fast animation solutions for rather difficult physical problems. Moreover, it addresses one of the major difficulties in creating realistic scenes with virtual actors.

The challenge of computer animation is to break down physical models for complex structures as textiles, approximate them efficiently, and run fast simulations with intelligent numerical methods. Furthermore, interactivity and collisions with other objects in the scene are challenges that have motivated much creative work over the recent years.

The range of methods proposed in literature is quite large. The techniques vary from simplified methods designed for real-time applications to sophisticated methods that were designed to reproduce measured material properties. Rendering cloth is especially difficult because of its complex material properties. Software rendering can deal with these properties fairly easily, once they have been acquired, but remains too slow for interactive applications. Hardware accelerated rendering often provides a way to achieve interactive renderings, unfortunately complex materials aren't directly supported. We will demonstrate how interactive rendering with complex materials can nonetheless be achieved

1. Physical Models

(*Olaf Etzmuss*)

Cloth models have been designed with different objectives. A common objective in computer graphics is to generate pretty and convincing pictures and films. For that purpose physics may be ignored or simplified significantly. A different objective is to preserve physical, measured properties in order to map real materials onto a simulated cloth. This, for instance, is indispensable in e-commerce applications, in which a customer selects clothes based on a simulation. In computer graphics this also should lead to an animation that is fast and allows interaction with a complex scene. But the user is prepared to wait a bit longer for the results to achieve a physically sound simulation.

According to these considerations, we will first start with a model has the former objective. After that we will describe how discrete and continuous models aim to preserve real material properties.

All models have in common that they discretize the cloth by a polygonal mesh. The vertices of this mesh are called particles or (mass) nodes. The mesh topology defines, how the particles interact and exert forces on one another.

1.1. Discrete Models

Given the mesh describing the cloth, forces on each particle are computed depending on its position and velocity and the positions and velocities 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 $\mathbf{x}_i$ is computed by

$$F(x,v) = m_i \cdot \frac{d^2\mathbf{x}_i}{dt^2}. \tag{1}$$

Here x denotes the vector containing all particle positions and v the vector of all particle velocities. Note that, since

$\mathbf{s}(u,v)$	deformed surface
$\mathbf{r}(u,v)$	(local, partial) rest state of surface
$\mathbf{d}(u,v)$	displacement
$\mathbf{x}_i$	particle positions
$\mathbf{v}_i$	particle velocity
ε	strain (tensor)
σ	stress (tensor)
C	elastic tensor
D	viscous tensor
$\langle \mathbf{a}, \mathbf{b} \rangle$	scalar product of vectors **a** and **b**
Δ	Laplacian $\mathbf{s}_{xx} + \mathbf{s}_{yy} + \mathbf{s}_{zz}$
$\mathbf{s}_u$	partial derivative of **s** with respect to u

Table 1: *Notation in this section*

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.

1.1.1. Mass-spring systems

In mass-spring systems, particle interaction is solely modelled by linear springs.

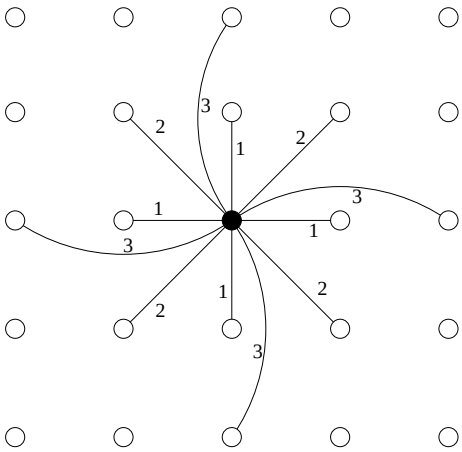


Figure 1: *Provot's mass-spring system with (1) structural springs, (2) shear springs, and (3) bending springs*

Provot[76] 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 shown in figure 1.

Forces by linear springs between two particles at $\mathbf{x}_i$ and $\mathbf{x}_j$ are given by

$$F_{ij}^e(x) = k_{ij}(\|\mathbf{x}_i - \mathbf{x}_j\| - l_{ij}) \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}, \tag{2}$$

where k_{ij} is the elastic modulus of this spring and l_{ij} its rest length. The spring constant depends on the type of the spring. For the structural forces there are very large constants, whereas for the bend and shear forces the springs have small values.

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.

Furthermore, we need viscous forces to account for energy dissipation due to internal friction. These forces damp out kinetic energy and depend on the velocity of the object. It is very popular to model these for each spring by

$$F_{ij}^d(x) = d_{ij}(\mathbf{v}_i - \mathbf{v}_j). \tag{3}$$

Since these terms are linear they are particularly well suited for the numerical integration. However, there are two major disadvantages of this simple term as it also penalises a rigid rotation of a spring. Moreover, high damping of a structural spring prevents the object from bending. Hence, this simplified damping makes the deformable object move rather stiffly. These effects are alleviated by modelling a stiff, damped spring accurately by

$$F_{ij}^d = d_{ij} \frac{\langle \mathbf{v}_i - \mathbf{v}_j, \mathbf{x}_i - \mathbf{x}_j \rangle}{\|\mathbf{x}_i - \mathbf{x}_j\|^2} (\mathbf{x}_i - \mathbf{x}_j) \tag{4}$$

This is just the linear damping term (3) projected onto the direction of the spring. Unfortunately, in many cases this term complicates the implicit time integration.

Finally, in order to run the simulation, we only have to sum up all spring forces and plug them into equation (1).

In several mass-spring systems[76, 22, 53] another popular idea is exploited. It is motivated by a biphasic behaviour of textile materials as shown in figure 2 , i.e. initially the material yields to an exerted stress easily but appears to be extremely stiff in a second phase. This effect is imitated by rather small spring constants that model the first phase. In order to model the second, almost rigid phase, the system is post-processed after each time step 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. Such a post-processing is justified for simple mass-spring systems that do not model specific material properties anyway. Moreover, the result depends on the order in which the spring elongations are corrected.

Although Provot's mass-spring system does not model

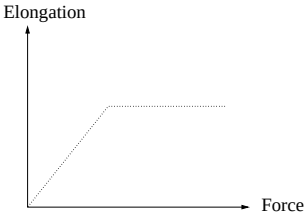


Figure 2: *Biphasic spring modelled by post-correction*

any specific material and is not related to properties of real clothes, it is capable of producing visually very pleasing animations that are sufficient in many computer graphics applications.

1.1.2. Representations of Cloth as discrete Mechanism

In their book on cloth modelling Donald House and David Breen state that *"Cloth is a mechanism, not a continuous material"*. Consequently, some attempts have been made to model clothes by the interaction of discrete threads that are interwoven in textiles.

Some discrete systems that have been developed in computer animation for the animation of clothes and other surfaces have the advantage that they allow fast simulations. In particular, particle systems have been successfully used for rapid animations. We can consider the quadrilateral mesh that is described by the mass nodes and structural springs in Provot's mass-spring system as a network of interwoven threads, in which one thread is given by a chain of structural springs. Different threads can interact at the mass points, where shear, bend, or other internal forces apply. In order to model the interaction of threads, more complex forces than pure spring-forces are added to the system and yield a more general particle system.

Most particle systems use potential functions for tension, bend, and shear energy. These energies are chosen to correspond to standard experiments (Kawabata[57]) to measure textile properties. Hence, the measurements from one experiment are used to model one specific energy function. All energies are modelled on a rectangular grid, where each particle interacts with its four direct neighbours. The grid is aligned with two distinct directions that are apparent in textiles (in woven materials they are called weft and warp direction). The materials show different properties in these directions and each experiment has to be carried out for both directions.

The 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 $\mathbf{x}_0$ is

$$E_t = \sum_{i=1}^{4} \begin{cases} \frac{1}{2}C_{t_1,i}(\|\mathbf{x}_0-\mathbf{x}_i\| - l_i - h_{t_1,i})^3 & \text{if} \quad \|\mathbf{x}_0-\mathbf{x}_i\| \geq l_i \\ \frac{1}{2}C_{t_2,i}(\|\mathbf{x}_0-\mathbf{x}_i\| - l_i - h_{t_2,i})^5 & \text{if} \quad \|\mathbf{x}_0-\mathbf{x}_i\| \leq l_i \end{cases}, \tag{5}$$

where l_i are the rest lengths between particles and $C_{t,i}$ and $h_{t,i}$ are material parameters. They can be used to fit measured data by a piecewise linear curve. The energy is computed from a strain $(\|\mathbf{x}_0-\mathbf{x}_i\| - l_i)$, and the strain-stress relation is modelled piecewise cubic or quintic. 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

$$E_s = \sum_{i=1}^{4} \frac{1}{2}C_s(\phi_i - \frac{\pi}{2} - h_{s,i})^2 \tag{6}$$

and the bend energy as

$$E_b = \sum_{i=1}^{2} \frac{1}{2}C_b(\psi_i - \pi - h_{b,i})^2. \tag{7}$$

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 3.

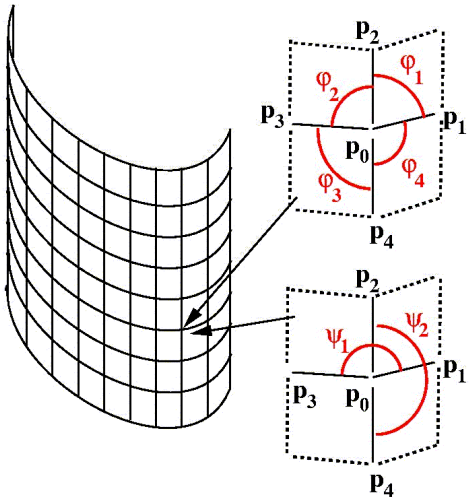


Figure 3: *Shear and bend energy in a particle system*

All derived energies are combined to compute the final forces to be plugged into equation (1):

$$F = -\text{grad}(E_t + E_s + E_b + E_{\text{external}}).$$

In this section, only elastic forces have been discussed. Viscous forces should be modelled in the fashion of section 1.2.6. Note that the well-known cloth simulation by Baraff

and Witkin[2] also represents is a particle system. However, it was designed for triangle meshes and without the objective of fitting to real material data.

1.1.3. Triangular meshes

Until now, we only have considered particle systems based on rectangular meshes. Triangular mass-spring systems are widely used as well and can be constructed with almost the same set of forces. However, their physical properties are hard to control and depend on the topology of the mesh. Furthermore, they usually show a very strong transversal contraction. This motivated Volino[85] to extend the concept of triangular mass-spring systems. 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 on the elongations of all edges of all faces incident to the considered particle. The coefficients of these dependencies are the material constants and allow a flexible modelling of the physical properties.

1.2. Continuous Models

Although clothes are not homogeneous, continuous objects, modelling them as discrete mechanism involves complications. As we cannot represent each single thread in a textile by an edge in the mesh, we have to choose a certain resolution of the object. If we want to be independent of this resolution, we need to represent a patch of textile as a continuous material, which allows us to use low resolution models without loosing basic material properties.

From a continuum model a consistent discretization can be derived. Consistency here means that the computed solution converges to the accurate solution for the continuum when the resolution is increased. That allows us to switch from one resolution to another without changing the properties of the cloth. Therefore we will describe the foundations of the continuous theory and present a particle system that can approximate this theory. This section will be concluded by a paragraph about energy dissipation in cloth.

1.2.1. Descriptions of Strain

Continuum mechanics is the standard theory to describe and model deformable objects, and the following elaborations are based on several text books[7, 9, 63, 81].

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. In the case of a one-dimensional spring, these entities are scalars. 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.

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[23]). The deformation of a regular surface embedded in $\mathbf{R}^3$ is described by a strain tensor with respect to a certain undeformed reference state. In this equilibrium state, denoted by $\mathbf{r}$, the object is not deformed, and the elastic energy is zero. Let $\mathbf{r}$ be parametrised over a domain $U \times V$. Under forces the rest state deforms to a state $\mathbf{s}(u,v)$. The displacement is a mapping $\mathbf{d}$ defined by $\mathbf{d}(u,v) = \mathbf{s}(u,v) - \mathbf{r}(u,v)$ as depicted in figure 4.

The difference of the first fundamental forms I_s and I_r of the current state and the equilibrium state of the object describes the in-plane strain and defines a nonlinear strain tensor[59]

$$\tilde{G} = \tfrac{1}{2}(I_s - I_r) = \tfrac{1}{2}\begin{pmatrix} \langle \mathbf{s}_u, \mathbf{s}_u \rangle & \langle \mathbf{s}_u, \mathbf{s}_v \rangle \\ \langle \mathbf{s}_u, \mathbf{s}_v \rangle & \langle \mathbf{s}_v, \mathbf{s}_v \rangle \end{pmatrix} - \tfrac{1}{2}\begin{pmatrix} \langle \mathbf{r}_u, \mathbf{r}_u \rangle & \langle \mathbf{r}_u, \mathbf{r}_v \rangle \\ \langle \mathbf{r}_u, \mathbf{r}_v \rangle & \langle \mathbf{r}_v, \mathbf{r}_v \rangle \end{pmatrix}. \quad (8)$$

For planar surfaces, the deformation is defined uniquely by the difference of the metrics of these states. As a piece of cloth is a surface in three-dimensional space, the curvature tensors (second fundamental forms) have to be taken into account as well. Terzopoulos and Fleischer[82] developed a model for animated surfaces based on the the energy due to these tensors.

Commonly, the rest state $\mathbf{r}$ is assumed to be the identity mapping. Then equ. $\tilde{G}$ coincides with Green's strain tensor

$$G = \frac{1}{2}\begin{pmatrix} \langle \mathbf{s}_u, \mathbf{s}_u \rangle - 1 & \langle \mathbf{s}_u, \mathbf{s}_v \rangle \\ \langle \mathbf{s}_u, \mathbf{s}_v \rangle & \langle \mathbf{s}_v, \mathbf{s}_v \rangle - 1 \end{pmatrix}.$$

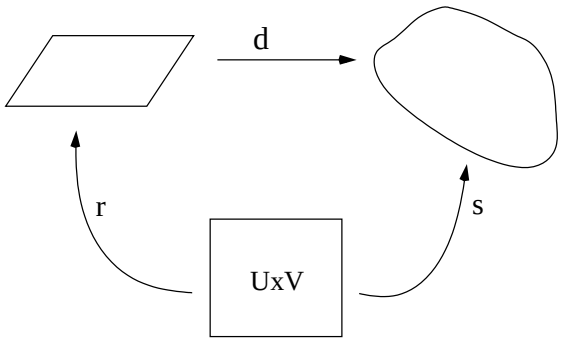


Figure 4: *The reference configuration: the rest state is parametrised by a mapping* $\mathbf{r}$ *on a space* $U \times V$*. By deformation* $\mathbf{d}$ *it transforms into the deformed (strained) configuration, which is parametrised by the mapping* $\mathbf{s}$*.*

Green's tensor, unfortunately, is nonlinear and yields fourth order terms in the energy formulation, and these are computationally very costly and lead to various numerical problems. Linear elasticity theory would allow much faster animations. 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 $\mathbf{d}$ in the components of Green's tensor, here for two dimensions:

$$\begin{aligned} \langle \mathbf{s}_u, \mathbf{s}_u \rangle - 1 &= \langle \mathbf{e}_1 + \mathbf{d}_u, \mathbf{e}_1 + \mathbf{d}_u \rangle - 1 \\ &= 2\langle \mathbf{d}_u, \mathbf{e}_1 \rangle + O(\mathbf{d}^2) \\ \langle \mathbf{s}_v, \mathbf{s}_v \rangle - 1 &= \langle \mathbf{e}_2 + \mathbf{d}_v, \mathbf{e}_2 + \mathbf{d}_v \rangle - 1 \\ &= 2\langle \mathbf{d}_v, \mathbf{e}_2 \rangle + O(\mathbf{d}^2) \\ \langle \mathbf{s}_u, \mathbf{s}_v \rangle &= \langle \mathbf{e}_1 + \mathbf{d}_u, \mathbf{e}_2 + \mathbf{d}_v \rangle \\ &= \langle \mathbf{d}_u, \mathbf{e}_2 \rangle + \langle \mathbf{d}_v, \mathbf{e}_1 \rangle + O(\mathbf{d}^2), \end{aligned} \tag{9}$$

where $(\mathbf{e}_1, \mathbf{e}_2)$ is the Cartesian basis of $\mathbf{R}^2$. Thus, Cauchy's tensor can be written as

$$\varepsilon = \begin{pmatrix} \mathbf{d}_u^1 & \frac{1}{2}(\mathbf{d}_v^1 + \mathbf{d}_u^2) \\ \frac{1}{2}(\mathbf{d}_v^1 + \mathbf{d}_u^2) & \mathbf{d}_v^2 \end{pmatrix},$$

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 linearisations in the deformations, i.e. the displacement $\mathbf{d}$ (figure 4) is assumed to be small, and all terms of order two or higher in $\mathbf{d}$ are neglected in the strain tensor. For this reason the linear theory is not appropriate for highly flexible objects like clothes. 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 remain small.

In this section we have learnt that a linearisation with respect to a global reference frame is not applicable to cloth animation. However, we will see later that a linearisation with respect to a local reference frame is feasible.

1.2.2. The equation of motion

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:

$$\sigma_{ij} = C_{ijkl}\, \varepsilon_{kl} \tag{10}$$

C is a symmetric rank-4 tensor containing the material properties. Here, symmetry means $C_{ijkl} = C_{klij}$ as well as $C_{ijkl} = C_{jikl}$. Hooke's law is used to compute the stress tensor in the equation of motion of a continuous elastic material:

$$\rho \frac{\partial^2 s}{\partial t^2} - \operatorname{div} \sigma = f, \tag{11}$$

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). Equation (11) 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 eliminates all spatial derivatives in the equation and reduces the PDE to an ordinary differential equation (ODE) in time t that can be solved by any suitable integration method. This way, we reduce the PDE (11) to the ODE (1).

As we will see in chapter 2, the higher order terms that are induced by Green's tensor lead to complication for rapid animations with large time steps. A linearisation of these terms could alleviate these problems.

1.2.3. Bend forces

Bend forces cannot be derived from the standard strain tensor because they are out-of-plane forces and arise only due to a volumetric property of the object (an ideal surface does not exhibit bending forces). Hence, there are basically two possibilities of deriving bending forces. First, we can simply add some bending forces to the in-plane forces. Second, the cloth can be modelled as a thin volumetric object.

In order to add bending forces, we add an energy that is also derived from a volumetric model. Typically we choose the thin plate energy, which we project onto the surface normal $\mathbf{n}$:

$$\begin{aligned} f &= \frac{1}{2}\langle \mathbf{n}, \partial \int B_1 s_{uu}^2 + B_2 s_{vv}^2 du dv \rangle \\ &= \langle \mathbf{n}, B_1 \mathbf{s}_{uuuu} + (B_1 + B_2)\mathbf{s}_{uuvv} + B_2 \mathbf{s}_{vvvv} \rangle. \end{aligned} \tag{12}$$

B_1 and B_2 are the elastic material constants, called bending moduli, for the respective directions. With this model, the in-plane and bend strain are decoupled and provide a simple model for computation.

A more comprehensive description is obtained by modelling cloth as thin volumetric objects. Shell theory provides the appropriate mechanism. Shells are thin objects which span over surfaces. The shell is parameterised by a mid-surface $\mathbf{s}$:

$$\mathbf{x}(u, v, w) = \mathbf{s}(u, v) + w\mathbf{d}(u, v),$$

i.e. the object position is described by the mid-surface parameters u, v and the height over the mid-surface w in the direction of a director $\mathbf{d}$ that has unit length.
From this description a three-dimensional strain and stress tensor is derived. Shells comprise (in-plane) membrane forces as well as bending forces. Some authors[26, 50] model clothes accurately by a 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.

These applications of shell theory have not been targeted at animation, but they serve well as a reference solution that animation systems can be compared with.

1.2.4. Description of Clothes in a Linear Theory

A characteristic property of textiles is orthotropy. Linear, 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 symmetry axes or directions in a woven material are the weft and warp directions. 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. The transverse contraction is often omitted because it is not apparent in usual scenes in which clothes are worn by a virtual avatar. This allows us to set the Poisson number to zero, i.e. $C_{iijj} = 0, i \neq j$. Only two Young moduli $k_1 := C_{1111}$ and $k_2 := C_{2222}$ and the shear modulus $\mu := C_{1212}$ remain as elastic constants to model the in-plane stress. Additionally, the bending moduli B_1 and B_2 describe the curvature elasticity in the weft and warp directions.

Unfortunately, the strain/stress relation in textile materials is highly nonlinear, and approximations to these nonlinear properties have been an issue in computer animation. Generally, the nonlinear stress-strain relation is modelled by a piecewise linear function. This can be achieved easily by updating the the elastic tensor according to the current slope of the strain/stress curve[24]. The updating is carried out before each time step.

1.2.5. Continuous Theory and Particle Systems

As we have seen, is is not possible to model cloth objects linearly because Cauchy's stress tensor cannot handle large displacements. Green's strain tensor, however, impedes a rapid numerical solution and makes implicit time integration very difficult (see chapter 2).

In this section, we will locally linearise the diagonal components of the strain tensor to alleviate the solution of the ODE. From this model we can derive a particle system that models continuous objects by a finite difference discretisation. This elastic model will be summarised here briefly. For details we refer to the full article[27].

The key is an approximation of the stress tensor. Given the deformed surface $\mathbf{s}(u,v)$, we approximate the stress tensor by

$$\sigma = \begin{pmatrix} k_1(\|\mathbf{s}_u\| - 1) & \frac{1}{2}\mu < \mathbf{s}_u, \mathbf{s}_v > \\ \frac{1}{2}\mu < \mathbf{s}_u, \mathbf{s}_v > & k_2(\|\mathbf{s}_v\| - 1) \end{pmatrix}, \qquad (13)$$

where k_1, k_2, and μ are the elastic material constants.

This tensor is plugged into the equation of motion for continuous objects (11). In particular, in the strain tensor the dominating diagonal components have been linearised with respect to a local rest frame. Thus, the numerical solution is alleviated.

From the diagonal components of the stress tensor we derive the following expression for the stress by finite difference discretization:

$$\sigma_{ii} = \frac{k_{1,2}}{l^2}(\|\mathbf{x}_i - \mathbf{x}_j\| - l),$$

where l is the rest distance between the particles at $\mathbf{x}_i$ and $\mathbf{x}_j$. The shear term $\sigma_{12} = \frac{\mu}{2} < \mathbf{s}_u, \mathbf{s}_v >$ as given in equation (13) is also approximated by finite differences. It is computed from the four direct neighbours of the central particle in the mesh as follows:

$$\sigma_{12} = \frac{\mu}{8l^2} < \mathbf{x}_{\text{upper}} - \mathbf{x}_{\text{lower}}, \mathbf{x}_{\text{right}} - \mathbf{x}_{\text{left}} >$$

The shear strain is measured by the scalar product of the angles formed by all edges incident to a particle very similar to the particle system already presented in section 1.1.2.

Linear spring forces result from approximating the divergence of the diagonal entries in the stress tensor

$$f_{i,j} = \frac{k_{1,2}}{l^2}\left[(\mathbf{x}_i - \mathbf{x}_j) - l\frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}\right], \qquad (14)$$

Note that the $f_{i,j}$ are actually force densities as we have continuous objects. Analogously to the diagonal components of the stress tensor, the corresponding force terms are computed as $\frac{\partial \sigma_{12}}{\partial v}\frac{1}{\|\mathbf{s}_u\|}\mathbf{s}_u$ and $\frac{\partial \sigma_{21}}{\partial u}\frac{1}{\|\mathbf{s}_v\|}\mathbf{s}_v$ (for a derivation see[27]).

Additionally to the stretch and shear forces, which are derived from the stress tensor (13), we add a thin plate energy according to equation (12). The resulting forces can be discretized again by finite differences, i.e. the fourth order derivatives are computed by fourth order difference operators. The surface normal $\mathbf{n}$ at a particle is computed as the usual vertex normal.

For rectangular meshes it can be verified that this particle system approximates the accurate continuum model with linear precision, i.e. we have convergence to the accurate solution when the resolution is increased. For general quadrilateral meshes, the solution still converges, but does not possess the approximation property.

1.2.6. Viscosity

Figure 5 shows a typical strain/stress curve that can be measured when a textile patch is stretched and relaxed. Hysteresis effects as in this figure are due to energy dissipation. When the material deforms under external forces, the strain/stress 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.

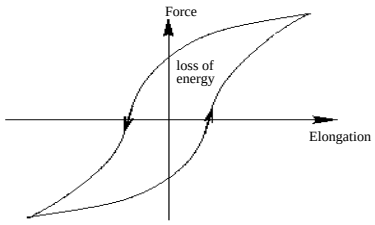


Figure 5: *Hysteresis*

There are several causes for energy dissipation when modelling deformable objects. We will focus exclusively on intrinsic effects, i.e. those that do not depend directly on external forces like friction due to fluid flow (wind). Internal friction depends on the relative motion of parts of the deformable object, that is, on the strain rate. The strain rate tensor is defined as

$$\nu_{ij} = \frac{d\varepsilon_{ij}}{dt}. \tag{15}$$

The forces generated by internal friction are often called viscous in analogy to Newtonian fluids. A viscous stress can be computed analogously to equation (10):

$$\sigma^{\nu}_{ij} = D_{ijkl}\,\nu_{kl}, \tag{16}$$

where the "viscosity tensor" D has the same structure as the elastic tensor C. The viscous stress is added to the purely elastic stress. 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. Note that the spring damping forces as state in equation (4) can be derived from this model.

In order to fit the typical behaviour of cloth, a more complex and visco-elastic model has to be chosen. For instance a constant Q model allows to reproduce a measured hysteresis curve quite elegantly [36].

2. Numerical Simulation

(*Michael Hauth*)

2.1. Methods for Numerical Integration

As we have seen in the previous section mechanical systems are often given as a second order ordinary differential equation accompanied by initial values

$$\begin{aligned} x''(t) &= f_v\big(t, x(t), x'(t)\big), \\ x(t_0) &= x_0, x'(t_0) = v_0. \end{aligned} \tag{17}$$

The differential equation can be transformed into a first order system by introducing velocities as a separate variable:

$$\begin{bmatrix} x(t) \\ v(t) \end{bmatrix}' = \begin{bmatrix} v(t) \\ f_v\big(t, x(t), v(t)\big) \end{bmatrix}, \quad \begin{bmatrix} x(t_0) \\ v(t_0) \end{bmatrix} = \begin{bmatrix} x_0 \\ v_0 \end{bmatrix}. \tag{18}$$

For the next few sections it will be convenient to write this ODE in the more abstract form

$$y'(t) = f\big(t, y(t)\big), \quad y(t_0) = y_0, \tag{19}$$

before we will come back to the special setting (18) for eventually gaining computational advantages.

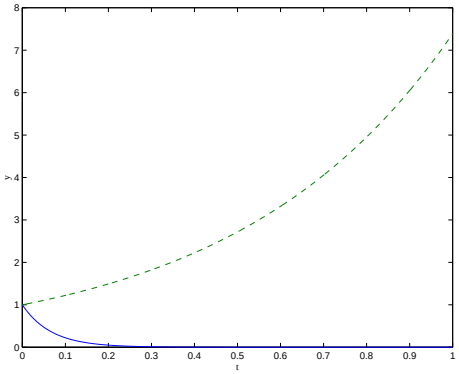


Figure 6: *Solutions of example 1 for* $\lambda = 2$ *(dashed) and* $\lambda = -15$ *(solid)*

Throughout the following discussion we will use the following examples:

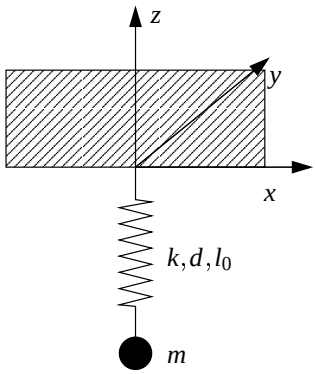
1. $y' = \lambda y, y(0) = 1$ with $\lambda = 2, -15$ for $t \in [0,1]$ (figure 6).
2. The overdamped wave equation $y'' = \lambda/2\, y + \lambda y'$ with $\lambda = 5$ for $t \in [0.10]$ and starting values $y(0) = 0, y'(0) = 1$. It has the analytical solution $y(t) = 1/15\sqrt{15}e^{1/2\left(-5+\sqrt{15}\right)t} - 1/15\sqrt{15}e^{-1/2\left(5+\sqrt{15}\right)t}$
3. This example is based on a simple mechanical system (figure 7(a)): A particle p with mass m connected to the origin using a spring with stiffness k, damping d and rest length l_0, is pulled down by gravity. This setting is described by the ODE

$$\frac{d^2z}{dt^2} = \frac{k(l_0 - z)}{m} - \frac{d}{m}\frac{dz}{dt} + g_z. \tag{20}$$

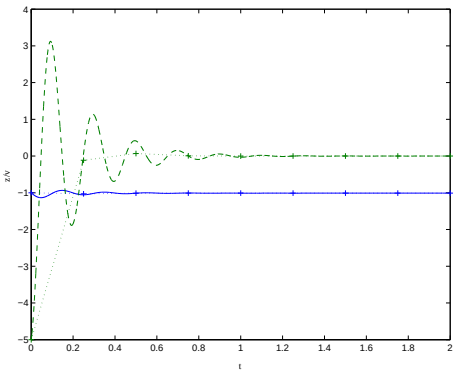
We set the parameters $m = 0.1$, $k = 100$, $l_0 = -1$, $d = 1$, $g = -10\, v_{0,z} = -5$ and simulate the interval $t \in [0,2]$ (figure 7(b)).

2.1.1. Explicit methods

The oldest and most simple method of integration is the so called forward or explicit Euler method. Time is discretised into slices of length h. To get a formula for advancing a



(a) The mechanical system of example 3.



(b) Exact solution of example 3: z (solid), z' (dashed) and an implicit Euler solution (see below) for large timesteps.

Figure 7: *Example 3.*

timestep, h the differential quotient on the left hand side of (19) is replaced by the forward difference quotient

$$\frac{y(t+h)-y(t)}{h} \approx y'(t) = f(t,y(t)). \tag{21}$$

Thus we get the integration formula for advancing a single timestep

$$y(t+h) = y(t) + hf(t,y(t)). \tag{22}$$

Iterating this method gives a sequence of numerical approximations $Y_n \approx y(t_n) =: y(t_0+nh)$. Geometrically this method can be interpreted as straightly following the tangent of the solution and then recalculating the slope for the next step.

There are several criteria for evaluating an integration method:

- convergence
- accuracy
- stability
- efficiency

Convergence means that for $h \to 0$ the numerical solutions Y_n meet the analytical. All useful methods must be convergent, so we won't discuss non-convergent methods or criteria for convergence. More interesting is the accuracy or order of a method. By this we mean how fast a method converges for $h \to 0$, or with other words how accurate the solution is for a given h. By using a taylor expansion for the exact solution after a single timestep

$$y(t+h) = y(t) + hy'(t) + h^2/2y''(t) + O(h^3) \tag{23}$$

we find that for the numerical approximation Y_1 produced by an explicit Euler step

$$y(t_1) - Y_1 = O(h^2). \tag{24}$$

If we continue the method using the *numerical solution* Y_1 as a starting value for the next timestep we lose[38] a power of h for the global error

$$y(t_n) - Y_n = O(h). \tag{25}$$

This means that the explicit Euler method converges linearly or has *order* 1. We will analyse stability and efficiency of the method later.

As a next step we will introduce methods of higher order. For this a centered difference estimation for $y'(t+h/2)$ (19) is used

$$\frac{y(t+h)-y(t)}{h} \approx y'(t+h/2) = f(t,y(t+h/2)). \tag{26}$$

and thus as an iteration scheme

$$Y_{n+1} = Y_n + f(t,y(t_n+h/2)). \tag{27}$$

But how do we find $y(t_n+h/2)$? For an estimation we use an explicit Euler step to get

$$k_1 = Y_n + \tfrac{h}{2} f(t,Y_n) \tag{28}$$
$$Y_{n+1} = Y_n + hk_1, \tag{29}$$

the so called *explicit midpoint* rule. The estimation by forward Euler, although not very accurate is good enough, as the function evaluation is multiplied by the timestep to advance to the next approximation. So by a taylor expansion we find a local error of $O(h^3)$ leading to a global error of

$$Y_n - y(t_n) = O(h^2) \tag{30}$$

for the explicit midpoint rule.

Generalizing the idea of using function evaluations at s intermediate points $t+c_jh$ leads to the Runge-Kutta methods defined by a Runge-Kutta matrix (a_{ij}), weights b_i, abscissae

c_j and the equations

$$k_i = Y_n + h\sum_{j=1}^{s} a_{ij}k'_j$$
$$\text{with } k'_i = f(t_n + c_i h, k_i) \text{ for } \quad i = 1,\ldots,s$$
$$Y_{n+1} = Y_n + h\sum_{i=1}^{s} b_i k'_i \tag{31}$$

The coefficient set can comfortably be specified like in table 2. If the matrix (a_{ij}) is strictly upper, all inner stages k_i only

c_1	a_{11}	a_{12}	$\cdots$	a_{1s}
c_2	a_{12}	a_{22}	$\cdots$	a_{2s}
$\vdots$	$\vdots$		$\ddots$	$\vdots$
c_s	a_{s1}	a_{s2}	$\cdots$	a_{ss}
	b_1	b_2	$\cdots$	b_s

Table 2: *General Runge–Kutta method*

depend on k_j with $j < i$ and thus can be computed one after the other.

The most famous one is the method by Runge and Kutta given in (3) together with the explicit midpoint rule interpreted as a Runge-Kutta method. The method by Runge and Kutta possesses order 4.

0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0
	0	1

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	2/6	2/6	1/6

Table 3: *Explicit midpoint and "the" Runge-Kutta method*

By using algebraic relations for the coefficients, it is possible to construct explicit Runge-Kutta methods of arbitrary high order resulting in many inner stages with many function evaluations. But for most practical applications order 4 is sufficient.

Having constructed all these method we now will try them on our examples.

The diagrams in figure 8 were produced by solving the examples using different timesteps and measuring the number of floating point operations needed for achieving the specified accuracy when compared with the (analytical) reference solution. In the work-precision diagram the *y*-axis shows the error $\|Y_{t_{end}} - y(t_e nd)\|$ as a function of the required number of floating point operations. The first example with $\lambda = 2$ (figure 8(a)) shows exactly the expected behaviour: when reducing the time step and thus investing more work, the numerical solutions converge against the reference solution. Moreover the slope of the curves in the double logarithmic plot exactly match the order of the method. But in all the other examples (figure 8(b)-8(d)) this behaviour only shows up after an initial phase, where the solver produces completely wrong results. This is the point where stability comes in. We will now analyse this by using the simplest example where it occurs, i.e. example 1 with $\lambda < 0$.

2.1.2. Stability

The equation for example 1 is called *Dahlquist's test equation*

$$y' = \lambda y, \qquad \lambda \in \mathbb{C}. \tag{32}$$

Its exact solution to an initial value $y(0) = y_0$ is given by

$$y(t) = e^{\lambda t} y_0. \tag{33}$$

This equation is a tool for understanding and evaluating the stability of integration methods. We have seen, that in the damped case characterised by $\mathrm{Re}\,\lambda < 0$ convergence is only achieved by very small timesteps. In this case, since the exponent is negative, the analytical solution is bounded for $t \to \infty$. Therefore a meaningful numerical method is required to deliver a bounded solution. An integration scheme that yields a bounded solution is called *stable*.

If we apply the explicit Euler's method with a fixed step size h to (32) the n-th point of each solution is given by:

$$Y_n = (1 + h\lambda)^n y_0 \tag{34}$$

For the explicit Euler scheme the numerical solution given by (34) is bounded if and only if $|1 + h\lambda| < 1$, i.e. for $h\lambda$ in the unit ball around -1. A similar analysis can be carried out for the other methods and also results in restrictions of the admissible step size.

This analysis explains the sharp bend in figures 8(b)-8(d). Only when the step size drops below a certain limit dictated by λ (by $h < \lambda^{-1}$ in case of the forward Euler method) the numerical solutions can converge. If the damping is increased, i.e. $\mathrm{Re}\,\lambda \to -\infty$, then for the explicit Euler necessarily $h \to 0$ for the solution to be stable. This means the step size is artificially limited and it cannot be increased beyond the stability limit in order to save work sacrificing some - but not all - accuracy.

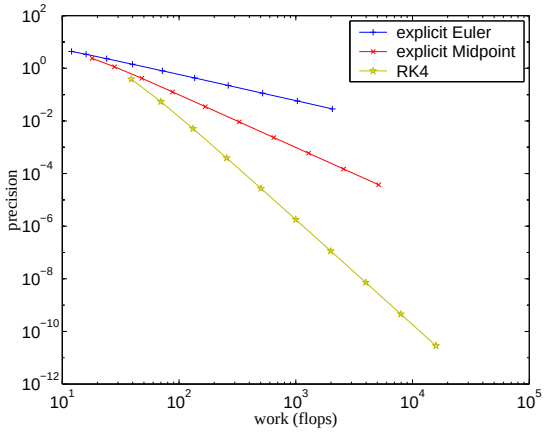
2.1.3. The implicit Euler method

To construct a method that better suits our needs we go back to (19) and substitute the differential quotient by a *backward* difference quotient for $y(t+h)$

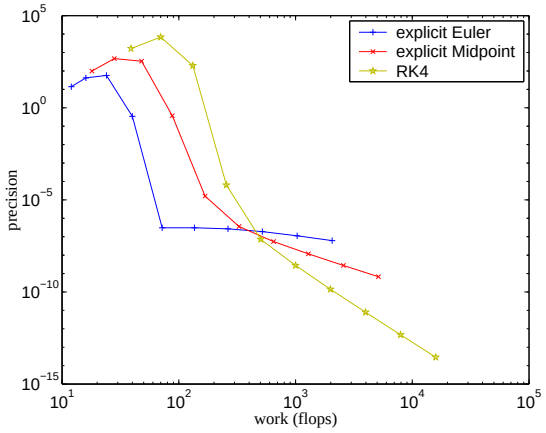
$$\frac{y(t+h) - y(t)}{h} \approx y'(t+h) = f(t+h, y(t+h)). \tag{35}$$

This time this results in the integration formula

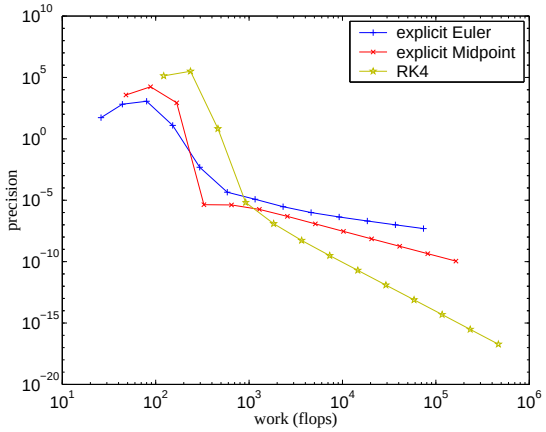
$$Y_{n+1} = Y_n + h f(t+h, Y_{n+1}), \tag{36}$$



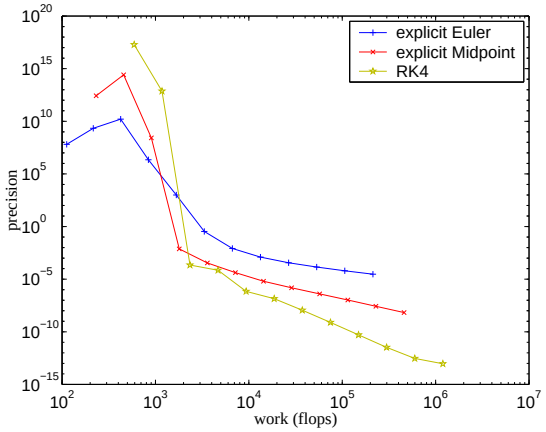
(a) Example 1a: $y' = 2y$



(b) Example 1b: $y' = -15y$



(c) Example 2: $y'' = -5y' - 5y$



(d) Example 3

Figure 8: *Work precision diagrams for the explicit Euler, explicit midpoint and RK4 methods.*

the so called backward or implicit Euler method. As its explicit variant this method can be shown to have order 1. Now the numerical solution only is given implicitly as the solution of the possibly nonlinear equation

$$Y_{n+1} - hf(t+h, Y_{n+1}) - Y_n = 0. \tag{37}$$

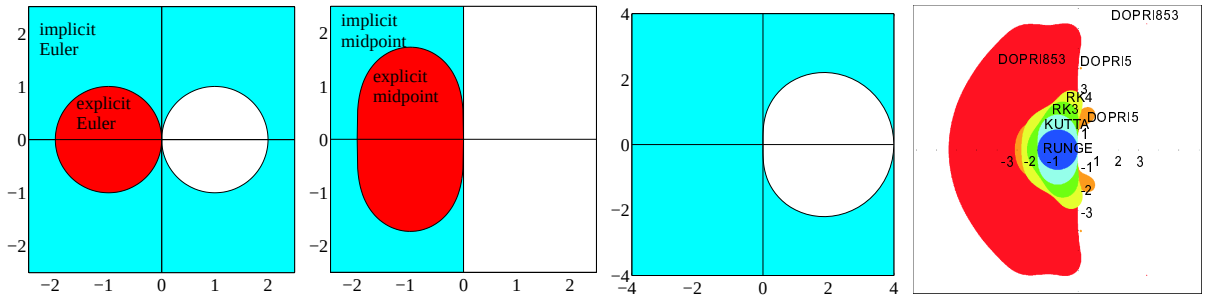
If we apply this method to the Dahlquist equation we get the recurrence formula

$$Y_n = (1 - h\lambda)^{-n} y_0. \tag{38}$$

The numerical solution Y_n remains bounded for $|(1 - h\lambda)^{-1}| < 1$. If we assume $\lambda < 0$, this holds for arbitrary $h > 0$. Thus there is no restriction on stepsize, the method is *unconditionally stable*. Figure 10 shows the work-precision diagrams for the implicit Euler method and our examples. We observe that we never loose stability and we especially do no miss the solution by several orders of magnitude compared to the explicit methods, although we loose some accuracy when the timesteps become large.

As a practical tool for graphically visualizing the stability properties of a method we define the *stability region* $\mathcal{S}$ to be the set of parameters, for which the integration method yields a bounded solution:

$$\mathcal{S} := \{z := h\lambda \in \mathbb{C} : \text{ the numerical integration of equation (32) withstep size } h \text{ and parameter } \lambda \text{ is stable}\}. \tag{39}$$



(a) Euler methods (b) midpoint rules (c) BDF(2) (d) RK methods

Figure 9: *Stability regions (shaded) of the methods.*

Methods that contain the complete left half-plane in $\mathcal{S}$ are called *A-stable* or *unconditionally stable*. They are well suited for the stable integration of stiff equations. Obviously, the implicit Euler scheme is A–stable, whereas its explicit counterpart is not. The stability regions of all presented methods are shown in figure 9.

After reviewing the process that led us to the definition of the stability region, we can outline a more general idea that will allow us to determine easily the stability of more complex methods. The idea for analysing both Euler methods applied to (32) was to find a closed expression describing the *stability function* $\mathcal{R}$. This function maps the initial value y_0 to the value Y_1, performing a single step of the method

$$\mathcal{R} : y_0 \mapsto Y_1. \tag{40}$$

Thus $Y_n = R(h\lambda)^n y_0$. For the explicit Euler method we found in (34)

$$\mathcal{R}(z) = 1 + z, \tag{41}$$

for the implicit version in (38)

$$\mathcal{R}(z) = \frac{1}{1-z}. \tag{42}$$

The definition for the stability region now reads

$$\mathcal{S} = \{z \in \mathbb{C} : |\mathcal{R}(z)| < 1\}. \tag{43}$$

2.1.4. Methods of higher order

To find a higher order method, we go back to equation (27) and do not replace the $y(t+h/2)$ but take the formula as an implicit definition of $y(t+h)$. We get the *implicit midpoint rule*

$$Y_1 = Y_0 + hf\left(\frac{Y_1+Y_0}{2}\right). \tag{44}$$

Further on we use a simplified notation for advancing one step, writing Y_0 and Y_1 instead of Y_n and Y_{n+1}.

Alternatively the midpoint rule can be derived as a *collocation method*[39] with $s = 1$ internal nodes, i.e. by constructing a polynomial interpolating the particle trajectories at a given, fixed set of s nodes[39]. This idea allows for the construction of implicit Runge-Kutta methods with arbitrary order. In contrast to explicit methods the matrix (a_{ij}) ceases to be strictly lower triangular. These methods are computationally more expensive, so we just stick to the midpoint rule. Its stability function is given by

$$\mathcal{R} = \frac{1+z/2}{1-z/2}. \tag{45}$$

As $\mathcal{R} \leq 1$ for any $\mathrm{Re}\, z < 0$ the implicit midpoint rule is *A*-stable.

As another possible choice we now introduce *multistep methods*. They are computationally inexpensive because they have no inner stages and some of them are A-stable. A multistep method with k steps is of the general form

$$\sum_{j=0}^{k} \alpha_j Y_{k-j+1} = h \sum_{j=0}^{k} \beta_j f_{k-j+1}, \tag{46}$$

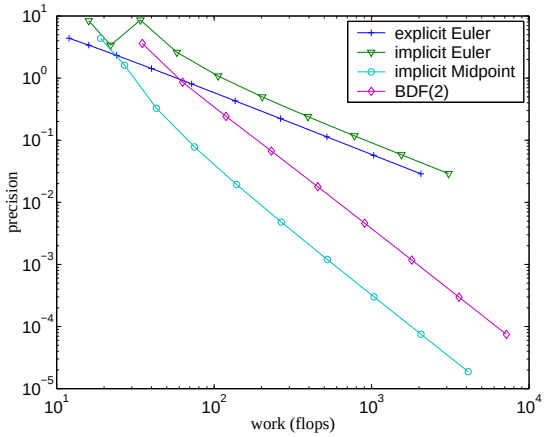
with $f_{n+j} := f(t_{n+j}, Y_{n+j})$. Here we also have 'history points' with negative indices. The coefficient α_k is required to be nonzero; for variable time step sizes the coefficients depend on the last stepsizes, which we have omitted here for the ease of demonstration. Important special cases are the class of *Adams methods* where $\alpha_0 = \cdots = \alpha_{k-2} = 0$:

$$Y_1 = Y_0 + h \sum_{j=0}^{k} \beta_j f_{k-j+1} \tag{47}$$

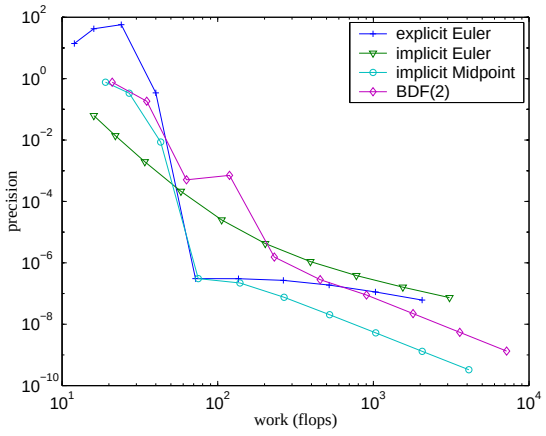
and the class of *BDF–methods* (**b**ackward **d**ifferentiation **f**ormulas) with $\beta_0 = \cdots = \beta_{k-1} = 0$:

$$\sum_{j=0}^{k} \alpha_j Y_{k-j+1} = h\beta_k f_1. \tag{48}$$

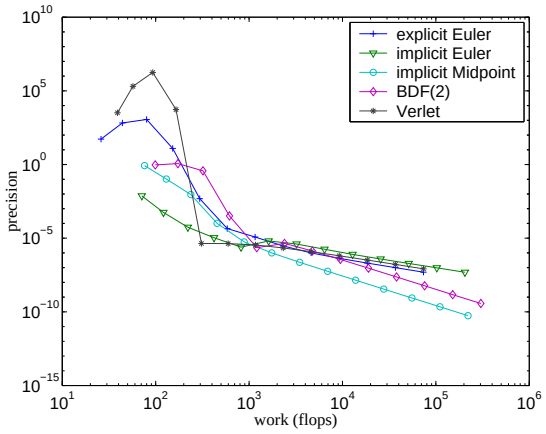
If the formula involves the right-hand side f_1 at the new ap-



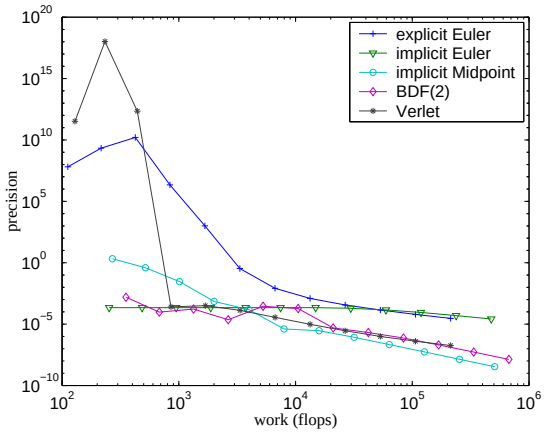
(a) Example 1a: $y' = 2y$



(b) Example 1b: $y' = -15y$



(c) Example 2: $y'' = -5y' - 5y$



(d) Example 3

Figure 10: *Work precision diagrams for the implicit Euler, implicit midpoint, BDF(2) and Verlet (whenever possible) methods. The results for Euler are for comparison and the same as in figure 8.*

proximation point Y_1 the method is said to be implicit. BDF-methods are always implicit. The coefficients can again be constructed by a collocation approach. BDF-methods exist up to order 6, higher order methods loose consistency for any choice of coefficients[39].
The stability regions of implicit and explicit Adams methods are bounded and located around the origin, thus they are not interesting for large time steps.
BDF-methods were the first to be developed to deal with stiff equations and possess an unbounded stability region covering a sector within the negative complex half-plane. Therefore they are among the most widely used methods today. For $k+1$ points, these methods possess order $k+1$ and only one nonlinear system has to be solved, whereas s coupled systems have to be solved for an s-stage implicit Runge-Kutta method.

The BDF-method for $k = 1$ is just the implicit Euler method, for k=2 the method is given as

$$Y_1 = \frac{4}{3}Y_0 - \frac{1}{3}Y_{-1} + \frac{2}{3}hf(t+h, Y_1) \qquad (49)$$

The coefficients for higher order methods are given in table 4. The stability region of BDF(2) and the other implicit methods are shown in figure 9.

α_0	α_1	α_2	α_3	α_4	α_5	α_6	β
$\frac{3}{2}$	-2	$\frac{1}{2}$					1
$\frac{11}{6}$	-3	$\frac{3}{2}$	$-\frac{1}{3}$				1
$\frac{25}{12}$	-4	3	$-\frac{4}{3}$	$\frac{1}{4}$			1
$\frac{137}{60}$	-5	5	$-\frac{10}{3}$	$\frac{5}{4}$	$-\frac{1}{5}$		1
$\frac{147}{60}$	-6	$\frac{15}{2}$	$-\frac{20}{3}$	$\frac{15}{4}$	$-\frac{6}{5}$	$\frac{1}{6}$	1

Table 4: *BDF Methods*

2.1.5. The Verlet method

As a last method we will discuss a scheme commonly referred to as leapfrog or Stoermer-Verlet method. It is especially efficient if (17) is given as the second order system

$$x''(t) = f_v\big(t, x(t)\big), \tag{50}$$

i.e. $f_v\big(t, x(t), x'(t)\big) = f_v\big(t, x(t)\big)$. It is not applicable to general first order systems of the form (19).

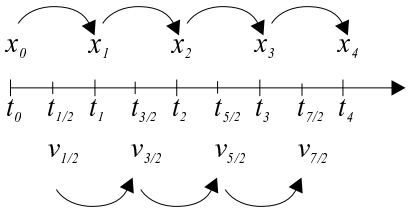


Figure 11: *Staggered grids for the Verlet method.*

To derive it, we use centered differences at a staggered grid (figure 11) i.e. we now approximate v at $t+(2i+1)h/2$ and x at $t+ih$ by centered differences

$$\frac{v_{n+1/2} - v_{n-1/2}}{h} = f(x_n) \tag{51}$$

$$\frac{x_{n+1} - x_n}{h} = v_{n+1/2} \tag{52}$$

thus

$$v_{n+1/2} = v_{n-1/2} + hf(x_n) \tag{53}$$

$$x_{n+1} = x_n + hv_{n+1/2}. \tag{54}$$

The method possesses order 2 as one can see by substituting (53) into (54) resulting in the second order centered difference

$$\frac{x_{n+1} - 2x_n + x_{n-1}}{h} = f(x_n). \tag{55}$$

From this equation an alternative formulation of the Verlet scheme as a multistep method can be derived

$$v_n - v_{n-1} = hf(x_n) \tag{56}$$

$$x_{n+1} - x_n = hv_n, \tag{57}$$

which omits the half steps and staggered grids from above.

Now for second order equations which do not possess the form of (50) one may replace $f(x_n)$ by $f(x_n, v_{n-1})$ at the expense of some stability. Correctly the replacement had to be with $f(x_n, v_n)$ but this would result in a implicit method. Now we can apply the method to examples 2 and 3.

Figure 10 shows the work precision diagrams for the implicit methods. Even for large time steps these methods give approximations to the exact solution. After a somewhat uneven convergence phase all the explicit methods converge smoothly for decreasing time step to the exact solution with a slope given by their order.

Up to now we have omitted a discussion of the stability properties of the leapfrog method. From the examples it can be observed that the method cannot be unconditionally stable. Indeed some more delicate computations show that the method only delivers a bounded solution for arbitrary $h > 0$ for purely oscillatoric equations, i.e. for second order ordinary differential equations of the form (50) (with no damping term) and a contractive right hand side. Nevertheless the method remains well behaved in the presence of low damping. This explains its importance in molecular and celestial dynamics, where all systems are conservative. In figure 12 we applied the Verlet method to the undamped wave equation $y'' = 5y$ and it is clearly one of the best choices over a wide range of accuracy requirements.

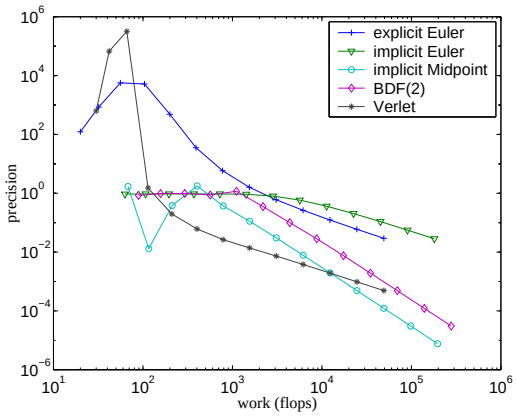


Figure 12: *Work precision diagram for the explcit/implicit Euler, implicit midpoint, BDF(2) and Verlet methods for the wave equation $y'' = -5y$ without damping.*

2.1.6. Selecting an efficient method

Which method is best for a certain application? This question is nearly impossible to answer a priori. The only choice is to try a set of methods and to evaluate which one performs best. Choosing the methods to try, though can be done based on theoretical considerations and observations of the problem at hand. A possible strategy is shown in figure 13.

The same statement holds for predicting the efficiency of

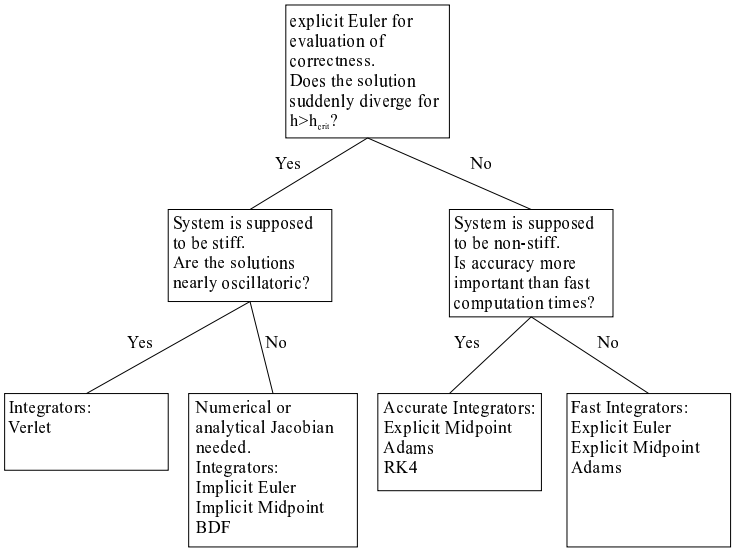


Figure 13: *Selecting a method.*

a method. Generally implicit methods require more work per step. On the other hand one may be able to use time steps that are several magnitudes larger than the ones explicit methods would allow. Although accuracy will suffer, the integration won't be unstable (see figure for example 7(b)). If evaluations of the right hand side function are cheap, a step with RK(4) is faster than an implicit step with BDF(4). On the other hand if it is cheap to compute a good sparse approximation to the jacobian, it may be more efficient to solve the linear system with a few cg iterations than to perform 4 full function evaluations.

2.2. Solving nonlinear systems

All implicit methods require the solution of a nonlinear system. The implicit Euler method for example reduces our integration problem to the solution of the nonlinear system

$$Y_1 - hf(Y_1) - Y_0 = 0. \tag{58}$$

The other methods yield a system of similar form, namely

$$Y_1 - hf(\frac{1}{2}(Y_1 + Y_0)) - Y_0 = 0 \tag{59}$$

$$Y_1 - \frac{2}{3}f(Y_1) + \left(-\frac{4}{3}Y_0 + \frac{1}{3}Y_{-1}\right) = 0 \tag{60}$$

for the midpoint and BDF(2) rule, respectively. This is a nonlinear system of dimension $6N$. This system must be solved with Newton's method to allow for arbitrary step sizes independent of λ. Simpler methods for nonlinear systems would compensate the advantage of A-stability because the number of iterations would increase proportionally to the stiffness parameter $|\lambda|$. We now will work out an approach for implementing Newton's method efficiently.

2.2.1. Newton's method

For the nonlinear system $G(Y) = 0$ we compute a numerical solution by the following algorithm:

Algorithm 1: Newton's Method

(1) **for** $k = 1, 2, \ldots$ **until** convergence **do**
(2) Compute $G(Y^{(k)})$.
(3) Compute $J^{(k)} = \frac{\partial}{\partial Y} G(Y^{(k)})$.
(4) Solve $J^{(k)} s^{(k)} = -G(Y^{(k)})$.
(5) $Y^{(k+1)} := Y^{(k)} + s^{(k)}$
end

Applying Newton's method reduces the problem to the successive solution of linear systems. In a classical Newton method this is achieved by Gaussian elimination. This introduces a lot of non-zero elements into the factors. Although reordering techniques alleviate the effects, this approach is too expensive for the present application.
A lot of authors[2, 86] use iterative methods to solve the linear system. We will also use the conjugate gradient method here to solve the linear systems in each Newton step. Unfortunately this changes the convergence behaviour of the outer Newton method, which is referred to as an inexact Newton method[78], given by algorithm 2.

Algorithm 2: Inexact Newton Method

(1) **for** $k = 1, 2, \ldots$ **until** convergence **do**
(2) Compute $G(Y^{(k)})$.
(3) Compute $J^{(k)} = \frac{\partial}{\partial y}(Y^{(k)})$.
(4) Find $s^{(k)}$ with $J^{(k)} s^{(k)} = -G(Y^{(k)})) + r^{(k)}$,

such that $\|r^{(k)}\| \leq \eta_k \|G(Y^{(k)})\|$.
(5) $Y^{(k+1)} := Y^{(k)} + s^{(k)}$
end

2.2.2. Residual control

The error of the iterative solution of the linear system is formulated in terms of the residual, which is easily computationally accessible, whereas the actual error cannot be computed. The tolerance of the linear iteration is decreased proportionally to the monotonically decreasing residual of the nonlinear iteration.
An analysis of this method[78] shows that it converges under rather weak additional assumptions. If the classical Newton method converges and the scalar tolerances $\eta^{(k)}$ are uniformly bounded by an $\eta < 1$, the inexact method converges. In literature the $\eta^{(k)}$ are referred to as *forcing terms*. Note that this additional assumption is also necessary: For $\eta = 1$, $s^{(k)} = 0$ would be admissible and the iteration would stagnate.
The inexact method then at least converges linearly, whereas Newton converges superlinearly. By choosing the η_k to converge to zero sufficiently fast[78], the convergence of the inexact Newton method can be forced to have an order > 1. In a neighbourhood of the solution the convergence usually speeds up. By extrapolating the solution of the previous time step we obtain a good initial value for the new solution and the method converges quickly using the constant bound $\eta = 0.02$ without imposing a too strict tolerance on the linear solver.

2.2.3. Inexact simplified Newton methods

The efficiency of the Newton method can be further improved by another approximation. In the simplified version of Newton's method the Jacobian $J^{(k)}$ is approximated by $J^{(0)}$. Such a scheme can be rewritten in the form of an inexact Newton method, if the linear system is written as follows and J is chosen as approximation to $J^{(k)}$

$$\begin{aligned} Js^{(k)} &= -G(Y^{(k)}) + (J - J^{(k)})s^{(k)} + r^{(k)} \\ &=: -G(Y^{(k)}) + \tilde{r}^{(k)} \end{aligned} \tag{61}$$

The residual $r^{(k)}$ is replaced by the larger $\tilde{r}^{(k)}$, which can be bounded if $J \approx J^{(k)}$. By choosing $\tilde{\eta}^{(k)}$ appropriately, the method still converges. In fact, we trade some accuracy approximating $J^{(k)}$ against accuracy in solving the linear system and up to a certain limit the method still behaves as before.

This degree of freedom can be further exploited be even not computing $J^{(0)}$ but a sparser approximation of it. In our system[41] we modelled cloth using the continuity based approach from section 1.2.5. The choice of the approximated Jacobian is motivated by observing that the dominating stiffness is induced into the system by linear spring forces from (14). These are split into a linear and a nonlinear part.

We approximate the right-hand side of the system by the linear expression

$$F_{\text{lin}}(x,v) = \sum_{j|(i,j)\in E} \left[\frac{k_{ij}}{l_{ij}^2}(x_i - x_j) + \frac{d_{ij}}{l_{ij}^2}(v_i - v_j) \right] \tag{62}$$

Writing this equation in matrix notation

$$F_{\text{lin}}(x,v) = Kx + Dv, \tag{63}$$

we approximate the full Jacobian of the system by the Jacobian of $F_{\text{lin}}(x,v)$. Then we apply (61) with

$$J = I - h\gamma \begin{pmatrix} 0 & 0 \\ K & D \end{pmatrix}, \tag{64}$$

where h is the time step and γ depends on the integration method used. In the implementation this system of dimension $6N$ can be reduced to a system of dimension $3N$ by exploiting the linear relation between position and velocity (section 2.3).
This choice of the Jacobian has two major advantages over the full Jacobian. First, J is inexpensive to compute and only changes when either the material constants or the step size changes. Second, we reduce the 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 solver. The resulting algorithm is surprisingly simple.

Algorithm 3: Inexact Simplified Newton's Method

(1) Compute $J \approx \frac{\partial}{\partial Y} G(Y^{(0)})$.
(2) **for** $k = 1, 2, \ldots$ **until** convergence **do**
(3) Compute $G(Y^{(k)})$.
(4) Find $s^{(k)}$ with $Js^{(k)} = -G(Y^{(k)})) + r^{(k)}$,
such that $\|r^{(k)}\| \leq \tilde{\eta}_k \|G(Y^{(k)})\|$.
(5) Update $Y^{(k+1)} := Y^{(k)} + s^{(k)}$
end

2.2.4. Adaptive time stepping

Newton's method can also be used to control the step size of the ODE solver. If the convergence of Newton's method is poor, the time step h is reduced such that the solution of the previous time step is a better start value for the current time step and achieves a faster convergence. On the other hand, the number of Newton iterations necessary is a criteria for the behaviour of the integrator and has already been used for an order selection algorithm[40] of the Radau integrator. In our implementation the number of Newton iterations decides whether to increase or decrease the time step h.

2.3. Assembly of a solver

All tools have been presented to assemble a numerical integrator and specialise it to mechanical systems. As an example we provide a solver based on BDF(2) and the simplified

inexact Newton method with adaptive time stepping. We first derive the scheme with fixed stepsize

$$\frac{3}{2}Y_1 - 2Y_0 + \frac{1}{2}Y_{-1} = hf(Y_1). \tag{65}$$

Splitting $Y = [x, v]^T$ and $f = [f_x, f_v]^T$ into position and velocity part

$$x_1 - \frac{4}{3}x_0 + \frac{1}{3}x_{-1} = \frac{2}{3}hv_1 \tag{66}$$

$$v_1 - \frac{4}{3}v_0 + \frac{1}{3}v_{-1} = \frac{2}{3}hf_v(x_1, v_1). \tag{67}$$

Substituting for x_1 leaves one equation

$$v_1 - \frac{4}{3}v_0 + \frac{1}{3}v_{-1} = \frac{2}{3}hf_v\big(\frac{2}{3}hv_1 + \frac{4}{3}x_0 - \frac{1}{3}x_{-1}, v_1\big). \tag{68}$$

For applying Newton's Method we have to differentiate with respect to v_1 obtaining the equation for the update

$$\Big(I - \frac{4}{9}h^2\frac{\partial}{\partial x}f_v(x,v) - \frac{2}{3}h\frac{\partial}{\partial v}f_v(x,v)\Big)s^{(k)} = -g(x_1^{(k)}, v_1^{(k)}) \tag{69}$$

with the right hand side

$$g(x_1, v_1) = v_1 - \frac{4}{3}v_0 + \frac{1}{3}v_{-1} - \frac{2}{3}hf_v(x_1, v_1). \tag{70}$$

We also update the space part x during Newton iteration, simplifying the function evaluation

$$x_1^{(k+1)} = \frac{4}{3}x_0 - \frac{1}{3}x_{-1} + \frac{3}{2}hv_1^{(k+1)} = x_1^{(k)} + \frac{3}{2}hs^{(k)}. \tag{71}$$

Now we introduce the approximation to the Jacobian, rewriting the system (69) to

$$(I - \frac{4}{9}h^2K - \frac{2}{3}hD)s^{(k)} = -g(x_1^{(k)}, v_1^{(k)}). \tag{72}$$

We omitted the variable stepsize. If h changes, equation (65) and derived scalars change to

$$Y_1 - \frac{(1+\omega)^2}{1+2\omega}Y_0 + \frac{\omega^2}{1+2\omega}Y_{-1} = h_{\text{new}}\frac{1+\omega}{1+2\omega}f_1 \tag{73}$$

with $\omega = \frac{h_{\text{new}}}{h_{\text{old}}}$. In particular (72) (resp. (69)) changes to

$$\Big(I - \big(\frac{1+\omega}{1+2\omega}\big)^2h^2K - \frac{1+\omega}{1+2\omega}hD\Big)s^{(k)} = -\tilde{g}(x_1^{(k)}, v_1^{(k)}) \tag{74}$$

with

$$\tilde{g}(x_1, v_1) = v_1 - \frac{(1+\omega)^2}{1+2\omega}v_0 + \frac{\omega^2}{1+2\omega}v_{-1} - \frac{1+\omega}{1+2\omega}hf_v(x_1, v_1). \tag{75}$$

The update for x becomes

$$x_1^{(k+1)} = \frac{(1+\omega)^2}{1+2\omega}x_0 - \frac{\omega^2}{1+2\omega}x_{-1} + \frac{1+\omega}{1+2\omega}h_n ewv_1^{(k+1)} = x_1^{(k)} + \frac{1+\omega}{1+2\omega}h_n ews^{(k)}. \tag{76}$$

Thus we have the following complete integrator.

Algorithm 4: BDF2-Solver

(1) $\omega = 1$
(2) // timestepping loop
while $t < t_{end}$ **do**
(3) $t = t + h$
(4) Update if necessary K and D.
(5) If K, D or h changed precompute $A = I - \big(\frac{1+\omega}{1+2\omega}\big)^2h^2K - \frac{1+\omega}{1+2\omega}hD$
(6) $v_{-1} = v_0$, $v_0 = v_1$
(7) Precompute the constant terms in the integration formula (73) and (75)
(8) Initialise the starting values $x_1^{(0)}, v_1^{(0)}$ by extrapolation and compute $g = \tilde{g}(x_1^{(0)}, v_1^{(0)})$
(9) //Newton loop
while $\|\tilde{g}(x_1^{(k)}, v_1^{(k)})\| >$ `tol` and $k <$ `max_newt`
(10) Solve $As^{(k)} = -g$ such that $\|g - As^{(k)}\| \leq \eta\|g\|$.
(11) Compute $v_1^{(k+1)}$ and $x_1^{(k+1)}$ using (76)
(12) Update $g = g(x_1^{(k+1)}, v_1^{(k+1)})$
(13) $k = k + 1$
(14) **end while**
(15) **If** $k <$ `h_up` increase h
If $k >$ `h_down` decrease h
$\omega = \frac{h_{\text{new}}}{h_{\text{old}}}$
(16) **end do**

2.4. Comparison of methods

After we have established this theoretical background we are now able to describe many of the current simulation approaches for cloth simulation in a unifying way.

Baraff and Witkin[2] formulate nonlinear constraints and use their linear approximation for the construction of an implicit solution method commonly referred to as linear implicit Euler. This way the system to be solved also becomes linear and can be solved efficiently by a *cg*-method. This method corresponds to the solution of a nonlinear system with only one Newton iteration. Because the nonlinear part is not integrated, with high stiffness one may encounter problems[41, 25].

Desbrun et al.[22] used Provots model[76] (section 1.1.1) and a linear implicit method. But instead of linearizing the whole system, they split it in a linear and nonlinear part and use

a precomputed inverse of A for solving the linear part of the equations. They don't aim at solving the equation completely, as they don't integrate the nonlinear term explicitly. Instead the angular momentum is corrected to account for the nonlinear part. With this algorithm one can neither change the stepsize h nor deal with an Jacobian depending on t.

Debunne et al.[21] use a simplified version of a linear implicit Euler method. To solve the differential equation arising from their geometric nonlinear Green model, they apply an implicit Euler method using a single Newton iteration. The Jacobian is approximated by the 3x3 block-diagonal of the linear Cauchy tensor, allowing a very fast inversion and application.

3. Collisions

(*Bernd Eberhardt*)

3.1. Introduction

In the physically-based simulation of rigid or non-rigid bodies there are three main steps to be solved as efficiently as possible but also as accurately as necessary: Computation of the dynamics of any object, detection of collisions between objects, and computation of the collision response, i.e. of forces taking effect by collisions.

Thus, an important issue in modelling cloth is to determine the interaction or penetration of the cloth with itself and its surroundings. In fact, it would be only half of the story without it.

The use of particle systems in calculating textiles enables realistic modelling close to experimental data. This physically based modelling provides a realistic interaction between objects in a system. The first and arguably most important step in setting up a physically based modelling system is an accurate collision detection.

As described in the previous tutorial sections we calculate the trajectory of a particle via an integration of textile-specific differential (difference) equations solving an initial value problem. The initial values, i. e. locations and velocities of the particles, are well-defined before the start of a simulation via an implicit/explicit Euler-, Runge-Kutta- or another integration scheme. Then we detect penetration and have to react according to the situation and adjust velocities and/or locations, i. e. the new initial values.

Based on our intuition people can easily distinguish between a physically correct or insufficient treatment when collision is detected. Textiles, or more general deformable materials, are particular in collision detection. We may encounter arbitrary folding and hence we have to consider besides the classical collision with other objects also possible self-collisions of the textile itself. Therefore, we need apart from the physical laws for the motion equations a fast and very general treatment of collision effects for cloth-modelling.

Hence, we have two problems to solve: first, the detection of proximities where we find triangles close to each other and second, the reaction on this proximity where we want to correct and prevent interference of our objects.

The first problem of collision detection and other proximity queries has attracted substantial research during the last years; see e. g. Lin and Gottschalk[67] for a survey. Mostly bounding-box or -sphere tests are used to break down the complexity of checking each triangle with each other. For example if our moving objects are *convex polyhedra* fast algorithms have been developed that work in expected linear time, see e. g. [37, 73]. Other methods that overcome the restriction of convexity have been developed on the basis of various hierarchical bounding volume approaches: oriented bounding boxes used in OBBtrees [33], swept sphere volumes [64], and discretely oriented polytopes (k-DOPs) [60].

In addition, possibly colliding objects are identified by Sweep-and-Prune strategies [10]. As opposed to bounding volume hierarchies, regular grids partition the scene into voxels [4, 93]. Alternatively, graphics hardware [1] can be employed to detect collisions in image-space, which was even investigated for cloth modelling [84].

By these approaches interactive response times are possible on present hardware for models consisting of several thousand triangles. To achieve interactive response times in simulations that are governed by ODEs more sophisticated techniques are needed that allow for fast distance computations in even shorter time.

3.2. The Situation in Cloth Modelling

The situation for simulating accurately a cloth-like surface is somewhat more difficult to deal with than a usual distance computation of an ensemble of rigid bodies.

For calculating textiles we have complicated nonlinear differential equations to solve and the surface can adapt to almost any imaginable surface topology making the calculation of possible collisions difficult. Thus collision and self-collision detection are often the most time consuming part of the simulation algorithm.

In the past years very fast cloth simulation techniques have been developed to drive the cloth through simulation using large time-steps achieving almost real-time calculations. Having to deal with these large time-steps collision detection and its response is an even more difficult problem to solve because the cloth movement within one time-step can be quite large.

Moreover, calculation time can be dramatically affected by introducing collision-detection and -response. Since additional stiffness may be introduced when reacting on a collision, the numerical solution of the differential (difference)

equations may become instable using the large time-steps and hence we have to use smaller time-steps sometimes dramatically increasing calculation time. Therefore, special care should be taken in the response. This leads to collision foreseeing and preventing considerations formulating constraints on the movement of cloth-particles.

Accurate collision response is only possible with a measure of distance between close mesh surfaces. Using the distance an adaptive, constraint based approach for a collision response can be employed to avoid inter-penetrations.

The basic geometric elements for a collision detection algorithm may be the triangles (or polygons) of our virtual objects in the scene we are considering. Every surface involved in the simulation process, being part of the cloth itself or being part of the scenery, can also be assumed to be triangulated. For collision detection we basically have to make distance computations for each possible pair of triangles. Clearly this makes up an unrealistic task making up the so called *complexity problem*.

A lot of possible solutions how to break down this complexity problem have been proposed in the recent years. The idea is explained quickly: 'Do not test all the triangles'. But how do we find the triangles to be tested?

- Embed complex objects into *bounding volumes* which we can easily check on intersections. Moore already described this fundamental technique in 1988 [74].
- Do not check triangles far away from each other using grids or hierarchical structures.

These two aspects make up a fast collision detection algorithm and in the literature we find a whole variety of 'fast' and robust implementations.

3.3. Preliminary Tests, Bounding Volumes

Reducing complexity by using fast and not so many tests for complicated, complex objects is the basic idea and to this end. Moore and Wilhelm introduced Bounding Volumes (BV) in which they embed the objects to be tested.

These Bounding Volumes are geometric objects containing a group of objects, primitives or just one single complex object. Collision detection is then performed in two steps: First we detect only intersecting bounding volumes, then we check for collisions inside the volumes.

Many different types of geometric objects have been proposed to play the role of bounding volumes each having their pros and cons depending on the geometric properties of the colliding objects. However, there are some points to be considered when choosing the right bounding volume for your application:

- Is the collision test between two different bounding volumes efficient and fast?
- Is the construction of the many bounding volumes surrounding a complex object easy to perform? Can we do this on the fly?
- Are a lot of bounding volumes needed for the geometry of our complex objects?
- Do I need a lot of storage to describe an individual bounding volume?
- How well does the bounding volumes fit the model?

At this point we should mention at least Bounding Boxes (BBs) possibly being bounding volumes (BVs). Well, bounding boxes come in several variations:

AABB Axis aligned bounding boxes, where we use the usual bounding box aligned to the world space coordinate system [83]. Two three-dimensional coordinates are enough describing such a AABB: Simply calculate maximal and minimal point of the object to be encompassed. For these BVs test on intersection is performed very easily by simply checking the given maximal points. Two AABBs intersect, if and only if we have interval intersections along all of the three coordinate axes.

OOBB Object oriented bounding boxes, where we allow in addition rotated and translated bounding boxes [33].

Spheres Bounding spheres, where we try to encompass our object by small spheres. Again test on intersection of the spheres are easily computed and have just radius and center to be stored. But it is not easy to find a good covering and are not easily build on the fly.

***k*-DOPs** Discrete oriented polytope, which is is a convex polyhedron defined by k half-spaces enclosing the objects. This is the type of bounding volume we will consider further. Advantages are obviously fast convergence and easy check since we have the same test as for AABBs by simply checking interval intersections of the k half-spaces[60].

For collision detection in cloth modelling we have to consider the ever changing shape of our textile. Hence having build once an encompassing of our cloth for one time-step will not be a sufficient covering for the next. Therefore, because arbitrary object deformations cannot be expressed by rigid rotations and translations, orientable bounding volumes like OBBs[33], QuOSPOs[42], or Dynamically Aligned DOPs[92] are not suitable for cloth modelling.

We follow the ideas of Mezger et al. [72] and therefore choose common k-DOPs[60] as they show better convergence than spheres[49] or ordinary AABBs[83]. By convergence we mean the hierarchical series of bounding volume approximation of the surface (or object) to be checked.

3.3.1. *k*-DOPs

A k-DOP[60] (discrete oriented polytope) is a convex polyhedron defined by k half-spaces denoted as

$$H_i = \{x \in \mathbb{R}^m \mid n_i^T x \leq b, n_i \in N, b_i \in \mathbb{R}\}.$$

The normals n_i of the corresponding hyper-planes of all k-DOPs are discrete and form the small set $N = \{n_1, \ldots, n_k\} \subseteq \mathbb{R}^m$. For arithmetic reasons the entries of the normal-vectors are usually chosen from the set $\{-1, 0, 1\}$. In order to turn the intersection test for the polyhedrons into simple interval tests, the hyper-planes have to form $k/2$ parallel pairs. E.g. an axis aligned bounding box (AABB, 6-DOP) in $\mathbb{R}^3$ is given by $N_6 = \{(\pm 1, 0, 0), (0, \pm 1, 0), (0, 0, \pm 1)\}$, an octahedron (8-DOP) is generated by setting all normal components to ± 1. We usually use 14-DOPs ($N_{14} = N_6 \dot{\cup} N_8$) or 18-DOPs (AABB with clipped edges).

The easiest way to build the k-DOP bounding volume for a set of points is inserting them into a primarily empty k-DOP by updating its $k/2$ intervals accordingly. The overlap test between two k-DOPs is implemented by interval tests similar to the common AABB, indicating disjointness as soon as one pair of intervals is disjoint. Thus, the maximal number of interval tests is $k/2$ (in the overlapping case).

3.4. Hierarchy, Spatial Subdivision

Bounding volumes break down considerably the amount of triangles to be tested. However, we want to check only those objects (hence the triangles) which are close to each other. How do I quickly determine those objects within one particular region in space? This is a common problem in computer graphics not only in (physically based) simulations. Two ways of tackling the problem are well known:

- Grid subdivision and voxel-structures.
- Hierarchical subdivision such as octrees.

Hierarchical structures are known to reduce complexity enormously and thus should also be our choice in collision detection for cloth-modelling. Grid subdivision is the simplest way to divide space. The insertion of vertices into the grid cells can be performed easily and one should use hashing to store the sparse occupied voxel cells.

However as promised, we will build up here a hierarchical space partitioning. In addition we have to build the hierarchical space structure for each time-step dynamically and therefore have to have some sort of update mechanism. If we have arbitrarily moving objects this update can be very complicated. Octrees are hierarchical structures dividing space by filling objects into the structure. Why not doing it the other way round. We take our objects and build up a hierarchical structure in space.

The advantage is obvious if we consider objects with no constant geometric structure but having constant local topology such as deformable materials. However, as the objects freely move around the hierarchy has to be build efficiently on the fly in order not to generate a bottleneck and to generate new proximity information among the objects.

3.4.1. k-DOP Object Hierarchy

Let BV_k be the tightest k-DOP enclosing a set of vertices and $\widetilde{\bigcup}$ the operation forming the tightest k-DOP enclosing out of a set of k-DOPs. Then, like AABBs also k-DOP bounding volumes satisfy the equation

$$BV_k(V) = \widetilde{\bigcup}_{p \in P(V)} BV_k(p) \tag{77}$$

for a set of vertices V and an arbitrary partition $P(V)$. Hence, the optimal bounding volume for a node in the hierarchy can be easily computed by merging its child bounding volumes. Vice versa the hierarchy can be efficiently built using a top-down splitting method. Figure (14) shows two hierarchy levels for the 18-DOP-hierarchy of an avatar.

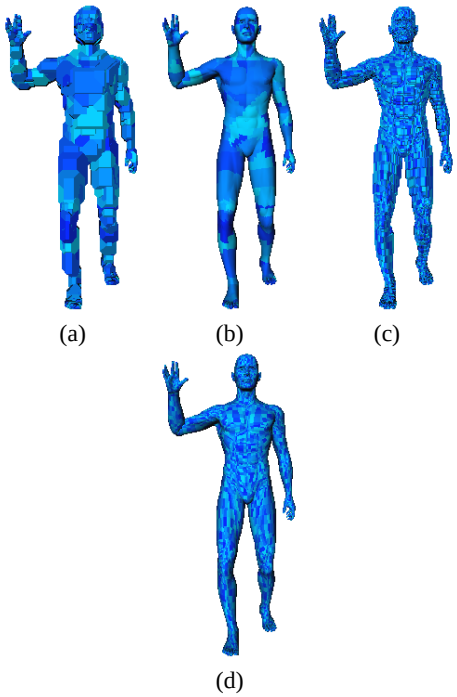


Figure 14: *Two levels of an 18-DOP-hierarchy. (a) and (c) show the 18-DOPs, (b) and (d) the corresponding regions on the surface.*

3.4.2. Node Split

As for the usual octrees we have to define how we are going to split a large k-DOP into smaller ones. As usual the bounding volumes are split according to the longest side. The longest side of a k-DOP is determined by the face pair with the maximum distance. The k-DOP is split parallel to this face pair through its center. As generally some polygons are cut by the splitting plane, they are assigned to that child node which would contain the smallest number of polygons. In the lower hierarchy levels, if all polygons happen to be cut, each of them is assigned to its own node. Finally, as the

corresponding vertices for the node are known, the *k*-DOPs can be optimally fitted to the underlying faces.

Although this method is simple, it turns out to be efficient on the one hand and to produce well balanced trees on the other hand. The complete hierarchy setup for objects holding several thousands of polygons can be performed within merely a second, allowing to add objects dynamically to the scene. To achieve optimal collision detection performance, the splitting continues until one single polygon remains per leaf.

3.5. Collision, Self-Collision, and Heuristics

The above collision detection technique is now applied to an actual cloth surface. For this we consider each polygon of our cloth-surface as a separate, individual object participating in the simulation. Upon these objects, together with the objects the cloth may collide with, we build the hierarchical structure described above. In going down to the individual polygons we may process collision regardless if we have polygons of different objects or polygons of the same (deformable) object.

In addition to this 'usual' setup we have for cloth additional information we we will use to speed up computation. Adjacency and surface curvature are two criteria which allow for additional information on possible collisions (we easily sort out triangles which can not collide). This is the basic idea of the ingenious collision detection algorithm proposed by P. Volino and N. Magnenat-Thalmann. In addition they observe the history of close regions to guarantee a consistent collision response[12]. Recent publications[87] additionally address *k*-DOPs as bounding volumes. Provot[77] describes a similar approach for the surface curvature heuristic.

The observation basically is the following: Following the basic detection procedure we seek those bounding volumes that overlap and will check again the child-bounding volumes being left with individual polygons to be checked. For cloth we consider as said the individual polygons as individual objects and hence their enclosing bounding volumes will intersect. Adjacent polygons however will never self-intersect. For efficiency we therefore should exclude those directly adjacent polygons.

Can we extend the region of non-self-intersecting triangles? "A flat surface cannot have self-collisions. If a surface has self-collisions it must be bent enough to form a loop." [88] A vector is searched that has positive dot product with all normals of the region. If such a vector exists and the projection of the region onto a plane in direction of the vector does not self-intersect, the region cannot self-intersect either.

This is an important optimization to the self-collision detection problem and should be included in the algorithm. However keep in mind that this is only possible because of the special situation in cloth modelling. To do this, a cone is maintained for every region representing a superset of the normal directions. The cone can be calculated during the bottom-up hierarchy update by very few arithmetic operations. The apex angle α of the cone represents the curvature of the region, indicating possible intersections if $\alpha \geq \pi$.

Still there remains the problem that hierarchy regions can have severe non-convex shape and therefore compromise the robustness of the surface curvature criterion. Figure (15) shows such a surface that self-intersects although the apex angle of its normal cone is rather small. Mezger et al.[72] divide such a mesh into several face groups and build an adjacency matrix for the groups. The curvature heuristic is not applied to non-adjacent groups during the self-collision test.

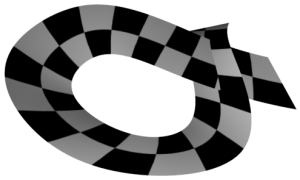


Figure 15: *Self-intersecting mesh with correlated normals but concave shape.*

3.6. Proximity, Distance and Collision Response

Whenever two colliding hierarchy leaves have been found, the distance between each pair of faces is calculated, and candidate pairs are detected and passed to the collision response. This problem has been extensively studied and can be calculated via the Gilbert-Johnson-Keerthy algorithm [31, 8]. Alternatively the Lin-Canny type algorithms ([68, 69, 73]) can be used.

The simplest response to a collision would be just to relocate the colliding particles. Hence imposing virtual springs having almost infinite stiffness and discontinuities. Clearly this affects the numerical performance. For animated scenes an efficient collision response method is crucial. In particular, numerical stability and performance in the context of large time steps have to be preserved and no additional stiffness should be introduced into the system.

Therefore, for an implementation we suggest the use of constraints based on the efficient implementation presented by Baraff and Witkin [3].

To allow large time steps and preserve stability collision detection and response aims at avoiding collisions before they occur. Therefore, not only colliding faces should be detected but also proximities, that is faces closer than a distance ε_{close}. In this case, the collision detection returns pairs of colliding or close faces together with the closest points on these faces. While the collision detection process returns

candidate faces, an adequate collision response has to be calculated for each involved particle. For that purpose, the collisions are sorted by the estimated distance of the two faces after the next time step. This distance is estimated by the current distance and the relative velocity.

We may now constrain the velocity in the normal direction of the close object as in [41, 77]. If the velocity is constrained, it has to be preset with appropriate values. The collision response distinguishes between two different cases:

1. Collisions of two deformable faces.
2. Collisions between a deformable face and a face of the pre-computed rigid environment.

In both cases only faces moving towards each other are handled. Their velocity in the constrained direction is set to assert a minimum distance d_{min}. This velocity is scaled by $\gamma = v_{rel} / \frac{d_{min}}{\Delta t}$ to simulate a resting contact, where v_{rel} is the relative velocity of the two faces and Δt is the current time step.

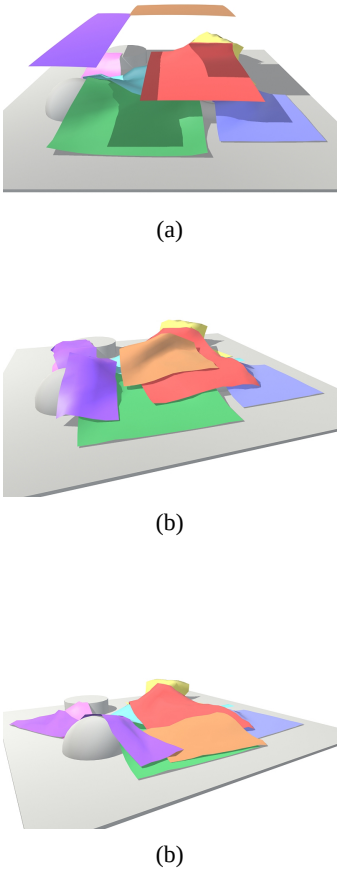
In the first case no velocity and therefore no momentum is transferred between the two faces of the object. In the second case additionally to the velocity used to assert the minimum distance, the velocity of the rigid object in the constrained direction has to be added to the velocity of the deformable object.

One may now constrain the velocity of the colliding particles in the direction n_s of the normal direction of the close object and preset it to

$$v = \frac{1}{\Delta t}\gamma(d_{min} - d) + \pi(v_r)\,, \tag{78}$$

where π is the projection onto the oriented direction n_s, d is the distance between the two faces and v_r is the velocity of the rigid object. If inter-penetrations occur, the faces are separated again, as long as their distance is smaller than a preset value.

Since only one direction of the particle velocity is constrained, the particle is still free to move according to the forces acting on it in the other directions. Hence it is easy to model stiction by constraining the other directions and replacing the projected velocity of the rigid object by the entire velocity of the rigid object in equation (78) such that the cloth moves with the rigid object. Collision constraints must be released when forces drag the particle away from the collision object. These release forces are estimated at the beginning of every integration step.



(a) (b) (b)

Figure 16: *Performed collision detection for a complex multiple layer cloth.* [28]

4. BTF Acquisition & Rendering

(*Reinhard Klein, Ralf Sarlette, Mirko Sattler*)

This subsection of the tutorial is structured in three parts. In the first part the term **BTF** ***B****idirectional* ***T****exture* ***F****unction* is explained and a short overview of the previous work is given. The second part deals with the automatic acquisition of BTF data and in detail the BTF data of clothing. The last part gives an outlook on compression and rendering this data on arbitrary meshes.

4.1. Introduction & BTF definition

In order to visualize real world objects with high fidelity, one has to acquire and render the physical reflection properties of an object surface. Therefore viewing and illumination directions have to be taken into account when rendering an object. Furthermore, an interactive change of the viewing position and the lighting conditions should be possible, to allow the user to simulate a desired scene. Currently, physical reflection properties are captured using light field techniques [32, 66, 19, 90, 89]. The drawback of most light field rendering approaches is that the measured material properties are coupled with a fixed geometry, thus not allowing to change the geometry or the material without remeasuring the object. These methods utilize either a fixed lighting environment or

Figure 17: *A pair of trousers rendered using BTF data (**Proposte** data set).*

a fixed camera position and are hence only suitable for static scenes or environments.

The term BRDF (Bidirectional Reflection Distribution Functions) was introduced by Nicodemus [29] in 1970. In computer graphics measured BRDF data is usually used to fit the parameters of different theoretical models [62, 65] or to create a lookup table.

By definiton, a BRDF does not contain any geometry information. Therefore these functions do not include self shadowing caused by the microstructural geometry of the surface, subsurface light transport, etc, because the surface is treated as if it consists of only one material. Hendrik et al. [45] proposed a method to capture spatial varying materials, which is again bound to the measured object.

Polynomial texture maps (PTM) as described by Malzbender et al.[71] are an extension to standard texture maps. As opposed to storing a color per pixel, these PTM's store second-order bi-quadratic polynomial coefficients per pixel. These polynomials model the changes that appear to a pixel color based on either light source position or viewing direction.

Another method to capture reflection properties is the usage of BTFs. The term BTF (bidirectional texture function) was introduced by Dana et al. [14, 13] in 1999. The term represents the appearance of a texture as a function of viewing and illumination direction, hence capturing effects caused by the macroscopic roughness of a surface, like self-shadowing, occlusion, inter-reflections or view-dependent reflections (see figure 18). On the other hand, by measuring flat material

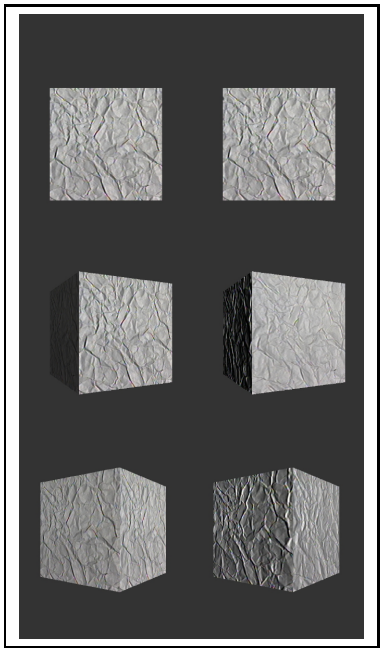


Figure 18: *A textured cube using CUReT [46] data. The sample "crumpled paper" is used. In this figure, from top to bottom, the cube is spinning around his vertical axis. The left cube shows normal texturing , while the right cube uses the BTF data.*

samples, BTFs are geometry independent.

Ignoring wavelength and fixing time, hence ignoring time dependent effects like fluorescence, a BTF can be considered as a specific 6D reflectance field:

$$BTF = f(\theta_i, \phi_i, x, y, \theta_r, \phi_r) \tag{79}$$

with incident (θ_i, ϕ_i) and reflected flux (θ_r, ϕ_r) at a specific surface point (x, y) [70, 14].

4.2. Acquisition of BTF data

The work of Dana et al. resulted in a 61 sample data set of real-world surfaces publicly available in the CUReT[46] database. The database was created to investigate the visual appearance of real-world surfaces, but does not satisfy all of the requirements for high quality cloth rendering. Besides the insufficient quality of some images, not all necessary viewing and illumination directions on the hemisphere were taken into account. Based on this work we build up a

measurement setup as described in 4.2.1. The quality of the measurements on the other hand has to be improved in some parts.

4.2.1. Measurement Setup

The equipment for the acquisition of the BTF data is mainly composed of a lamp, a camera, a robot and a personal computer. For the realistic reproduction of cloth appearance, the lamp has to approximate directional sunlight. To achieve this, we use a HMI-light source[†], which is widely used in the professional film- and photo-industry, in combination with a Fresnel-lens. As capturing device we utilize a programmable high resolution CCD-camera[‡]. The positioning of the sample is done by a robot arm[§]. All these parts are controlled by a personal computer.

Figure 19: *Picture of the laboratory. The black casings of the robot and light source are removed.*

To reduce the influence of scattered light, the lab is darkened and the hardware used for measurements is covered with dark casings. In the center of a semicircle-rail-system (170cm radius) the robot is placed on a laboratory bench. The camera is mounted on a wagon which resides on the rail-system. The light source is fixed on one end of the rail-system as shown in figure 19.
This computer controlled setup allows the reproduction of every constellation of the view/light-direction with reference to the sample surface (10cm x 10cm). See figure 20 for a sketch of the measurement setup.

4.2.2. Measurement Procedure

To achieve a sufficient measurement quality for rendering, the equipment has to be calibrated.

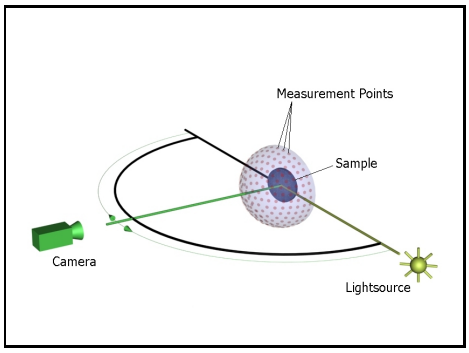


Figure 20: *Illustration of the measurement setup. A fixed light source and different camera positions are shown.*

- To reduce the positioning error caused by deviations between the robot and the rail-system, the sample holder mounted on the robot is tracked through the camera. Depending on the position of the camera on the rail-system, a correction movement for the robot has to be computed. To reduce the positioning error further, marker points, placed on the sample holder, are detected during the post-processing, allowing a software jitter correction.
- A geometric calibration[79] has to be applied to the camera to reduce geometric distortion, which is caused by the optical system of the camera.
- For each sample to be measured, the aperture of the camera has to be adjusted. Because the aperture stays fixed for all images of a sample, saturated and dark pixels have to be reduced as much as possible.
- To achieve the best possible color reproduction the combination of the camera and the light source has to be color calibrated[14]. Because of the use of products, often used for professional color photography, there are some solutions on the market. For the measurement of the camera color profile a special CCD-Camera standard card should be used.

A set of view/light-directions has to be chosen, fitting the reflection properties of the sample surface (e.g. see table 5). This can be a set of nearly equidistantly distributed directions, or their distribution can be denser in areas with a rapidly changing degree of reflection. During the measurement for every position of the data-set an image is taken and stored for post-processing. Thereafter, another image of the CCD-Camera standard card has to be taken to detect deviations occurred during the measurement process.

4.2.3. Post-processing

Every captured image is transformed from the camera raw format to RGB-color space using the camera color profile and the transformation profiles delivered by the camera manufacturer. Using the jitter correction derived from the marker points on the sample holder, the pixels covered by the sam-

† bron imaging HMI F575
‡ Kodak DCS760
§ Intelitek SCORBOT-ER 4u

ple are extracted from each image to generate *normtextures*, described in section 4.3.

4.3. The demonstration BTF database

Figure 21: *Sample normtextures from the demonstration database. The top row shows (from left to right)* ***IronLady, Stone, Proposte*** *with* $\theta_l = 0°$, $\phi_l = 0°$ *and* $\theta_v = 34°$, $\phi_v = 240°$. *The bottom row views use* $\theta_l = 51°$, $\phi_l = 280°$ *and* $\theta_v = 66°$, $\phi_v = 144°$.

For a startup we have created a demonstration BTF database. For this, we used a tileable texture maps with their corresponding height field. All images were rendered using the freeware raytracer POV-Ray [48] at a resolution of 512 by 512 pixels. Thereafter, these images were post-processed, creating *normtextures*. That is mapping the rendered images onto a frontal viewed square for further use as a texture map (Figure 22). The resolution of the *normtextures* is 128 by 128 pixels. Much of the resolution of the raw image is lost due to the extraction of the region of interest. Therefore cutting has to be done on a centered inner area of the raw image. The creation of *normtextures* allows blending of the textures at a triangle as described later. To have a consistent sampling density over the hemisphere we chose $\Delta\theta = 17°$ for each of the viewing and illumination directions. This yields in different $\Delta\phi$ shown in Table 5 and a different number of images per used θ segment. The sampling has 81 viewing and 81 illumination directions or 6561 images for all combinations. For practical measurements we have only 6480 images, because 81 positions would be occupied by both, the camera and the light source, and therefore can not be measured. Another method to generate a database is the synthesis, as described in [16, 18, 35, 70].

Figure 23 shows the general terminology used for measuring. At the vertex X we have a local coordinate system with the vertex normal N and a preferential direction W. The direction to the light source L is given by θ_l and ϕ_l; to the viewing position V by θ_v and ϕ_v.

The data in our demonstration database can easily be replaced by measured real-world data. The handling of denser data sets is possible, only limited through the amount of available memory. The database is indexed and a lookup table containing angles and image numbers is generated, allowing an easy database access at runtime. Three data sets were created: **IronLady, Stone** and **Proposte** (see figure 21). Each dataset is converted into the JPEG [47] format and needs about 27 megabytes disk space for storage.

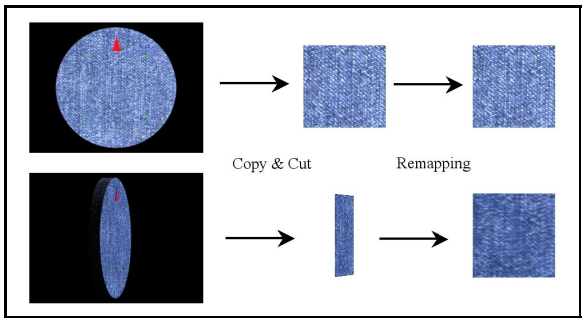


Figure 22: *Creating normtextures by remapping the cut image onto a square.*

θ [°]	$\Delta\phi$ [°]	No. of images
0	-	1
17	60	6
34	30	12
51	20	18
68	18	20
85	15	24

Table 5: *Sampling density of viewing and illumination angles of the BTF database.*

4.4. Rendering of BTF data

This part describes the rendering of a BTF data set on a given geometry. This involves the solution of the rendering equation [52]. Following the notation of [51] gives:

$$L_o(x,\vec{w}) = L_e(x,\vec{w}) \quad (80)$$
$$+ \int_S f_r(x,x' \to x,\vec{w}) L_i(x' \to x) V(x,x') G(x,x') dA'$$

with:
L_o: outgoing radiance $[Wm^{-2}sr^{-1}]$
L_e: emitted radiance $[Wm^{-2}sr^{-1}]$
L_i: incident radiance $[Wm^{-2}sr^{-1}]$
f_r: BRDF $[sr^{-1}]$
$\vec{w}$: outgoing direction
x: surface location
x': another surface location
V: visibility information (can x' be seen by x?)

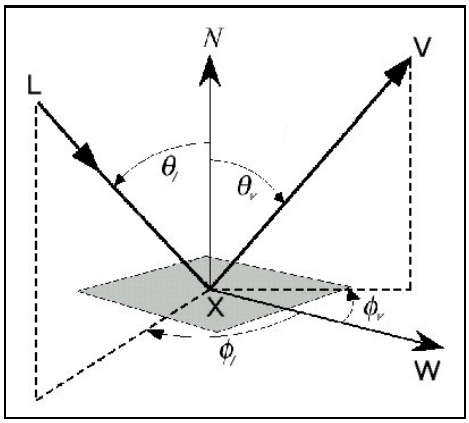


Figure 23: *General geometry used.*

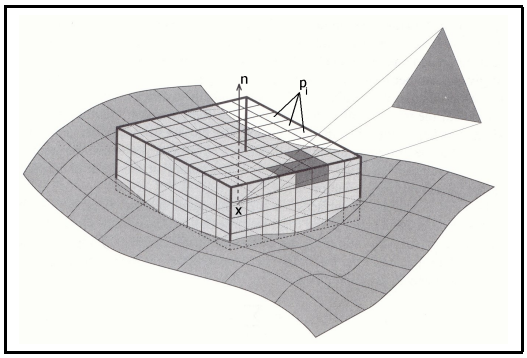


Figure 24: *The hemicube model at surface point* $\mathbf{x}$ *with normal* $\mathbf{n}$. p_i *are hemi pixels. Image modified from* [80].

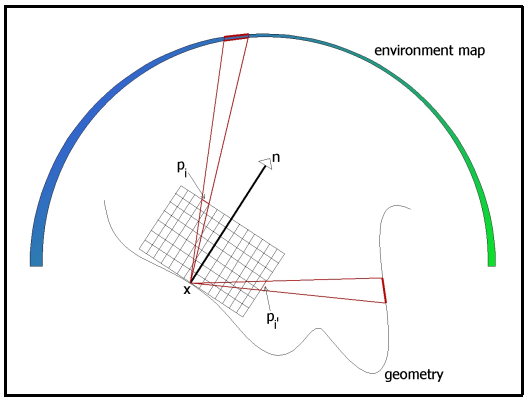


Figure 25: *2D sketch of figure 24. A hemicube at surface point* $\mathbf{x}$. *With hemicube pixel* $p_i \leftarrow$ *environment map and* $p_i' \leftarrow$ *geometry.*

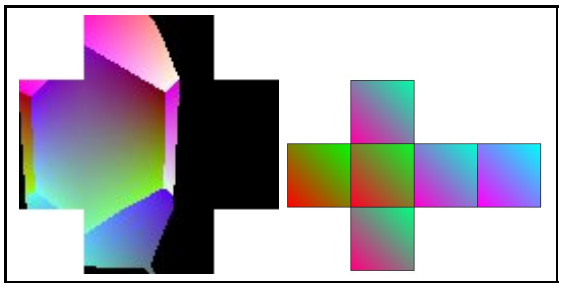


Figure 26: *Visibility map. Black is the mesh color, color encoded is the position on an environment map (shown on the right).*

G: geometry factor

We assume $L_e(x,\vec{w})$ to be zero. For creating realistic illumination conditions, image based lighting techniques [34, 44] can be applied. The usage of high-dynamic range images even improves this results [20]. Using an hemicube [11] method (figure 24) pre-computed visibility information calculations and integration over the incoming radiance can be done as shown in [30].
By color encode the position of an environment map into the visibility map (figure 26) and use this as a lookup for the currently used environmental illumination, we are able to do a fast evaluation of the term:

$$L_i(x' \rightarrow x)V(x,x')G(x,x') \tag{81}$$

in (80). We approximate the integral part of(80) as follows:

$$\begin{aligned}\int_S f_r(x,x' \rightarrow x,\vec{w})L_i(x' \rightarrow x)V(x,x')G(x,x')dA' \quad (82)\\ \approx \sum_i f_r(x,p_i \rightarrow x,\vec{w})L_i(p_i \rightarrow x)V(x,p_i)G(x,p_i)\\ \approx \sum_i f_r(x,p_i \rightarrow x,\vec{w})\tilde{L}_i\end{aligned}$$

with i as the index of the pixels p_i of the hemicube (figure 25) and $\tilde{L}_i$ as

$$\tilde{L}_i = \begin{cases} L_i(p_i \rightarrow x)V(x,p_i)G(x,p_i) & : if\ p_i \leftarrow environment map \\ 0 & : if\ p_i \leftarrow geometry \end{cases} \tag{83}$$

Self-shadowing is handled with the second case ($p_i \leftarrow$ *geometry*). This could also be replaced by a color encoded geometry storing illumination intensities to handle inter reflections.
We interpret our BTF database as discreet solutions to the integral part of (80). The reflection properties f_r are naturally stored in each BTF image. We use (81) as input for the lookup into the database. The main challenge using this approach is the multitexturing of all BTF images of all visible light directions per vertex. Which means that for all pixels of the visibility map, which are not marked as occluding geometry, the corresponding BTF image has to be weighted with the pixel color and in a second step all these pictures have to be combined. This can not be handled by today's graphics hardware and a suitable re-parametrization of the BTF data is necessary.

For this reason until now the following implementation was done. The illumination of the geometry is provided by assuming light coming uniformly from all directions and additional point light sources.
At runtime, for each frame, for all vertices, all relevant angles are computed : θ_v, ϕ_v, θ_l, ϕ_l. Using n point light sources, θ_l, ϕ_l becomes $\theta_l(n)$, $\phi_l(n)$.
With the above mentioned lookup table, the texture maps can be extracted out of the database, which are nearest to the current angles and used for blending. This is done once for every vertex. Figure 27 shows the blending and weighting of an arbitrary triangle with vertices V_1, V_2, V_3 and corresponding vertex normals N_1, N_2, N_3. I is the intensity dependent of the regarded light source. To determine,

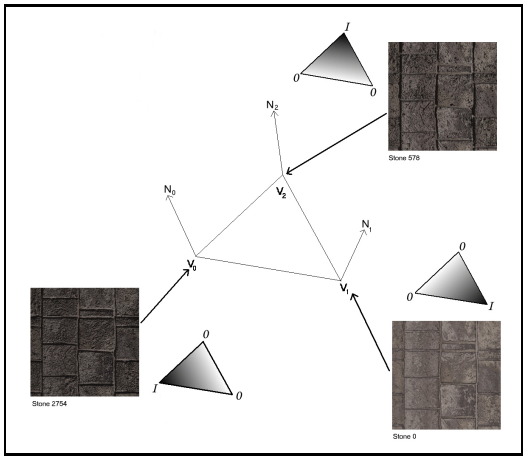


Figure 27: *Blending three looked up normtextures for usage on a triangle. Weighting occurs according to the grey shaded triangles.*

whether the light source is visible from a specific vertex, the world coordinates of the light source are transformed into the local coordinate system of the vertex. With a simple lookup into the visibility maps the visibility of the light sources can be determined for each vertex. Naturally this method works only, as long as the light source is not within the convex hull of the mesh. This disadvantage can be overcome by using also a depth test with the projected light source.
Since the correct illumination is stored in the images from the BTF database no further illumination calculations are necessary. The weighting derived from the visibility maps and the three images shown in figure 27 is computed in one step. To approximate the term caused by the light coming uniformly from all directions the corresponding textures with light direction $\theta = 0$ and $\phi = 0$ are used. These are weighted by the calculated brightness derived from the visibility maps and blended in a second pass with the first retrieved textures [30].

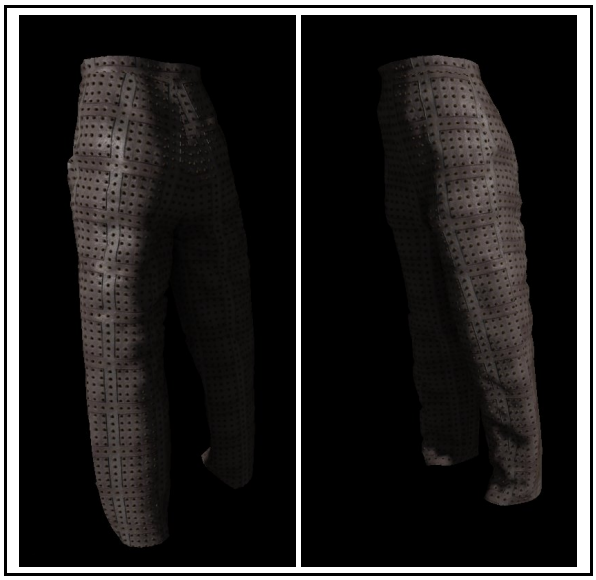


Figure 28: *Two views of a pair of trousers using the **IronLady** data set.*

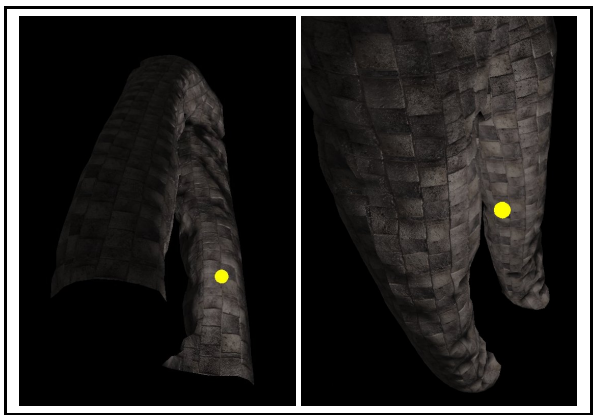


Figure 29: *Two views of a pair of trousers using the **Stone** data set. The single point light source is marked as a yellow dot.*

Using this method geometry induced self-shadowing is automatically determined during the evaluation of the visibility maps. This is essential in order to visualize folds which have a mayor impact on the realistic outlook of clothing. For an example see figures 28 and 29, which shows several different views of the a pair of trousers using the *IronLady* and the *Stone* data set.
To texture an arbitrary geometry with a BTF data set for each vertex the viewing angle (θ_v, ϕ_v) and for all point light sources, the light direction $(\theta_{l,i}, \phi_{l,i})$ is computed for each frame. Static geometry with 4200 vertices can be viewed and illuminated by several point light sources at around 20

frames per second on an 1,4 GHz Athlon using a GeForce3 graphics card. To handle dynamic meshes the mentioned visibility maps must be updated too, which can not be done at interactive rates by now.

5. Advanced Local Illumination

(*Katja Daubert, Jan Kautz*)

Traditionally hardware renderers only support the Phong lighting model in combination with Gouraud shading using point light sources. However, the Phong lighting model is strictly empirical and physically implausible. Gouraud shading also tends to undersample the highlight unless a highly tessellated surface is used.

Advanced lighting algorithms allow the usage of other more complex reflectance models to simulate more compelling materials such as velvet and brushed metal.

Todays graphics hardware supports per-pixel lighting with which diffuse and specular bump maps can be rendered at interactive rates. New algorithms have been proposed that can even render bump maps with better reflectance models and including shadows.

These topics will be addressed in this part of the tutorial.

5.1. Lighting Models

First we would like to explain what a lighting model really is, and then define it mathematically.

When light hits a perfectly reflective surface, i.e. a mirror, the reflected light leaves the surface at the same angle it hit the surface. However, perfect mirrors do not exist in reality. In contrast, most surface have a very complex micro-structure. This micro-structure makes different materials appear differently.

When light hits such a surface, it is not reflected towards a single direction, but rather to a cone of directions. If the surface is completely diffuse, light even scatters equally in all directions.

In computer graphics the bidirectional reflectance distribution function (BRDF or also reflectance model) is used to describe the way a surface reflects light. A reflectance model can be seen as a material description that modulates the intensity of the light that arrives at the surface. For every light incident direction it tells you how much light is being scattering to which exitant direction. For example the Blinn-Phong model that is used by OpenGL can be described as:

$$\begin{aligned} f_r(\hat{\omega}_o, \hat{\omega}_i) &= k_d + k_s \frac{(\hat{n} \cdot \hat{h})^N}{\hat{n} \cdot \hat{\omega}_i}, \\ \hat{h} &= \frac{\hat{\omega}_i + \hat{\omega}_o}{|\hat{\omega}_i + \hat{\omega}_o|}, \end{aligned}$$

where $\hat{\omega}_i$ is the incident light direction, $\hat{\omega}_o$ is the exitant light direction (i.e. viewing direction), $\hat{h}$ is the half-way vector between $\hat{\omega}_i$ and $\hat{\omega}_o$, all of which are in coordinates relative to the surface, i.e. relative to the local tangent frame consisting of the normal $\hat{n}$, the tangent $\hat{t}$, and the bi-normal $\hat{b}$. The parameters k_d, k_s, and N describe the diffuse coefficient, the specular coefficient, and the Phong exponent.

This not a complete lighting model, since the BRDF only tells you how light is scattered. A lighting model includes much more: how the light intensity decreases with the distance from the light source (e.g. quadratically), what kind of light sources are supported (e.g. point light or parallel light), if shadows are included, and so on. We will use the following simple lighting model for the rest of the chapter (which is similar to the OpenGL lighting model):

$$\begin{aligned} L(\hat{\omega}_o) &= f_r(\hat{\omega}_o, \hat{\omega}_i) L_i(\hat{\omega}_i)(\hat{n} \cdot \hat{\omega}_i), \\ L_i(\hat{\omega}_i) &= \begin{cases} \frac{I}{r^2} & \text{for point lights} \\ I & \text{for parallel light} \end{cases} \end{aligned} \tag{84}$$

This lighting model uses only a single point or parallel light source that has the brightness I, and is r units away from the illuminated surface. $L(\hat{\omega}_o)$ describes the radiance leaving at the surface point in direction $\hat{\omega}_o$ towards the eye; this is then perceived by the eye.

The standard OpenGL lighting model does not allow to change the function f_r, it always uses the Blinn-Phong model introduced above. In the next section, we will explain how this can be changed. Furthermore standard OpenGL only evaluates $L(\hat{\omega}_o)$ at every vertex and uses Gouraud shading to interpolate values within the triangle.

5.2. Rendering with Arbitrary Reflectance Models

In computer graphics, when we talk about materials or material properties, what we are really talking about is the reflectance properties of a surface that define how light arriving at the surface is scattered. In this part of the tutorial we will explain how surfaces with almost arbitrary materials can be rendered at interactive rates.

The main idea is to approximate a given reflectance model (e.g. for velvet or brushed metal) so that it can be used with OpenGL. A new algorithm called “separable decomposition” is employed [54].

Each of the two directions that a BRDF uses can be modeled as a 2D parameter, hence a reflectance model usually depends on 4 parameters. For an accurate representation this 4D function could just be sampled, but graphics hardware does not support 4D texture and a lot of memory would be needed for this representation.

Instead a separable decomposition is used, which approximates the 4D function with a product of two 2D functions.

$$\begin{aligned} f_r(\hat{\omega}_o, \hat{\omega}_i) &= g(\hat{\omega}_o) \cdot h(\hat{\omega}_i), \\ L(\hat{\omega}_o) &= g(\hat{\omega}_o) h(\hat{\omega}_i) L_i(\hat{\omega}_i)(\hat{n} \cdot \hat{\omega}_i), \end{aligned}$$

Using texture mapping the equation for $L(\hat{\omega}_o)$ can easily be evaluated on the graphics hardware. Each of these 2D functions $g(\hat{\omega}_o)$ and $h(\hat{\omega}_i)$ can be sampled and stored in a texture map. For every vertex of every polygon you have to compute $\hat{\omega}_o$ and $\hat{\omega}_i$ and use it as texture coordinates. Then the polygon has to be texture mapped with the textures containing $g(\hat{\omega}_o)$ and $h(\hat{\omega}_i)$ and the computed texture coordinates. Blending has to be set to modulate, so that $g(\hat{\omega}_o)$ and $h(\hat{\omega}_i)$ are multiplied together. The term $L_i(\hat{\omega}_i)(\hat{n} \cdot \hat{\omega}_i)$ can be multiplied to the result of $g(\hat{\omega}_o) \cdot h(\hat{\omega}_i)$ by enabling OpenGL lighting with only a diffuse component.

Rendering of arbitrary materials using this approximation is very fast because it boils down to computing texture coordinates and blending two texture maps together.

The main trick is to reparameterize the original 4D reflectance model in a smart way, such that the approximation works well. We refer the reader to [54], [56], or [91] for more detailed descriptions. See Figure 30 for an example rendered with this technique at real-time rates.

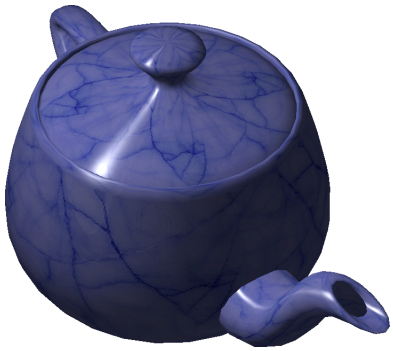


Figure 30: *Hardware accelerated (>60Hz) rendering of an anisotropic marble teapot.*

5.3. Bump Mapping

Blinn [5] has shown how wrinkled surfaces can be simulated by only perturbing the normal vector, without changing the underlying surface itself. The perturbed normal is then used for the lighting calculations instead of the original surface normal. This technique is generally called bump mapping.

If we have another look at the lighting model equation (see Equation 84), we can see a dependence on the normal $\hat{n}$. As mentioned in Section 5.1, the lighting is usually only evaluated at every vertex and not within a triangle, so the normals from the vertices are used to evaluate the Equation 84.

In order to simulate wrinkles, bump mapping requires a per-pixel normal, which is used for the evaluation of this equation. Graphics cards now support complex per-pixel operation which allow to perform this bump mapping technique at interactive rates [58].

Bump mapping is fairly simple to implement with these new features. These features include per-pixel dot-products, multiplication, addition, subtraction, so lighting models using only these operations can be used to do bump mapping. For every pixel we simply have to evaluate the lighting model.

Usually the Blinn-Phong model that was introduced in Section 5.1 is used to do bump mapping, because this model only mainly uses dot-products. Let us have a look at the lighting model using the Blinn-Phong reflectance model:

$$L(\hat{\omega}_o) \;=\; k_d I(\hat{n}\cdot\hat{\omega}_i) + k_s I(\hat{n}\cdot\hat{h})^N$$

If this is used in conjunction with bump mapping, the first term of the sum is usually entitled diffuse bump mapping and the second term is entitled specular bump mapping. Using the new per-pixel operations, this formula can be easily computed at every pixel. First, the normals are encoded in a texture map. Then $\hat{\omega}_i$ and $\hat{h}$ are computed on a per-vertex basis (will be interpolated across the triangle). Now, the graphics card has to be configured, such that it computes the equation above. For more details, please see [58].

5.4. Bump Mapping with a Spatially Varying Reflectance Model

As just mentioned, bump mapping usually uses simple lighting model such as the Blinn-Phong model [6] for the lighting calculations. While this is an appropriate and fast method to do bump mapping, it is not very flexible. The Blinn-Phong model does not have many parameters that can be tweaked to change the appearance of the bumpy surface and the chosen parameters (i.e. at least the exponent) have to remain constant over a polygon.

We will introduce a new bump mapping technique [55] which uses a modified version of the Blinn-Phong model, which offers more flexibility concerning the parameters. And what's more, those parameters can even change on a per-pixel level. See Figure 31 for an example of what can be done.

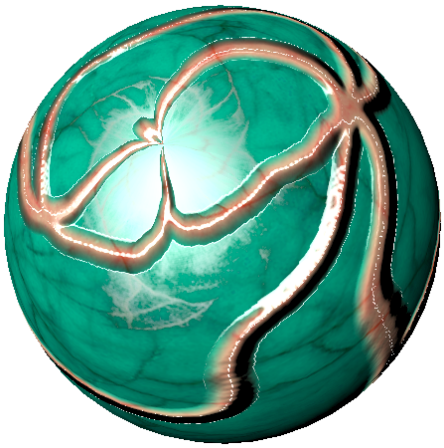


Figure 31: *Marble sphere with elevated "veins" using the spatially varying modified Blinn-Phong model.*

The lighting model using the modified Blinn-Phong model can be written as follows:

$$L(\hat{\omega}_o) = k_d(\hat{n}\cdot\hat{\omega}_i) + k_s\left(1-\left(\hat{h}\cdot\frac{\hat{t}}{\alpha_x}\right)^2-\left(\hat{h}\cdot\frac{\hat{b}}{\alpha_y}\right)^2\right).$$

This lighting models uses new parameters. The specular part does not depend on the normal $\hat{n}$ anymore, but on the tangent $\hat{t}$ and the bi-normal $\hat{b}$. These two vectors are divided by α_x resp. α_y, which have to be in the range $[0,1]$. The smaller these values are, the smaller the highlight will be. The model is anisotropic, which means, that the shape and the orientation of the highlight depends on the orientation of the surface. If different α_x and α_y are chosen, the model is anisotropic, otherwise it is isotropic. In the anisotropic case, the tangent and bi-normal define the main orientation of the highlight.

The implementation of this new lighting model works very much like standard Blinn-Phong bump mapping, only that the $\hat{t}/\alpha_x$ has to be stored in one texture map, and $\hat{b}/\alpha_y$ has to be stored in a second texture map. Of course, the graphics card has to be set up, that it performs the necessary dot-products etc., but this can be done using OpenGL extensions or DirectX 8.

Other BRDFs can be used as well, if they can be implemented with the supported per-pixel operations. If not than an upcoming feature called dependent texture lookup can be used to implement arbitrary functions on the graphics hardware. Dependent texture lookup

uses the values from a texture map to lookup into another texture map, which nothing else than evaluating an arbitrary function. We refer the reader to [55] for more details.

6. A Spatially Variant BRDF Model for Textiles

(*Katja Daubert, Jan Kautz*)

One of the challenges of modeling and rendering realistic cloth or clothing is that individual stitches or knits can often be resolved from normal viewing distances. Especially with coarsely woven or knitted fabric, the surface cannot be assumed to be flat, since occlusion and self-shadowing effects become significant at grazing angles. This rules out simple texture mapping schemes as well as bump mapping. Similarly, modeling all the geometric detail is prohibitive both in terms of the memory requirements and rendering time. On the other hand, it seems possible to compose a complex fabric surface from copies of individual weaving or knitting patterns unless the viewer gets close enough to the fabric to notice the periodicity.

In this section we will explain a fast and memory-efficient method for modeling and rendering fabrics that is based on replicating weaving or knitting patterns. The method assumes we have one or a small number of stitch types, which are repeated over the garment. Using a geometric model of a single stitch, we first compute the lighting (including indirect lighting and shadows) using the methods described in [15]. By sampling the stitch regularly within a plane we then generate a view dependent texture with per-pixel normals and material properties.

In the next sections we will first take a look at the BRDF model we use to capture the light and view dependent data for a stitch or weave. We will then explain how to render clothing using this model and then briefly sketch how to acquire data and fit the model to it, to capture different stitches or weaves. More information can be found in [17].

6.1. Representation

We employ a spatially varying BRDF representation to capture the variation of the optical properties across the material. The two spatial dimensions are point sampled into a 2D array. For each entry in the array we store different parameters for a Lafortune reflection model [61], a lookup table, as well as the normal and tangent.

An entry's BRDF $f_r(\vec{l},\vec{v})$ for the light direction $\vec{l}$ and the viewing direction $\vec{v}$ is given by the following equation:

$$f_r(\vec{l},\vec{v}) = T(\vec{v}) \cdot f_l(\vec{l},\vec{v}), \tag{85}$$

where $f_l(\vec{l},\vec{v})$ denotes the Lafortune model and $T(\vec{v})$ is the lookup table. The Lafortune model itself consists of a diffuse part ρ and a sum of lobes:

$$f_l(\vec{l},\vec{v}) = l'_z \cdot \left(\rho + \sum_i \left[C_{x_i} l'_x v'_x + C_{y_i} l'_y v'_y + C_{z_i} l'_z v'_z \right]^{N_i} \right) \tag{86}$$

Before evaluating the lobe we transform the light and viewing direction into the local coordinate system given by the sample point's average normal and tangent, yielding $\vec{l}'$ and $\vec{v}'$. For more information on the parameters refer to [17]. The lookup table $T(\vec{v})$ stores color and alpha values for each of the original viewing directions. Values for directions not stored in the lookup table are obtained by interpolation.

The alpha value stored in the lookup table is used to evaluate the transparency. It is not considered in the multiplication with f_l, but used during rendering to determine if there is a hole in the model at a certain point for a given viewing direction. The alpha values are interpolated similarly to the color values.

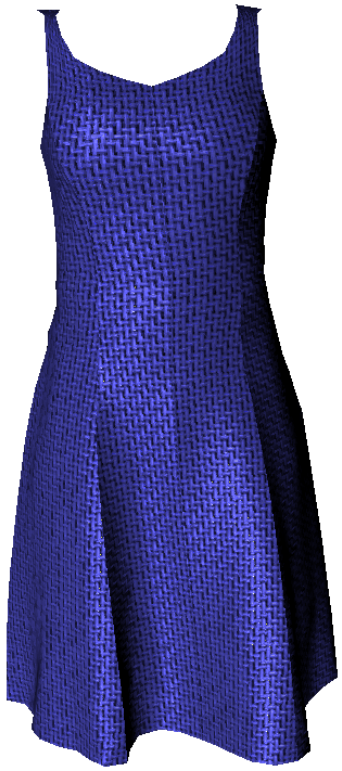


Figure 32: *The BRDF to render the weave of this dress consists of one lobe. Although it is possible to render BRDFs with more than one lobe, more passes are required to do so.*

6.2. Rendering

Now let's assume we have the geometry of a garment, given e.g. as a triangle mesh with normals, tangents and texture coordinates, and wish to render it using the appearance of a stitch or weave, captured in the model explained above. On modern graphics cards the rendering can be done in hardware, without reading back the frame buffer. However, the model is too complex to be evaluated in a single pass. We have to split the computation into several passes and combine the results at the end. We store intermediate results as images of the garment, in which each pixel on the garment color-codes the result of this pass. At the end we render a viewport filling quad and use multi-texturing with the intermediate results to obtain the final image.

First, let's take a look at how to evaluate the color table $T(\vec{v})$. The values stored in the color table resemble a stack of textures, with each texture corresponding to a different viewing direction $\vec{v}$. We can think of the color table as a 3D texture, with the viewing direction varying in the third dimension. The texture slices, however, are computed for the setting in which the underlying surface is flat,

with the normal point straight up. Obviously, for a general garment, this is not true. This again means we have to compute the viewing direction relative to the garment normals, or, in other words, map the global viewing direction into texture space, which is defined by the garment's per-vertex tangents, bi-normals and normals.

From the mapped viewing direction we then need to decide which slice in the texture stack to use. The texture slices in the lookup table are computed for a fixed, known, set of directions. We set the the vertex's texture coordinate for the third dimension (r-coordinate) to point to the slice corresponding to the direction nearest our transformed viewing direction. ¶

What happens though, if the normals for the three vertices of a garment triangle diverge strongly, and different slices are selected across the triangle? Clearly, this would lead to incorrect interpolations across the third texture dimension. We take care of this case using multi-texturing in the following way. For each vertex we specify three different sets of texture coordinates. The first set maps all vertices of a triangle into the texture slice needed for the first vertex, the second set of texture coordinates maps the triangle into the slice corresponding to the second vertex and so forth. When rendering, we blend the three textures in such a way, that a texture is faded to zero as we approach the two vertices not corresponding to the current slice. This way, in a triangle with strongly diverging normals, the resulting slices are blended naturally over the triangle. The result of this pass is an image of the garment, in which each garment pixel holds the result for $T(\vec{v})$.

Next, we have to evaluate the lobes. The main problem here, is that current graphics hardware is not capable of computing x^{N_i} on a per-pixel basis. The solution here is to precompute a texture holding the results for x^{N_i} and use dependent texturing, as explained in [55].

The remaining steps simply combine the results for $T(\vec{v})$, the lobes evaluation with ρ and l'_z, which is easy to do using register combiners.

6.3. Data Acquisition and Fitting

Now we know how to render a garment using an already computed BRDF model, let's take a look at how we can fit parameters for a model representing our own stitch. We will need a model of the stitches micro-geometry to do so.

Then we are ready to obtain the required data. As mentioned above, the spatial variations of the fabric pattern are stored as a 2D array of BRDF models. Apart from radiance samples $r(\vec{l},\vec{v})$ for all combinations of viewing and light directions, we also need an average normal, an average tangent, and an alpha value for each viewing direction for each of these entries.

We use an extension of Heidrich et al.'s algorithm [43] to triangle meshes [15], which allows us to compute the direct and indirect illumination of a triangle mesh or parametric surface (representing the stitch geometry) for a given viewing and light direction per vertex in hardware (for details see [15]). In order to account for masking and

¶ In principle, on modern graphics cards, the mapping and setting of the r-coordinate can be computed in a vertex program, which would result in better rendering rates. However, computing the approximation of the arccos and arctan function, as well as an integer cast in a vertex program is quite tricky to code.

Figure 33: *Woolen sweater rendered using our approach (knit and perl loops).*

parts of the repeated geometry being visible through holes, we paste together multiple copies of the geometry.

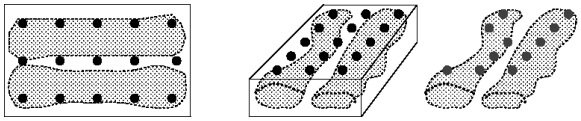


Figure 34: *Computing the sampling locations for the radiance values. Left: top view, middle: projection, right: resulting sampling locations, discarding samples at holes.*

Now we need to collect the radiance data for each sampling point. We obtain the 2D sampling locations by first defining a set of evenly spaced sampling points on the top face of the model's bounding box, as can be seen on the left in Figure 34. Then we project these points according to the current viewing direction (see Figure 34 in the middle) and collect the radiance samples from the surface visible through these 2D projections (see Figure 34 right), similarly to obtaining a light field.

As the stitch geometry can have holes, there might be no surface visible at a sampling point for a certain viewing direction. We store this information as a boolean transparency in the alpha channel for that sample. Multiple levels of transparency values can be obtained by super-sampling, i.e. considering the neighboring pixels.

In order to compute the normals, we display the scene once for each viewing direction with the normals coded as color values. An average normal is computed by adding the normals separately for each sampling point and averaging them at the end.

Once we have acquired all the necessary data, we use it to find an optimal set of parameters for the Lafortune model for each entry in the array of BRDFs. This fitting procedure can be divided into two major steps which are applied alternately. At first, the parameters of the lobes are fit. Then, in the second step, the entries of the lookup table are updated. Now the lobes are fit again and so on.

Given a set of all radiance samples and the corresponding viewing and light directions acquired for one sampling point, the fitting of the parameters of the Lafortune model f_l requires a non-linear optimization method. As proposed in [61], we applied the Levenberg-Marquardt algorithm [75] for this task.

After fitting the lobe parameters, we need to adapt the sampling point's lookup table $T(\vec{v})$. Each entry of the table is fit separately. This time only those radiance samples of the sampling point that correspond to the viewing direction of the current entry are considered. The optimal color for one entry minimizes the following set of equations:

$$\left(r(\vec{l}_1,\vec{v}),\ldots,r(\vec{l}_R,\vec{v})\right)^T = T(\vec{v})\left(f_l(\vec{l}_1,\vec{v}),\ldots,f_l(\vec{l}_R,\vec{v})\right)^T \quad (87)$$

where $r(\vec{l}_1,\vec{v}),\ldots,r(\vec{l}_R,\vec{v})$ are the radiance samples of the sampling point with the common viewing direction $\vec{v}$ and the distinct light directions $\vec{l}_1,\ldots,\vec{l}_R$. The currently estimated lobes are evaluated for every light direction yielding $f_l(\vec{l}_i,\vec{v})$. Treating the color channels separately, Equation 87 can be rewritten by replacing the column vector on its left side by $\vec{r}(\vec{v})$, the vector on its right side by $\vec{f}(\vec{v})$, yielding $\vec{r}(\vec{v}) = T(\vec{v})\cdot\vec{f}(\vec{v})$. The least squares solution to this equation is given by

$$T(\vec{v}) = \frac{\langle\vec{f}(\vec{v})|\vec{r}(\vec{v})\rangle}{\langle\vec{f}(\vec{v})|\vec{f}(\vec{v})\rangle} \quad (88)$$

where $\langle\cdot|\cdot\rangle$ denotes the dot product. This is done separately for every color channel and easily extends to additional spectral components.

To further improve the result we alternately repeat the steps of fitting the lobes and fitting the lookup table. The iteration stops as soon as the average difference of the previous lookup table's entries to the new lookup table's entries is below a certain threshold.

In addition to the color, each entry in the lookup table also contains an alpha value indicating the opacity of the sample point. This value is fixed for every viewing direction and is not affected by the fitting process. Instead it is determined through ray-casting during the data acquisition phase.

6.3.1. Mip-Map Fitting

The same fitting we have done for every single sample point can also be performed for groups of sample points. Let a sample point be a texel in a texture. Collecting all radiance samples for four neighboring sample points, averaging the normals, fitting the lobes and the entries of the lookup table then yields the BRDF corresponding to a texel on the next higher mip-map level.

By grouping even more sample points, further mip-map levels can be generated. The overall effort per level stays the same since the same number of radiance samples are involved at each level.

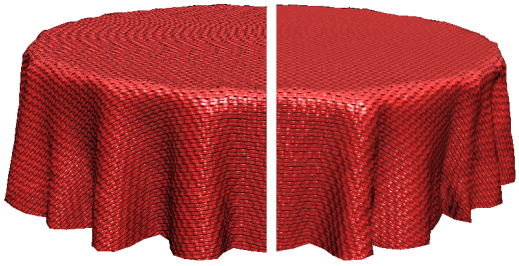


Figure 35: *A roughly woven tablecloth. Left: Aliasing artifacts are clearly visible if no mip-mapping is used. Right: using several mip-mapping layers.*

References

1. G. Baciu, W. Wong, and H. Sun. Hardware-Assisted Virtual Collisions. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST, Taipei, Taiwan*, pages 145–151, 1998. 17

2. 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. 4, 14, 16

3. David Baraff and Andrew Witkin. Large Steps in Cloth Simulation. *Computer Graphics*, 32(Annual Conference Series):43–54, 1998. 20

4. R. Bigliani and J. W. Eischen. Collision Detection in Cloth Modeling. In *Cloth and Clothing in Computer Graphics*. ACM SIGGRAPH, 1999. 17

5. J. Blinn. Simulation of Wrinkled Surfaces. In *Proceedings SIGGRAPH*, pages 286–292, August 1978. 28

6. J. Blinn. Models of Light Reflection For Computer Synthesized Pictures. In *Proceedings SIGGRAPH*, pages 192–198, July 1977. 28

7. Dieter Braess. *Finite Elemente.* Springer, 1997. 4

8. S. Cameron. Enhancing GJK: Computing Minimum Penetration Distances Between Convex Polyhedra. In *Proceedings of the Int. Conf. On Robotics and Automation*, pages 3112–3117, 1997. 20

9. Philippe G. Ciarlet. *Mathematical Elasticity. Vol. I.* North-Holland Publishing Co., Amsterdam, 1992. 4

10. J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments. In *Symposium on Interactive 3D Graphics*, pages 189–196, 218, 1995. 17

11. M. F. Cohen and D. P. Greenberg. The hemicube: A radiosity solution for complex environments. *Symposium on Computational Geometry*, 19(3):31–40, 22–26 July 1985. 25

12. Martin Courshesnes, Pascal Volino, and Nadia Magnenat Thalmann. Versatile and Efficient Techniques for Simulating Cloth and Other Deformable Objects. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference

Series, pages 137–144. ACM SIGGRAPH, Addison Wesley, 1995. 20

13. Kristian J. Dana and Shree K. Nazar. 3D Textured Surface Modeling. *WIAGMOR Workshop CVPR '99*, 1999. 22

14. Kristian J. Dana, Bram van Ginneken, Shree K. Nazar, and Jan J. Koenderink. Reflectance and Texture of Real-World Surfaces. *ACM Transactions on Graphics*, 1999. 22, 23

15. K. Daubert, W. Heidrich, J. Kautz, J.-M. Dischler, and Hans-Peter Seidel. Efficient Light Transport Using Precomputed Visibility. Technical Report MPI-I-2001-4-003, Max-Planck-Institut für Informatik, 2001. 29, 30

16. Katja Daubert, Hendrik P. A. Lensch, Wolfgang Heidrich, and Hans-Peter Seidel. Efficient cloth modeling and rendering. pages 63–70. 24

17. Katja Daubert, Hendrik P. A. Lensch, Wolfgang Heidrich, and Hans-Peter Seidel. Efficient Cloth Modeling and Rendering. In *Proc. of Eurographics Workshop on Rendering*, 2001. 29

18. Katja Daubert and Hans-Peter Seidel. Hardware-based volumetric knit-wear. *Eurographics '02*, 2002. 24

19. P. Debevec, T. Hawkins, C. Tchou, H.-P. Duiker, W. Sarokin, and M. Sagar. Acquiring the reflectance field of a human face. *Proceedings of ACM SIGGRAPH 2000*, 2000. 21

20. Paul Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8. 25

21. 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. 17

22. Mathieu Desbrun, Peter Schröder, and Alan Barr. Interactive Animation of Structured Deformable Objects. In *Graphics Interface*, pages 1–8, June 1999. 2, 16

23. Manfredo P. DoCarmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1976. 4

24. 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. 6

25. 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. 16

26. 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. 5

27. Olaf Etzmuß and Joachim Groß. Deriving a particle system from continuum mechanics for the animation of deformable objects. *to appear in IEEE Transactions on Visualization and Computer Graphics*, 2002. 6

28. Olaf Etzmuß, Michael Keckeisen, Stefan Kimmerle, Johannes Mezger, Michael Hauth, and Markus Wacker. A Cloth Modelling System for Animated Characters. In *Proceedings Graphiktag*, 2001. 21

29. J. J. Hsia I. W. Ginsberg F. E. Nicodemus, J. C. Richmond and T. Limperis. Reflectance nomenclature and directional reflectance and emissivity. *Applied Optics*, pages 1474–1475, 1970. 22

30. B. Ganster, R. Klein, M. Sattler, and R. Sarlette. Real-time shading of folded surfaces. *CGI 2002 Proc.*, 2001. 25, 26

31. E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space. *IEEE Journal of Robotics and Automation*, 4(2), 1988. 20

32. Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *Computer Graphics*, 30(Annual Conference Series):43–54, 1996. 21

33. S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996. 17, 18

34. Ned Greene. Environment mapping and other applications of world projection. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986. 25

35. Eduard Gröller, René T. Rau, and Wolfgang Straßer. Modeling textiles as three dimensional textures. In Xavier Pueyo and Peter Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 205–214, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4. 24

36. 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. 7

37. Leonidas J. Guibas, David Hsu, and Li Zhang. H-walk: hierarchical distance computation for moving convex bodies. in proceedings of the fifteenth annual symposium on computational geometry. pages 265–273, Miami, FL, USA, January 1999. Association for Computing Machinery. 17

38. E. Hairer and G. Wanner. Solving Ordinary Differential Equations I & II. Springer-Verlag, Berlin, 1996. 8

39. E. Hairer and G. Wanner. Solving Ordinary Differential Equations II. Springer-Verlag, Berlin, 1996. 11, 12

40. Ernst Hairer and Gerhard Wanner. Stiff differential equations solved by Radau methods. *J. Comput. Appl. Math.*, 111(1-2):93–111, 1999. 15

41. Michael Hauth and Olaf Etzmuß. A High Performance Solver for the Animation of Deformable Objects using Advanced Numerical Methods. In *Proc. Eurographics 2001*, 2001. 15, 16, 21

42. Taosong He. Fast Collision Detection Using QuOSPO Trees. *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 55–62, 1999. 18

43. W. Heidrich, K. Daubert, J. Kautz, and H.-P. Seidel. Illuminating Micro Geometry Based on Precomputed Visibility. In *Proceedings SIGGRAPH*, July 2000. 30

44. Wolfgang Heidrich. View-independent environment maps. In *Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '98*, 1998. 25

45. P. Hendrik and J. Kautz. Image-based reconstruction of spatially varying materials. *Proceedings of Eurographics Rendering Workshop t'01*, 2001. 22

46. http://www.cs.columbia.edu/CAVE/curet/. 22

47. http://www.ijg.org/. Independent jpeg group. 24

48. http://www.povray.org. 24

49. Philip M. Hubbard. Approximating Polyhedra with Spheres for Time-Critical Collision Detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996. 18

50. Lukasy Janski and Volker Ulbricht. Numerical simulation of mechanical behaviour of textile surfaces. *Zeitschrift für Angewandte Mathematik und Mechanik*, 80(S2):S525–S526, 2000. 5

51. Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Natick, MA, 2001. 24

52. J. T. Kajiya. The rendering equation. volume 20, pages 143–150, August 1986. 24

53. 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. 2

54. J. Kautz and M. McCool. Interactive Rendering with Arbitrary BRDFs using Separable Approximations. In *10th Eurographics Rendering Workshop 1999*, pages 281–292, June 1999. 27

55. J. Kautz and H.-P. Seidel. Towards Interactive Bump Mapping with Anisotropic Shift-Variant BRDFs. In *Hardware Workshop 2000*, pages 51–58, August 2000. 28, 29, 30

56. J. Kautz, C. Wynn, J. Blow, C. Blasband, A. Ahmad, , and M. McCool. Achieving Real-Time Realistic Reflectance. *Game Developer Magazine*, January 2001. 27

57. S. Kawabata. *The Standardization and Analysis of Hand Evaluation*. The Textile Machinery Society of Japan, Osaka, 1980. 3

58. M. Kilgard. *A Practical and Robust Bump-mapping Technique for Today's GPUs*. NVIDIA Corporation, April 2000. Available from http://www.nvidia.com. 28

59. E. Klingbeil. *Tensorrechnung für Ingenieure*. BI Wissenschaftsverlag, 1989. 4

60. James T. Klosowski, Martin Held, Joseph S. B. Mitchell, Henry Sowizral, and Karel Zikan. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998. 17, 18

61. E. Lafortune, S. Foo, K. Torrance, and D. Greenberg. Non-Linear Approximation of Reflectance Functions. In *SIGGRAPH '97 Proceedings*, pages 117–126, August 1997. 29, 31

62. Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, and Donald P. Greenberg. Non-linear approximation of reflectance functions. *Proc. SIGGRAPH 97*, pages 117–126, 1997. 22

63. L. D. Landau and E. M. Lifschitz. *Elastizitätstheorie*. Akademischer Verlag, 1989. 4

64. Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Department of Computer Science, University of North Carolina, Chapel Hill, 1999. 17

65. Gregory J. Ward Larson. Measuring and modeling anisotropic reflection. *Proc. SIGGRAPH 92*, pages 265–272, 1992. 22

66. Marc Levoy and Pat Hanrahan. Light field rendering. *Computer Graphics*, 30(Annual Conference Series):31–42, 1996. 21

67. M. Lin and S. Gottschalk. Collision Detection between Geometric Models: A Survey. In *Proceedings of IMA Conference on Mathematics of Surfaces*, 1998. 17

68. M. C. Lin and J. F. Canny. A Fast Algorithm for Incremental Distance Calculation. In *IEEE International Conference on Robotics and Automation*, pages 1008–1014, 1991. 20

69. M. C. Lin and D. Manocha. Fast Interference Detection Between Geometric Models. *The Visual Computer*, 11(10):542–551, 1995. 20

70. Xinguo Liu, Yizhou Zu, and Heung-Yeung Shum. Synthesizing Bidirectional Texture Functions for Real-World Surfaces. *SIGGRAPH 2001*, 2001. 22, 24

71. Tom Malzbender, Dan Gelb, and Hans Wolters. Polynomial texture maps. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 519–528. ACM Press / ACM SIGGRAPH, 2001. 22

72. Johannes Mezger, Stefan Kimmerle, and Olaf Etzmuß. Progress in Collision Detection and Response Techniques for Cloth Animation. (Submitted), 2002. 18, 20

73. B. Mirtich. VClip: Fast and Robust Polyhedral Collision Detection. *ACM Transactions on Graphics*, 17(3):177–208, 1998. 17, 20

74. M. Moore and J. Wilhelms. Collision detection and response for computer animation. volume 22 of *Annual Conference Series*, pages 289–298. ACM SIGGRAPH, July 1988. 18

75. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, 1992. ISBN 0-521-43108-5. 31

76. Xavier Provot. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior. In *Graphics Interface '95*, pages 147–154, 1995. 2, 16

77. Xavier Provot. Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garments. In *Graphics Interface '97*, pages 177–189, 1997. 20, 21

78. W. C. Rheinboldt. *Methods for Solving Systems of Nonlinear Equations*, volume 70 of *CBMS-NSF regional conference series in applied mathematics*. SIAM, second edition, 1998. 14, 15

79. Luc Robert. Camera calibration without feature extraction. *Computer Vision and Image Understanding: CVIU*, 63(2):314–325, 1996. 23

80. Francois Sillion and Claude Puech. *Radiosity and Global Illumination.* Morgan Kaufmann, San Francisco, CA, 1994. 25

81. Hans Stephani and Gerhard Kluge. *Theoretische Mechanik.* Spektrum Akademischer Verlag, 1995. 4

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

83. G. van den Bergen. Efficient Collision Detection of Complex Deformable Models using AABB Trees. *Journal of Graphics Tools*, 2(4):1–14, 1999. 18

84. T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast Cloth Animation on Walking Avatars. In *Computer Graphics Forum (Proc. of Eurographics)*, 2001. 17

85. 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. 4

86. P. Volino and N. Magnenat-Thalmann. Implementing fast Cloth Simulation with Collision Response. In *Computer Graphics International Proceedings*, 2000. 14

87. P. Volino and N. Magnenat-Thalmann. Implementing fast Cloth Simulation with Collision Response. In *Computer Graphics International*, June 2000. 20

88. P. Volino and N. Magnenat-Thalmann. *Virtual Clothing.* Springer, 2000. 20

89. M. H. Chu W-C. Chen, J-Y. Bouguet and R. Grzeszczuk. Light field mapping: Efficient representation and hardware rendering of surface light fields. *Proceedings of ACM SIGGRAPH 2002*, 2002. 21

90. Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface light fields for 3D photography. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 287–296. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. 21

91. C. Wynn. BRDF-Based Lighting. Technical report, NVIDIA Corporation, 2000. 27

92. G. Zachmann. Rapid Collision Detection by Dynamically Aligned DOP-Trees. *Proc. of IEEE, VRAIS'98 Atlanta*, 1998. 18

93. Dongliang Zhang and Matthew M.F. Yuen. Collision Detection for Clothed Human Animation. *Proceedings of the Pacific Graphics*, 2000. 17