Physikalisch basierte Gewebesimulation in Echtzeit

Diplomarbeit

Universität Rostock, Fachbereich Informatik
Lehrstuhl für Computergraphik

vorgelegt von
Cords, Hilko
geboren am 04.04.1978 in Oldenburg

Gutachter: Prof. Dr.-Ing. D. Jackèl, Prof. Dr.-Ing. H. Schumann
Betreuer: Prof. Dr.-Ing. D. Jackèl

Abgabedatum: 06.04.2004

Kurzfassung Die realistische Simulation von Kleidung und Geweben ist seit Mitte der 80er Jahre Forschungsgebiet in der Computergraphik. Zahlreiche Anwendungen in den Bereichen des Films, der Virtual Reality und der Computerspiele erfordern eine realistische Kleidungsdarstellung und -animation. Diese Arbeit analysiert vorhandene Methoden und Verfahren der Kleidungssimulation mit Bezug auf Echtzeitumgebungen. Darauf aufbauend wird ein Konzept zur physikalisch basierten Echtzeit-Kleidungssimulation mit Hilfe von Feder-Masse Systemen entwickelt. Zum einen werden explizite Integrationsverfahren (Euler, Midway, Runge-Kutta) und zum anderen wird die implizite Euler-Integration untersucht. Die wirkenden Kräfte und zugehörige Differentiationen werden für die implizite Integration mit einer neuen, im Rahmen dieser Arbeit entwickelten physikalisch basierten Methode berechnet.

Weitere, für eine realistische Kleidungssimulation wichtige Konzepte zur Echtzeitdarstellung von Kollisionen, Wind und Schnitten werden ebenso präsentiert wie Methoden zur Verbesserung der Darstellungsqualität (z.B. die für ein Shading notwendige Normalenberechnung, Felldarstellung). Die Realisierbarkeit, die Grenzen und Möglichkeiten der vorgestellten Konzepte werden anhand einer prototypischen Implementierung betrachtet. Diese ist zugleich Grundlage für eine Bewertung der entwickelten Verfahren.

Abstract Realistic cloth simulation has been a topic of research in the computer graphics community since the mid 1980s. Numerous applications within the virtual reality, the film and computer game industry require a realistic graphical representation and animation of clothes. This thesis analyzes existing methods and principles of cloth simulation with reference to real-time environments. Based on the analysis a concept of physically-based real-time cloth simulation with mass-spring systems is presented. Explicit integration methods (Euler, Midway, Runge-Kutta) as well as the implicit backward Euler method are examined. The forces and differentiations for the implicit Euler method are calculated with a new, physically-based method.

In addition, this thesis discusses the principles of collisions, wind, cuts and the image quality (e.g. calculation of normals, fur) related to real-time cloth simulation. The feasibility and the possibilities of the developed concepts has been validated by a prototype implementation. Based on the implementation an evaluation of the developed concepts is presented.

Classification (ACM CCS 1998): I.3.7, I.3.5, I.6.8.

Keywords: Cloth Simulation, Cloth Animation, Cloth Modeling, Real-Time Cloth, Fur, Implicit Integration, Physically-based Modeling, Physically-based Simulation, Cloth-Cloth Collision, Selfcollision

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Simulation, Animation und realistische Darstellung von Geweben und Kleidung[1] ist seit Mitte der 80er Jahre ein Forschungsthema in der Computergraphik. Wie in anderen Gebieten der Informatik hat die Steigerung der Prozessorgeschwindigkeiten auch in der Kleidungssimulation die Möglichkeiten erheblich erweitert. Anwendung findet sie vor allem im Bereich des (Trick-) Films, der Virtual-Reality und der Computerspiele. Aber auch für Textilindustrie, Architekten und Raumausstatter ist eine der Realität entsprechende Simulation von Kleidung und Geweben von Interesse. Zahlreiche weitere Anwendungsmöglichkeiten sind z.B. in den Medien und der darstellenden Kunst gegeben. Diese Arbeit diskutiert die Möglichkeiten von Kleidungssimulationen in Echtzeitumgebungen, die gerade für Virtual-Reality-Anwendungen und Computerspiele von großer Bedeutung sind.

Die explizite Integration eines Feder-Masse Systems in Verbindung mit einer Kleidungssimulation besitzt die Nachteile, daß nur kleine Zeitschritte und kleine Federkonstanten simuliert werden können: Die Simulation beansprucht viel Rechenzeit, und die simulierte Kleidung besitzt eine unrealistische, hohe Elastizität. Diese Gründe motivieren die Verwendung der impliziten Integration, die erstmals 1998 in Verbindung mit einer Kleidungssimulation vorgestellt wurde [BW98].

Ein Konzept der expliziten und impliziten Integration von physikalisch basierten Kräften mit Bezug auf die Kleidungssimulation wird in

[1] In dieser Arbeit werden die Begriffe Gewebe und Kleidung verwendet. Die üblich Bezeichnung dieser Thematik lautet „cloth simulation“, auch wenn es sich bei der simulierten „cloth“ (engl. Tuch, Stoff) z.B. um ein Kleid handelt. Es sei bedacht, dass Kleidung nicht notwendigerweise aus Gewebe bestehen muss.

dieser Arbeit entwickelt und mit Hinblick auf eine Implementierung und die dabei zu beachtenden Besonderheiten betrachtet. Kern des Konzeptes ist eine physikalisch basierte, neue Art der Berechnung der wirkenden Kräfte und ihrer Differentiationen für die implizite Integration. Weitere, für eine realistische Gewebesimulation entscheidende Konzepte zur Kollisionserkennung, -behandlung und zur Simulation von Wind und Schnitten werden mit Hinblick auf eine Echtzeitumgebung präsentiert. Des Weiteren werden Konzepte zur Verbesserung der Echtzeit-Darstellungsqualität des simulierten Gewebes vorgestellt: Die für ein Shading notwendigen Oberflächennormalen elastischer Körper (wie z.B. Gewebe) verändern sich unentwegt während der Simulation, so dass ein schnelles Verfahren zur Normalenberechnung erforderlich ist. Zudem werden Methoden zur Echtzeitdarstellung von Schlagschatten und Fell entwickelt. Alle Konzepte wurden im Rahmen dieser Arbeit in einer prototypischen Implementierung realisiert, um sie in der Praxis bewerten und vergleichen zu können. Aufgrund der Beschränkung der Echtzeit muss ein Kompromiss zwischen Geschwindigkeit und Darstellungsqualität eingegangen werden. Dieser Kompromiss entfällt in Nicht-Echtzeitumgebungen.

Der Aufbau des verbleibenden Textes ist wie folgt organisiert: Im nachfolgenden Kapitel werden zunächst die grundlegenden Begriffe, Modelle und physikalischen Zusammenhänge erläutert, auf die im weiteren Verlauf der Arbeit aufgebaut wird. Es schließt sich ein Abschnitt über die Grundlagen der Kleidungssimulation an. Kapitel 3 gibt einen Überblick über thematisch verwandte Arbeiten und betrachtet diese mit Hinblick auf Echtzeitumgebungen. Das Konzept der impliziten Integration physikalisch basierter Kräfte wird in Kapitel 4 entwickelt. Dort werden auch weitergehende Konzepte zur Kollisionserkennung und -behandlung, zum Shading des Gewebes und zur Darstellung von Wind, Rissen und Fell allgemein gültig vorgestellt. Die Betrachtung im Rahmen einer konkreten Implementierung schließt sich in Kapitel 5 an. Es folgt eine Bewertung der entstandenen Verfahren in der Praxis (Kapitel 6). Abschließend wird eine Zusammenfassung der Arbeit und ein Ausblick gegeben.

An dieser Stelle sei auch erwähnt, dass in dieser Arbeit bei einigen Begriffen die englische Bezeichnung gewählt wurde. In diesen Fällen erschien eine Übersetzung nicht sinnvoll, da die jeweiligen Begriffe entweder keine sinnvolle Entsprechung in der deutschen Sprache besitzen oder aber allgemein in der Computergraphik unter der entsprechenden Bezeichnung verwendet werden.

Kapitel 2

Grundlagen

Die für diese Arbeit grundlegenden Begriffe und Modelle werden in diesem Kapitel vorgestellt. Zunächst werden einige formale Definitionen eingeführt, um die Basis zum Verständnis der folgenden mathematischen und physikalischen Modelle zu schaffen. Anschließend werden für diese Arbeit entscheidende physikalische Grundlagen kurz erläutert. Darauf folgt eine Einführung in für diese Arbeit wichtige numerische Integrationsverfahren und das konjugierte Gradientenverfahren, welches zur numerischen Lösung von linearen Gleichungssystemen herangezogen werden kann und als solches eine weitere Basis für diese Arbeit darstellt. Schließlich werden die Grundlagen der Gewebe- und Kleidungssimulation vorgestellt.

2.1 Mathematisch-physikalische Grundlagen

2.1.1 Definitionen und Begriffe

2.1.1.1 Mathematische Notation

In dieser Arbeit werden mathematische Formeln mit folgender Semantik gesetzt[1]:

$$\text{Skalare}: \quad a, b, c, \ldots$$

$$\text{Spaltenvektoren}: \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_x \\ x_y \\ x_z \end{pmatrix}, \mathbf{y}, \mathbf{z}, \ldots$$

[1] Damit wird eine weit verbreitete Art der mathematischen Notation verwendet, wie sie in zahlreichen physikalischen und mathematischen Veröffentlichungen zu finden und auch in der Computergraphik üblich ist.

Zeilenvektoren :	$\mathbf{x}^T = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}, \mathbf{y}^T, \mathbf{z}^T, \ldots$
Matrizen :	$\mathcal{M}, \mathcal{A}, \mathcal{B}, \ldots$
Einheitsmatrix der Ordnung n :	$\mathcal{E}_{n \times n}$
Normierte Vektoren :	$\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}, \ldots$
Element eines Vektors :	$\mathbf{x}^a = \begin{pmatrix} x_1 \\ \vdots \\ x_a \\ \vdots \\ x_n \end{pmatrix}^a = x_a, \ldots$
Skalarprodukt :	$\mathbf{x} \cdot \mathbf{y}, \mathbf{a} \cdot \mathbf{b}, \ldots$
Ableitungen :	$\frac{\partial f}{\partial x} = f', \frac{\partial f}{\partial x \partial x} = f'', \ldots$
Zeitableitungen :	$\frac{\partial x}{\partial t} = \dot{x}, \frac{\partial x}{\partial t \partial t} = \ddot{x}, \ldots$

2.1.1.2 Mathematische Bezeichnungen

Die in dieser Arbeit verwendeten Variablen werden wie folgt bezeichnet:

Position :	$\mathbf{x}$
Abstandsvektor :	$\Delta\mathbf{x} = \mathbf{x_2} - \mathbf{x_1}$
Geschwindigkeit :	$\mathbf{v} = \frac{\partial \mathbf{x}}{\partial t} = \dot{\mathbf{x}}$
Beschleunigung :	$\mathbf{a} = \frac{\partial \mathbf{x}}{\partial t \partial t} = \frac{\partial \mathbf{v}}{\partial t} = \ddot{\mathbf{x}}$
Differenzgeschwindigkeit :	$\Delta\mathbf{v} = \mathbf{v_2} - \mathbf{v_1}$
Kraft :	$\mathbf{F}$
Zeitschritt :	h

2.1.1.3 Simulationsparameter

Die in dieser Arbeit verwendeten Variablen zur Definition von Simulationsparametern, die mit denen im Source-Code des im Rahmen dieser Arbeit entstandenen Programms `ClothSimulator` verwendeten übereinstimmen, werden wie folgt bezeichnet:

Simulationsschrittweite :	$h \quad [h] = s$
Anzahl der verwendeten Partikel :	n

$$\begin{aligned} \text{Federkonstante}: &\quad D \quad [D] = kg/s^2 \\ \text{Federdämpfung}: &\quad k_d \quad [k_d] = 1 \\ \text{Globale Dämpfung}: &\quad k_{global} \quad [k_{global}] = 1 \\ \text{Masse}: &\quad m \quad [m] = kg \end{aligned}$$

2.1.1.4 Vektorgradient

In den Berechnungen dieser Arbeit wird des Öfteren ein Vektorfeld in Richtung eines Vektors abgeleitet (Vektorgradient). Dieses mathematische Konstrukt sei an dieser Stelle in Anlehnung an [BSMM99], [Pri02] und [Mac00] für den dreidimensionalen Fall gegeben:

$$\mathcal{M} = \operatorname{grad} \mathbf{f}(\mathbf{x}) = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial \begin{pmatrix} f_x(x,y,z) \\ f_y(x,y,z) \\ f_z(x,y,z) \end{pmatrix}}{\partial \begin{pmatrix} x \\ y \\ z \end{pmatrix}} = \begin{pmatrix} \frac{\partial f_x}{\partial x} & \frac{\partial f_x}{\partial y} & \frac{\partial f_x}{\partial z} \\ \frac{\partial f_y}{\partial x} & \frac{\partial f_y}{\partial y} & \frac{\partial f_y}{\partial z} \\ \frac{\partial f_z}{\partial x} & \frac{\partial f_z}{\partial y} & \frac{\partial f_z}{\partial z} \end{pmatrix} \tag{2.1}$$

2.1.2 Gewichtskraft

Die Gewichtskraft eines Körpers in Bezug auf einen anderen Körper (z.B. die Erde) ergibt sich aus dem Newtonschen Gravitationsgesetz:

$$F_g = G \cdot \frac{m_1\, m_2}{r_{12}^2} \tag{2.2}$$

$$\begin{aligned} F_g &: \text{Gravitationskraft} \quad [F] = \frac{kg\, m}{s^2} \\ G &: \text{Gravitationskonstante} \quad [G] = N\, \frac{m^2}{kg^2} \\ m_1, m_2 &: \text{Massen der Körper} \quad [m] = kg \\ r_{12} &: \text{Schwerpunktsabstand der Körper} \quad [r] = m \end{aligned}$$

Die Gewichtskraft, die auf einen Körper der Masse m auf der Erde wirkt, wird folgendermaßen bestimmt:

$$F_g = G \cdot \frac{m\, m_{Erde}}{r_{12}^2} \tag{2.3}$$

mit

$r_{12} \approx$ Radius der Erde

Dies ergibt

$$F_g = g \cdot m \tag{2.4}$$

mit

$$g = 9,80665\frac{m}{s^2} \quad \text{für Meereshöhe in etwa } 45°\text{geographischer Breite.}$$

Die Fallbeschleunigung g ist eine gute Näherung für jede beliebige Position auf der Erdoberfläche, jedoch ist g genau genommen wegen der nichtsphärischen Form der Erde und der durch die Erdrotation hervorgerufenen Zentrifugalkraft vom Breitengrad abhängig. Zudem ist g auch von der Höhe des Messpunktes über der Erdoberfläche abhängig[2]. Für andere Bezugssysteme als die Erde ergeben sich entsprechend andere Fallbeschleunigungen.

Für die Anwendung im Rahmen dieser Arbeit ist die Genauigkeit von Gleichung 2.4 ausreichend. Da Gleichung 2.4 nur von der Masse des Probekörpers abhängt und diese für den nichtrelativistischen Fall als konstant angesehen werden kann, wirkt zu jedem Zeitpunkt t eine konstante Gewichtskraft auf den Körper.

Es sei auch erwähnt, dass unabhängig von der Masse eines Körpers jeder Körper nach dem 2. Newtonschen Axiom $F = m \cdot \ddot{x}$ die gleiche Fallbeschleunigung $\ddot{x} = \frac{F_g}{m} = \frac{m \cdot g}{m} = g$ erfährt.

2.1.3 Das ungedämpfte Federpendel

In diesem Abschnitt wird die Basisgleichung eines Federpendels mit verbundenem Massepunkt m behandelt (vgl. [Dem98] und [Stö99]). Dabei handelt es sich um einen reibungsfreien idealisierten Fall, da bei ausgedehnten Körpern zusätzlich Effekte wie Drehmoment und Luftwiderstand betrachtet werden müssten. Das gedämpfte Federpendel wird nach Vorstellung der Reibungskräfte in Abschnitt 2.1.4.1 behandelt.

Auf einen an einer Feder (Ruhelänge s_0) befestigten Massepunkt m wirkt

[2] Man bedenke den Erdradius von $r_{Erde} \approx 6378\,km$, der quadratisch in das Newtonsche Gravitationsgesetz einfließt, so dass auch hier kleine Höhenveränderungen (bis in den km-Bereich) je nach Anwendung vernachlässigbar sind.

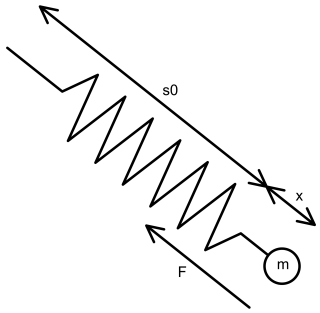

Abb. 2.1: Die wirkende Kraft eines Federpendels hängt nur von der Auslenkung aus der Ruhelage ab.

folgende aus dem Hook'schen Gesetz hervorgehende Kraft(vgl. Abb. 2.1):

$$F_F = -D \cdot x \tag{2.5}$$

F_F : Rückstellkraft $\quad [F] = \dfrac{kg\,m}{s^2}$

D : Federkonstante $\quad [D] = \dfrac{kg}{s^2}$

x : Auslenkung aus der Ruhelage $s_0 \quad [x] = m$

Die Kraft F_F wirkt in Richtung der Feder. D ist eine Konstante, die von dem Torsionsmodul des Federmaterials und der Dicke der Feder abhängt und maßgeblich die wirkende Kraft beeinflusst. Die Rückstellkraft ist linear von der Auslenkung aus der Ruhelage abhängig.

An dieser Stelle sei noch erwähnt, dass bei einem realen Federpendel nur innerhalb gewisser Grenzen der Auslenkung ein lineares Kraftgesetz gültig ist.

2.1.4 Reibungskraft

Die Reibungskraft ist eine der Bewegung eines Körpers entgegenwirkende Kraft. Sie tritt auf, wenn sich der Körper in Berührung mit einem anderen Körper oder durch eine Flüssigkeit oder ein Gas bewegt. Reibungskräfte wirken parallel zu der Berührungsfläche (vgl. [Stö99]).

Die auftretenden Reibungskräfte bei Berührung zweier Körper sind für diese Arbeit nicht von entscheidender Bedeutung: Es sei lediglich erwähnt, dass sie unterschieden werden in Haftreibung, Gleitreibung und Rollreibung. Wichtiger ist für diese Arbeit die Reibungskraft, die bei Bewegung eines Körpers durch Gase (z.B. Luft) oder Flüssigkeiten entsteht. Diese berechnet

sich nach dem Stokes'schen Reibungsgesetz:

$$F_r = -k \cdot \dot{x} = -k \cdot v \tag{2.6}$$

Dabei bezeichnet k die Reibungskonstante und v die Geschwindigkeit des Körpers[3]. Die der Bewegungsrichtung entgegengesetzte Reibungskraft ist linear abhängig von der Geschwindigkeit v.

2.1.4.1 Das gedämpfte Federpendel

Unter Berücksichtigung der im vorhergehenden Abschnitt beschriebenen Reibungskraft verändert sich auch die wirkende Kraft des in Abschnitt 2.1.3 vorgestellten ungedämpften Federpendels. Die im gleichen Abschnitt vorgestellte Gleichung 2.5 zur Berechnung der wirkenden Kraft eines Federpendels wird um die Reibungskraft (Dämpfung) erweitert:

$$F_{Feder} = -D \cdot x + F_r = -D \cdot x - k \cdot \dot{x} \tag{2.7}$$

$$m \cdot \ddot{x} = -D \cdot x - k \cdot \dot{x} \tag{2.8}$$

2.1.5 Numerische Integrationsverfahren

In diesem Abschnitt werden grundlegende numerische Integrationsverfahren vorgestellt, auf die im weiteren Verlauf dieser Arbeit aufgebaut wird. Grundlage für alle vorgestellten Verfahren ist das Anfangswertproblem

$$\begin{aligned} y' &= f(x, y) \\ y(x_0) &= y_0. \end{aligned} \tag{2.9}$$

Das Prinzip der vorgestellten Verfahren zur Lösung dieses Problems besteht darin, für die gesuchte Funktion $y(x)$ an ausgewählten Stützstellen x_i Näherungswerte y_i zu ermitteln. Dabei werden in der Regel äquidistante Stützstellen mit der definierten Schrittweite h verwendet (vgl. dazu [BSMM99]):

$$x_i = x_0 + i\,h \quad (i = 0, 1, 2, ...) \tag{2.10}$$

Die vorgestellten Verfahren unterscheiden sich in Bezug auf ihre Komplexität[4] und ihre Genauigkeit. Anhand dieser Parameter muss abgewogen wer-

[3] Das ursprüngliche Stokes'sche Reibungsgesetz bezieht sich auf einen kugelförmigen Körper. Durch eine Anpassung der Reibungskonstante kann jedoch eine ausreichend genaue und allgemein gültige Reibungskraft bestimmt werden.

[4] Die jeweilige Komplexität bestimmt die zur Berechnung benötigte Rechenzeit.

den, welches Verfahren für die jeweilig zu lösende Aufgabe am besten geeignet ist[5]. Es handelt sich bei allen Integrationsverfahren um iterative Verfahren.

Für alle Verfahren werden Beispiele gegeben, da die Umsetzung für ein einfaches physikalisches System mit der Position x, der Geschwindigkeit $\dot{x}$ und der Beschleunigung $\ddot{x}$ wegen der komplexeren Differentialgleichung nicht intuitiv durchgeführt werden kann. Zudem entspricht die Realisierung der Integrationsverfahren in der später folgenden Umsetzung in ein Computerprogramm größtenteils den gegebenen Beispielen.

2.1.5.1 Explizites Euler-Verfahren

Eines der am schnellsten berechenbaren Verfahren ist das Eulersche Polygonzugverfahren[6]. Durch Integration erhält man aus Gleichung 2.9 die Integraldarstellung:

$$y(x) = y_0 + \int_{x_0}^{x} f(x, y(x))\, dx \tag{2.11}$$

Diese ist Ausgangspunkt für die Approximation (vgl. [BSMM99]):

$$y(x_1) = y_0 + \int_{x_0}^{x_0+h} f(x, y(x))\, dx \approx y_0 + h\, f(x_0, y_0) = y_1 \tag{2.12}$$

Verallgemeinert zum expliziten Euler-Verfahren ergibt sich folgende Schreibweise:

$$y_{i+1} = y_i + h\, f(x_i, y_i) \quad (i = 0, 1, 2, ...;\ y(x_0) = y_0) \tag{2.13}$$

Diese Formulierung entspricht einer Taylorentwicklung 1.Ordnung, die auch gleichzeitig den Fehler angibt:

$$y(x_1) = y(x_0 + h) = y_0 + f(x_0, y_0)\, h + \frac{y''(\xi)}{2}\, h^2 \tag{2.14}$$

mit $x_0 < \xi < x_0 + h$. Die Fehlerabweichung liegt somit in der Größenordnung h^2. Die Genauigkeit kann durch Verringern der Schrittweite h erhöht werden.

[5] Im Rahmen dieser Arbeit wurden alle hier vorgestellten Verfahren implementiert und mit Bezug auf Performance und Möglichkeiten verglichen (Siehe Kapitel 6).

[6] In Veröffentlichungen wird dieses oft auch lediglich „Euler-Integration“ oder „explizites Euler-Verfahren“ genannt. Diese Begriffe werden auch in dieser Arbeit verwendet.

In der Praxis zeigt sich, dass bei Halbierung der Schrittweite h auch der Fehler der Näherung y_i etwa halbiert wird (vgl. [BSMM99]).

Das explizite Euler-Verfahren ist aufgrund seiner Einfachheit innerhalb einer Implementierung ein sehr schnelles und intuitiv umsetzbares Verfahren. Da es jedoch relativ ungenau ist, kommt es gerade bei komplexeren Funktionen leicht zu numerischen Instabilitäten, die nur durch Verringern der Schrittweite h verhindert werden können.

Beispiel Um ein eindimensionales physikalisches System, bestehend aus einem Massepunkt m, auf welchen die Kraft F wirkt, mit dem expliziten Euler-Verfahren bei einer Schrittweite von $t = h$ zu lösen, müssen folgende Schritte durchgeführt werden:

1. Beschreibung des Systems:

$$m\,\ddot{x} = f(t, x, \dot{x}) = F(x, v) \quad \text{Startposition} : t_0, x_0, v_0, F_0 \tag{2.15}$$

2. Berechnung von Δx und Δv:

$$\Delta x = h \cdot v_0 \tag{2.16}$$
$$\Delta v = h \cdot m^{-1} \cdot F_0 \tag{2.17}$$

3. Berechnung von x_h und v_h zum Zeitpunkt $t = t_0 + h$:

$$x_h = x_0 + \Delta x \tag{2.18}$$
$$v_h = v_0 + \Delta v \tag{2.19}$$

Daraus ergibt sich die wirkende Kraft zum Zeitpunkt $t = h$ mit

$$f_h = f(x_h, v_h). \tag{2.20}$$

2.1.5.2 Runge-Kutta-Verfahren

Ein weiteres, sehr häufig genutztes explizites Integrationsverfahren ist das Runge-Kutta-Verfahren. Im Folgenden werden zwei mögliche Runge-Kutta-Verfahren beschrieben: das Runge-Kutta-Verfahren 2. Ordnung und das Runge-Kutta-Verfahren 4. Ordnung (vgl. [Yan01]). Die Fehlerabweichung liegt bei Ersterem im Bereich von h^3 und bei Letzterem im Bereich von h^5, so dass bei geeigneter Schrittweite h eine hohe Genauigkeit, vor allem im Vergleich zum expliziten Euler-Verfahren, erreicht werden kann. Somit wird eine erhöhte numerische Stabilität gewährleistet.

Das Prinzip beider Verfahren basiert auf einem Vorhersage-Korrektur Schema. Die Vorhersage für y_{i+1} wird mit dem expliziten Euler-Verfahren bestimmt. Die zugehörige Korrektur wird implizit mit der zuvor bestimmten Vorhersage berechnet.

	x	y	$k = h \cdot f(x, y)$
1.Schritt	x_0	y_0	k_1
2.Schritt	$x_0 + h$	$y_0 + k_1$	k_2

Tabelle 2.1: Runge-Kutta-Verfahren 2. Ordnung

Runge-Kutta-Verfahren 2. Ordnung Der gesuchte Wert von y_1 berechnet sich nach dem in Tabelle 2.1 aufgetragenen, zweistufigen Verfahren. Es ergibt sich:

$$x_1 = x_0 + h \tag{2.21}$$

$$y_1 = y_0 + \frac{1}{2} \cdot (k_1 + k_2) \tag{2.22}$$

Beispiel Um ein eindimensionales physikalisches System bestehend aus einem Massepunkt m, auf welchen die Kraft F wirkt, mit dem Runge-Kutta-Verfahren 2. Ordnung bei einer Schrittweite von $t = h$ zu lösen, müssen folgende Schritte durchgeführt werden:

1. Beschreibung des Systems:

$$m\,\ddot{x} = f(t, x, \dot{x}) = F(x, v) \quad \text{Startposition} : t_0, x_0, v_0, F_0 \tag{2.23}$$

2. Berechnung von k_n $(n\,\epsilon\,\{1, 2\})$:

$$k_1 = \Delta v_{k_1} = h \cdot m^{-1} \cdot F_0 \tag{2.24}$$

$$v_{k_2} = v_0 + \Delta v_{k_1} \tag{2.25}$$

$$x_{k_2} = x_0 + h \cdot v_{k_2} \tag{2.26}$$

$$\text{Berechung der wirkenden Kraft} \quad F_{k_2} = F(x_{k_2}, v_{k_2})$$

$$k_2 = \Delta v_{k_2} = h \cdot m^{-1} \cdot F_{k_2} \tag{2.27}$$

3. Berechnung von x_h und v_h zum Zeitpunk $t = t_0 + h$:

$$v_h = v_0 + \frac{k_1 + k_2}{2} \tag{2.28}$$

$$x_h = x_0 + h \cdot v_0 \tag{2.29}$$

Runge-Kutta-Verfahren 4. Ordnung Dieses Verfahren entspricht dem Prinzip nach dem Runge-Kutta-Verfahren 2. Ordnung. Der gesuchte Wert

von y_1 berechnet sich nach dem in Tabelle 2.2 aufgetragenen, vierstufigen Verfahren. Es ergibt sich:

$$x_1 = x_0 + h \tag{2.30}$$

$$y_1 = y_0 + \frac{1}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4)) \tag{2.31}$$

Beispiel Um ein eindimensionales physikalisches System bestehend aus einem Massepunkt m, auf welchen die Kraft F wirkt, mit dem Runge-Kutta-Verfahren 4. Ordnung bei einer Schrittweite von $t = h$ zu lösen, müssen folgende Schritte durchgeführt werden:

1. Beschreibung des Systems:

$$m\,\ddot{x} = f(t, x, \dot{x}) = F(x, v) \quad \text{Startposition} : t_0, x_0, v_0, F_0 \tag{2.32}$$

2. Berechnung von k_n $(n\,\epsilon\,\{1, 2, 3, 4\})$:

$$k_1 = \Delta v_{k_1} = h \cdot m^{-1} \cdot F_0 \tag{2.33}$$

$$x_{k_1} = x_0 + h \cdot (v_0 + \Delta v_{k_1}) \tag{2.34}$$

$$v_{k_2} = v_0 + \frac{1}{2} \cdot \Delta v_{k_1} \tag{2.35}$$

$$x_{k_2} = x_0 + \frac{h}{2} \cdot v_{k_2} \tag{2.36}$$

$$\text{wirkende Kraft} \quad F_{k_2} = F(x_{k_2}, v_{k_2})$$

$$k_2 = \Delta v_{k_2} = h \cdot m^{-1} \cdot F_{k_2} \tag{2.37}$$

$$v_{k_3} = v_0 + \frac{1}{2}\Delta v_{k_2} \tag{2.38}$$

$$x_{k_3} = x_0 + \frac{h}{2} \cdot v_{k_3} \tag{2.39}$$

$$\text{wirkende Kraft} \quad F_{k_3} = F(x_{k_3}, v_{k_3})$$

$$k_3 = \Delta v_{k_3} = h \cdot m^{-1} \cdot F_{k_3} \tag{2.40}$$

$$v_{k_4} = v_0 + \Delta v_{k_3} \tag{2.41}$$

$$x_{k_4} = x_0 + h \cdot v_{k_4} \tag{2.42}$$

$$\text{wirkende Kraft} \quad F_{k_4} = F(x_{k_4}, v_{k_4})$$

$$k_4 = \Delta v_{k_4} = h \cdot m^{-1} \cdot F_{k_4} \tag{2.43}$$

Es ergibt sich:

$$v_h = v_0 + \frac{1}{6} \cdot (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4) \tag{2.44}$$

$$x_h = x_0 + h \cdot v_0 \tag{2.45}$$

	x	y	$k = h \cdot f(x, y)$
1.Schritt	x_0	y_0	k_1
2.Schritt	$x_0 + \frac{h}{2}$	$y_0 + \frac{k_1}{2}$	k_2
3.Schritt	$x_0 + \frac{h}{2}$	$y_0 + \frac{k_2}{2}$	k_3
4.Schritt	$x_0 + h$	$y_0 + k_3$	k_4

Tabelle 2.2: Runge-Kutta-Verfahren 4. Ordnung

2.1.5.3 Midway-Verfahren

Das explizite Midway-Verfahren ist ein Integrationsverfahren 2. Ordnung. Es ist wie folgt anzuwenden:

	x	y	k
1.Schritt	x_0	y_0	$k_1 = \frac{h}{2} \cdot f(x, y)$
2.Schritt	$x_0 + h$	$y_0 + k_1$	$k_2 = h \cdot f(x, y)$

Tabelle 2.3: Midway-Verfahren

Die Berechnung entspricht dabei der des Runge-Kutta-Verfahrens 2. Ordnung (vgl. Abschnitt 2.1.5.2), wobei k_1 und k_2 durch die vorstehend dargestellten Berechnungsvorschriften ersetzt werden. Die Berechnung eines physikalischen Systems erfolgt ebenfalls entsprechend.

2.1.5.4 Euler-Cromer-Methode

Eine kleine Veränderung der in den vorhergehenden Abschnitten beschriebenen expliziten Integrationsverfahren führt zu einer starken Verbesserung der Genauigkeit und damit bei komplexen Systemen zu einer besseren Stabilität (vgl. [LHSM99]): Die Berechnung von x_h erfolgt nicht auf die in den Beispielen der vorhergehenden Abschnitte dargestellte Art $x_h = x_0 + h \cdot v_0$, sondern mit

$$x_h = x_0 + h \cdot v_h. \tag{2.46}$$

Da v_h zur Berechnungszeit von x_h bereits bekannt ist, ist der Aufwand der Gleiche. Ein Vergleich des Euler- und des Euler-Cromer-Verfahrens ist in Abbildung 2.2 dargestellt. Die numerische Lösung des Euler-Cromer-Verfahrens ist wesentlich genauer als die des Euler-Verfahrens. Zudem wird die Energieerhaltung mit dem Euler-Cromer-Verfahrens (bis auf wenige Schwankungen) besser approximiert (vgl. [LHSM99]).

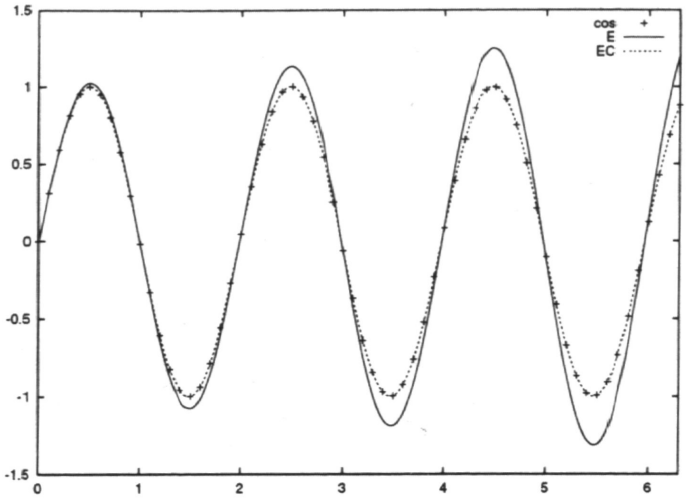

Abb. 2.2: Vergleich von Euler- und Euler-Cromer-Methode. Dargestellt sind die exakte und die beiden numerisch mit dem Euler- und mit dem Euler-Cromer-Verfahren berechneten Lösungen der Bewegungsgleichung eines harmonischen Oszillators $m\ddot{x} = -kx$ mit der Amplitude 1, Phasenverschiebung 0, m=1kg, k=$10\frac{kg}{s^2}$, h=0,01s. Analytische Lösung: $\sin(t \cdot x)$. Das Euler-Verfahren approximiert die gesuchte Funktion wesentlich schlechter als das Euler-Cromer-Verfahren. Quelle: [LHSM99]

2.1.5.5 Implizites Euler-Verfahren

Eine Veränderung des expliziten Euler-Verfahrens (vgl. Gleichung 2.13) führt zum impliziten Euler-Verfahren:

$$y_{i+1} = y_i + h\, f(x_{i+1}, y_{i+1}) = y_i + h\, f(x_i + \Delta x, y_i + \Delta y) \tag{2.47}$$

Der Unterschied besteht darin, dass die Funktion f nun bei $x = x_{i+1}$ statt $x = x_i$ evaluiert wird (vgl. Abb. 2.3). Aus diesem Grund wird zur Bestimmung von y_i eine nichtlineare Integrationsmethode benötigt. Damit wächst der Aufwand zur Lösung dieser Gleichung erheblich. Gleichzeitig jedoch werden wesentlich exaktere Lösungen berechnet und die Stabilität steigt erheblich.

Es wird der Ansatz gewählt, Gleichung 2.47 nicht exakt zu lösen, sondern eine Taylorentwicklung 1. Ordnung für f durchzuführen und die dadurch entstehende Gleichung zu lösen:

$$y_{i+1} = y_i + h\, f(x_{i+1}, y_{i+1}) \approx y_i + h \cdot \left(f_0 + \frac{\partial f}{\partial x} \cdot \Delta x + \frac{\partial f}{\partial y} \cdot \Delta y \right) \tag{2.48}$$

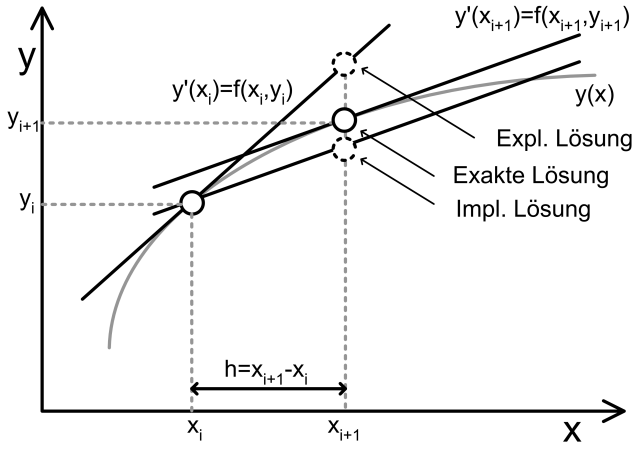

Abb. 2.3: Vergleich von explizitem und implizitem Verfahren zur Berechnung der numerischen Lösung.

In diesem Fall gilt unter Berücksichtigung von Gleichung 2.9 und Gleichung 2.10: $\Delta x = h$. Umformen ergibt die nun zu lösende Gleichung:

$$y_{i+1} - y_i = h \cdot \left(f_0 + \frac{\partial f}{\partial x} \cdot h + \frac{\partial f}{\partial y} \cdot \Delta y \right) \tag{2.49}$$

$$\Delta y = h \cdot \left(f_0 + \frac{\partial f}{\partial x} \cdot h + \frac{\partial f}{\partial y} \cdot \Delta y \right) \tag{2.50}$$

$$\left(1 - h \cdot \frac{\partial f}{\partial y} \right) \cdot \Delta y = h \cdot \left(f_0 + \frac{\partial f}{\partial x} \cdot h \right) \tag{2.51}$$

Δy kann direkt berechnet werden, wenn $\frac{\partial f}{\partial y}$ und $\frac{\partial f}{\partial x}$ an der aktuellen Position bekannt sind.

Beispiel zur impliziten Integration Um ein dreidimensionales physikalisches System bestehend aus einem Massepunkt m, auf welchen die Kraft $\mathbf{F_0}$ wirkt, mit dem impliziten Euler-Verfahren bei einer Schrittweite von $t = h$ zu lösen, müssen folgende Schritte durchgeführt werden:

1. Beschreibung des Systems:

$$m\,\ddot{\mathbf{x}} = f(t, \mathbf{x}, \dot{\mathbf{x}}) = \mathbf{F}(\mathbf{x}, \mathbf{v}) \quad \text{Startposition} : t_0, \mathbf{x_0}, \mathbf{v_0}, \mathbf{F_0} \tag{2.52}$$

2. Implizite Euler-Rückwärtsmethode:

$$\Delta \mathbf{x} = h \cdot (\mathbf{v_0} + \Delta \mathbf{v}) \tag{2.53}$$

$$\Delta \mathbf{v} = h \cdot m^{-1} \cdot \mathbf{F}(\mathbf{x_0} + \Delta \mathbf{x}, \mathbf{v_0} + \Delta \mathbf{v}) \tag{2.54}$$

3. Taylorentwicklung von $\mathbf{F}$:

$$\mathbf{F}(\mathbf{x_0} + \Delta\mathbf{x}, \mathbf{v_0} + \Delta\mathbf{v}) = \mathbf{F_0} + \frac{\partial \mathbf{F}(\mathbf{x_0}, \mathbf{y_0})}{\partial \mathbf{x}} \cdot \Delta\mathbf{x} + \frac{\partial \mathbf{F}(\mathbf{x_0}, \mathbf{y_0})}{\partial \mathbf{v}} \cdot \Delta\mathbf{v} \tag{2.55}$$

4. Ersetzen von $\mathbf{F}$ durch Taylorentwicklung:

$$\Delta\mathbf{x} = h \cdot (\mathbf{v_0} + \Delta\mathbf{v}) \tag{2.56}$$

$$\Delta\mathbf{v} = h \cdot m^{-1} \cdot \left(\mathbf{F_0} + \frac{\partial \mathbf{F}(\mathbf{x_0}, \mathbf{y_0})}{\partial \mathbf{x}} \cdot \Delta\mathbf{x} + \frac{\partial \mathbf{F}(\mathbf{x_0}, \mathbf{y_0})}{\partial \mathbf{v}} \cdot \Delta\mathbf{v}\right) \tag{2.57}$$

5. Ersetzen von $\Delta\mathbf{x}$:

$$\Delta\mathbf{v} = h \cdot m^{-1} \cdot \left(\mathbf{F_0} + \frac{\partial \mathbf{F}(\mathbf{x_0}, \mathbf{y_0})}{\partial \mathbf{x}} \cdot h \cdot (\mathbf{v_0} + \Delta\mathbf{v}) + \frac{\partial \mathbf{F}(\mathbf{x_0}, \mathbf{y_0})}{\partial \mathbf{v}} \cdot \Delta\mathbf{v}\right) \tag{2.58}$$

6. Umformen:

$$\left(\mathcal{E}_{3\times3} - h\, m^{-1} \frac{\partial \mathbf{F}}{\partial \mathbf{v}} - h^2 m^{-1} \frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right) \Delta\mathbf{v} = h \cdot m^{-1} \cdot \left(\mathbf{F_0} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \cdot h \cdot \mathbf{v_0}\right) \tag{2.59}$$

Dies entspricht dem folgenden Gleichungssystem (es sei beachtet, dass $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ und $\frac{\partial \mathbf{F}}{\partial \mathbf{v}}$ konstant sind):

$$\mathcal{A}_{3\times3} \cdot \Delta\mathbf{v} = \mathbf{b} \qquad \mathcal{A}_{3\times3} = const;\ \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = const \tag{2.60}$$

Es handelt sich in diesem Fall um ein Gleichungssystem, bestehend aus drei Gleichungen mit drei Unbekannten: $\Delta v_1, \Delta v_2, \Delta v_3$. Dieses ist in der Regel lösbar. $\Delta\mathbf{v}$ kann nun z.B. mit dem Gauß'schen Eliminationsverfahren oder dem in Abschnitt 2.1.6 beschriebenen Konjugierten Gradientenverfahren bestimmt werden.

2.1.6 Konjugiertes Gradientenverfahren

In dieser Arbeit wird der von David Baraff in [BW98] vorgestellte Lösungsansatz verwendet, die entstehende Differentialgleichung mit Hilfe des Konjugierten Gradientenverfahrens[7] zu lösen.

[7] Dieses wird im weiteren Verlauf dieser Arbeit auch mit CG-Verfahren abgekürzt (CG - conjugate gradient method), um den zahlreichen englischsprachigen Veröffentlichungen zum Thema Kleidungssimulation zu entsprechen.

Das Konjugierte Gradientenverfahren ist eine der am weitesten verbreiteten Methoden zum Lösen von spärlich besetzten Gleichungssystemen. Diese Arbeit kann keinen vollständigen Überblick über dieses Verfahren geben, aus diesem Grund sei auf [Yan01] und [She94] verwiesen, wo sich jeweils sehr ausführliche Herleitungen und Beschreibungen des CG-Verfahren finden lassen.

Mit dem CG-Verfahren können spärlich besetzte Gleichungssysteme folgender Form gelöst werden:

$$\begin{aligned} \mathcal{A} \cdot \mathbf{x} &= \mathbf{b} \\ \mathcal{A} &= const \\ \mathbf{b} &= const \end{aligned} \tag{2.61}$$

Mit diesem Gleichungssystem kann folgende Funktion der gleichen Dimension aufgestellt werden:

$$\mathbf{f}(\mathbf{x}) = \frac{1}{2} \cdot \mathbf{x}^T \mathcal{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c \tag{2.62}$$

Es kann gezeigt werden, dass das globale Minimum von $\mathbf{f}$ Lösung des Gleichungssystems $\mathcal{A} \cdot \mathbf{x} = \mathbf{b}$ ist, wenn $\mathcal{A}$ symmetrisch und positiv-definit[8] ist [AB03]. Zur Bestimmung des Minimums werden konjugierte Gradienten genutzt.

Es handelt sich um ein iteratives Verfahren, welches jedoch sehr schnell konvergiert. Der Startvektor $\mathbf{x}$, die Matrix $\mathcal{A}$, der Vektor $\mathbf{b}$, die Fehlertoleranz ε und die maximale Anzahl von Iterationsschritten i_{max} werden dem Algorithmus übergeben. Dieser besitzt folgende zwei Abbruchbedingungen:

1. die maximale Anzahl von Iterationsschritten i_{max} ist erreicht
2. $\varepsilon < \|\mathcal{A} \cdot \mathbf{x} - \mathbf{b}\| \quad \Leftrightarrow \quad \varepsilon^2 < (\mathcal{A} \cdot \mathbf{x} - \mathbf{b})^T \cdot (\mathcal{A} \cdot \mathbf{x} - \mathbf{b})$

Die Laufzeit bis zur Konvergenz hängt maßgeblich von dem Parameter ε ab. In der Praxis zeigt sich, dass die Anzahl der Iterationen kaum den Wert 10 übersteigt. Lediglich im Falle von numerischen Instabilitäten (Explosionen) wird bis zum Wert i_{max} iteriert. In Tabelle 2.4 ist der Algorithmus des CG-Verfahrens als Pseudocode gegeben (vgl. [She94] und [BW98]).

[8] $\mathcal{A}$ ist positiv-definit gdw. $\mathbf{x}^T\mathcal{A}\mathbf{x} > 0$ für alle $\mathbf{x} \neq \mathbf{0}$

$i = 0$
$\mathbf{r} = \mathbf{b} - \mathcal{A}\mathbf{x}$
$\mathbf{d} = \mathbf{r}$
$\delta_{new} = \mathbf{r}^T\mathbf{r}$
$\delta_0 = \delta_{new}$
while $i < i_{max}$ and $\delta_{new} > \varepsilon^2\delta_0$ do
    $\mathbf{q} = \mathcal{A}\mathbf{d}$
    $\alpha = \frac{\delta_{new}}{\mathbf{d}^T\mathbf{q}}$
    $\mathbf{x} = \mathbf{x} + \alpha\mathbf{d}$
    $\mathbf{r} = \mathbf{r} - \alpha\mathbf{q}$
    $\delta_{old} = \delta_{new}$
    $\delta_{new} = \mathbf{r}^T\mathbf{r}$
    $\beta = \frac{\delta_{new}}{\delta_{old}}$
    $\mathbf{d} = \mathbf{r} + \beta\mathbf{d}$
    $i = i + 1$

Tabelle 2.4: CG-Verfahren: Algorithmus. Eingabewerte: ϵ, δ_0, $\mathbf{b}$, $\mathcal{M}$

2.2 Grundlagen der Gewebesimulation

Nachdem in den vorhergehenden Abschnitten die für diese Arbeit erforderlichen physikalisch-mathematischen Grundlagen und Zusammenhänge dargestellt wurden, folgt in diesem Abschnitt eine Einführung in die Grundlagen der Gewebe- und Kleidungssimulation. Zunächst wird ein Überblick über die wirkenden Kräfte innerhalb eines Gewebes gegeben. Dann werden die beiden üblichen Repräsentationen von Geweben und deren Eigenschaften vorgestellt. Anschließend folgt ein kurzer Abriss über Kollisionen und die gerade bei kollidierenden elastischen Körpern zu beachtenden Besonderheiten.

2.2.1 Methoden der Gewebesimulation

In diesem Abschnitt werden die beiden üblichen Repräsentationen von Gewebe auf dem Computer und deren prinzipielle Berechnung vorgestellt. Unterschieden wird zwischen der kontinuierlichen und der Partikel-Repräsentation. Erstere wird mit Hilfe der Finiten Element Methode (im Folgenden: FE-Methode) und Letztere mit geeigneten Integrationsverfahren simuliert. Da die FE-Methode viel Rechenzeit benötigt und bis dato noch keine Echtzeitanimationen von Kleidung mit Hilfe der FE-Methode existieren, liegt der

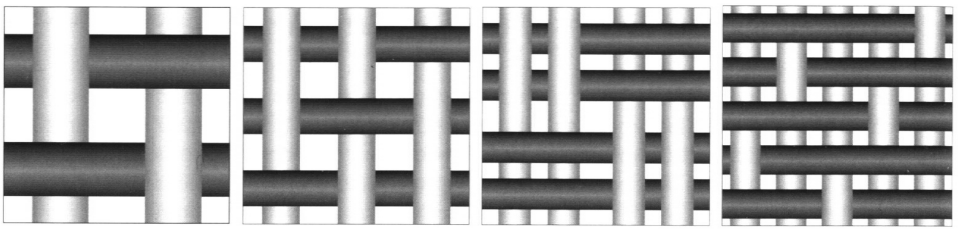

Abb. 2.4: Verschiedene maschinell gewebte Muster: Standard, Twirl, Korb, Satin. Quelle: [VM00b]

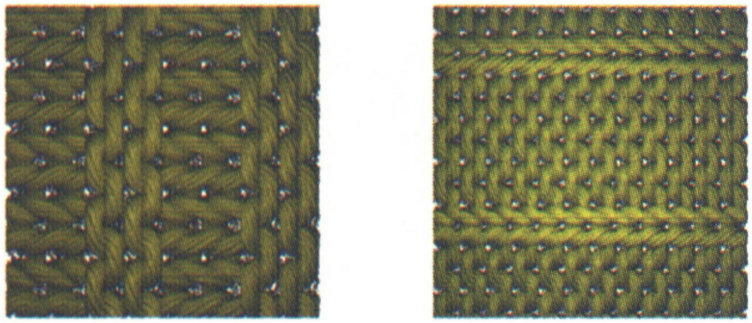

Abb. 2.5: Detailaufnahmen eines gestrickten Materials. Quelle: [HB00]

Schwerpunkt auf der Präsentation der Partikel-Repräsentation, die zudem in der im Rahmen dieser Arbeit implementierten Simulationsumgebung und auch allgemein in der Praxis üblicherweise verwendet wird.

2.2.1.1 Wirkende Kräfte in einem Gewebe

Die reale Physik eines Gewebes ist ausschließlich von den physikalischen Eigenschaften des verwendeten Fadens abhängig. Dessen mechanischen Eigenschaften wie Steifigkeit, Elastizität oder Torsionsspannung bestimmen maßgeblich das Verhalten des Gewebes. Zudem beeinflusst die Art der Verarbeitung des Fadens zu einem Gewebemuster das Verhalten des Gewebes. Verschiedene gewebte Muster sind in Abbildung 2.4 dargestellt.

Der Ansatz, die Physik eines gewebten Fadens und damit auch ein gewebtes Material zu simulieren, würde theoretisch zu vielversprechenden realistischen Ergebnissen führen[9]. Im Rahmen der Echtzeit-Computergraphik kann dieses mikroskopische und komplexe physikalische Verhalten nicht simuliert werden: Es sei bedacht, wieviele Knotenpunkte allein in einem kleinen Ausschnitt gewebten oder gestrickten Materials vorliegen (vgl. Abb. 2.5).

In der Praxis wird lediglich versucht, das makroskopische Verhalten ei-

[9]Ein Ansatz zur Simulation von gestricktem Material wurde z.B. in [HB00] vorgestellt.

nes Gewebes zu simulieren[10]. Dazu wird der Ansatz gewählt, die Physik des Gewebes direkt zu modellieren. Der Schwerpunkt liegt dabei wie oft in der Computergraphik auf dem physikalisch korrekten Aussehen, nicht auf der physikalischen Korrektheit. Allgemein üblich ist es, die Physik des Gewebes unter Berücksichtigung seiner hervorstechenden physikalischen Eigenschaften (Gewebe besitzt eine bestimmte Dehnbarkeit, eine bestimmte Steifigkeit und eine bestimmte Scherbarkeit) auf drei wirkende Kräfte zu reduzieren:

- Stretch Forces (Zugkräfte)
- Bend Forces (Neigungskräfte)
- Shear Forces (Scherkräfte)

Die Stretch-, Bend- und Shear-Forces simulieren die wirkenden Kräfte innerhalb eines Gewebes, die durch Dehnung, Steifigkeit bzw. Scherung desselben entstehen (vgl. dazu Abb. 2.6). Mit Hilfe dieser drei Kräfte wird das Verhalten von Geweben ausreichend gut beschrieben. Im Folgenden werden in Anlehnung an die zahlreichen englischsprachigen Veröffentlichungen die vorstehend aufgeführten englischsprachigen Bezeichnungen verwendet.

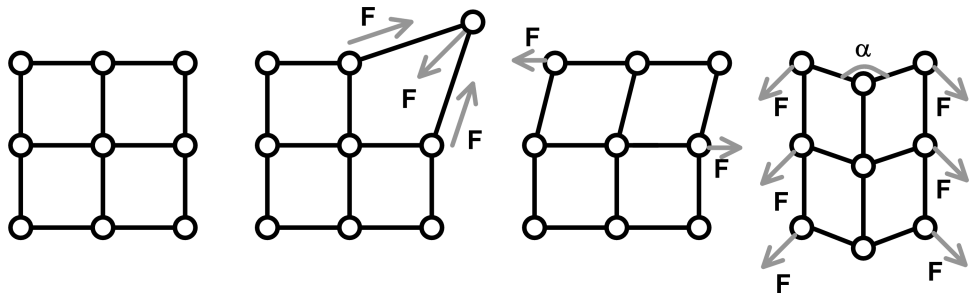

Abb. 2.6: Wirkende Kräfte in einem Material: Ausgangspunkt (es wirken keine Kräfte), Stretch Forces, Shear Forces, Bend Forces (3D).

Das Ziel der Simulation ist es, diese Kräfte geeignet zu repräsentieren und damit eine der Realität entsprechende Gewebedarstellung zu erzielen. Verschiedene reale Materialien können mit einer Variation des Betrags dieser Kräfte modelliert werden. Weitergehende physikalische Phänomene wie etwa das Auftreten von Rissen bei einer bestimmten Dehnung werden in der Regel vernachlässigt. Derartige Effekte treten nur bei Sonderfällen auf, die in der Kleidungssimulation oft nicht benötigt werden. In dieser Arbeit wird ein Konzept zur Simulation von Rissen vorgestellt (siehe Kapitel 4).

[10]Dies gilt, von vereinzelten Versuchen abgesehen, sowohl für die Echtzeit- als auch die Nicht-Echtzeit-Computergraphik.

2.2.1.2 Kontinuierliche Repräsentation

Der Ansatz der kontinuierlichen Repräsentation von Geweben und Kleidung hat seine Ursprünge in der Kontinuumsmechanik, in der die benötigten Ergebnisse zumeist mit der FE-Methode berechnet werden. Das zu betrachtende Gewebe wird als Kontinuum betrachtet, so dass diese Darstellung der Realität relativ nahe kommt, da keinerlei Diskretisierung[11] notwendig ist. Bei der Berechnung entsteht ein gekoppeltes, nicht-lineares Differentialgleichungssystem, in das die wirkenden Kräfte global für den gesamten Stoff einfließen[12]. In einer physikalisch basierten Simulation sollten dabei alle in der Realität wirkenden Kräfte wie Gravitation, Luftreibung, Steifigkeitskräfte des Materials etc. einfließen.

Diskussion Wie bereits oben angedeutet, ist die FE-Methode ausgesprochen rechenintensiv, da die Lösung eines nichlinearen DGL-Systems erforderlich ist. Dies ist der Preis für die ausgesprochen hohe Genauigkeit des Verfahrens. Zudem kommt es bei der kontinuierlichen Modellierung zu einigen Problemen, die wiederum separat behandelt werden müssen: Zum einen ist ein Gewebe in der Regel nicht homogen, zum anderen kommt es bei starken Faltenwürfen zu Problemen während der Interpolation (Überschwingen der Approximation), die nur durch komplexere nicht-lineare Funktionen behoben werden können (vgl. [AH89]).

Bislang ist noch kein Echtzeit-Kleidungssimulationsverfahren basierend auf einer FE-Methode vorgestellt worden[13]. Da eine hohe physikalische Genauigkeit in der Computergraphik/-animation oft nicht erforderlich bzw. z.T. auch gar nicht gefragt ist (der physikalisch korrekte Schein ist in der Regel ausreichend), ist dieses Verfahren nicht geeignet zur Echtzeit-Gewebesimulation.

2.2.1.3 Partikel Repräsentation

Die Partikel Repräsentation stellt die üblicherweise verwendete Darstellung zur Simulation von Geweben dar. Das darzustellende Gewebe wird als Partikelsystem betrachtet und dementsprechend diskretisiert (vgl. Abb. 2.7). Als Simulationsbasis dient in der Regel ein Feder-Masse Partikelsystem. Jedem

[11] Zur Berechnung mit der FE-Methode ist auch eine Diskretisierung notwendig, die dann zumeist polynomiell interpoliert wird. Aber dennoch wird das Material als Kontinuum betrachtet.

[12] Zum Vergleich: Bei dem folgenden Partikel-System fliessen die wirkenden Kräfte abgesehen von der Gravitation lediglich lokal für das jeweilige Partikel ein.

[13] Es existiert auch kein Verfahren, welches in der Dimension der Echtzeit liegt. Ein relativ schnelles Verfahren wurde in [EKS03] vorgestellt.

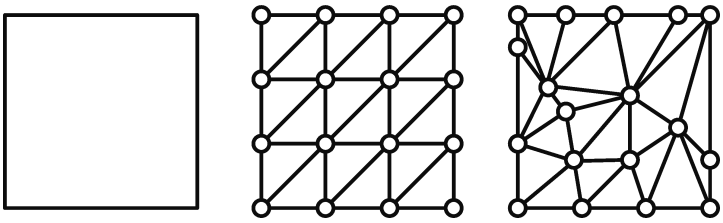

Abb. 2.7: Ein Gewebe wird als Partikelsystem diskretisiert: homogene und inhomogene Diskretisierung.

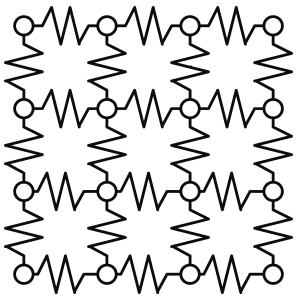

Abb. 2.8: Darstellung der Stretch-Forces in einem Partikelsystem mit Federn.

Partikel wird eine Masse entsprechend der Masse des umgebenden Gewebes zugeordnet. Zumindest die Stretch-Forces werden durch Federn dargestellt (vgl. Abb 2.8). Für die Darstellung der Bend- und Shear-Forces gibt es verschiedene Ansätze. In [BW98] wurde der Ansatz vorgestellt, die beiden Kräfte durch eigene Funktionen in Abhängigkeit des zugehörigen Winkels darzustellen (vgl. Abb. 2.9). Dieses Vorgehen erfordert eine separate Behandlung der Shear- und Bend-Forces innerhalb der Simulation und auch die Entwicklung einer geeigneten mathematischen Formulierung dieser beiden Kräfte.

Ein anderer Ansatz besteht darin, die Shear- und Bend-Forces ebenso wie die Stretch-Forces mit Federn zu repräsentieren. Ein Beispiel für ein homogenes Partikelsystem ist in Abb. 2.10 gegeben. Diese Art der Darstellung unterscheidet nicht mehr zwischen den verschiedenen Kräften. Sie alle können mit Hilfe des Feder-Masse Partikelsystems dargestellt und berechnet werden. Dieser Ansatz wird im Rahmen dieser Arbeit verwendet. Die Darstellung der wirkenden Kräfte mit Hilfe von Federn innerhalb einer Simulation ist in Abbildung 2.11 dargestellt.

Inhomogenitäten innerhalb des Materials und verschiedene Materialeigenschaften können durch Massen- und Federvariationen, bzw. Variationen der Federkonstanten und ungedehnten Federlängen in den entsprechenden Gleichungen hervorgerufen werden. Zudem können Materialeigenschaften über die Topologie des Feder-Masse Systems definiert werden.

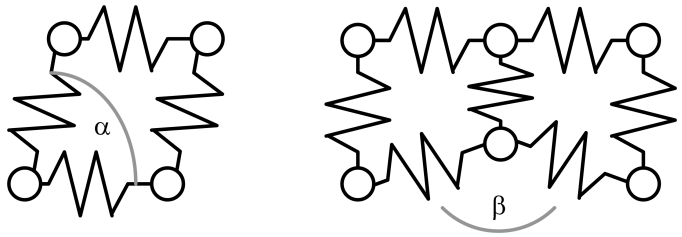

Abb. 2.9: Links: Darstellung der Shear-Forces mit Winkelabhängigkeiten. Rechts: Darstellung der Bend-Forces mit Winkelabhängigkeiten.

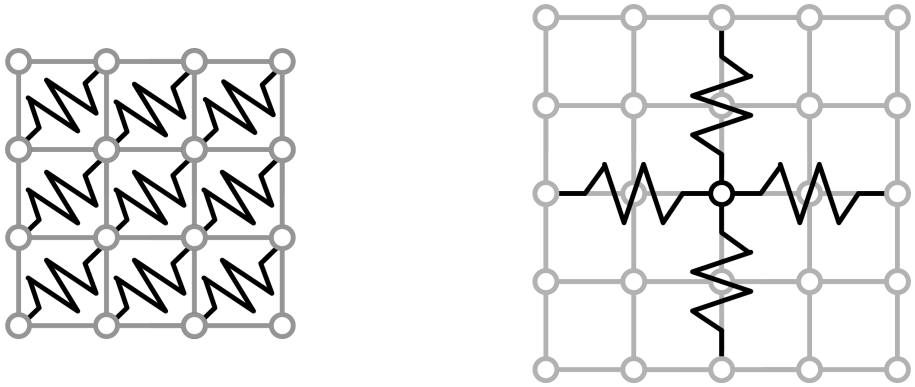

Abb. 2.10: Links: Darstellung der Shear-Forces mit Federn. Rechts: Darstellung der Bend-Forces (lediglich dargestellt für das mittlere Partikel) mit Federn.

Dämpfung Die Definition von Dämpfungen ist im Rahmen der numerischen Simulation von großer Bedeutung: Komplexe Systeme neigen zur Explosion aufgrund von numerischen Instabilitäten. Mit Hilfe von Dämpfungen kann die Stabilität erheblich verbessert werden.

Super-Elastischer Effekt Ein Problem der Modellierung mit Hilfe von Federn sei an dieser Stelle erwähnt: Die in Abschnitt 2.1.3 vorgestellte mathematische Darstellung einer Feder beschreibt eine Kraft in linearer Abhängigkeit zur Auslenkung aus der Ruhelage. Diese Darstellung gilt selbst in der Realität bei einer Feder nur für gewisse Auslenkungen, da ab einer gewissen Auslenkung z.B. materielle Verformungen auftreten, so dass die Feder nicht mehr in die ursprüngliche Ruhelage zurückkehrt. Auch für Gewebe und Kleidung gilt dieses lineare Kraftgesetz nur mit Einschränkungen: Die Dehnbarkeit eines Gewebes folgt im Allgemeinen keiner linearen Kraftregel und theoretisch können die Federn bei entsprechend großen wirkenden Kräften beliebige Längen annehmen. Dadurch erhält das simulierte Gewebe bei kleinen Federkonstanten und dementsprechend stark dehnbaren Federn einen gewissen gummiartigen Anschein. Dieser entspricht nicht der Realität, da

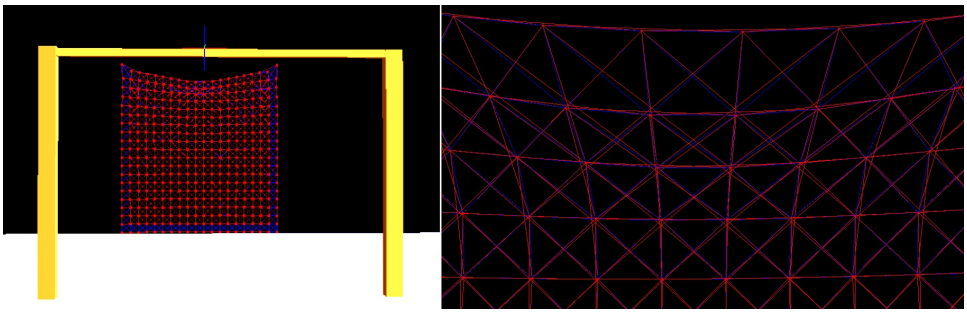

Abb. 2.11: Darstellung der Stretch- (blau), Shear- und Bend-Forces (jew. rot) in der Praxis. Gerendert mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

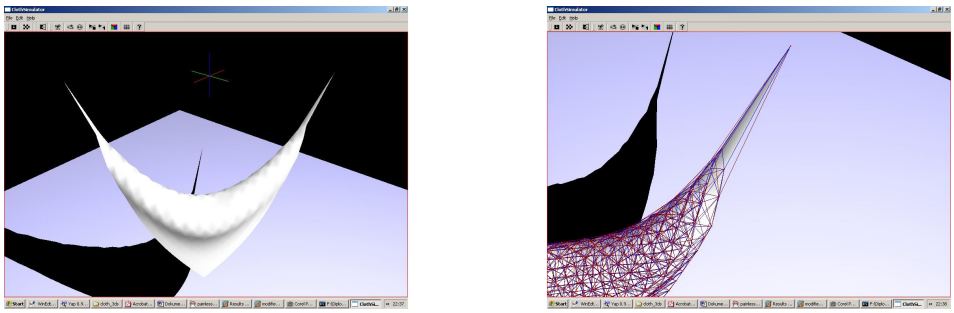

Abb. 2.12: Super-Elastischer Effekt. Links: Hängendes Gewebe (2 Constraints). Rechts: Problem an den Constraints: Federn sind stark gedehnt. Gerendert mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

die Steifigkeit von realer Kleidung wesentlich höher ist. Stark wahrnehmbar ist dieser Effekt z.B. wenn ein Gewebe an wenigen Constraints[14] aufgehangen ist, so dass die wenigen Federn, die von diesem Partikel ausgehen, die gesamte Masse des Gewebes tragen. Sie sind dementsprechend sehr stark gedehnt und hinterlassen einen unrealistischen Eindruck (vgl. Abb. 2.12). Dieses Phänomen wird Super-Elastischer Effekt genannt.

2.2.2 Kollisionen

In dem Moment, in dem das zu simulierende Gewebe sich nicht mehr im leeren Raum befindet, werden Kollisionen mit anderen Objekten entschei-

[14]In diesem Fall bezeichnen Constraints statische Partikel, die ihre Position nicht verändern können bzw. sich auf vorgegebenen Bahnen bewegen. Diese Darstellung entspricht physikalischen Zwangsbedingungen, die die Anzahl der Freiheitsgrade verringern. Eine anschaulichere Beschreibung wäre Aufhängepunkte. In Anlehnung an englischsprachige Veröffentlichungen werden sie in dieser Arbeit jedoch Constraints oder Constraint-Points genannt.

dend, um die Interaktion des Gewebes mit der Umgebung beschreiben zu können. Zudem kann es aufgrund der sehr freien Beweglichkeit des Gewebes zu Kollisionen mit sich selber kommen. Bei der Animation von elastischen Körpern, zu denen auch Kleidung gehört, wird deshalb zwischen zwei Arten von Kollisionen unterschieden. Diese werden in der Regel separat behandelt, um die jeweilige Kollision möglichst effektiv erkennen und behandeln zu können:

- cloth-surface collision (cloth-rigid environment collision)
- cloth-cloth collision (self-collision)

Kollisionen werden allgemein in einem zweistufigen Prozess behandelt:

1. collision detection
2. collision response

Kollisionserkennung ist ein komplexer Prozess, dessen Aufwand mit der Anzahl der verwendeten Polygone quadratisch wächst. Intuitive Ansätze überprüfen alle Polygone einer Szene miteinander und entscheiden, ob eine Kollision vorliegt. Dieses Vorgehen überschreitet mit steigender Anzahl von Polygonen schnell eine in akzeptabler Zeit berechenbare Grenze.

Da die Kollisionserkennung ein in der Computergraphik ausgesprochen häufig auftretendes Problem darstellt, existieren zahlreiche Veröffentlichungen zu dieser Thematik. Optimierungsansätze basieren auf der Reduktion von Komplexität und zumeist auf der Verwendung von Bounding Boxes/Volumes, Projektionen zur Verringerung der Dimension, Space Subdivision Methods oder der Verwendung von Hierarchien. Aktuelle Ansätze werden in Kapitel 3 vorgestellt.

Die Behandlung einer aufgetretenen Kollision wird auch in zahlreichen Arbeiten behandelt. Einige Ansätze werden in Kapitel 3 vorgestellt.

Kapitel 3

Verwandte Arbeiten

In den folgenden Abschnitten werden einige für die Echtzeit-Simulation von Kleidung wichtige Veröffentlichungen vorgestellt. Da eine Vielzahl von Veröffentlichungen im Bereich der Kleidungssimulation existiert, wurde daraus eine Auswahl für diese Arbeit relevanter getroffen. Zunächst werden die grundlegenden Arbeiten der Kleidungssimulation vorgestellt und anschließend die aktuellen Arbeiten zum Thema behandelt.

3.1 Historischer Überblick

Die ersten Arbeiten zur Kleidungssimulation auf dem Computer wurden in den 80er Jahren veröffentlicht. Ein Überblick über die Geschichte der Kleidungssimulation ist in [NG96] zu finden.

Die Arbeit von Weil J.Weil stellte 1986 auf der Siggraph einen der ersten geometrischen Ansätze zur Kleidungsvisualisierung vor [Wei86]. Sein Ansatz bestand aus einem zweistufigen Prozess zur Darstellung eines rechteckigen Gewebestückes, welches an verschiedenen Constraints befestigt war. Benutzt wurde eine aus der Physik bekannte Funktion (vgl. [Kuy97])

$$y = \frac{a}{2}\left(e^{x/a} + e^{-x/a}\right) = a\cosh(x/a), \tag{3.1}$$

die ein hängendes Seil oder Kabel beschreibt und die auf den dreidimensionalen Fall erweitert wurde. Topologisch wurde das Gewebe mit einem 2D-Netz von geometrischen 3D-Punkten modelliert. Zwischen den Constraints wurde die obige Funktion interpoliert und mit einem Unterteilungsalgorithmus wurde das Netz unter weiterer Berücksichtigung der obigen Funktion diesen Werten angepasst (vgl. Abb. 3.1). Dieser Ansatz ist ein rein geometrischer, mechanische Eigenschaften des Gewebes werden nicht berücksichtigt.

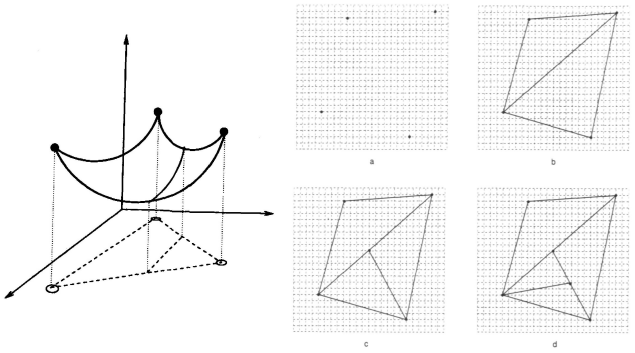

Abb. 3.1: Geometrischer Ansatz zur Kleidungssimulation von J.Weil. Links: Einmal unterteiltes Dreieck im Raum. Rechts: Unterteilung des Netzes. Quelle: [HB00], [NG96]

Die Arbeit von Feynman Im Jahre 1987 wurde von C.Feynman in seiner Master Thesis [Fey87] ein erster physikalisch basierter Ansatz zur Kleidungssimulation vorgestellt. Das Gewebe wurde auch durch ein 2D-Netz im 3D-Raum dargestellt. Feynman stellte eine Energiefunktion auf, die sich aus dem Elastizitäts-, dem Steifigkeits- und dem Gravitationspotential zusammensetzte. Aus dem Minimum dieser Energie ergaben sich die statischen Endpositionen des Gewebes. Feynman stellte eine Simulation von hängendem Gewebe und einer von Gewebe bedeckten Kugel vor (vgl. Abb. 3.2).

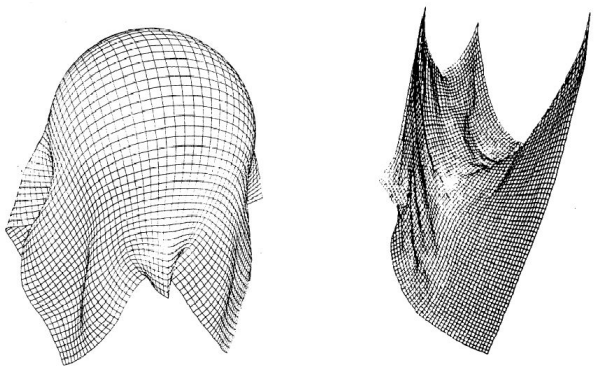

Abb. 3.2: Quelle: Ergebnisse des physikalisch basierten Ansatzes zur Kleidungssimulation von C.Feynman. [NG96]

Die Arbeit von Haumann Im selben Jahr wurde von Haumann das erste Feder-Masse System zur Kleidungssimulation vorgestellt. Das darzustellende Gewebe wurde mit Hilfe von Punktmassen diskretisiert und die Physik des Gewebes wurde mit zwei Arten von Federn dargestellt: Edge-

und Hinge[1]-Springs. Diese entsprechen den in Abschnitt 2.2.1.1 vorgestellten Stretch- und Shear-Springs. Haumanns System unterstützte Luftreibung, Kollisionserkennung und benutzte zur Integration die in Abschnitt 2.1.5.1 vorgestellte explizite Euler-Methode. Damit ist Haumanns Arbeit als eine der grundlegenden Arbeiten der Kleidungssimulation zu bewerten, deren Prinzipien heute noch hauptsächlich verwendet werden.

Zahlreiche weitere Arbeiten folgten in den nächsten Jahren, wobei sich ein Teil der Forschung auch darauf bezog, Verfahren zur verbesserten und vereinfachten Modellierung von realer Kleidung oder Prozesse zur Verbesserung des Realismus wie z.B. Faltenwurf zu entwickeln.

3.2 Stand der Technik

3.2.1 Methoden der Gewebesimulation

3.2.1.1 Die Arbeit von Baraff und Witkin

Die Arbeit „Large Steps in Cloth Simulation“ [BW98] von David Baraff und Andrew Witkin hat große Veränderungen in der Kleidungssimulation bewirkt. Sie stellen in ihrer Arbeit ein Konzept vor, mit dem eine stabile Kleidungssimulation bei relativ großen Zeitschritten realisiert werden kann. Das Konzept basiert auf der Verwendung eines Feder-Masse Partikelsystems und einer impliziten Integration (vgl. Abschnitt 2.1.5.5), wobei das entstehende Gleichungssystem mit dem Konjugierten Gradientenverfahren (vgl. Abschnitt 2.1.6) gelöst wird. Dieses Vorgehen gewährleistet eine sehr stabile Simulation, wobei die Einschränkungen anderer Integrationsverfahren, wie z.B. des expliziten Euler-Verfahrens in Bezug auf die kleine berechenbare Zeitschrittweite, nicht mehr vorhanden sind. Die bisherige Beschränkung auf lediglich sehr kleine Zeitschritte, die dementsprechend lange Rechenzeiten mit sich brachte, verhinderte bis dato eine realistisch aussehende schnelle Animation. Die 1998 mit der impliziten Integration erzielten Renderzeiten lagen im Bereich von 2s pro Frame bei einem aus 2600 Partikeln bestehenden Gewebestück und im Bereich von 10s pro Frame bei einem aus 4530 Partikeln bestehenden Kleid. Die für die Simulation verwendete Hardware ist nicht bekannt.

In der Arbeit wird zwischen den drei in einem Gewebe wirkenden Kräften Stretch-, Shear- und Bend-Forces unterschieden (vgl. Abschnitt 2.2.1.1). Dabei werden die Stretch-Forces mit Hilfe von Federn und die Shear- und

[1]Scharnier, Angelpunkt

Bend-Forces mit Winkelabhängigkeiten beschrieben (vgl. Abschnitt 2.2.1.3). Die jeweiligen mathematischen Formulierungen wurden nicht physikalisch basiert sondern empirisch definiert eingeführt.

Die Arbeit von Baraff und Witkin war Grundlage für die meisten folgenden Arbeiten zum Thema Kleidungssimulation und hat als solche einen neuen Abschnitt in der Kleidungssimulation eingeleitet. Implementierungen wie Maya-Cloth, die Kleidungsfunktionalität von Renderman[2] aber auch freie Projekte wie Freecloth, basieren auf den Ansatz von Baraff und Witkin.

3.2.1.2 Weitere Arbeiten

Nach der Einführung der impliziten Integration in die Kleidungssimulation wurden zahlreiche weitere Arbeiten veröffentlicht, die zumeist auf der impliziten Integration aufbauend einige Neuerungen oder Verbesserungen präsentierten. In diesem Abschnitt werden diejenigen Arbeiten kurz vorgestellt, die eine erwähnenswerte Neuerung darstellen oder neue Impulse in Bezug auf eine Echtzeitsimulation von Kleidung geben.

Die Arbeiten von Desbrun, Schröder, Meyer, Debunne u. Barr In der Arbeit „Interactive animation of structured deformable objects“ [DSB99] wird eine Methode vorgestellt, die mit Hilfe der impliziten Integration nach Baraff und der Einführung eines Post-Prozesses gute Ergebnisse auch in interaktiven Umgebungen erzielt. Mit dem Konzept des Post-Prozesses kann das Problem des Super-Elastischen Effektes behoben werden. So wird nach dem eigentlichen Zeitschritt ein inverser Dynamikschritt angewendet: Es wird eine Maximalauslenkung für alle Federn definiert und diese darf nicht überschritten werden. In dem Post-Prozess werden alle Federlängen überprüft und ihre Position wird iterativ bei Überschreitung der erlaubten Maximallänge verändert. Mit dieser Methode kann es nicht mehr zu extrem ausgedehnten Federlängen kommen, die ihrerseits keinerlei Entsprechung bei realer Kleidung oder realem Gewebe finden.

In der Arbeit „Interactive Animation of Cloth-like Objects in Virtual Reality“ [MDDB00] der gleichen Arbeitsgruppe wird ein hybrides Modell vorgestellt, welches eine Kombination aus explizitem und implizitem Algorithmus verwendet. Es wird eine gute Stabilität erreicht und die interaktive Animation von kleineren Gewebestücken ist möglich.

In einer weiteren Arbeit über verformbare Objekte ([DDCB01]) wird ein

[2] D. Baraff und A. Witkin sind z.Zt. auch bei Pixar tätig und arbeiten weiterhin an der Thematik Kleidungs-/Elastische Körpersimulation.

zeit-adaptiver Ansatz vorgestellt, in dem abhängig von den jeweiligen Positionsveränderungen nach einem Zeitschritt entschieden wird, ob der Zeitschritt verkleinert werden muss. Dieses Vorgehen beschleunigt die Berechnung vor allem in Fällen, in denen noch keine Kollisionen vorliegen. In der Echtzeit-Animation kann dieses Prinzip jedoch nicht angewendet werden, da konstante Zeitschritte für eine flüssige Animation oftmals erforderlich sind.

Die Arbeit von Wang In der Arbeit „Adaptive Cloth Simulation“ [Wan02] wird ein raum-adaptiver Ansatz zur Kleidungssimulation vorgestellt, der die Beschränkung auf ein statisch unterteiltes Gewebe aufhebt. Abhängig von den Krümmungswinkeln zwischen den einzelnen Elementen wird bei Überschreitung eines Grenzwertes dynamisch eine weitere Unterteilung eingefügt. Als Basis dient wiederum die implizite Integration. Mit diesem Vorgehen kann die Performance erheblich verbessert werden, da in der Regel weniger Partikel zu behandeln sind als bei einer homogenen Mesh-Unterteilung.

Die Arbeit von Villard und Borouchaki Die Arbeit „Adaptive Meshing for Cloth Animation“ [VB02] stellt ebenfalls ein adaptives Meshing-Verfahren vor. Es besitzt keinerlei Nachteile im Vergleich zu einer homogenen Mesh-Unterteilung. Der vorgestellte Algorithmus halbiert etwa die Anzahl der zu behandelnden Partikel. Die Renderzeit verringert sich etwa um den Faktor 8 bei gleichen visuellen Ergebnissen, da wesentlich weniger Partikel und Federn zur Darstellung des Gewebes verwendet werden. Siehe dazu auch Abbildung 3.3.

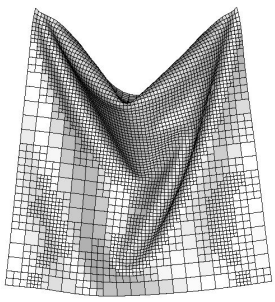

Abb. 3.3: Adaptive Mesh-Unterteilung nach Villard. Quelle: [VB02]

Die Arbeit von Oshita und Makinouchi In der Arbeit „Real-Time Cloth Simulation with Sparse Particles“ [OM01] wird ein Verfahren vorgestellt, welches mit ausgesprochen wenigen Partikeln gute Ergebnisse liefert

(vgl. Abb. 3.4). Das darzustellende Gewebe wird mit nur wenigen Partikeln repräsentiert, deren Positionen und Geschwindigkeiten berechnet werden. Es folgt ein geometrisch basierter Ansatz, in dem diese Teilchen mit Hilfe von Bezier-Kurven interpoliert werden. Dadurch werden etwaige harte Kanten beseitigt. Zusätzlich können durch die Wahl der Bezierparameter noch wesentlich feinere Falten erzeugt werden als sie das grobe Partikelsystem darstellen könnte. Dieses Verfahren kann in Echtzeitumgebungen realisiert werden.

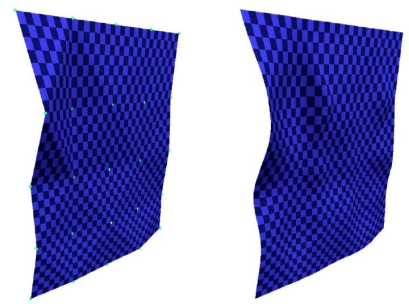

Abb. 3.4: Kleidungssimulation mit wenigen Partikeln nach dem Ansatz von Oshita und Makinouchi. Links: Grobes Mesh. Rechts: Interpolation mit Bezier-Kurven. Quelle: [OM01]

Die Arbeit von Heidrich, Seidel et al. In der Arbeit „Efficient Cloth Modeling and Rendering“ [DLHS01] wird ein Modell zur Kleidungssimulation vorgestellt, das sowohl blickpunktabhängige Effekte wie Verdeckung und Selbstschattierung als auch Beleuchtungseffekte betrachtet. Ziel ist es, nicht mehr rein eben wirkende Materialien darzustellen und damit die realistische Erscheinung zu verbessern. Als Basis dient eine zusätzliche Datenstruktur, in der Parameter für ein Lafortune-Reflektionsmodell, eine Zuordnungstabelle sowie Normalen und Tangenten abgespeichert sind. Diese Vorgehensweise gestattet es z.B., verschiedene Strickmuster plastischer und somit überzeugender darzustellen.

Die Arbeit von Cheng, Xu, Shi und Shum In der Arbeit „Physically-based Real-time Animation of Draped Cloth“ wird ein Verfahren vorgestellt, welches sich explizit mit hängenden Geweben, wie z.B. Gardinen, beschäftigt. Der Ansatz besteht darin, die Gardine als eine Aneinanderreihung von vertikalen Pendeln zu modellieren, die sich unabhängig voneinander bewegen. Nach jedem Zeitschritt werden diese Pendel mit Kurvenfunktionen wie z.B. Splines interpoliert. Dadurch ist der nötige Rechenaufwand wesentlich geringer als bei einem 2D-Netz - dennoch werden sehr überzeugende Ergebnisse erzielt (vgl. Abb. 3.5).

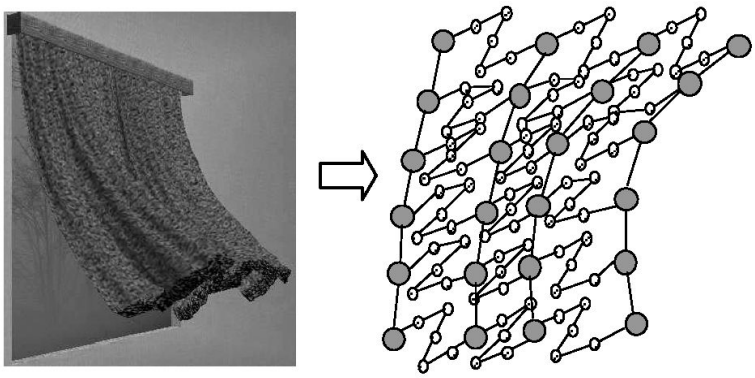

Abb. 3.5: Semi-rigid rods (etwa: halb-starre Stäbe). Nach dem Ansatz von Cheng, Xu et al. Quelle: [CXJS01]

Die Arbeit von Etzmuss, Keckeisen und Straßer Die Arbeit „A Fast Finite Element Solution for Cloth Modelling“ [EKS03] ist eine der wenigen aktuellen Arbeiten, die sich noch mit der Finiten-Element Methode in Bezug auf Kleidungssimulation beschäftigt. Das Prinzip reduziert das nichtlineare Elastizitätsproblem auf ein ebenes, lineares Problem pro Zeitschritt. Damit wird eine relativ schnelle Berechenbarkeit im Vergleich zu herkömmlichen Finite-Element Methoden erreicht. Dennoch bewegt sich diese Methode jenseits der in Echtzeit berechenbaren Grenze: Die Simulation eines aus 6711 Dreiecken bestehenden T-Shirts benötigt ohne Kollisionsbehandlung pro Frame bereits 3 Sekunden.

Die Arbeit von Vassilev, Spanlang und Chrysanthou In der Arbeit „Fast Cloth Animation on Walking Avatars“ [VSC01] werden zwei neue Ansätze vorgestellt: Zum einen wird die Funktionalität der Vertex- und Fragmentshader moderner Graphikkarten genutzt, um nicht nur die Normalen von kollidierenden Objekten zu bestimmen, sondern auch um die Geschwindigkeiten der einzelnen Vertexes in Bezug auf die Partikel zu interpolieren. Damit wird eine verhältnismäßig schnelle Simulation erreicht: Der vorgestellte Algorithmus benötigt für die Berechnung eines aus 672 Partikeln bestehenden Rocks $0.25s$ pro Frame und für ein aus 2300 Partikeln bestehendem Kleid $0.3s$ pro Frame. Die Angaben schließen eine bildbasierte Kollisionserkennung mit ein, die Vergleiche im Tiefenbuffer durchführt.

Die Arbeit präsentiert auch ein neues Konzept zur Beseitigung des Super-Elastischen Effektes: Falls eine Feder eine vorgegebene Maximallänge überschreitet, werden die Geschwindigkeiten in Richtung der Feder auf den Wert Null gesetzt, so dass eine weitere Ausdehnung nicht möglich ist.

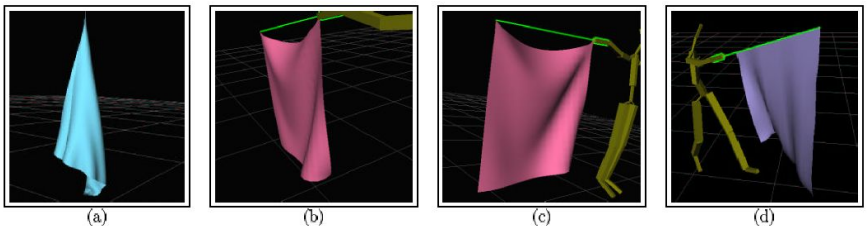

Abb. 3.6: Ergebnisse von Kang, Choi und Cho Quelle: [KCC00]

Die Arbeiten von Kang, Choi und Cho In der Arbeit „Fast and stable Animation of Cloth with an Approximated Implicit Method“ [KCC00] wird ein approximiertes implizites Integrationsverfahren vorgestellt, mit dem ein mit mehreren hundert Partikeln dargestelltes Gewebe interaktiv animiert werden kann. Der Ansatz besteht darin, alle von h^2 abhängigen Terme zu vernachlässigen und damit den Berechnungsaufwand zu verringern. Trotz der Approximation werden stabile Lösungen erreicht (vgl. Abb. 3.6).

In der gleichen Arbeit wird auch ein verbessertes Konzept zur Beseitigung des Super-Elastischen Effektes vorgestellt: Bei Überschreitung einer Maximallänge einer Feder werden die Positionen angepasst. Dieser Ansatz entspricht dem in [Pro97] vorgestellten. Die neue Idee besteht darin, eine Ordnung zu definieren, nach der die Partikel in ihrer Position angepasst werden. Die vorgestellte Ordnung ist eine nach Federlängen sortierte Liste. Dadurch kommt es während der Positionsveränderungen kaum noch zu Schleifen: Das Problem, dass eine Partikelposition verändert wird und diese Positionsveränderung die Länge einer anderen Feder über den Maximalwert hinaus vergrößert, tritt weniger häufig auf.

In der weiterführenden Arbeit „An efficient animation of wrinkled cloth with approximated implicit integration“ [KCCL01] wird ein auf einem approximierten impliziten Integrationsverfahren basierendes Verfahren vorgestellt, mit dem realistische Faltenwürfe erzeugt werden können. Der Ansatz besteht darin, das grobe Partikelsystem mit leicht überschwingenden Freiformkurven zu interpolieren und so keine physikalisch basierten aber dennoch sehr überzeugenden Falten zu generieren.

Dieser Ansatz wird in der Arbeit „Bilayered Approximate Integration for Rapid and Plausible Animation of Virtual Cloth with Realistic Wrinkles“ [KC02] zu einer Zweilayer-Methode weiterentwickelt. Das globale Verhalten des zu simulierenden Gewebes wird mit einem groben Netz (Partikelsystem) implizit simuliert. Ein weiteres sehr feines Netz wird erzeugt, dessen Parameter mit Hilfe des groben Netzes interpoliert werden. Dieses feine Netz betrachtet nur interne Kräfte des Gewebes und wird mit Hilfe eines expliziten

Integrationsverfahrens simuliert. Somit können feine Faltenwürfe dargestellt werden, ohne dass ein sehr feines Netz global mit allen wirkenden externen Kräften simuliert werden muss (vgl. Abb. 3.7). Diese Methode gestattet realistische Animationen in interaktiven Echtzeitumgebungen.

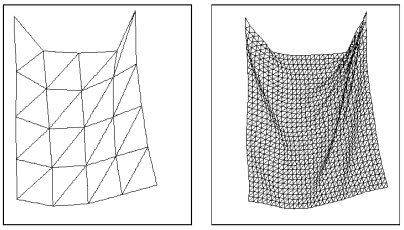

Abb. 3.7: Ergebnisse von Kang, Choi und Cho: Zweilayertechnik. Links: Global simuliertes, grobes Mesh. Rechts: Feines Mesh für realistischen Faltenwurf. Quelle: [KC02]

Die Arbeiten von Volino, Magnenat-Thalman et al. Die Arbeitsgruppe um P. Volino und N. Magnenat-Thalman ist seit langem im Bereich der Kleidungssimulation tätig und hat bereits zahlreiche Arbeiten zum Thema veröffentlicht: u.a. [HBVM02], [MVC02], [AMF03], [CM02], [VM00b].

Der Großteil der Arbeiten bezieht sich auf die realistische Darstellung von Geweben und Kleidung. Die vorgestellten Prinzipien unterliegen somit nicht der Beschränkung der Echtzeit und lassen sich in der Regel nicht in Echtzeit berechnen.

In [CM02] wird ein hybrides Verfahren vorgestellt, welches physikalische mit geometrischen Ansätzen verbindet, um die Rechenzeit zu verringern und auch in interaktiven Umgebungen genutzt werden kann. So wird die Annahme getroffen, dass Kleidung immer in einem gewissen Rahmen vom Körper herunterhängt. Man denke an einen Ärmel eines Pullovers bei einem augestreckten Arm. Dafür wird die aus der Physik bekannte Formel für ein hängendes Seil auf den dreidimensionalen Fall erweitert (vgl. die Arbeit von Weil, Abschnitt 3.1). Dieser Ansatz wird mit einem Partikelsystem kombiniert und es werden gute Ergebnisse bei Kleidungsstücken erzielt.

3.2.1.3 Diskussion

Die von Baraff in die Kleidungssimulation eingeführte implizite Integration ermöglicht es, wesentlich steifere Feder-Masse Systeme bei größeren Zeitschritten zu simulieren. Dadurch kann die Federartigkeit des simulierten Gewebes, die bei expliziten Verfahren auftritt, stark reduziert werden. Mit der impliziten Integration können sehr realistisch anmutende Simulationsergebnisse erzielt werden. Viel versprechend sind auch die vorgestellten adapti-

ven Ansätze oder vor allem die Benutzung von mehrstufigen Meshes, da der benötigte Berechnungsaufwand wesentlich verringert werden kann. Die meisten der oben vorgestellten Arbeiten bieten jedoch keine Echtzeitsimulation. Sie benötigen oftmals eine Zeit pro Frame, die im Sekundenbereich liegt. Dafür ist z.T. jedoch auch die aufwendige Kollisionserkennung verantwortlich, so dass in Echtzeitumgebungen Kompromisse zwischen Geschwindigkeit und Stimmigkeit der Simulation eingegangen werden müssen.

3.2.2 Parallele Gewebesimulation

Die in der Gewebesimulation verwendeten Feder-Masse Partikelsysteme besitzen die Eigenschaft, dass die auf ein Partikel wirkenden Kräfte in der Regel nur lokal von den umgebenen Partikeln oder global wirkenden Kräften wie der Gravitation abhängen (vgl. Kapitel 2.2). Aus diesem Grunde bietet sich eine Parallelisierung der Simulation eines derartigen Partikelsystems an. Jedem Prozess werden bestimmte Partikel und die mit diesen über Federn verbundenen Partikel zugeordnet. Die wirkenden Kräfte können dann parallel berechnet werden. Es entsteht ein gewisser Overhead, da die Prozesse z.T. die Parameter von gleichen Partikeln benötigen. Bei steigender Anzahl von Partikeln in dem System verringert sich dieser Overhead jedoch. Zudem existieren Soft- und Hardwarelösungen, mit denen Prozesse auf den selben Speicher zugreifen können[3].

Auch das in Kapitel 2.1.6 vorgestellte Konjugierte Gradientenverfahren, dessen größter Aufwand darin besteht, große Matrizen mit einem Vektor zu multiplizieren, kann gut parallelisiert werden. Die Matrix wird in eine Anzahl Zeilen aufgeteilt, die der Anzahl der zur Verfügung stehenden Prozesse entspricht. Jeder Prozess kann nun die entsprechenden Multiplikationen und Additionen durchführen und schließlich werden die Ergebnisse wieder zu einem Vektor zusammengefügt.

Ansätze zur Parallelisierung der impliziten Kleidungssimulation werden in [LGPT02], [ZFV02] und [RRZ00] vorgestellt. Dabei beziehen sich die Arbeiten [LGPT02] und [RRZ00] auf Realisierungen auf parallelen Computern und [ZFV02] bezieht sich auf Realisierungen in einem Cluster. Die erreichten Beschleunigungen bei der Verwendung von 8 bis 16 Prozessoren im Vergleich zu einem Prozessor variieren im Bereich von $2,5$ bis zum Faktor 6, je nach verwendeter Methode und Anzahl der Partikel. Die parallele Gewebesimulation ist somit sehr gut geeignet für interaktive Umgebungen.

[3] Softwarelösung: z.B. OpenMP. Hardwarelösungen: z.B. Cray Y-MP, Convex C-2, Cray C-90.

3.2.3 Methoden der Kollisionserkennung

Kollisionserkennung ist ein Standardproblem in der Computergraphik. Dementsprechend können die geläufigen Standard-Verfahren auch in der Kleidungssimulation genutzt werden. Die Kollisionserkennung von extrem elastischen Körpern wie Gewebe mit Objekten benötigt jedoch sehr viel Rechenzeit (vgl. [Pro97]). Noch mehr Rechenzeit wird zur Erkennung von Selbstkollisionen benötigt. Aus diesem Grund ist eine optimierte Kollisionserkennung für eine Echtzeitumgebung vonnöten.

3.2.3.1 Kollision zweier Dreiecke

Die Grundlage für alle folgenden Verfahren ist es, feststellen zu können, ob zwei Dreiecke miteinander kollidieren. In der Regel wird geprüft, ob separierende Ebenen senkrecht zu den jeweiligen Kanten existieren, die die beiden Dreiecke trennen. Andere Verfahren legen zwei Ebenen durch die beiden Dreiecke und überprüfen, ob die Schnittgerade der beiden Ebenen (diese existiert, wenn die Ebenen nicht koplanar zueinander sind) durch beide Dreiecke verläuft. Je nach Bedarf wird zwischen den folgenden Kollisionstypen unterschieden (vgl. [Bar98], [VM00b]):

- Polygon-Kanten Kollision
- Polygon-Vertex Kollision
- Kanten-Kanten Kollision
- Kanten-Vertex Kollision
- Vertex-Vertex Kollision

3.2.3.2 Bounding Volumes

Die Methode der Bounding Volumes ist ein in der Computergraphik allgemein weit verbreitetes Verfahren zur Reduktion von Komplexität. In der Kollisionserkennung wird das Verfahren genutzt, um nicht jedes Polygon eines aus zahlreichen Polygonen bestehenden Objektes auf etwaige Kollisionen hin zu überprüfen. Stattdessen wird ein umfassender Volumenkörper (Bounding Volume) um das Objekt gelegt und zunächst lediglich dieser Körper auf etwaige Kollision hin überprüft. Liegt keine Kollision vor, so liegt auch keine Kollision mit einem innerhalb des Bounding Volumes liegenden Polygon vor. Im anderen Fall werden die innerhalb des Bounding Volumes liegenden Polygone auf Kollisionen überprüft. Da aber in der Regel bei einer animierten Szene pro Frame nur einige Kollisionen vorliegen, wird die Komplexität

erheblich verringert. Als Bounding Volumes werden in der Regel einfache geometrische Körper verwendet, die eine schnelle Überprüfung ermöglichen: z.B. Quader, Kugeln oder Zylinder (bounding boxes, bounding spheres, bounding cylinders). Vergleiche zu dieser Thematik [FGL03], [FvFH00], [VM00b], [Wat00] und [WW92]. Das Prinzip der Bounding Volumes bietet sich an, mit den folgenden Verfahren kombiniert zu werden.

3.2.3.3 Projektionsverfahren

Projektionsverfahren begründen sich auf dem folgenden Prinzip: Wenn Objekte geometrisch kollidieren, so kollidieren auch ihre Projektionen. Auf diese Art kann eine Kollision im Raum bei reduzierter Dimension erkannt werden. Die Verfahren variieren zwischen 2D-Projektionen auf eine Ebene (in der Regel achsenparallel) und 1D-Projektionen auf eine Gerade (in der Regel Koordinatenachse). Nun können effektive Intervall-Vergleiche durchgeführt werden. Im Falle von 1D-Projektionen kollidieren Objekte genau dann, wenn alle drei korrespondierenden Intervalle kollidieren. Zudem können effektive Sortierverfahren angewendet werden, um die Kollisionserkennung zu optimieren. Auch das Projektionsverfahren kann sinnvoll mit anderen Kollisionserkennungsverfahren kombiniert werden, um eine weitere Performanceverbesserung zu erzielen.

3.2.3.4 Raumunterteilungsverfahren

Eine weitere Methode zur Reduktion der Komplexität besteht darin, den 3D-Raum in kleinere Räume zu unterteilen. Unterschieden wird zwischen der regulären und der hierarchischen Unterteilung, die im Folgenden vorgestellt werden:

Reguläre Unterteilung Bei der regulären Unterteilung wird der Raum in ein reguläres dreidimensionales Gitter von Quadern unterteilt. Diese Elemente des Gitters werden auch Voxel genannt. Da es sich um ein reguläres Gitter handelt, können einzelne Vertexes, Objekte oder Polygone sehr schnell einem bestimmten Voxel zugeordnet werden(vgl. [ZY00]). Befinden sich zwei Polygone innerhalb des selben Voxels, so werden diese hinsichtlich einer Kollision überprüft. Andere Ansätze verwenden ein sehr feines Voxelgitter und entscheiden in dem Moment, in dem zwei Polygone das selbe Voxel schneiden, dass eine Kollision vorliegt.

Hierarchische Unterteilung Bei einer hierarchischen Unterteilung wird der darzustellenden Geometrie im Gegensatz zur Regulären Unterteilung

Genüge getragen. So ermöglicht die hierarchische Unterteilung eine in Bezug auf die darzustellende Geometrie optimierte Unterteilung. Weit verbreitete hierarchische Unterteilungen sind die Octree- und Binary Space-Unterteilung (vgl. [MKE03], [VM00b]). Rekursive Algorithmen ermöglichen aufgrund der Hierarchie eine sehr schnelle Entscheidung, ob Kollisionen vorliegen.

Ein Problem der hierarchischen Unterteilung ist die Aktualisierung der Hierarchie bei bewegten oder deformierbaren Objekten. Diese Aktualisierung kann unter Umständen einen großen Aufwand erfordern.

3.2.3.5 Sonstige Verfahren

Die Arbeit von Fuhrmann, Groß und Luckas und Choi und Ko In der Arbeit „Interactive Animation of Cloth including Self Collision Detection“ [FGL03] und der Arbeit „Stable but Responsive Cloth“ [CH02] wird ein sehr schnelles Verfahren zur Detektion von Selbstkollisionen beschrieben: Anstatt aufwendiger Dreieckskollisionenberechnungen wird der Abstand von Partikelpärchen betrachtet. Unterschreitet dieser einen vorgegebenen Minimalwert ε (Mindestabstand) werden die beiden Partikel als kollidierend betrachtet. Dabei darf der Mindestabstand nicht zu gering gewählt werden, da die Kollision eines Partikels genau zwischen zwei benachbarten Partikeln stattfinden kann.

Diese Methode ist sehr schnell und bei einer entsprechend feinen Triangulierung auch verhältnismäßig genau. Die gesamte Kollisionserkennung ist lediglich auf Abstandsvergleiche beschränkt und kann z.B. mit Bounding Boxes noch effektiver gestaltet werden. Wenn eine Kollision nicht erkannt wird, besteht jedoch keine Möglichkeit, diese wieder rückgängig zu machen. Somit ist das vorgestellte Prinzip als approximatives Verfahren zu sehen, welches sich jedoch aufgrund seiner schnellen Berechenbarkeit gerade für interaktive Systeme anbietet.

Die Arbeit von Wloka In der Arbeit „Interactive Cloth Simulation“ [Wlo01] (von nVidia für nVidia-Grafikprozessoren veröffentlicht) wird ein einfaches Verfahren vorgestellt, Kollisionen von Partikeln mit einer Kugel innerhalb eines Vertexshaders zu erkennen und zu beheben. Dieser Ansatz verspricht eine sehr schnelle Kollisionserkennung und -behandlung, die hardwarebeschleunigt direkt auf der Graphikkarte ausgeführt wird. Aufgrund der großen Beschränktheit von aktuellen Vertex- und Fragmentshadern ist dieses Konzept jedoch nicht sehr flexibel und nur in bestimmten Einzelfällen nutzbar.

3.2.3.6 Probleme der Kollisionserkennung bei diskreten Zeitschritten

Da es sich bei simulierter Kleidung in der Regel um 2D-Netze im 3D-Raum handelt, die als solche keine Volumenausdehnung besitzen, kann es bei der Kollisionserkennung zu folgendem Problem kommen: Falls zwei Polygone am Zeitpunkt t_0 nah beieinander liegen und sich aufeinander zu bewegen, kann es vorkommen, dass sie sich bis zum Zeitpunkt $t_0 + \Delta t$ durch einander „hindurch“ bewegt haben und nicht kollidierten. In diesem Fall würde keines der oben vorgestellten Verfahren eine Kollision erkannt haben, obwohl zwischen den beiden diskreten Zeitpunkten t_0 und $t_0 + \Delta t$ eine Kollision stattgefunden hat. Dieses Problem kann durch Time Bounding Volumes, die die Position der Objekte sowohl zum Zeitpunkt t_0 als auch zum Zeitpunkt $t_0 + \Delta t$ einschließen, behoben werden (vgl. [Pro97], [BFA02a]). Das Time Bounding Volume umschließt damit alle Positionen, die zwischen den Zeitpunkten t_0 und $t_0 + \Delta t$ bei entsprechend kleinen Zeitschritten eingenommen wurden. Werden diese Bounding Volumes zur Kollisionserkennung eingesetzt, können auch Kollisionen erkannt werden, die zwischen den diskreten Zeitpunkten stattgefunden haben.

3.2.4 Methoden der Kollisionsbehandlung

Die Methoden der Kollisionsbehandlung werden allgemein nach zwei Ansätzen unterschieden:

- Dynamische Behandlung
- Geometrische Behandlung

Dabei bezeichnet die dynamische Behandlung eine Kollisionsbehandlung, die direkt mit physikalisch basierten Kräften auf das entsprechende Partikel wirkt und in dem folgenden Integrationsschritt berücksichtigt wird. Die Geometrische Behandlung bezeichnet Kollisionsbehandlungen, die direkt auf Positionen, Geschwindigkeiten und/oder Beschleunigungen wirkt. Diese wird in der Regel in einem Postprocess durchgeführt.

Allgemein sollten bei der Kollisionsbehandlung die Energieerhaltung und das Reaktionsprinzip (Actio=Reactio, 3. Newtonsche Gesetz) berücksichtigt werden, um eine überzeugende Simulation zu gewährleisten.

Bei der Kleidungssimulation werden Selbstkollisionen in der Regel dynamisch und Kollisionen zwischen Kleidung und Festkörpern in der Regel geometrisch behandelt.

3.2.4.1 Dynamische Kollisionsbehandlung

In der Arbeit „Large Steps in Cloth Simulation“ [BW98] wird eine constraint-basierte Kollisionsbehandlung vorgestellt: Bei Kollisionen der Kleidung mit sich selber wird den betreffenden Partikeln eine der Bewegungsrichtung entgegengesetzt wirkende Kraft hinzugefügt, so dass sie auseinander gezogen werden. Diese Kraft wird im Rahmen der impliziten Integration behandelt. Bei Kollisionen mit Festkörpern wird ein geometrisch basierter Ansatz gewählt: Das entsprechende Partikel wird auf die Oberfläche des Festkörpers verschoben. Dabei wird nicht die Position direkt geändert, sondern ein Verschiebungsvektor zu Δx addiert, der somit in die Integration mit einfließt.

In [FGL03] wird eine constraint-basierte Kollisionsbehandlung vorgestellt: Wenn zwei Partikel einen definierten Mindestabstand unterschreiten, wird zwischen ihnen eine temporäre Feder gespannt, die bewirkt, dass die Partikel sich nicht zu nahe kommen. Nach einer vorgegebenen Lebensdauer oder bei zunehmendem Abstand wird die temporäre Feder wieder entfernt. Auf diese Art werden die Kollisionen direkt im Rahmen der Integration mit behandelt, ohne eine explizite Kollisionsbehandlung entwickeln zu müssen.

3.2.4.2 Geometrische Kollisionsbehandlung

In [Pro97] wird eine auf Geschwindigkeiten basierende geometrische Kollisionsbehandlung vorgestellt: Bei einer Kollision wird die Geschwindigkeit an der Normalen des Kollisionspunktes inelastisch reflektiert, wobei eine Konstante den Energieverlust bei der Reflektion definiert. Die Konstante kann auch benutzt werden, um verschiedene Materialeigenschaften zu simulieren.

Die meisten Arbeiten zum Thema Kleidungssimulation lösen das Problem der Kollision von Kleidung mit Festkörpern folgendermaßen: Es kann leicht entschieden werden, ob ein Dreieck oder Partikel sich innerhalb eines Körpers befindet (insbesondere bei einfachen geometrischen Körpern: Kugeln, Quader, Zylinder, etc.). Im Falle einer Kollision werden die Positionen des beteiligten Dreiecks oder die Position des Partikels in einem Post-Prozess derart angepasst, dass sich das Dreieck wieder außerhalb des Körpers befindet. Die Ansätze variieren zwischen einer Verschiebung entlang der Normalen der Festkörperoberfläche oder einer Verschiebung entgegengesetzt der Geschwindigkeit der beteiligten Partikel. Derartige Kollisionsbehandlungen bringen immer die Gefahr einer etwaigen Explosion des Partikelsystems mit sich, da dem System auch Energie hinzugefügt

werden kann. Diese unerwünschten Effekte können durch ausgeglichene Dämpfungsfunktionen und kleinere Zeitschritte behoben werden.

3.2.4.3 Sonstige Kollisionsbehandlungen

In der Arbeit „Untangling Cloth“ [BWK03] werden die neuen Verfahren der Kollisionserkennung und -behandlung vorgestellt, die in dem Pixar-/Disney-Film „Monsters Inc.“ verwendet wurden. Der Schwerpunkt liegt dabei auf der Erkennung und Beseitigung von Kollisionen, die z.B. vorliegen, wenn ein T-Shirt getragen wird und der Arm eng am Oberkörper anliegt. In dieser Situation kommt es zu dem Problem, dass das T-Shirt unter der Achsel in den Körper eindringen muss. Herkömmliche Verfahren können diese Situation nicht behandeln, da sie davon ausgehen, dass ein Gewebe niemals in einen anderen Körper eindringt. Sie führen in dieser Situation in der Regel zu einer Explosion des Partikelsystems. Das vorgestellte Verfahren kann jedoch nicht in einer Echtzeit-Umgebung realisiert werden: Die vorgestellte Kollisionserkennung liegt im Sekundenbereich pro Frame.

3.2.5 Vergleich verschiedener Integrationsverfahren

In der Arbeit [VM01] werden verschiedene Integrationsverfahren in Hinblick auf ihre Effizienz bei der Kleidungssimulation verglichen. Betrachtet werden das explizite Midpoint-Verfahren, das explizite Runge-Kutta-Verfahren 5. Ordnung und das implizite Euler-Verfahren unter Verwendung der Konjugierten Gradientenmethode mit verschiedenen Iterationstiefen (1,2,4 und 8). Bei den im Folgenden dargestellten Ergebnissen sollte beachtet werden, dass ein allgemein gültiger Vergleich dieser Verfahren nur schwerlich möglich ist, da die Ergebnisse in einem hohen Maße von dem zu simulierenden Kontext abhängen: So ist z.B. die gewünschte Genauigkeit der Simulation, die Steifigkeit des Systems, die gewünschte Geschwindigkeit der Berechnung[4] und der Simulationskontext[5] mitentscheidend dafür, welches Integrationsverfahren für einen konkreten Anwendungsfall am besten geeignet ist.

Ein Hauptkriterium für die Wahl des Integrationsverfahrens ist in der Regel die Performance. Auch diese ist von zahlreichen Parametern und dem zu simulierenden Kontext abhängig:

- Berechnungszeit, die für eine Iteration benötigt wird - diese hängt von der Komplexität des verwendeten Integrationsverfahrens ab

[4] Diese ist insbesondere im Zusammenhang mit einer Echtzeitsimulation entscheidend.

[5] Soll z.B. möglichst schnell gegen einen statischen Zustand konvergiert werden oder eine komplette Simulation bei konstanten diskreten Zeitschritten erfolgen, was für eine flüssige Animation z.B. zwingend erforderlich wäre.

- Zeitschrittgröße, um eine bestimmte Genauigkeit oder numerische Stabilität zu gewährleisten

- gewünschte Genauigkeit - diese steigt bei einer Verminderung der Zeitschrittweite oder der Verwendung eines Integrationsverfahrens höherer Ordnung

- numerische Stabilität, die wiederum die maximale Zeitschrittweite limitiert

Das in [VM01] verwendete Framework (entwickelt in C++) zur Ermittlung der Vergleichswerte unterstützt zwei verschiedene diskrete mechanische Repräsentationen der Kleidungsobjekte:

- ein vollständiges Oberflächenelastizitätsmodell

- ein vereinfachtes Federmassemodell

Ersteres erlaubt die Simulation anisotroper Elastizitätseigenschaften. Letzteres stellt jede Kante des Meshes mit einer Feder dar und ist damit eines der einfachsten Modelle der Kleidungssimulation überhaupt. Alle Messungen wurden ohne Kollisionserkennung durchgeführt, um die Ergebnisse direkt vergleichen zu können.

Das implizite Euler-Verfahren mit reduzierter Anzahl von Konjugierten Gradienteniterationen ist in seiner Geschwindigkeit durchaus vergleichbar mit den expliziten Verfahren Midpoint und Runge-Kutta. Mit lediglich einer Iteration ist es nur unwesentlich langsamer als das einfache Midpoint-Verfahren.

In der Arbeit werden die Integrationsverfahren auch anhand zweier Beispielszenarien verglichen. Zum einen wird ein freier Fall ohne Luftwiderstand und zum anderen der Fall eines hängenden Gewebes mit einer fixen Kante simuliert. Beim freien Fall stellt sich heraus, dass das Runge-Kutta-Verfahren wesentlich flexiblere Parameter zulässt (z.B. Steifigkeit) als das Midpoint-Verfahren. Das implizite Euler-Verfahren benötigt mindestens 4 konjugierte Gradienteniterationen, um eine ähnliche Präzision wie das Runge-Kutta-Verfahren zu erreichen.

Bei der Simulation eines hängenden Gewebes werden die Vorteile der impliziten Integration offensichtlicher: Bis zum ersten Durchlaufen des Ruhezustandes ohne Instabilitäten benötigt die Simulation mit dem Runge-Kutta-Verfahren 500 Iteration mit einem maximalen Zeitschritt von $t = 0.001s$. Die Simulationszeit beträgt 150 Sekunden. Das implizite Euler-Verfahren benötigt je nach Anzahl der Iterationen und Zeitschrittweite etwa $20 - 200$

Iterationen bei Zeitschritten von $t = 0.025..0.0025s$. Die Simulationszeit beträgt etwa 30 Sekunden und ist damit um das fünffache schneller als das Runge-Kutta-Verfahren.

Somit stellt das implizite Euler-Verfahren eine gute Basis für die Kleidungssimulation dar. An dieser Stelle sei noch ein weiteres Ergebnis der Arbeit [VM01] erwähnt: Die hohe Stabilität des impliziten Euler-Verfahrens, die es erlaubt, relativ große Zeitschritte zu wählen, darf nicht mit einer hohen Genauigkeit verwechselt werden - es gilt die Regel, je größer der Zeitschritt, desto größer die Ungenauigkeit. In Bezug auf die Genauigkeit der Verfahren, kann das Runge-Kutta-Verfahren durchaus mit dem impliziten Euler-Verfahren verglichen werden. Gerade bei Echtzeitanwendungen kann jedoch oftmals auf eine hohe Genauigkeit zu Gunsten einer hohen Geschwindigkeit verzichtet werden.

3.2.6 Existierende Implementierungen

Im Internet sind einige Programme (z.T. auch mit Sourcecode) zu finden, die eine Kleidungssimulation auf Basis eines Feder-Masse-System und der expliziten Euler-Integration durchführen. Diese Programme besitzen durchgehend beispielhaften Charakter und simulieren lediglich sehr kleine (bis max. 625 Partikel) und nur homogene quadratische Meshes, unterstützen maximal eine Kugel als Kollisionsobjekt und simulieren Gravitation. Die jeweiligen Simulationsgeschwindigkeiten variieren sehr stark. Alle Programme mit Ausnahme von `D3DCloth`, welches DirectX verwendet, benutzen OpenGL als Graphiklibrary.

- `Cloth Simulator 2000` [PG00] - Ein quadratisches Tuch bewegt sich innerhalb eines Würfels.
- `SimCloth` [Gar03] - Dieses Programm ermöglicht es, mit Kanonenkugeln auf ein rechteckiges, hängendes Stück Gewebe zu schiessen, wobei sich das Gewebe entsprechend der Kollisionsgeschwindigkeit und -position der Kugel bewegt.
- `Cloth` [Ada03] - Ein quadratisches Stück Gewebe fällt auf eine Kugel und rutscht ab.
- `Cloth` [Jac03] - Ein quadratisches Stück Gewebe fällt unter Einwirkung der Gravitation, wobei ein frei gewähltes Partikel als Constraint definiert werden kann.
- `Cloth Simulation` [Isa02] - Ein quadratisches Tuch ist an zwei Constraint-Punkten aufgehängt und fällt auf eine Kugel.

- `Cloth` [Bro02] - Ein rechteckiges Tuch, dessen eine Seite vollständig als Constraint definiert ist, kann interaktiv im Raum bewegt werden. Dieses Programm unterstützt Texture-Mapping.

- `ClothSample` [Mac01] - Ein rechteckiges Tuch kann an vier verschiedenen Constraint-Punkten aufgehängt werden. Texture-Mapping wird unterstützt.

- `Powierz` [Mat] - Ein rechteckiges Tuch fällt auf eine Kugel.

- `D3DCloth` [Pow03] - Dieses Programm konnte im Rahmen dieser Arbeit nicht compiliert werden, es sei jedoch der Vollständigkeit halber erwähnt. Es ist auch das einzige Programm, welches eine Kleidungssimulation unter DirectX vornimmt.

Zwei weitere Programme (`Clothy` und `Freecloth`) werden in den folgenden beiden Abschnitten genauer behandelt, da diese im Funktionsumfang den obigen Programmen überlegen sind und in Kapitel 6 für Performancevergleiche herangezogen werden. `Freecloth` nimmt dabei eine Sonderstellung ein, da es die einzig existierende freie Implementierung der impliziten Integration nach Baraff [BW98] darstellt.

Im Bereich der Nicht-Echtzeit-Simulation sei die Kleidungsfunktionalität von `Maya` erwähnt, die auch auf der impliziten Integration nach Baraff basiert. In Ermangelung einer Test-Version konnte diese jedoch nicht in Bezug auf ihre Performance untersucht werden. Auf welche Art die Kleidungssimulation in `3D Studio Max` durchgeführt wird, konnte nicht ermittelt werden.

3.2.6.1 `Clothy`

`Clothy` [Lan99b] ist ein beispielhaftes Kleidungssimulationsprogramm, welches zur Simulation Feder-Masse-Systeme benutzt. Es können quadratische, homogene Meshes beliebiger Größe erzeugt werden. Diese können mit den expliziten Simulationsverfahren Euler, Midway und Runge-Kutta 4. Ordnung simuliert werden. Parameter, wie z.B. die Schrittweite h und Federkonstanten, können frei gewählt werden. Als Kollisionsobjekt steht eine positionierbare Kugel zur Verfügung. Die Darstellung erfolgt lediglich durch Rendern der Partikel und der diese verbindenden Federn.

3.2.6.2 `Freecloth`

`Freecloth` [Pri03] ist eine freie Implementierung des von D. Baraff vorgestellten Konzeptes der impliziten Integration. Die wirkenden Kräfte werden

auf die von Baraff in [BW98] entwickelte Art berechnet. Somit bietet sich dieses Programm zum Vergleich mit dem in dieser Arbeit entwickelten Konzept an. In `Freecloth` können Simulationsparameter frei gewählt werden. Es unterstützt folgende Constraints: 1,2 u. 4 Constraint Points, einen runden und einen eckigen Tisch. Die Tische werden lediglich durch Constraint Points dargestellt, die die Form des Tisches beschreiben. `Freecloth` unterstützt Texture-Mapping und Shading. Ein Screenshot von `Freecloth` ist in Abbildung 6.14, Abschnitt 6.3.5 zu finden.

Kapitel 4

Neues Konzept zur Echtzeit-Gewebesimulation

In Bezug auf die Geschwindigkeit ist die explizite Euler Integration aufgrund ihrer Einfachheit nicht zu übertreffen. Aufgrund der zahlreichen Nachteile, gerade in Bezug auf die Kleidungssimulation[1], die die explizite Euler-Integration besitzt, wird die implizite Euler-Integration motiviert, die erstmals in Verbindung mit einer Kleidungssimulation in [BW98] vorgestellt wurde. Im ersten Teil dieses Kapitels wird nun ein Konzept zur impliziten Kleidungssimulation in Echtzeit entwickelt, welches die entstehenden Gleichungssysteme mit Hilfe der CG-Methode löst. Die wirkenden Kräfte werden dabei nicht wie von D. Baraff in [BW98] vorgeschlagen empirisch eingeführt, sondern den entsprechenden physikalischen Gesetzen folgend berechnet. Der mathematische Aufwand ist aus diesem Grunde relativ hoch, da zahlreiche Differentiationen von Kräften zur Geschwindigkeit und zur Position im dreidimensionalen Raum analytisch berechnet werden müssen. Die in einem Gewebe wirkenden Kräfte (stretch-, shear- und bend-forces) werden ausschließlich mit Hilfe von Federn dargestellt, so dass eine Gewebemodellierung anschaulich durchgeführt werden kann. Zudem können der Realität entsprechende Dämpfungs- und Reibungskräfte definiert werden, die dem System eine hohe Stabilität geben.

Des Weiteren werden Konzepte zur Kollisionserkennung und -behandlung und zur Verbesserung der graphischen Ausgabequalität in einer Echtzeitumgebung vorgestellt. Die Konzepte zu Kollisionserkennung und -behandlung basieren dabei auf den in Kapitel 3 vorgestellten Ansätzen. Alle Kon-

[1] Super-elastischer Effekt; geringe Schrittweite; es kommt leicht zu Instabilitäten, die nur durch ein Verringern der Schrittweite h beseitigt werden können; lediglich kleine Federkonstanten D gewährleisten eine stabile Simulation

zepte sind allgemein, d.h. unabhängig von etwaigen Graphiklibraries und einer konkreten Implementierung gehalten. Konkrete Hinweise zu einer Implementierung sind in Kapitel 5 zu finden.

4.1 Ablauf der Kleidungssimulation

Das hier vorgestellte System zur Kleidungssimulation besitzt das in Abbildung 4.1 dargestellte Ablaufdiagramm. Nach der Integration und der Berechnung der aktuellen Positionen und Geschwindigkeiten aller Partikel werden Wind, etwaige Schnitte und Kollisionserkennung und -behandlung in einem Postprocess realisiert. Im Falle eines gewünschten Shadings wird eine Pipeline durchlaufen, in der die Normalen berechnet, Texturing durchgeführt und Fell und Schlagschatten berechnet werden.

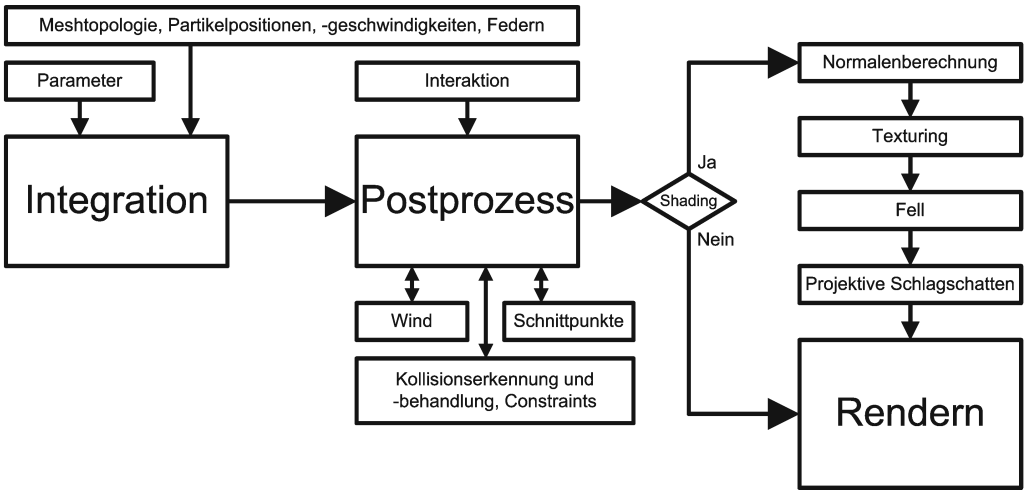

Abb. 4.1: Ablaufdiagramm des in diesem Abschnitt vorgestellten Konzeptes zur Kleidungssimulation.

4.2 Physikalisch basierte implizite Kleidungssimulation

Die impliziten Euler-Integration führt zu folgender Differentialgleichung (vgl. Abschnitt 2.1.5.5):

$$\left(\mathcal{E}_{3\times 3} - h\,m^{-1}\frac{\partial \mathbf{F}}{\partial \mathbf{v}} - h^2 m^{-1}\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\right)\Delta\mathbf{v} = h \cdot m^{-1} \cdot \left(\mathbf{F} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \cdot h \cdot \mathbf{v}\right) \qquad (4.1)$$

In Kapitel 2.1.5.5 handelt es sich um ein dreidimensionales physikalisches System, so dass $\mathbf{F}$, $\mathbf{v}$ und $\mathbf{x}$ jeweils dreikomponentige Vektoren darstellen. Bei den entsprechenden Matrizen $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ und $\frac{\partial \mathbf{F}}{\partial \mathbf{v}}$ handelt es sich um 3×3 Matrizen.

Im Folgenden wird ein Partikelsystem bestehend aus n Partikeln betrachtet, so dass die Vektoren auf $3{\cdot}n$ Komponenten und die Matrizen auf $3{\cdot}n \times 3{\cdot}n$ Matrizen erweitert werden:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \mathbf{x_3} \\ \vdots \\ \mathbf{x_n} \end{pmatrix} = \begin{pmatrix} x_{1x} \\ x_{1y} \\ x_{1z} \\ x_{2x} \\ x_{2y} \\ x_{2z} \\ x_{3x} \\ \vdots \\ x_{nz} \end{pmatrix} \quad \mathbf{v} = \begin{pmatrix} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \vdots \\ \mathbf{v_n} \end{pmatrix} = \begin{pmatrix} v_{1x} \\ v_{1y} \\ v_{1z} \\ v_{2x} \\ v_{2y} \\ v_{2z} \\ v_{3x} \\ \vdots \\ v_{nz} \end{pmatrix} \quad \mathbf{f} = \begin{pmatrix} \mathbf{f_1} \\ \mathbf{f_2} \\ \mathbf{f_3} \\ \vdots \\ \mathbf{f_n} \end{pmatrix} = \begin{pmatrix} f_{1x} \\ f_{1y} \\ f_{1z} \\ f_{2x} \\ f_{2y} \\ f_{2z} \\ f_{3x} \\ \vdots \\ f_{nz} \end{pmatrix} \tag{4.2}$$

Die Matrizen verändern sich entsprechend. Mit der CG-Methode muss zur Bestimmung von $\Delta\mathbf{v}$ ein Gleichungssystem bestehend aus $3 \cdot n$ Differentialgleichungen gelöst werden. Um dieses Gleichungssystem aufstellen zu können müssen $\mathbf{F}$, $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ und $\frac{\partial \mathbf{F}}{\partial \mathbf{v}}$ berechnet werden. $\mathbf{F}$ setzt sich zunächst aus der Gravitation, den wirkenden Feder- und Federdämpfungskräften und der globalen Reibungskraft additiv zusammen. Ein Beispiel für einen Ausschnitt eines möglichen Partikelsystems ist in Abbildung 4.2 dargestellt. Die Kräfte sind für das Partikel p_m gegeben. Der Kraftvektor $\mathbf{F}$ dieses Partikelsystems besitzt z.B. die Form

$$\begin{pmatrix} \vdots \\ \mathbf{F_m} \\ \mathbf{F_o} \\ \mathbf{F_p} \\ \mathbf{F_q} \\ \vdots \end{pmatrix} \tag{4.3}$$

mit $\mathbf{F_m} = \mathbf{F_{F_o}} + \mathbf{F_{D_o}} + \mathbf{F_{F_p}} + \mathbf{F_{D_p}} + \mathbf{F_{F_q}} + \mathbf{F_{D_q}} + \mathbf{F_g} + \mathbf{F_r}$.

Reibungskräfte wie Gleit- oder Haftreibung werden in diesem Modell vernachlässigt, diese können jedoch direkt und ohne großen Aufwand hinzugefügt werden. Topologische und strukturelle Merkmale des Gewebes

können über Art und Parameter der Federn definiert werden. Weitere Kräfte, die z.B. durch Wind oder Kollisionen hervorgerufen werden, werden zunächst im folgenden mathematischen Modell nicht betrachtet - diese können dem System während eines Postprozesses hinzugefügt werden.

$\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ und $\frac{\partial \mathbf{F}}{\partial \mathbf{v}}$ können durch Differentiation von $\mathbf{F}$ unter Zuhilfenahme von Gleichung 2.1, Abschnitt 2.1.1.4 bestimmt werden. Für das in Abbildung 4.2 gegebene Beispiel des Partikels p_m eines aus n Partikeln bestehenden Partikelsystems berechnet sich $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ z.B. folgendermaßen (siehe auch Formel 4.3 - $\mathbf{F_m}$ hängt nur von den Positionen und Geschwindigkeiten der Partikel p_m, p_o, p_p und p_q ab. Allgemein ist die auf ein Partikel wirkende Kraft in diesem Modell nur von seiner eigenen Position und Geschwindigkeit und den Positionen und Geschwindigkeiten aller mit diesem Partikel über Federn verbundenen Partikel abhängig.):

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial \mathbf{F_0}}{\partial \mathbf{x_0}} & \cdots & & \\ \vdots & & & \\ \frac{\partial \mathbf{F_m}}{\partial \mathbf{x_0}} & \frac{\partial \mathbf{F_m}}{\partial \mathbf{x_1}} & \cdots & \frac{\partial \mathbf{F_m}}{\partial \mathbf{x_n}} \\ & & & \vdots \\ & & \cdots & \frac{\partial \mathbf{F_n}}{\partial \mathbf{x_n}} \end{pmatrix} \tag{4.4}$$

$$= \begin{pmatrix} & \cdots & & & & & & & & \\ \vdots & \ddots & & & & & & & & \\ 0 & \cdots & 0 & \frac{\partial \mathbf{F_m}}{\partial \mathbf{x_m}} & \frac{\partial \mathbf{F_m}}{\partial \mathbf{x_o}} & \frac{\partial \mathbf{F_m}}{\partial \mathbf{x_p}} & \frac{\partial \mathbf{F_m}}{\partial \mathbf{x_q}} & 0 & \cdots & 0 \\ & & & & & & & & \ddots & \vdots \\ & & & & & & & & \cdots & \end{pmatrix} \tag{4.5}$$

Wird für $\mathbf{F_m}$ die in Formel 4.3 gegebene Summe eingesetzt, sind für die Differentiation $\frac{\partial \mathbf{F_m}}{\partial \mathbf{x_y}}$ mit $y = m, o, p, q$ jeweils nur die Sumanden relevant, die von $\mathbf{x_y}$ abhängen. So ergeben sich in diesem Beispiel folgende Matrixelemente:

$$\frac{\partial \mathbf{F_m}}{\partial \mathbf{x_m}} = \frac{\partial(\mathbf{F_{F_o}} + \mathbf{F_{D_o}} + \mathbf{F_{F_p}} + \mathbf{F_{D_p}} + \mathbf{F_{F_q}} + \mathbf{F_{D_q}})}{\partial \mathbf{x_m}} \tag{4.6}$$

$$\frac{\partial \mathbf{F_m}}{\partial \mathbf{x_o}} = \frac{\partial(\mathbf{F_{F_o}} + \mathbf{F_{D_o}})}{\partial \mathbf{x_o}} \tag{4.7}$$

$$\frac{\partial \mathbf{F_m}}{\partial \mathbf{x_p}} = \frac{\partial(\mathbf{F_{F_p}} + \mathbf{F_{D_p}})}{\partial \mathbf{x_p}} \tag{4.8}$$

$$\frac{\partial \mathbf{F_m}}{\partial \mathbf{x_q}} = \frac{\partial(\mathbf{F_{F_q}} + \mathbf{F_{D_q}})}{\partial \mathbf{x_q}} \tag{4.9}$$

An dieser Stelle können die Antisymmetrieeigenschaften der durch eine Feder hervorgerufenen Kräfte ausgenutzt werden (Die Antisymmetrie ist in

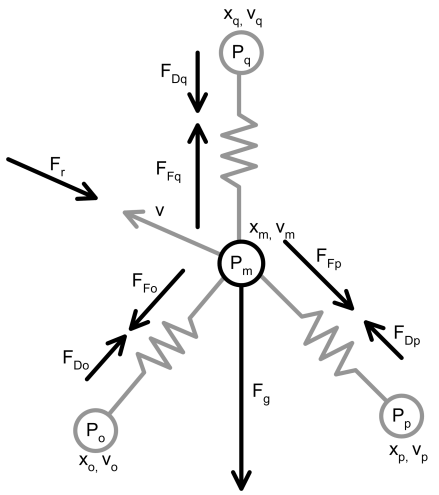

Abb. 4.2: Die wirkenden Kräfte für das Partikel p_m, welches beispielhaft über drei Federn mit drei anderen Partikeln verbunden ist. Dargestellt sind die in diesem Modell verwendeten Kräfte: Gewichtskraft, Federkräfte, Federdämpfungen und die globale Dämpfung.

den Ergebnissen der in den nächsten Abschnitten folgenden Berechnungen erkennbar). Die Berechnung der Differentiationen für alle anderen Partikel und für $\frac{\partial \mathbf{F}}{\partial \mathbf{v}}$ erfolgt entsprechend.

Die konkrete Berechnung der wirkenden Kräfte und ihrer Differentiationen für Partikel bzw. über Federn verbundene Partikelpaare werden im Folgenden dargestellt.

4.2.1 Berechnung der Kräfte und ihrer Differentiationen

4.2.1.1 Gewichtskraft

Die Gewichtskraft ist entsprechend Abschnitt 2.1.2 gegeben:

$$\mathbf{F_g} = m \cdot \mathbf{g} \tag{4.10}$$

Da die Gewichtskraft bei konstanter Masse eine konstante Größe und Orientierung unabhängig von Position und Geschwindigkeit besitzt, ergeben sich die Orts- und Geschwindigkeitsableitung zu Null.

4.2.1.2 Federkraft $\mathbf{F_F}$

Die in 2.1.3 gegebene, die wirkende Kraft eines ungedämpften Federpendels beschreibende Gleichung $F_F = -D \cdot (x - x_0)$ wird auf den dreidimensionalen Fall erweitert:

$$\mathbf{F_F} = +D\ (\|\mathbf{s}\| - s_0) \cdot \frac{\mathbf{s}}{\|s\|} \tag{4.11}$$

$$= -D\ (s_0 - \|\mathbf{s}\|) \cdot \frac{\mathbf{s}}{\|s\|} \tag{4.12}$$

mit

$$\mathbf{s} = \mathbf{x_2} - \mathbf{x_1} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} x_{21} - x_{11} \\ x_{22} - x_{12} \\ x_{23} - x_{13} \end{pmatrix} = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{pmatrix} \tag{4.13}$$

$$s_0 = \|\mathbf{x_{1_0}} - \mathbf{x_{2_0}}\| \qquad \text{(Länge der ungedehnten Feder)} \tag{4.14}$$

$$\|\mathbf{s}\| = \sqrt{s_1^2 + s_2^2 + s_3^2} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{4.15}$$

Umformen ergibt:

$$\mathbf{F_F} = +D\ (\|\mathbf{s}\| - s_0) \cdot \frac{\mathbf{s}}{\|\mathbf{s}\|} \tag{4.16}$$

$$= +D\ (\|\mathbf{x_2} - \mathbf{x_1}\| - s_0) \cdot \frac{\mathbf{x_2} - \mathbf{x_1}}{\|\mathbf{x_2} - \mathbf{x_1}\|} \tag{4.17}$$

$$= +D\ (\mathbf{x_2} - \mathbf{x_1}) \cdot (1 - \frac{s_0}{\|\mathbf{x_2} - \mathbf{x_1}\|}) \tag{4.18}$$

$$= +D\ (\mathbf{x_2} - \mathbf{x_1}) \cdot \frac{\|\mathbf{x_2} - \mathbf{x_1}\| - s_0}{\|\mathbf{x_2} - \mathbf{x_1}\|} \tag{4.19}$$

Da die Federkraft $\mathbf{F_F}$ nur von den Positionen $\mathbf{x_1}$ und $\mathbf{x_2}$ der beiden zugehörigen Partikel abhängt, ergibt sich die Differentiation zur Geschwindigkeit zu Null. Die Differentiation zum Ort erfolgt im nächsten Abschnitt.

4.2.1.3 Differentiation der Federkraft zum Ort: $\frac{\partial \mathbf{F_F}}{\partial \mathbf{x}}$

Im Folgenden wird die Federkraft $\frac{\partial \mathbf{F_F}}{\partial \mathbf{x}}$ zwischen zwei Partikeln, die mit einer Feder verbunden sind, zum Ort differenziert. Grundlage für die Differentiation ist die im vorhergehenden Abschnitt beschriebene Gleichung zur Beschreibung der wirkenden Kraft eines Federpendels und die Definition des Vektorgradienten:

$$\mathcal{M} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_x}{\partial x} & \frac{\partial f_x}{\partial y} & \frac{\partial f_x}{\partial z} \\ \frac{\partial f_y}{\partial x} & \frac{\partial f_y}{\partial y} & \frac{\partial f_y}{\partial z} \\ \frac{\partial f_z}{\partial x} & \frac{\partial f_z}{\partial y} & \frac{\partial f_z}{\partial z} \end{pmatrix} \tag{4.20}$$

Sei

$$c = \|\mathbf{x2} - \mathbf{x1}\|^2 \tag{4.21}$$

so ergibt sich für das erste Element der Vektorgradientenmatrix $\frac{\partial \mathbf{F}_x}{\partial x_1}$:

$$\frac{\partial \mathbf{F}_x}{\partial x_1} = -D\left(1 - \frac{s_0}{\sqrt{c}}\right) + \frac{D \cdot (x_2 - x_1) \cdot s_0 \cdot (-2\,x_2 + 2\,x_1)}{2 \cdot \sqrt{c^3}} \tag{4.22}$$

$$= -D\left(1 - \frac{s_0}{\sqrt{c}}\right) + \frac{D \cdot (x_2 - x_1) \cdot s_0 \cdot (x_1 - x_2)}{\sqrt{c^3}} \tag{4.23}$$

$$= -D\left(1 - \frac{s_0}{c^{\frac{1}{2}}}\right) - D \cdot \frac{(x_2 - x_1)^2 \cdot s_0}{c^{\frac{3}{2}}} \tag{4.24}$$

Die anderen acht Ableitungen ergeben sich nach dem gleichen Schema:

$$\frac{\partial \mathbf{F}_{\mathbf{F}x}}{\partial y_1} = -D \cdot \frac{(x_2 - x_1) \cdot s_0 \cdot (y_2 - y_1)}{c^{\frac{3}{2}}} \tag{4.25}$$

$$\frac{\partial \mathbf{F}_{\mathbf{F}x}}{\partial z_1} = -D \cdot \frac{(x_2 - x_1) \cdot s_0 \cdot (z_2 - z_1)}{c^{\frac{3}{2}}} \tag{4.26}$$

$$\frac{\partial \mathbf{F}_{\mathbf{F}y}}{\partial x_1} = -D \cdot \frac{(x_2 - x_1) \cdot s_0 \cdot (y_2 - y_1)}{c^{\frac{3}{2}}} \tag{4.27}$$

$$\frac{\partial \mathbf{F}_{\mathbf{F}y}}{\partial y_1} = -D\left(1 - \frac{s_0}{c^{\frac{1}{2}}}\right) - D \cdot \frac{(y_2 - y_1)^2 \cdot s_0}{c^{\frac{3}{2}}} \tag{4.28}$$

$$\frac{\partial \mathbf{F}_{\mathbf{F}y}}{\partial z_1} = -D \cdot \frac{(y_2 - y_1) \cdot s_0 \cdot (z_2 - z1)}{c^{\frac{3}{2}}} \tag{4.29}$$

$$\frac{\partial \mathbf{F}_{\mathbf{F}z}}{\partial x_1} = -D \cdot \frac{(x_2 - x_1) \cdot s_0 \cdot (z_2 - z_1)}{c^{\frac{3}{2}}} \tag{4.30}$$

$$\frac{\partial \mathbf{F}_{\mathbf{F}z}}{\partial y_1} = -D \cdot \frac{(y_2 - y_1) \cdot s_0 \cdot (z_2 - z_1)}{c^{\frac{3}{2}}} \tag{4.31}$$

$$\frac{\partial \mathbf{F}_{\mathbf{F}z}}{\partial z_1} = -D\left(1 - \frac{s_0}{c^{\frac{1}{2}}}\right) - D \cdot \frac{(z_2 - z_1)^2 \cdot s_0}{c^{\frac{3}{2}}} \tag{4.32}$$

Darstellung in Matrizenschreibweise und anschließende Umformungen ergeben:

$$\frac{\partial \mathbf{F_F}}{\partial \mathbf{x_1}} = \begin{pmatrix} -D\,(1-\frac{s_0}{c^{\frac{1}{2}}}) - D\cdot\frac{(x_2-x_1)^2\cdot s_0}{c^{\frac{3}{2}}} & -D\cdot\frac{(x_2-x_1)\cdot s_0\cdot(y_2-y_1)}{c^{\frac{3}{2}}} & -D\cdot\frac{(x_2-x_1)\cdot s_0\cdot(z_2-z_1)}{c^{\frac{3}{2}}} \\ -D\cdot\frac{(x_2-x_1)\cdot s_0\cdot(y_2-y_1)}{c^{\frac{3}{2}}} & -D\,(1-\frac{s_0}{c^{\frac{1}{2}}}) - D\cdot\frac{(y_2-y_1)^2\cdot s_0}{c^{\frac{3}{2}}} & -D\cdot\frac{(y_2-y_1)\cdot s_0\cdot(z_2-z1)}{c^{\frac{3}{2}}} \\ -D\cdot\frac{(x_2-x_1)\cdot s_0\cdot(z_2-z_1)}{c^{\frac{3}{2}}} & -D\cdot\frac{(y_2-y_1)\cdot s_0\cdot(z_2-z_1)}{c^{\frac{3}{2}}} & -D\,(1-\frac{s_0}{c^{\frac{1}{2}}}) - D\cdot\frac{(z_2-z_1)^2\cdot s_0}{c^{\frac{3}{2}}} \end{pmatrix} \tag{4.33}$$

$$= \frac{-D\cdot s_0}{c^{\frac{3}{2}}}\cdot\begin{pmatrix} (x_2-x_1)^2 & (x_2-x_1)\cdot(y_2-y_1) & (x_2-x_1)\cdot(z_2-z_1) \\ (x_2-x_1)\cdot(y_2-y_1) & (y_2-y_1)^2 & (y_2-y_1)\cdot(z_2-z_1) \\ (x_2-x_1)\cdot(z_2-z_1) & (y_2-y_1)\cdot(z_2-z_1) & (z_2-z_1)^2 \end{pmatrix} - D\cdot(1-\frac{s_0}{c^{\frac{1}{2}}})\cdot\mathcal{E}_{3\times 3} \tag{4.34}$$

$$= \frac{-D\cdot s_0}{c^{\frac{3}{2}}}\cdot\begin{pmatrix} x_2-x_1 \\ y_2-y_1 \\ z_2-z_1 \end{pmatrix}\cdot\begin{pmatrix} x_2-x_1 & y_2-y_1 & z_2-z_1 \end{pmatrix} - D\cdot(1-\frac{s_0}{c^{\frac{1}{2}}})\cdot\mathcal{E}_{3\times 3} \tag{4.35}$$

$$= \frac{-D\cdot s_0}{c^{\frac{3}{2}}}\cdot\Delta\mathbf{x}\cdot(\Delta\mathbf{x})^T - D\cdot(1-\frac{s_0}{c^{\frac{1}{2}}})\cdot\mathcal{E}_{3\times 3} \tag{4.36}$$

mit $c = \|\mathbf{x2}-\mathbf{x1}\|^2$.

Die Antisymmetrie von $\mathbf{x_1}$ und $\mathbf{x_2}$ ist vor allem in Gleichung 4.33 zu erkennen: Die auf zwei Partikel wirkenden Kräfte einer zwischen diesen Partikeln gespannten Feder besitzen den gleichen Betrag und entgegengesetzte Orientierung. Anhand von Gleichung 4.33 ist auch erkennbar, dass es sich um eine symmetrische Matrix handelt.

4.2.1.4 Federdämpfung $\mathbf{F_d}$

Die in Abschnitt 2.1.4.1 vorgestellte Gleichung zur Berechnung der Dämpfungskraft $F_d = -k \cdot v$ eines eindimensionalen Federpendels wird auf den dreidimensionalen Fall erweitert. Dabei sei bedacht, dass die Dämpfungskraft lediglich in Richtung der Feder wirkt.

$$\mathbf{F_d} = \Big((\mathbf{x_2} - \mathbf{x_1}) \cdot (\mathbf{v_2} - \mathbf{v_1})\Big) \cdot \frac{k_d}{\|\mathbf{x_2} - \mathbf{x_1}\|} \cdot \frac{\mathbf{x_2} - \mathbf{x_1}}{\|\mathbf{x_2} - \mathbf{x_1}\|} \tag{4.37}$$

$$= (\mathbf{\Delta x} \cdot \mathbf{\Delta v}) \cdot \frac{k_d}{\|\mathbf{\Delta x}\|} \cdot \frac{\mathbf{\Delta x}}{\|\mathbf{\Delta x}\|} \tag{4.38}$$

$$= (\mathbf{\Delta x} \cdot \mathbf{\Delta v}) \cdot \frac{k_d}{\|\mathbf{\Delta x}\|^2} \cdot \mathbf{\Delta x} \tag{4.39}$$

$$= k_d \cdot \frac{(\Delta x_1 \Delta v_1 + \Delta x_2 \Delta v_2 + \Delta x_3 \Delta v_3)}{{\Delta x_1}^2 + {\Delta x_2}^2 + {\Delta x_3}^2} \cdot \mathbf{\Delta x} \tag{4.40}$$

4.2.1.5 Differentiation der Federdämpfung zum Ort: $\frac{\partial \mathbf{F_d}}{\partial \mathbf{x}}$

Gleichung 4.40 sei in Komponentenschreibweise gegeben:

$$\mathbf{F_d} = k_d \cdot \begin{pmatrix} x_{2x} - x_{1x} \\ x_{2y} - x_{1y} \\ x_{2z} - x_{1z} \end{pmatrix} \cdot \frac{\Big((x_{2x} - x_{1x})(v_{2x} - v_{1x}) + (x_{2y} - x_{1y})(v_{2y} - v_{1y}) + (x_{2z} - x_{1z})(v_{2z} - v_{1z})\Big)}{(x_{2x} - x_{1x})^2 + (x_{2y} - x_{1y})^2 + (x_{2z} - x_{1z})^2} \tag{4.41}$$

Unter Berücksichtigung der Berechnungsregel für den Vektorgradienten

$$\frac{\partial \mathbf{F_d}}{\partial \mathbf{x_1}} = \begin{pmatrix} \frac{\partial F_{d_x}}{\partial x_{1_x}} & \frac{\partial F_{d_x}}{\partial x_{1_y}} & \frac{\partial F_{d_x}}{\partial x_{1_z}} \\ \frac{\partial F_{d_y}}{\partial x_{1_x}} & \frac{\partial F_{d_y}}{\partial x_{1_y}} & \frac{\partial F_{d_y}}{\partial x_{1_z}} \\ \frac{\partial F_{d_z}}{\partial x_{1_x}} & \frac{\partial F_{d_z}}{\partial x_{1_z}} & \frac{\partial F_{d_z}}{\partial x_{1_z}} \end{pmatrix} \tag{4.42}$$

werden im Folgenden die einzelnen Komponentenableitungen bestimmt. Es

ergibt sich für $\frac{\partial F_{dx}}{\partial x_{1x}}$:

$$\frac{\partial F_{dx}}{\partial x_{1x}} = \frac{-k_d}{(x_{2x}-x_{1x})^2+(x_{2y}-x_{1y})^2+(x_{2z}-x_{1z})^2} \cdot$$
$$\Big((x_{2x}-x_{1x})\cdot(v_{2x}-v_{1x})+(x_{2y}-x_{1y})\cdot(v_{2y}-v_{1y})+(x_{2z}-x_{1z})\cdot(v_{2z}-v_{1z})\Big)$$
$$+\frac{k_d\cdot(x_{2x}-x_{1x})\cdot(-v_{2x}+v_{1x})}{(x_{2x}-x_{1x})^2+(x_{2y}-x_{1y})^2+(x_{2z}-x_{1z})^2} -$$
$$\frac{k_d\cdot(x_{2x}-x_{1x})\cdot(-2\cdot x_{2x}+2\cdot x_{1x})}{\Big((x_{2x}-x_{1x})^2+(x_{2y}-x_{1y})^2+(x_{2z}-x_{1z})^2\Big)^2} \cdot$$
$$\frac{(x_{2x}-x_{1x})\cdot(v_{2x}-v_{1x})+(x_{2y}-x_{1y})\cdot(v_{2y}-v_{1y})+(x_{2z}-x_{1z})\cdot(v_{2z}-v_{1z})}{\Big((x_{2x}-x_{1x})^2+(x_{2y}-x_{1y})^2+(x_{2z}-x_{1z})^2\Big)^2} \tag{4.43}$$

Mit Hilfe von Vektoren dargestellt, entspricht dies:

$$\frac{\partial F_{dx}}{\partial x_{1x}} = \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot(\Delta\mathbf{x}\cdot\Delta\mathbf{v})+$$

$$\frac{k_d\cdot(\Delta x_x)\cdot(-\Delta v_x)}{\|\Delta\mathbf{x}\|^2} - \frac{k_d\cdot(\Delta x_x)\cdot(-2\Delta x_x)\cdot(\Delta\mathbf{x}\cdot\Delta\mathbf{v})}{\|\Delta\mathbf{x}\|^4} \tag{4.44}$$

Umformen ergibt:

$$\frac{\partial F_{dx}}{\partial x_{1x}} = \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot(\Delta\mathbf{x}\Delta\mathbf{v}+\Delta x_x\Delta v_x) + \frac{2\,k_d\,\Delta\mathbf{x}\Delta\mathbf{v}\,(\Delta x_x)^2}{\|\Delta\mathbf{x}\|^4} \tag{4.45}$$
$$= \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot\Big(\Delta\mathbf{x}\Delta\mathbf{v}+\Delta x_x\Delta v_x - \frac{2\,\Delta\mathbf{x}\Delta\mathbf{v}\,(\Delta x_x)^2}{\|\Delta\mathbf{x}\|^2}\Big) \tag{4.46}$$

Für die 8 anderen Ableitungen ergibt sich nach der gleichen Vorgehensweise:

$$\frac{\partial F_{dy}}{\partial x_{1x}} = \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot\Big(\Delta x_y\Delta v_x - \frac{2\,\Delta\mathbf{x}\Delta\mathbf{v}\,\Delta x_y\Delta x_x}{\|\Delta\mathbf{x}\|^2}\Big) \tag{4.47}$$
$$\frac{\partial F_{dz}}{\partial x_{1x}} = \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot\Big(\Delta x_z\Delta v_x - \frac{2\,\Delta\mathbf{x}\Delta\mathbf{v}\,\Delta x_z\Delta x_x}{\|\Delta\mathbf{x}\|^2}\Big) \tag{4.48}$$
$$\frac{\partial F_{dx}}{\partial x_{1y}} = \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot\Big(\Delta x_x\Delta v_y - \frac{2\,\Delta\mathbf{x}\Delta\mathbf{v}\,\Delta x_x\Delta x_y}{\|\Delta\mathbf{x}\|^2}\Big) \tag{4.49}$$

$$\frac{\partial F_{dy}}{\partial x_{1_y}} = \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot \left(\Delta\mathbf{x}\Delta\mathbf{v} + \Delta x_y \Delta v_y - \frac{2\ \Delta\mathbf{x}\Delta\mathbf{v}\ (\Delta x_y)^2}{\|\Delta\mathbf{x}\|^2}\right) \tag{4.50}$$

$$\frac{\partial F_{dz}}{\partial x_{1_y}} = \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot \left(\Delta x_z \Delta v_y - \frac{2\ \Delta\mathbf{x}\Delta\mathbf{v}\ \Delta x_z \Delta x_y}{\|\Delta\mathbf{x}\|^2}\right) \tag{4.51}$$

$$\frac{\partial F_{dx}}{\partial x_{1_z}} = \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot \left(\Delta x_x \Delta v_z - \frac{2\ \Delta\mathbf{x}\Delta\mathbf{v}\ \Delta x_x \Delta x_z}{\|\Delta\mathbf{x}\|^2}\right) \tag{4.52}$$

$$\frac{\partial F_{dy}}{\partial x_{1_z}} = \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot \left(\Delta x_y \Delta v_z - \frac{2\ \Delta\mathbf{x}\Delta\mathbf{v}\ \Delta x_y \Delta x_z}{\|\Delta\mathbf{x}\|^2}\right) \tag{4.53}$$

$$\frac{\partial F_{dz}}{\partial x_{1_z}} = \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot \left(\Delta\mathbf{x}\Delta\mathbf{v} + \Delta x_z \Delta v_z - \frac{2\ \Delta\mathbf{x}\Delta\mathbf{v}\ (\Delta x_z)^2}{\|\Delta\mathbf{x}\|^2}\right) \tag{4.54}$$

Als Matrix zusammengefasst und umgeformt ergibt sich das Ergebnis:

$$\begin{aligned}
\frac{\partial \mathbf{F_d}}{\partial \mathbf{x_1}} &= \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot \left(\Delta\mathbf{x}\Delta\mathbf{v} \cdot \mathcal{E}_{3\times 3} + \begin{pmatrix} \Delta x_x \Delta v_x & \Delta x_x \Delta v_y & \Delta x_x \Delta v_z \\ \Delta x_y \Delta v_x & \Delta x_y \Delta v_y & \Delta x_y \Delta v_z \\ \Delta x_z \Delta v_x & \Delta x_z \Delta v_y & \Delta x_z \Delta v_z \end{pmatrix} - \right. \\
&\quad \left. \frac{2\ \Delta\mathbf{x}\Delta\mathbf{v}}{\|\Delta\mathbf{x}\|^2} \cdot \begin{pmatrix} \Delta x_x \Delta x_x & \Delta x_y \Delta x_x & \Delta x_z \Delta x_x \\ \Delta x_x \Delta x_y & \Delta x_y \Delta x_y & \Delta x_z \Delta x_y \\ \Delta x_x \Delta x_z & \Delta x_y \Delta x_z & \Delta x_z \Delta x_z \end{pmatrix} \right) \qquad (4.55) \\
&= \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot \left(\Delta\mathbf{x}\Delta\mathbf{v} \cdot \mathcal{E}_{3\times 3} + \begin{pmatrix} \Delta x_x \\ \Delta x_y \\ \Delta x_z \end{pmatrix} \cdot \begin{pmatrix} \Delta v_x & \Delta v_y & \Delta v_z \end{pmatrix} - \right. \\
&\quad \left. \frac{2\ \Delta\mathbf{x}\Delta\mathbf{v}}{\|\Delta\mathbf{x}\|^2} \cdot \begin{pmatrix} \Delta x_x \\ \Delta x_y \\ \Delta x_z \end{pmatrix} \cdot \begin{pmatrix} \Delta x_x & \Delta x_y & \Delta x_z \end{pmatrix} \right) \qquad (4.56) \\
&= \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot \left(\Delta\mathbf{x}\Delta\mathbf{v} \cdot \mathcal{E}_{3\times 3} + \Delta\mathbf{x}\Delta\mathbf{y}^T - \frac{2\ \Delta\mathbf{x}\Delta\mathbf{v}}{\|\Delta\mathbf{x}\|^2} \cdot \Delta\mathbf{x}\Delta\mathbf{x}^T\right) \qquad (4.57)
\end{aligned}$$

4.2.1.6 Differentiation der Federdämpfung zur Geschwindigkeit: $\frac{\partial \mathbf{F_d}}{\partial \mathbf{v}}$

Die Basis zur Differentiation der Federdämpfung zur Geschwindigkeit ist Gleichung 4.41, die die Federdämpfung in Komponentenschreibweise dar-

stellt. Diese sei an dieser Stelle nochmals gegeben:

$$\mathbf{F_d} = k_d \cdot \begin{pmatrix} x_{2x} - x_{1x} \\ x_{2y} - x_{1y} \\ x_{2z} - x_{1z} \end{pmatrix} \cdot \frac{\Big((x_{2x} - x_{1x})(v_{2x} - v_{1x}) + (x_{2y} - x_{1y})(v_{2y} - v_{1y}) + (x_{2z} - x_{1z})(v_{2z} - v_{1z})\Big)}{(x_{2x} - x_{1x})^2 + (x_{2y} - x_{1y})^2 + (x_{2z} - x_{1z})^2} \tag{4.58}$$

Mit Hinblick auf die Bestimmung des Vektorgradienten werden im Folgenden die komponentenweise differenzierten Matrixelemente berechnet. So ergibt sich für $\frac{\partial F_{d_x}}{\partial v_{1x}}$:

$$\frac{\partial F_{dx}}{\partial v_{1x}} = \frac{k_d \cdot (x_{2x} - x_{1x}) \cdot (-x_{2x} + x_{1x})}{(x_{2x} - x_{1x})^2 + (x_{2y} - x_{1y})^2 + (x_{2z} - x_{1z})^2} \tag{4.59}$$

$$= \frac{-k_d \, \Delta x_x \, \Delta x_x}{\|\Delta \mathbf{x}\|^2} \tag{4.60}$$

Die anderen 8 Ableitungen ergeben sich entsprechend:

$$\frac{\partial F_{dy}}{\partial v_{1x}} = \frac{-k_d \, \Delta x_y \, \Delta x_x}{\|\Delta \mathbf{x}\|^2} \tag{4.61}$$

$$\frac{\partial F_{dz}}{\partial v_{1x}} = \frac{-k_d \, \Delta x_z \, \Delta x_x}{\|\Delta \mathbf{x}\|^2} \tag{4.62}$$

$$\frac{\partial F_{dx}}{\partial v_{1y}} = \frac{-k_d \, \Delta x_x \, \Delta x_y}{\|\Delta \mathbf{x}\|^2} \tag{4.63}$$

$$\frac{\partial F_{dy}}{\partial v_{1y}} = \frac{-k_d \, \Delta x_y \, \Delta x_y}{\|\Delta \mathbf{x}\|^2} \tag{4.64}$$

$$\frac{\partial F_{dz}}{\partial v_{1y}} = \frac{-k_d \, \Delta x_z \, \Delta x_y}{\|\Delta \mathbf{x}\|^2} \tag{4.65}$$

$$\frac{\partial F_{dx}}{\partial v_{1z}} = \frac{-k_d \, \Delta x_x \, \Delta x_z}{\|\Delta \mathbf{x}\|^2} \tag{4.66}$$

$$\frac{\partial F_{dy}}{\partial v_{1z}} = \frac{-k_d \, \Delta x_y \, \Delta x_z}{\|\Delta \mathbf{x}\|^2} \tag{4.67}$$

$$\frac{\partial F_{dz}}{\partial v_{1z}} = \frac{-k_d \, \Delta x_z \, \Delta x_z}{\|\Delta \mathbf{x}\|^2} \tag{4.68}$$

Zusammengefasst ergibt sich:

$$\frac{\partial \mathbf{F_d}}{\partial \mathbf{v}} = \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot \begin{pmatrix} \Delta x_x \Delta x_x & \Delta x_y \Delta x_x & \Delta x_z \Delta x_x \\ \Delta x_x \Delta x_y & \Delta x_y \Delta x_y & \Delta x_z \Delta x_y \\ \Delta x_x \Delta x_z & \Delta x_y \Delta x_z & \Delta x_z \Delta x_z \end{pmatrix} \tag{4.69}$$

$$= \frac{-k_d}{\|\Delta\mathbf{x}\|^2} \cdot \Delta\mathbf{x}\,\Delta\mathbf{x}^T \tag{4.70}$$

4.2.1.7 Globale Dämpfung/Reibungsverluste

Die globalen Reibungsverluste werden in diesem Modell durch die Stokes'sche Reibungskraft angenähert, die bei Bewegung eines Körpers durch Gase oder Flüssigkeiten entsteht (vgl. Kapitel 2.1.4). Die im selben Kapitel gegebene Gleichung wird auf den dreidimensionalen Fall erweitert:

$$\mathbf{F_r} = -k \cdot \dot{\mathbf{x}} = -k \cdot \mathbf{v} \tag{4.71}$$

Die Differentiation zum Ort und zur Geschwindigkeit können direkt bestimmt werden:

$$\frac{\partial \mathbf{F_r}}{\partial \mathbf{x}} = 0 \tag{4.72}$$

$$\frac{\partial \mathbf{F_r}}{\partial \mathbf{v}} = -k \cdot \mathcal{E}_{3\times 3} \tag{4.73}$$

4.2.2 Berechnung von $\Delta\mathbf{v}$

Die vorhergehenden Abschnitte beschreiben die Berechnung der wirkenden Kräfte und den zugehörigen Differentiationen. Diese wird für alle Partikel unter Ausnutzung der Symmetrieeigenschaften der Federkräfte durchgeführt und die wirkenden Kräfte werden pro Partikel addiert. Die Ergebnisse können in die implizite Euler-Gleichung eingesetzt werden (vgl. Kapitel 2.1.5.5):

$$\Big(\mathcal{E}_{3\times 3} - h\,m^{-1}\frac{\partial \mathbf{F}}{\partial \mathbf{v}} - h^2 m^{-1}\frac{\partial \mathbf{F}}{\partial \mathbf{x}}\Big)\Delta\mathbf{v} = h \cdot m^{-1} \cdot \Big(\mathbf{F} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}} \cdot h \cdot \mathbf{v}\Big) \tag{4.74}$$

Einsetzen der berechneten Kräfte und ihrer Differentiationen ergibt:

$$\mathcal{A} \cdot \Delta\mathbf{v} = \mathbf{b} \tag{4.75}$$

Dabei handelt es sich bei $\mathcal{A}$ um eine konstante $3n \times 3n$ Matrix und bei $\mathbf{b}$ und $\Delta\mathbf{v}$ um Vektoren der Dimension $3n$ (n: Anzahl der Partikel). Dieses spärlich besetzte Gleichungssystem kann mit der Konjugierten Gradientenmethode gelöst werden. Damit ist $\Delta\mathbf{v}$ bekannt und die Geschwindigkeiten nach dem nächsten Zeitschritt können mit $\mathbf{v}_h = \mathbf{v}_0 + \Delta\mathbf{v}$ bestimmt werden. Anhand der Geschwindigkeiten können nun auch die Positionen $\mathbf{x}_h$ nach dem nächsten Zeitschritt berechnet werden.

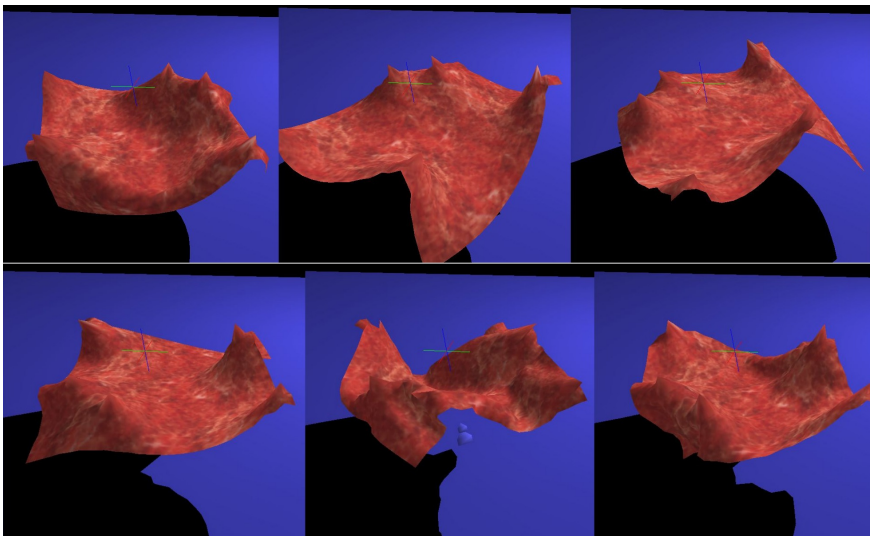

Abb. 4.3: Ein an 5 Constraint-Points hängendes Gewebe ($n = 25 \times 25$) wird durch eine kurzzeitige Rotation der Constraint-Points animiert. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

4.3 Postprocessing

In einem Postprozess werden dem oben beschriebenen physikalischen System zahlreiche weitere Eigenschaften zugewiesen, die in den folgenden Abschnitten beschrieben werden. So können gewisse Partikel statisch sein und sich deshalb nicht von der Stelle bewegen (Constraint-Points). Auch die Kollisionserkennung und -behandlung wird vollständig als Postprocess durchgeführt. Schließlich werden Effekte wie Wind oder Schneiden von Gewebe innerhalb des Postprozesses behandelt.

4.3.1 Constraint-Points

Für die Realisierung von Constraint-Points (siehe Abb. 4.3) gibt es prinzipiell zwei gleichberechtigte Ansätze, die jeweils für alle Constraint-Points einzeln durchgeführt werden:

- Nach der Berechnung der Positionen $\mathbf{x}_h$ wird die Position des Partikels i (Constraint-Point) $\mathbf{x}_h^{3i+t}$ $(t = 1, 2, 3)$ auf den ursprünglichen Wert $\mathbf{x}_0^{3i+t}$ gesetzt.
- Nach der Berechnung der Geschwindigkeiten $\mathbf{v}_h$ und vor der Berechnung der Positionen $\mathbf{x}_h$ wird die Geschwindigkeit des Partikels i (Constraint-Point) $\mathbf{v}_h^{3i+t}$ $(t = 1, 2, 3)$ auf den ursprünglichen Wert $\mathbf{v}_0^{3i+t}$ gesetzt. Diese entspricht bei einem statischen Partikel $\begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \frac{m}{s}$.

Der zweite Ansatz stellt gleichzeitig sicher, dass die Position des Constraint-Points unverändert und die Geschwindigkeit definiert ist. Aus diesem Grunde ist diesem Ansatz Vorzug zu gewähren, da beim ersten Ansatz undefinierte Geschwindigkeiten, die die Stabilität des Systems gefährden, auftreten können. Diesem Problem kann beseitigt werden, indem die Geschwindigkeiten anschließend noch angepasst werden.

Sollen sich die Constraint-Points auf vordefinierten Bahnen bewegen, so bietet sich ein hybrides Verfahren an, da vordefinierte Bahnen relativ aufwendig mit Hilfe von Geschwindigkeiten zu definieren sind. So wird der zweite Ansatz benutzt, um die Geschwindigkeiten zu behandeln. Anschließend werden die Positionen der Constraint-Points entsprechend der gewünschten Bahnbewegung verschoben.

4.3.2 Kollisionserkennung und -behandlung

Eine Kollisionserkennung und -behandlung für komplexe elastische Körper in Echtzeit durchzuführen ist wegen der hohen Komplexität ein nichttriviales Problem (vgl. Abschnitt 3.2.3 und 3.2.4). In dieser Arbeit wird eine sehr schnelle Kollisionserkennung zwischen Gewebe und Objekten benutzt, die in [FGL03] und [CH02] (vgl. Abschnitt 3.2.3.5) vorgestellt wurde. Dieser Ansatz wurde zur Erkennung von Gewebe-Gewebe Kollisionen prototypisch erweitert, ist aber dennoch zu langsam, um in Echtzeitumgebungen befriedigende Ergebnisse zu erzielen. Zudem besitzt die hier vorgestellte Methode zur Erkennung von Gewebe-Gewebe Kollisionen keine Allgemeingültigkeit, so dass nur in bestimmten Szenarien, wie z.B. bei einem auf den Boden fallenden Tuch, eine überzeugende Gewebe-Gewebe Kollisionserkennung und somit eine Kollisionsbehandlung stattfinden kann. In diesen Szenarien jedoch wird ein realistisch anmutendes Ergebnis erzielt.

4.3.2.1 Cloth-Object Kollisionen

Die Grundlage für die in dieser Arbeit verwendete Gewebe-Objekt Kollision besteht darin, die Partikel auf eine etwaige Kollision hin zu überprüfen. Existierende Kanten zwischen den Partikeln werden vernachlässigt. Dadurch wird der Aufwand erheblich gesenkt. Jedoch kommt es, je gröber das verwendete Mesh ist, zunehmend zu Problemen: Befindet sich ein Partikel innerhalb eines Objektes, so wird es auf die Oberfläche dieses Objektes versetzt (bzw. zuzüglich eines Δx Wertes: ein wenig über die Oberfläche). Bei Kugeln in Richtung der nächstgelegenen Oberflächennormalen und bei Quadern in Richtung der nächstgelegenen Oberfläche (siehe Abb. 4.4). Damit ist gewährleistet, dass kein Partikel sich innerhalb eines Objektes befindet.

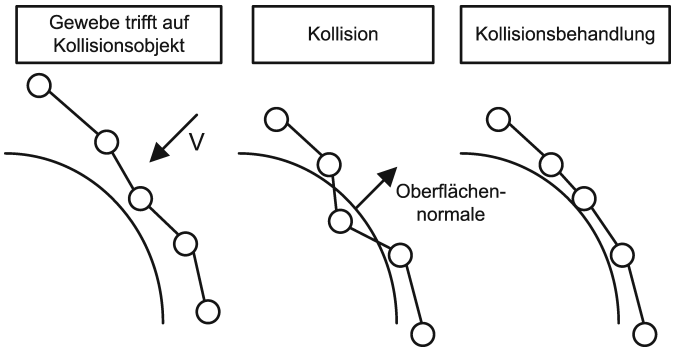

Abb. 4.4: Kollisionsbehandlung. Befindet sich ein Partikel des Feder-Masse Systems innerhalb eines Objektes, so wird es entlang der nächstgelegenen Oberflächennormalen auf die Oberfläche (bzw. leicht darüber) verschoben.

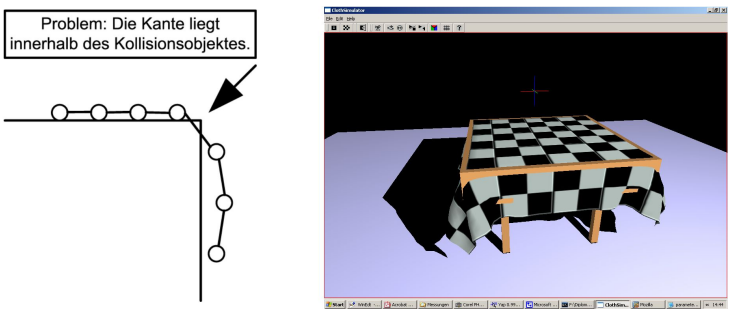

Abb. 4.5: Kantenproblem. Links: An scharfen Kollisionsobjektkanten oder sehr kleinen Kollisionsobjekten kann eine Kante des Gewebes innerhalb des Objektes liegen. Es kommt zu starken unerwünschten Artefakten (hier für den zweidimensionalen Fall dargestellt). Rechts: Beispiel eines Tisches mit Tischtuch ohne Beseitigung des Kantenproblems. Die Kanten des Tisches durchdringen das Tischtuch. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

Dies gilt jedoch nicht für die die Partikel verbindenden Kanten: Diese können sich sehr wohl innerhalb eines Objektes befinden - gerade bei scharfen Ecken und Kanten (z.B. Würfel) oder sehr kleinen Objekten kommt es zu starken unerwünschten Artefakten (siehe Abbildung 4.5).

Dieses Problem kann großenteils dadurch beseitigt werden, dass die Kollisionsobjekte ein wenig verdünnt gerendert werden (vgl. Abb. 4.6). Dabei sollte der Verkleinerungsfaktor derart gewählt werden, dass keine Kante das gerenderte, verschmälerte Objekt noch schneiden kann. Der Verkleinerungsfaktor ist proportional zur maximalen Kantenlänge (maximalen Federauslenkung), die an einer Kollision beteiligt ist. Bei zu kleinen Objekten bzw. zu groben Meshes werden etwaige Kollisionen gar nicht erkannt. Innerhalb des hier vorgestellten Modells bietet sich in diesem Fall an, das Mesh an den

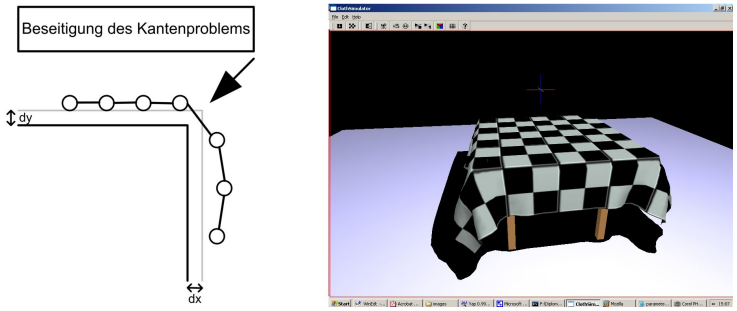

Abb. 4.6: Beseitigung des Kantenproblems. Links: Durch ein Verkleinern der gerenderten Kollisionsobjektes liegt keine Kante mehr innerhalb des Objektes. Rechts: Beispiel eines Tisches nach Beseitigung des Kantenproblems. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

möglichen Kollisionsstellen zu verfeinern oder die Kollisionsobjekte größer zu gestalten. Die Geschwindigkeit kann im Falle einer Kollision auf verschiedene Art modifiziert werden: Es können Reflektionseigenschaften definiert, die Geschwindigkeit kann auf Null gesetzt oder lediglich der Geschwindigkeitsanteil entgegengesetzt der Objektoberflächennormalenrichtung kann auf Null gesetzt werden. An dieser Stelle können auch eventuelle Haft- und Gleitreibungsverluste definiert werden. Das beschriebene Verfahren behandelt auch Kollisionen mit bewegten Objekten korrekt (siehe Abb. 4.7).

4.3.2.2 Cloth-Cloth Kollisionen

Die Gewebe-Gewebe Kollisionserkennung wird ähnlich der im vorhergehenden Abschnitt beschriebenen Gewebe-Objekt Kollisionserkennung durchgeführt: Alle nicht direkt benachbarten Partikelpärchen werden auf ihren Abstand hin überprüft. Unterschreitet dieser einen vorgegebenen Minimalwert, wird diese Situation als Kollision bewertet und behandelt. Die Partikel können sich nur bis zu einem vorgegebenen Minimalabstand annähern (vgl. Abbildung 4.8). Damit wird ein Durchstoßen des Gewebes mit sich selber verhindert und das Gewebe wird realitätsnäher simuliert.

Dieses Vorgehen führt jedoch im Zusammenhang mit Objektkollisionen oftmals nicht zu den gewünschten Ergebnissen, schlimmstenfalls zu einer Explosion des Systems. Ursache ist häufig das nachfolgende Gewebe, welches mit seiner Masse zu den kollidierenden Partikeln strebt und die Federn über die Explosionsgrenze hinaus spannt. Gute Ergebnisse werden jedoch bei zu Boden fallenden Geweben erzielt.

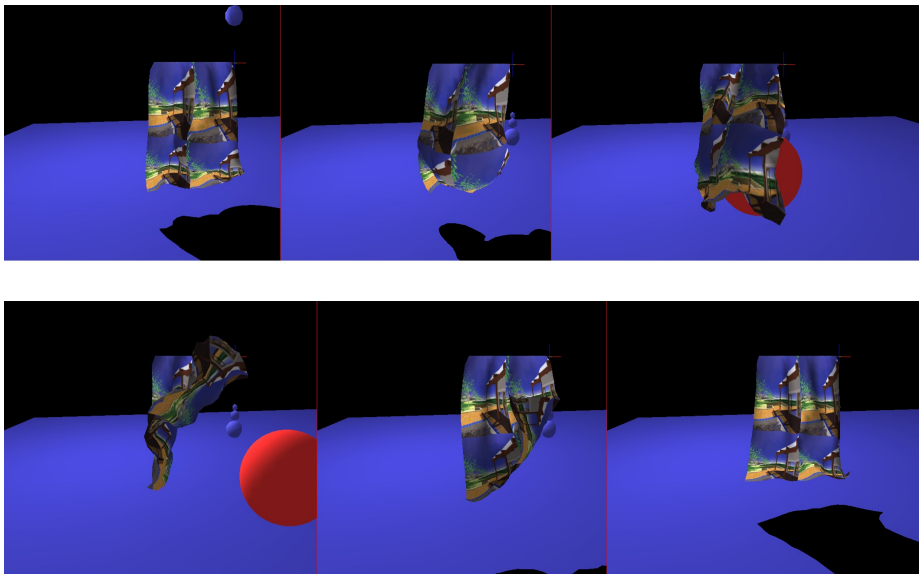

Abb. 4.7: Kollision mit einem bewegten Objekt: Eine Kugel bewegt sich durch ein hängendes Gewebe. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

4.3.3 Wind

Wind kann im Rahmen eines Postprozesses realisiert werden. In dieser Arbeit wird ein sehr einfaches Modell des Windes vorgestellt. Wirbel und Strömungsprofileigenschaften werden vernachlässigt. Es wird lediglich die Tatsache verwendet, dass die durch die Strömung wirkende Kraft proportional zu der senkrecht zur Strömungsrichtung stehenden Fläche ist.

In dieser Arbeit wird angenommen, dass die Fläche aller Dreiecke in etwa gleich groß ist. So kann jedem Partikel eine Geschwindigkeit in Richtung des Windes hinzugefügt werden, die von den Winkeln der an dieses Partikel angrenzenden Dreiecke zur Windrichtung abhängig ist. Allerdings erreicht das System schnell eine an die Windströmung angepasste statische Ausrichtung. Aus diesem Grund bietet es sich an, die Windrichtung oder die Windstärke zu variieren, um eine realistisch aussehende Bewegung des Gewebes im Wind zu erhalten (vgl. Abbildung 4.9).

4.3.4 Schnitte

Das vorliegende Gewebemodell von Partikeln und Federn bietet sich an, um Schnitte im Gewebe zu realisieren. So kann durch Entfernen von Verbindungsfedern, die das Gewebe zusammenhalten, ein Schnitt simuliert werden (vgl. Abb. 4.10).

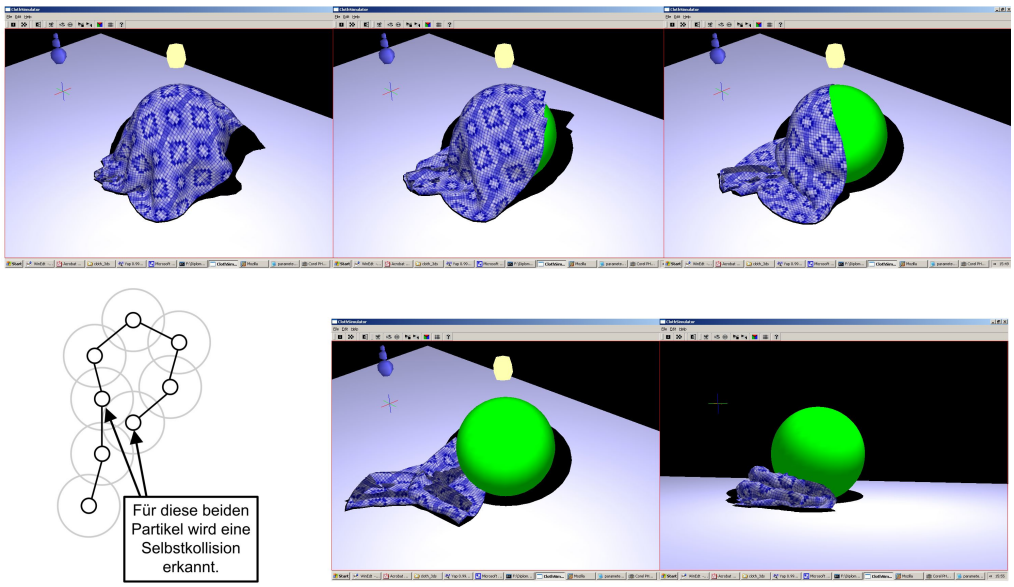

Abb. 4.8: Selbstkollision. Oben: Ein rundes Tuch (n=757) rutscht an einer Kugel ab. Unten links: Prinzip der Selbstkollision. Unten mitte: Endlage ohne Selbstkollisionserkennung. Unten rechts: Endlage mit Selbstkollisionserkennung. Gerendert mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

Das verbleibende Gewebe wird weiterhin korrekt simuliert, da lediglich weniger Federn im System vorhanden sind. Derartige Schnitte benötigen kaum Aufwand und können in interaktiven Umgebungen, wie z.B. mit einer virtuellen Schere oder einem virtuellen Skalpell, realisiert werden. Eine weitere Anwendung besteht darin, bei Überschreitung einer vorgegebenen Maximallänge einer Feder, diese zu entfernen. Damit kann das Reißen von Geweben bei zu intensiver Dehnung simuliert werden.

4.4 Verbesserung der Darstellungsqualität

Neben dem eigentlichen physikalischen System, welches die realistische Bewegung des Gewebes simuliert, wird der Realismus der Simulation hauptsächlich durch den Rendervorgang bestimmt. Man bedenke, dass mit den bisher vorgestellten Methoden lediglich ein Drahtgittermodell des Gewebes im Raume bewegt wird. In den folgenden drei Abschnitten werden Methoden zur realistischen Darstellung und Beleuchtung des Gewebes vorgestellt.

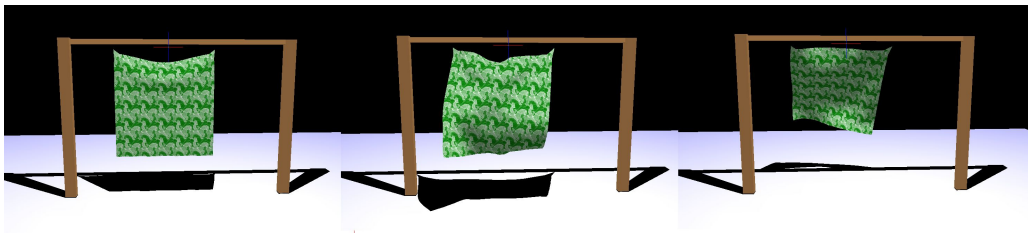

Abb. 4.9: Tuch im Wind. Links: Ohne Wind. Mitte und Rechts: Screenshot während der durch den Wind verursachten Bewegung des Tuches. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

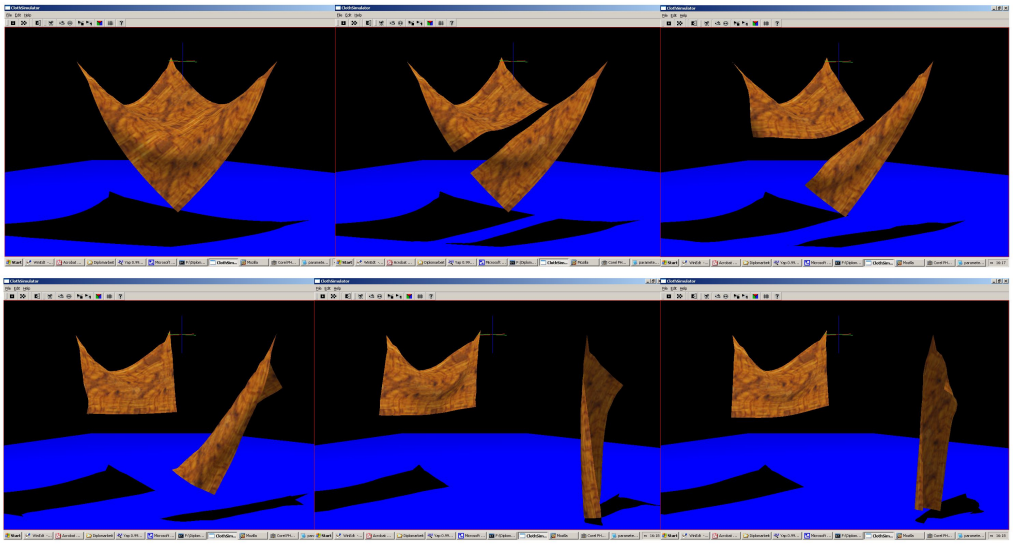

Abb. 4.10: Schneiden. Oben links: Ein an drei Constraints hängendes Tuch. Es wird ein Schnitt ausgeführt. Die beiden verbleibenden Tücher werden korrekt simuliert. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

4.4.1 Shading

Zur realistischen Darstellung des Gewebes ist ein Shading zur Darstellung von Beleuchtungseffekten aber auch Materialeigenschaften unabdingbar. Standardshadingmodelle wie Gouraud- oder Phong-Shading werden von aktueller Graphikkartenhardware unterstützt. Die Problematik der Beleuchtung eines extrem elastischen Körpers wie Gewebe liegt darin, dass die Oberflächennormalen sich permanent verändern. Aus diesem Grund ist eine komplette Neuberechnung der Oberflächennormalen vor jedem Rendervorgang durchzuführen. Im Rahmen dieser Arbeit wurde der folgende Ansatz gewählt (vgl. Abb. 4.11):

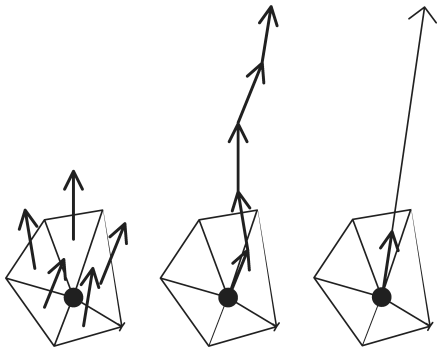

Abb. 4.11: Berechnung der Normalen. Die Normale eines Partikels (entspricht in diesem Fall einem Vertex) wird berechnet indem die Normalen angrenzender Dreiecke addiert werden und die Summe anschließend normiert wird.

1. Für alle Dreiecke werden die Dreiecksnormalen über ein Kreuzprodukt der Kanten mit anschließender Normierung und Orientierung[2] bestimmt.

2. Für jedes Partikel werden die Normalen der angrenzenden Dreiecke addiert und anschließend normiert. Es ergibt sich die Normale, die diesem Partikel zugeordnet ist.

Der große Nachteil dieses interpolierenden Ansatzes besteht darin, dass scharfe Kanten, wie etwa eine Tischdecke, die über eine Tischkante hängt, durch das Shadingmodell keine harten Reflektionssprünge besitzen, wie es in der Realität der Fall wäre (siehe Abb. 4.12). Die harten Kanten werden vom Beleuchtungsmodell wie abgerundet behandelt. Ein weiteres Problem besteht darin, dass durch die Interpolation der Normalen unter bestimmten Betrachtungswinkeln die Dreiecksstruktur des Gewebes erkennbar ist (vgl. Abb. 4.13). Dieses Problem kann durch eine andere Art der Normalenberechnung beseitigt werden. Dennoch erzielt diese Art der Normalenberechnung in der Regel realistisch anmutende Beleuchtungserscheinungen ohne stark wahrnehmbaren Artefakten und das erwähnte Kantenproblem ist in der Praxis häufig kaum wahrnehmbar oder kann durch eine feinere Meshunterteilung behoben werden.

Es sei auch erwähnt, dass der Aufwand zur Berechnung der Normalen nicht vernachlässigbar ist. Bei Simulationen von mehr als etwa 1000 Partikeln verringert die Normalenberechnung die maximal erreichbare Framerate erheblich. Es sind andere, schnellere Verfahrens zur Normalenberechnung

[2]Die Orientierung gewährleistet, dass alle Normalen von der Vorderseite des Gewebes ausgehend in die korrekte Richtung weisen.

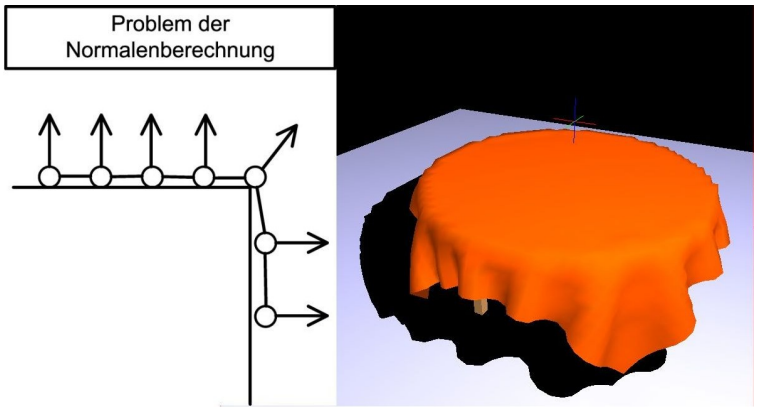

Abb. 4.12: Problem der vorgestellten Art der Normalenberechnung: Die Kanten werden aufgrund der interpolierten Normalen vom Shadingmodell wie abgerundet behandelt. Das Partikel auf der Kante würde zwei Normalen zur korrekten Beleuchtung benötigen. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

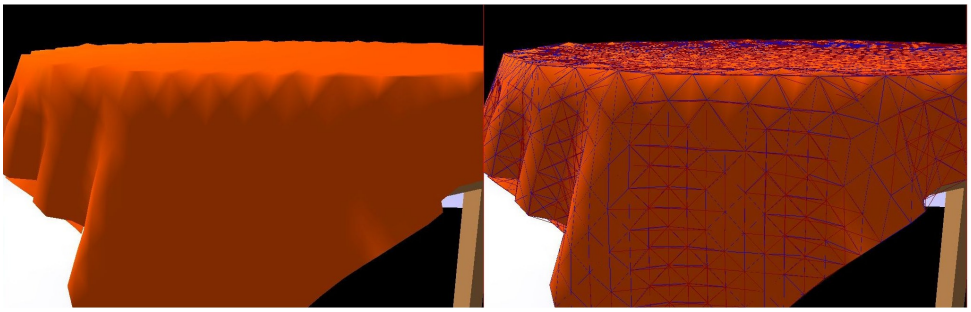

Abb. 4.13: Problem der vorgestellten Art der Normalenberechnung: An Kanten kann die Dreiecksstruktur des Meshes erkennbar werden.

denkbar. So ist es wegen der geringen Positionsveränderungen während eines Zeitschrittes z.B. oftmals nicht notwendig, dass alle Normalen pro Rendervorgang neu berechnet werden - eine Verteilung der Normalenberechnung auf mehrere Rendervorgänge würde die Performance verbessern.

4.4.2 Realismus

Für eine realistischere Darstellung des zu rendernden Gewebes bieten sich jegliche Echtzeitverfahren an, die auch in anderen Gebieten der Computergraphik zur Erhöhung des Realismus verwendet werden. Erwähnt seien Verfahren wie z.B. Texture-, Bump- und Environmentmapping. Aber auch Echtzeit-Schlagschattenverfahren (Projective Shadows, Shadowmapping oder Shadow Volumes) bieten sich zur Integration an. Zudem können die erweiterten

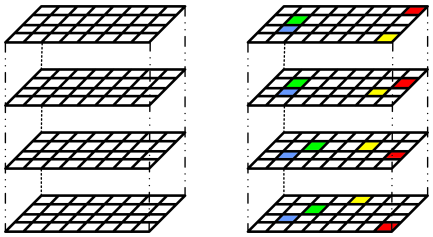

Abb. 4.14: Prinzip der Volumenvisualisierung anhand Texturen der Größe 8×4. Die Schichtung von Texturen erzeugt einen Voxelraum. In diesen kann gerendert werden. Im Beispiel rechts: 4 Linien verschiedener Farben (diese entsprechen den Fellhaaren).

Möglichkeiten moderner Graphikkarten genutzt werden, die vor allem mit den programmierbaren Vertex- und Fragmentshadern gegeben sind, um komplexere Beleuchtungsmodelle zu realisieren (vgl. [Cor03]). Mit Hilfe entsprechender Beleuchtungsmodelle wie dem Ashikhmin- oder dem Cook-Torrance-Beleuchtungsmodell können anisotrope Materialeigenschaften oder Reflektionseigenschaften, wie sie z.B. Seide besitzt, dargestellt werden.

Das im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator` unterstützt beispielhaft das Texturemapping und Projective Shadows auf die Grundebene (siehe dazu die Abbildungen auf den vorhergehenden Seiten).

4.4.3 Fell

Versuche, Verfahren aus der Volumenvisualisierung (Multi-Texture-Planes) zur Darstellung von Volumenausdehnungen, wie z.B. bei Wolle, zu realisieren, scheiterten im Rahmen dieser Arbeit. Die Volumenausdehnung von Wolle ist zu gering, als dass unter Verwendung des Tiefenbuffers realistische Ergebnisse erzielt werden können. Die entstandene Volumenvisualisierungseinheit wurde jedoch benutzt, um fellähliche Oberflächen darstellen zu können. Dabei wurde ein Verfahren genutzt, welches im DirectX 9.0 SDK anhand eines fellüberzogenen Würfels vorgestellt wurde.

Basis ist ein aus der Volumenvisualisierung bekanntes Verfahren: zahlreiche Texturen werden dicht übereinander dargestellt und repräsentieren den Voxel-Raum. Das darzustellende Volumen wird in den Voxelraum gerendert (siehe Abb. 4.14), wobei nicht genutzte Voxel einen Alphawert von 1.0 erhalten, so dass sie vollständig transparent sind. Für flüssige Farbverläufe und zur Reduzierung von Artefakten bei Rotationen bietet es sich an, auch anderen Voxeln eine gewisse Transparenz zuzuweisen. Die Anzahl und der verwendete Abstand der Volumen-Texturen bestimmen maßgeblich die Auflösung in z-Richtung. Zudem bestimmt die Anzahl auch in hohem Masse die maximal erreichbaren Frameraten.

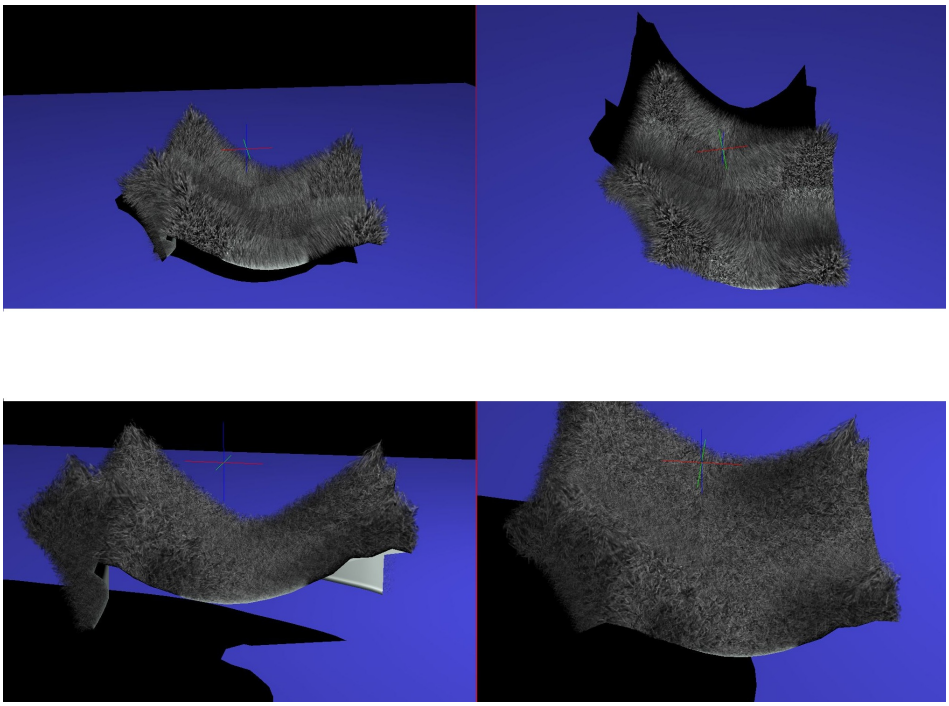

Abb. 4.15: Fell. Oben: strukturiertes, geordnetes Fell, Unten: chaotisch erscheinendes Fell. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

Bei einer Blickrichtung parallel zu den Volumen-Texturen sind die einzelnen Texturen erkennbar. Abhilfe schafft ein blickpunktabhängiges Umschalten der Richtung, in der die Texturen gezeichnet werden.

In den Voxelraum wird eine Anzahl von Linien unterschiedlicher Farbe und Orientierung gezeichnet (vgl. Abb. 4.14). In welchem Rahmen Farbe und Orientierung variieren bestimmt maßgeblich die Erscheinung des Fells: So kann ein chaotisch anmutendes Fell, aber auch ein gekämmt erscheinendes Fell erzeugt werden.

Ergebnisse des Fell-Algorithmus in Verbindung mit einer Gewebeanimation sind in Abb. 4.15 dargestellt. In der Simulation werden die Textureebenen entlang der jeweiligen Partikelnormalen verschoben. Der Vorteil dieses Verfahrens ist es, zahlreiche Linien darstellen zu können, viel mehr als die Graphikkarte eigentlich in Echtzeit darstellen könnte. Zudem kann das Textureblending benutzt werden, um die einzelnen Linien nicht allzu scharf wirken zu lassen. Dennoch besitzen die einzelnen Linien eine dreidimensionale Position im Raum, so dass Parallaxeneffekte dem Fell bei Rotation einen realistischen dreidimensionalen Anschein geben. Dieser realistische Effekt wird durch die große Beweglichkeit des simulierten Gewebes noch verstärkt.

Kapitel 5

Implementierung

In den folgenden Abschnitten wird die Umsetzung der in Kapitel 4 vorgestellten Konzepte diskutiert. Im Rahmen dieser Arbeit entstand zum einen das Programm `ClothSimulator`, welches die interaktive Echtzeit-Simulation von Kleidung und Geweben ermöglicht. Es werden die folgenden numerischen Integrationsmethoden, sowohl in der Standard-Implementierung als auch der Cromer-Variation unterstützt:

- Euler
- Midway
- Runge-Kutta 2. Ordnung
- Runge-Kutta 4. Ordnung
- implizites Euler-Verfahren in Verbindung mit der CG-Methode

Ziel der Implementierung ist es, die Umsetzbarkeit der verschiedenen Integrationsverfahren und vor allem des in Kapitel 4 entwickelten Konzeptes zur impliziten Kleidungssimulation und dessen Möglichkeiten in der Praxis sowie in Bezug auf die Performance bewerten zu können.

Des Weiteren ist im Rahmen dieser Arbeit das Programm `ClothEditor` entstanden, um nicht nur rein quadratische und homogene Gewebestrukturen simulieren zu können. Dieses Programm ermöglicht das einfache Erstellen von beliebigen Gewebeschnitten als inhomogenes Mesh, das Erzeugen von einfachen Kleidungsstücken, wie T-Shirts oder Hosen, aber auch das Erstellen von Rotationskörpern, um die Möglichkeiten der vorgestellten Konzepte auch über die Kleidungssimulation hinaus demonstrieren zu können.

In den anschließenden Abschnitten wird zunächst eine kurze Einführung in die verwendete Implementierungsumgebung und die Grundlagen der Implementierung gegeben. Darauf folgend werden die Implementierung und die bei einer Umsetzung auftretenden Probleme der in Kapitel 4 gegebenen Konzepte diskutiert.

5.1 Auswahl der Implementierungsumgebung

Die Implementierung der Programme `ClothSimulator` und `ClothEditor` erfolgte auf einem Pentium IV/2400MHz mit einer auf dem nVidia GeForce4 Ti4800 SE pro Chip basierenden Graphikkarte unter Windows XP Professional und VS C++ 6.0.

Als Graphikbibliothek kam OpenGL 1.4 unter Verwendung von Glut 1.7 zum Einsatz. OpenGL ist in der Wissenschaft die am weitesten verbreitete Lösung und wurde somit auch im Rahmen dieser Arbeit verwendet. Zudem ist OpenGL auf den verschiedensten Plattformen verfügbar und bietet damit z.B. einen entscheidenden Vorteil gegenüber DirectX. Aus dem gleichen Grunde wurde zur Erzeugung des GUI QT 3.0.0 verwendet. Ansonsten wurden ausschließlich Standardlibraries genutzt, die jedem C++ Compiler beiliegen. Damit sind die entwickelten Programme unabhängig von dem Betriebssystem Windows und können unter jedem Betriebssystem compiliert werden, unter dem OpenGL und QT verfügbar sind (Linux, Solaris, ...).

Die Parameterfiles werden mit einem einfachen, freien XML-Parameterfile-Parser eingelesen, der dem Source-Code beiliegt.

5.2 Grundlagen der Implementierung

Der Kern des Programmes `ClothSimulator` besteht aus drei Klassen:

- `AppWin`
- `OGLFrame`
- `Cloth`

`AppWin` initialisiert und behandelt das QT Widget zur Menüführung. `OGLFrame` initialisiert das OpenGL Widget und behandelt die innerhalb desselben emittierten Signale. Schließlich ist die Klasse `Cloth` für die Initalisierung, Simulation und Darstellung der Kleidung zuständig. Weitere

für den Kern des Programmes irrelevante Klassen wie etwa `fur` oder `rts_shadow` werden verwendet.

Während der Entwicklung von `ClothSimulator` wurde gerade bei der Klasse `Cloth` der Schwerpunkt auf eine geschwindigkeitsoptimierte Programmierweise gelegt. In den Integrations- und Kraftberechnungsmethoden wurde in der Regel auf die Verwendung von dynamischen Objekten verzichtet. Eine Testimplementierung, in der die Klasse `Cloth` rein statische Objekte benutzt, besitzt einen Geschwindigkeitsvorteil von etwa 10%. Es wurde aber dennoch auf eine rein statische Implemtierungsweise verzichtet, da aufgrund des Memory-Sharings ansonsten nur ein Cloth-Objekt zur gleichen Zeit erzeugt werden könnte.

Die ersten Versionen von `ClothSimulator` führten alle für die Simulation entscheidenden Rechnungen im `double` Bereich aus (8 Bytes, Mantisse: 52 Bits, Exponent: 11 Bits). Später wurden alle Rechnungen im `float` Bereich durchgeführt (4 Bytes, Mantisse: 23 Bits, Exponent: 8 Bits). Der Geschwindigkeitsvorteil liegt bei 33%. In der vorliegenden Version wird der zu verwendende Daten-Typ in `Cloth.hpp` definiert, so dass er vor der Compilierung festgelgt werden kann. Da die Positionen und Geschwindigkeiten der Partikel zum nächsten Zeitschritt lediglich von den aktuellen Positionen und Geschwindigkeiten abhängen, handelt es sich um eine selbstkorrigierende Simulation, in der numerische Fehler, die nicht zur Explosion führen, mit zunehmender simulierter Zeit wieder korrigiert werden. Aus diesem Grund nimmt die Stabilität des Systems bei einer Datentypänderung von `double` nach `float` auch nur unmerklich ab. Die numerischen Ungenauigkeiten, die zur Explosion führen, liegen nicht auf einer hinteren Nachkommastelle, sondern führen mit großen Zahlenwerten zu extremen Positions- oder Geschwindigkeitsveränderungen.

In einem Parameterfile, welches während des Initialisierungsvorganges des `Cloth`-Objektes geparst wird, werden zahlreiche Parameter für die Simulation gespeichert: das mit dem `ClothEditor` erzeugte, zu ladende Meshfile und zugehörige Transformationen bzw. die Parameter für ein reguläres, generiertes Gitter, physikalische Parameter und Faktoren, Texturen, Simulationsparameter, Informationen über Kollisionsobjekte und Parameter für Fell und Wind.

5.3 Datenstrukturen

Ein Vektor der Dimension n wird folgendermaßen gespeichert:

```
_CLOTH_DATA_TYPE_* vec;
vec = new _CLOTH_DATA_TYPE_ [n];
```

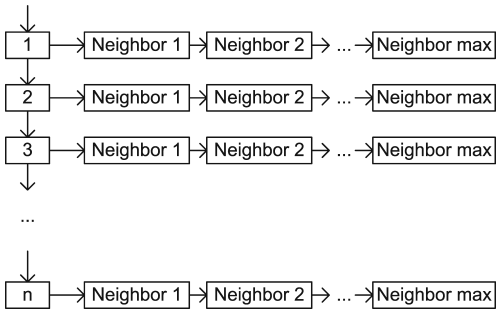

Abb. 5.1: Die verwendete Datenstruktur.

Die dreidimensionalen Positionen von n Partikeln werden in einem Vektor $\mathbf{p}$ der Größe $3 \cdot n$ gespeichert. Dabei entsprechen $\mathbf{p}_x^i$, $\mathbf{p}_y^i$ und $\mathbf{p}_z^i$ den Positionen `p[i*3+0]`, `p[i*3+1]` und `p[i*3+2]`.

Matrizen besitzen eine eigene Datenstruktur, die Wissen über die Topologie des Meshes ausnutzt, um den benötigten Speicher zu reduzieren. Es sei bedacht, dass ein kleines $30 \times 30 = 900$ Partikel großes quadratisches Mesh mit `double`-Präzision unkomprimiert bereits $(3 \cdot 30^2)^2 \cdot 8\,\text{Bytes} = 58319999\,\text{Bytes} \approx 55\,\text{MB}$ an Speicherplatz für einen einzigen Vektorgradienten benötigt. Bei einem nicht mehr in Echtzeit simulierbaren Gewebe mit 200×200 Partikeln wären es bereits 107 GB!

Da in der Regel lediglich nah beieinander liegende Partikel über Federn miteinander verbunden sind, handelt es sich bei allen Gradienten um spärlich besetzte Matrizen. Diese Eigenschaft wird derart ausgenutzt, dass pro Matrix-Zeile nur die Matrixelemente in einer Liste gespeichert werden, die von Null verschieden sind. Ein Maximalwert wird definiert, der die maximale Anzahl von Federn pro Partikel beschreibt. So wird dem Wissen Genüge getragen, dass es sich um spärlich besetzte Matrizen handelt. Auf diese Art werden die Matrix, die die Federbeziehungen aufnimmt, und die Matrizen A, $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ und $\frac{\partial \mathbf{F}}{\partial \mathbf{v}}$ (vgl. Kapitel 4) gespeichert (siehe dazu Abbildung 5.1).

Jedes Element der Datenstruktur ist eine Struktur, die Daten wie Federkonstante, Federdämpfung und Länge der ungedehnten Feder aufnimmt. Zudem speichert sie die Position des Nachbarn in der eigenen Nachbarliste, um bei späteren Rechnungen schnell die Symmetrieeigenschaften ausnutzen zu können - siehe dazu Abbildung 5.2.

Derart gespeichert benötigt ein $30 \times 30 = 900$ Partikel großes quadratisches Mesh mit `double`-Präzision und maximal 16 Federn pro Partikel $(3 \cdot 30^2) \cdot 16 \cdot 8\,\text{Bytes} < 0,4\,\text{MB}$ und ein 200×200 Partikel großes Mesh $< 15\,\text{MB}$ an Speicherplatz.

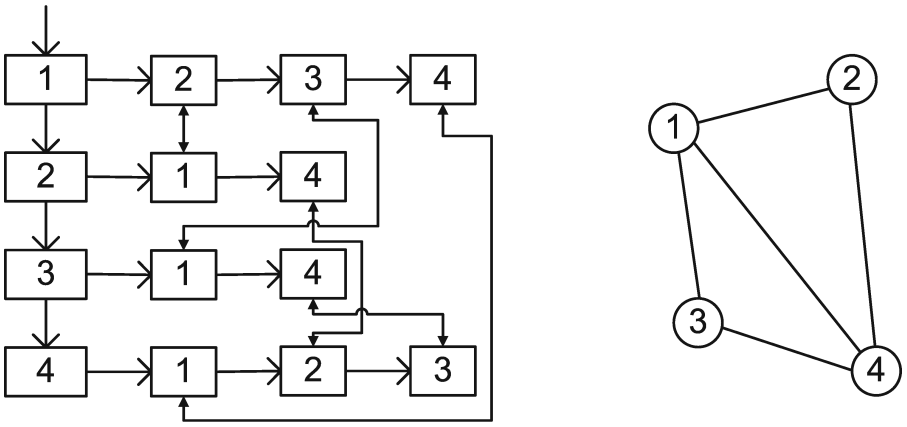

Abb. 5.2: Beispiel der verwendeten Datenstruktur für das rechts stehende Mesh.

5.4 Simulation

Es wurden die folgenden Integrationsverfahren jeweils in der Orginial- und in der Cromer-Variation implementiert:

- explizites Euler-Verfahren
- explizites Midway-Verfahren
- explizites Runge-Kutta-Verfahren 2. Ordnung
- explizites Runge-Kutta-Verfahren 4. Ordnung
- implizites Euler-Verfahren in Verbindung mit der CG-Methode

Die Implementierung der expliziten Integrationsverfahren entspricht dabei im Wesentlichen den in Abschnitt 2.1.5 gegebenen Beispielen. Die Implementierung zur Berechnung der wirkenden Kräfte und ihren Differentiationen des impliziten Euler-Verfahren entsprechen den in Kapitel 4 entwickelten mathematischen Gleichungen. Ein Beispiel ist zur Verdeutlichung der numerischen Programmierweise mit den im vorhergehenden Abschnitt beschriebenen Datenstrukturen an dieser Stelle gegeben. Die Berechnung der Differentiationen der Federkräfte zum Orte (vgl. Kapitel 4)

$$\frac{\partial \mathbf{F_F}}{\partial \mathbf{x_1}} = \frac{-D \cdot s_0}{c^{\frac{3}{2}}} \cdot \Delta\mathbf{x} \cdot (\Delta\mathbf{x})^T - D \cdot (1 - \frac{s_0}{c^{\frac{1}{2}}}) \cdot \mathcal{E}_{3\times 3} \tag{5.1}$$

$$c = \|\mathbf{x2} - \mathbf{x1}\|^2 \tag{5.2}$$

wird in der Implementierung folgendermaßen umgesetzt:

```
int y_iter;

// differentiation springforces position
for (int x=0; x<n; x++)                              // n: number of particles
  for (y_iter=0; y_iter<maximum_number_of_neighbors_per_spring; y_iter++)
                                                     // iterate through the neighbor list
    {
        // get the neighbor. now:  x: particle x1, y: particle x2
        int &y=m_massPointNeighbors[x][y_iter].neighbor;
        if (y!=(-1) && (y>x))                        // (y=-1): no more neighbors
                                                     // (y>x) : use the symmetrie
        {
            _CLOTH_DATA_TYPE_ x1[3]; _CLOTH_DATA_TYPE_ x2[3]; _CLOTH_DATA_TYPE_ xd[3];

            // get the positions
            x1[0]=(*v_x)[x*3+0]; x1[1]=(*v_x)[x*3+1]; x1[2]=(*v_x)[x*3+2];
            x2[0]=(*v_x)[y*3+0]; x2[1]=(*v_x)[y*3+1]; x2[2]=(*v_x)[y*3+2];

            _CLOTH_DATA_TYPE_ c=0.0;
            for (int i=0; i<3; i++) {
                xd[i]=x2[i]-x1[i];                   // calculate position difference x2-x1
                c+=xd[i]*xd[i]; }                    // square of distance, calculation of c

            // get length of unexpanded spring between x1 and x2
            _CLOTH_DATA_TYPE_ s0x1x2=m_massPointNeighbors[x][y_iter].s0;
            _CLOTH_DATA_TYPE_ c12=sqrt(c);           // c^(1/2) : distance
            _CLOTH_DATA_TYPE_ c32=sqrt(c*c*c);       // c^(3/2) : distance^3

            // gets D and mutiplies it with user-input factor
            _CLOTH_DATA_TYPE_ temp;
            temp=m_massPointNeighbors[x][y_iter].D;
            temp*=D_factor;

            _CLOTH_DATA_TYPE_ a=-temp*(1-(s0x1x2/c12));
            _CLOTH_DATA_TYPE_ b=-temp*s0x1x2/c32;

            for (int xx=0; xx<3; xx++)
              for (int yy=0; yy<3; yy++) {
                 // add a just for diagonal elements
                 if (xx==yy)
                    m_dfdx[x*3+xx][y_iter*3+yy]=a;
                 else
                    m_dfdx[x*3+xx][y_iter*3+yy]=0.;

                  m_dfdx[x*3+xx][y_iter*3+yy]+=b*xd[xx]*xd[yy];

                 // do the symetrie: F(x1x2)=-F(x2x1)
                 // "+" cause of -F and (x1-x0) instead of (x0-x1)!!
                 int neighbor_iter;
                 neighbor_iter=m_massPointNeighbors[x][y_iter].
                                this_particle_at_neighbor_iterator;
                 if (xx==yy)
                     m_dfdx[y*3+xx][neighbor_iter*3+yy]=a;
                 else
                     m_dfdx[y*3+xx][neighbor_iter*3+yy]=0.;

                 m_dfdx[y*3+xx][neighbor_iter*3+yy]+=b*xd[xx]*xd[yy];
               }
        }
        else // quit loop, no more neighbors
            if (y==-1) y_iter=maximum_number_of_neighbors_per_spring;
    }
```

Die Implementierung entspricht somit einer direkten numerischen Umsetzung der gegebenen Gleichung. Wie anhand des Source-Codes zu erkennen ist, wird die Symmetrie der Federn $F_{x_1x_2} = -F_{x_2x_1}$ zur Halbierung des Rechenaufwandes genutzt. Diese Eigenschaft wird, sofern möglich, bei allen Rechnungen in der Implementierung von `ClothSimulator` ausgenutzt.

Sobald $\Delta\mathbf{v}$ mit der entsprechenden Integrationsmethode bestimmt wurde, wird die Kollisionsbehandlung für die zukünftigen Positionen des Gewebes durchgeführt und die Geschwindigkeit im Falle einer Kollision angepasst. Anschließend werden die Positionen zum nächsten Zeitpunkt berechnet.

Ein Schnitt wird durchgeführt, indem alle Federn, die mit den betreffenden Partikel verbunden sind, aus der Datenstruktur gelöscht werden. Schließlich werden die betreffenden Partikel aus der Datenstruktur entfernt.

5.5 Kollisionserkennung und -behandlung

`ClothSimulator` unterstützt als einfache Kollisionsobjekte Kugeln, Quader, eine Bodenebene und eine kreisförmige Fläche. Diese Objekte stellen einen exemplarischen Charakter dar - es können beliebige Objekte hinzugefügt werden. Auf die Realisierung von komplexitätverringernden Ansätzen wie Bounding Volumes oder Raumunterteilungsverfahren wurde wegen der Einfachheit der Objekte verzichtet. Überprüft werden Partikelpositionen: Bei einer Kugel z.B. der Abstand zum Mittelpunkt. Unterschreitet der Abstand den Radius der Kugel liegt eine Kollision vor. Bei einem Quader wird anhand der acht Eckpunkte entschieden, ob sich ein Partikel innerhalb desselben befindet.

Wie in Abschnitt 4.3.2 beschrieben werden die Objekte verkleinert gerendert, um die unerwünschten Artefakte zu vermeiden, die durch das Überprüfen von den Partikeln (und nicht den Kanten) auf etwaige Kollisionen hin entstehen können. Des Weiteren wird zur Vermeidung von Artefakten ein Offsetwert mit dem Befehl `glPolygonOffset()` definiert. Dabei darf der verwendete Offsetwert nicht zu groß gewählt werden, da ansonsten an Stellen, an denen das Gewebe sich nahezu selber berührt, wieder Artefakte auftreten können. Dieser unerwünschte Effekt kann durch ein Sortieren der Gewebepolygone vor dem Rendern beseitigt werden.

Die Behandlung von Cloth-Object Kollisionen erfolgt geometrisch. Wie in Abschnitt 4.3.2 beschrieben, wird im Falle einer Kollision die Geschwindigkeit des betreffenden Partikels angepasst. An dieser Stelle können auch Haft- und Gleitreibungskräfte realisiert werden, die genau im Falle einer Kollision wirken. Bei der Implementierung von `ClothSimulator` wurde auf die Realisierung von Haft- und Gleitreibung verzichtet - diese können jedoch mit geringem Aufwand ergänzt werden.

Eine Cloth-Cloth Kollisionserkennung wurde lediglich exemplarisch und unoptimiert realisiert: Alle Partikelpärchen, ausgenommen unmittelbare Nachbarn, werden auf ihren Abstand hin überprüft. Unterschreitet dieser einen vorgegebenen Wert, so wird dieses Pärchen als kollidiert behandelt. Bei einer Kollision wird die Geschwindigkeit der betroffenen Partikel auf den Wert Null gesetzt. Diese Vorgehensweise liefert jedoch nur in speziellen Szenarien das gewünschte Ergebnis: Zu oft kommt es wegen nachbewegender Partikel zu einem Zustand, in dem sehr viele Partikel gleichzeitig kollidieren und somit allen Partikeln die Geschwindigkeit Null zugewiesen wird. Da sich alle beteiligten Partikel in diesem Moment nicht mehr bewegen, bleibt dieser Zustand erhalten und das Gewebe hängt wie an unsichtbaren Constraints frei im Raum. Gute Ergebnisse werden bei zu Boden fallenden Geweben und bei geringen Geschwindigkeiten erzielt. Jedoch ist der Aufwand der Vergleiche derart hoch, dass die Framerate um ein Vielfaches fällt und eine Echtzeit-Simulation nicht möglich ist. An dieser Stelle würde sich ein optimierendes Verfahren wie z.B. Bounding Volumes anbieten.

5.6 Verbesserung der Darstellungsqualität

Die für ein Shading erforderlichen Normalen werden wie in Abschitt 4.4.1 beschrieben berechnet. Das Mesh-Fileformat, welches `ClothEditor` schreibt, beinhaltet unter anderem eine Liste aller dem Gewebe zugehörigen Dreiecke in Form der drei zugehörigen Partikel. Somit kann über ein Kreuzprodukt und der Multiplikation eines Orientierungsfaktors die Normale für jedes Dreieck bestimmt werden. Für jedes Partikel werden alle Normalen der Dreiecke, von denen dieses Partikel ein Eckpunkt ist, addiert und anschließend normiert. Das Berechnen der Normalen ist ein in der Echtzeitsimulation nicht zu vernachlässigender Zeitfaktor (vgl. Kapitel 6).

`ClothSimulator` unterstützt projektive Schlagschatten von Kollisionsobjekten und dem Gewebe auf die Grundebene. Das Prinzip der projektiven Schlagschatten vernachlässsigt Schlagschatten von Objekten auf Objekte und Selbstschattierung und ist ein 2-Pass-Verfahren. Die Objekte werden im ersten Pass auf die Ebene projiziert und dort mit der entsprechenden Schattenfarbe oder aber dem ambienten Beleuchtungsanteil der Ebene gezeichnet. Im zweiten Pass werden dann die Objekte selbst gerendert. Der Projektionsursprung liegt an der Position der jeweiligen Punktlichtquelle und jedes zu zeichnende Polygon wird einzeln auf die Ebene projiziert. Kernstück des Algorithmus ist die Schattenprojektionsmatrix, deren Herleitung in [Cor03] zu finden ist.

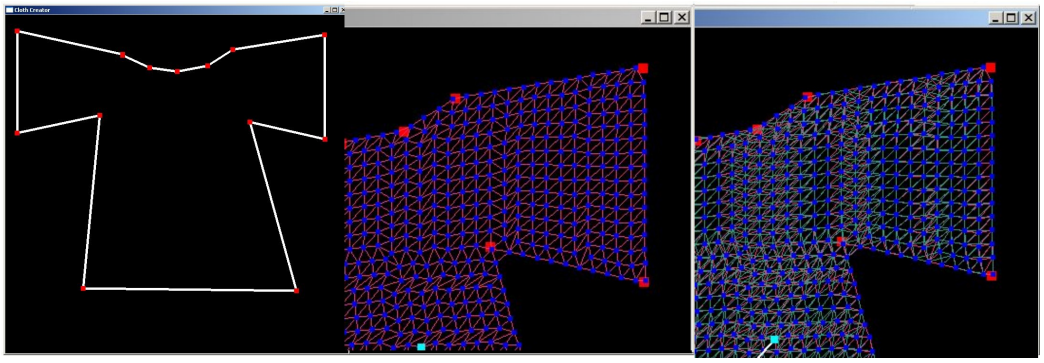

Abb. 5.3: Generierung eines inhomogenen Meshes mit `Clotheditor`. Links: Zunächst wird ein Polygonzug gezeichnet. Anschließend erfolgt die automatische Generierung des Meshes (mitte: stretch-forces, rechts: shear- und bend-forces, jeweils dargestellt mit Hilfe von Federn).

Der entwickelte Fellalgorithmus läuft folgendermaßen ab: Im Konstruktor wird zunächst der für den Voxelraum erforderliche Speicherplatz reserviert $(x \cdot y \cdot z \cdot 4(RGBA)\, Byte)$. Anschließend werden die einzelnen Fellhaare als Geraden in diesen Voxelraum gerendert. Die Position, Helligkeit, Länge und Orientierung der Geraden ist in einem vorgegebenen Rahmen zufällig. Die Geraden werden nicht mit einzelnen Punkten sondern mit einer Ausdehnung von 3×3 Punkten gerendert, wobei die Helligkeit nach folgender Matrix verringert wird:

$$\begin{pmatrix} 0,7 \cdot 0,7 & 0,7 & 0,7 \cdot 0,7 \\ 0,7 & 1 & 0,7 \\ 0,7 \cdot 0,7 & 0,7 & 0,7 \cdot 0,7 \end{pmatrix} \tag{5.3}$$

Dieses Vorgehen hat sich als Antialiasing in der Praxis bewährt. Auch empfiehlt es sich, allen Haaren einen gewissen Alphawert zuzuweisen - so erhält das Fell ein gewisses Volumen und die planare Schichtstruktur des Voxel-Volumens wird weniger ersichtlich. Anschließend werden die Volumendaten den verschiedenen Schichttexturen zugewiesen.

5.7 ClothEditor

Damit nicht nur reguläre, generierte Gewebe simuliert werden können, wurde das Programm `ClothEditor` (siehe Abb. 5.3) entwickelt. Mit diesem können beliebige planare Gewebeformen, planar-symmetrische Kleidungsstücke (wie z.B. einfache T-Shirts, Hosen und Kleider) und Rotationskörper erzeugt werden, wobei die Partikel, Schnittpunkte und Federverbindungen automatisch

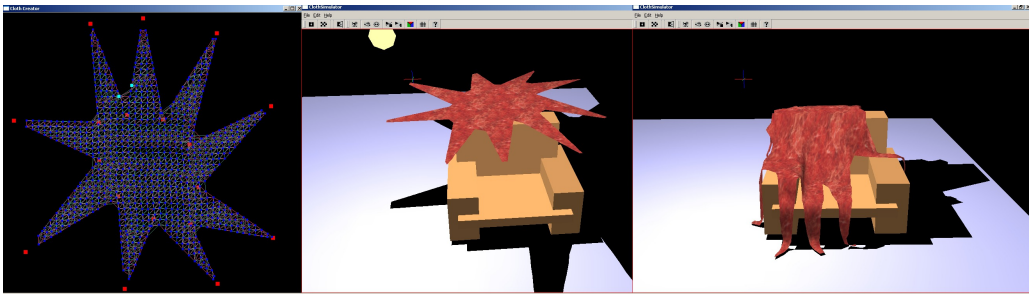

Abb. 5.4: Links: Eine planare Gewebeform wird mit `ClothEditor` erzeugt. Rechts: Simulation der erzeugten Gewebeform. Gerendert in Echtzeit.

generiert und einzeln nachbearbeitet werden können. Gespeichert werden diese in einem Mesh-File, welches die Positionen und Texturkoordinaten aller Partikel und deren Massen, die Federinformationen, Dreiecke und cut-points aufnimmt. Im Folgenden wird die Erzeugung der drei verschiedenen Gewebetypen beschrieben:

Eine planare Gewebeform (siehe Abb. 5.4) wird folgendermaßen generiert: Zunächst wird eine Liste von Eckpunkten des Gewebepolygons erzeugt. Es wird vorausgesetzt, dass keine Kante des Polygons eine andere schneidet. Anschließend wird das Polygon mit einem Mesh diskretisiert. Dazu wird ein homogenes Gitter über das Polygon gelegt. Ob ein Gitterpunkt innerhalb des Polygons liegt, wird danach entschieden, ob die Anzahl der Schnittpunkte einer horizontalen Geraden durch den Gitterpunkt mit dem Polygon rechtsseits und linksseits des Gitterpunktes ungerade ist. Danach werden die Meshpunkte der Polygonform angepasst: Dazu werden erst alle in einer Spalte und dann alle in einer Zeile liegenden Gitterpunkte, die zwei Nachbarn besitzen, derart angepasst, dass sie alle den gleichen Abstand besitzen und die, die nur einen Nachbarn besitzen, auf dem Polygon liegen (siehe Abb. 5.5). Die derart verschobenen Partikel besitzen eine Ordnung, da sie mit Hilfe eines regulären Gitters erzeugt wurden. Das mit Hilfe der Ordnung erzeugbare Mesh approximiert das vorgegebene Polygon ausreichend genau.

Abschließend werden die Federn und Dreiecke generiert. Dabei können die Federn entsprechend Abschnitt 2.2.1.1 erzeugt werden.

Die Koordinaten des planaren Gewebe-Meshes liegen im Bereich von $([0,1],[0,1])$. Die z-Koordinaten aller Partikel besitzen den Wert Null. Transformationen des Meshes werden im Parameterfile definiert. Dieses gilt auch für die beiden folgenden Gewebetypen.

Ein planar-symmetrisches Kleidungsstück wird folgendermaßen erstellt: Zunächst wird eine planare Gewebeform wie oben beschrieben erzeugt.

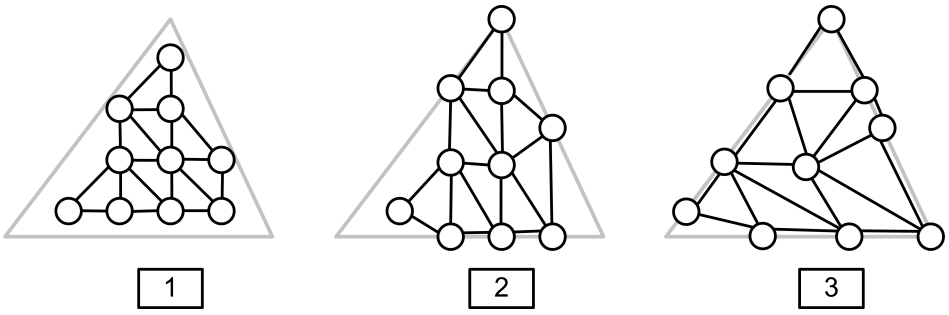

Abb. 5.5: Erzeugung eines inhomogenen Meshes, welches ein vorgegebenes Polygon approximiert: 1. Ein reguläres Gitter wird über den gewünschten Umriß gelegt. 2. Die Partikel werden vertikal derart verschoben, dass sie vertikal den gleichen Abstand besitzen und die Randpartikel auf dem Polygon liegen. 3. Das Gleiche für die horizontal liegenden Partikel.

Dieses Gewebe liegt vollständig in der durch die x- und y-Achse gegebenen Ebene. Nun wird dieses Gewebe verdoppelt, wobei die neuen Partikel lediglich eine andere z-Position besitzen: 1.0. Zu diesem Zeitpunkt existieren zwei ebene Stücken Gewebe, die in keiner Weise miteinander verbunden sind. Aus diesem Grund können Randpunkte selektiert werden, anhand derer Federn und Polygone zwischen diesen beiden einzelnen Gewebestücken erzeugt werden (jeweils zwei Randpunkte des in der xy-Ebene liegenden Gewebes werden selektiert - die zugehörigen zwei Punkte des kopierten Gewebes werden bestimmt und anhand dieser vier Punkte werden die Federn und Polygone generiert). Notwendige Transformationen aus dem Bereich $([0,1],[0,1],[0,1])$ heraus werden im Parameterfile definiert. Die Federn, die die Randpunkte verbinden, sind zu Beginn der Simulation stark gespannt (vgl. Abb. 5.6). Sie repräsentieren sozusagen die Nähte eines realen Kleidungsstückes. Zum „Bekleiden“ von Personen sollte zur Vermeidung von Explosionen zunächst eine kleine Schrittweite h gewählt werden. Diese Idee des „Bekleidens“ stammt aus [VM00b].

Ein Rotationskörper wird wie in der Computergraphik allgemein üblich erstellt: Ein Liste von Punkten (in diesem Fall aus dem Bereich ([0,1],[0,1],0)) wird um die y-Achse gedreht. Die Anzahl der Rotationsschritte kann frei gewählt werden. Die zugehörigen Federn und Dreiecke werden automatisch erzeugt.

Den drei unterschiedlichen Gewebetypen können in dem Programm `ClothEditor` noch zusätzlich Federn hinzugefügt bzw. entfernt werden. Zudem ist es möglich, eine vorgegebene Anzahl von Federn zufällig hinzuzufügen. Damit erhält das Modell neben der lokalen Steifigkeit, die durch die

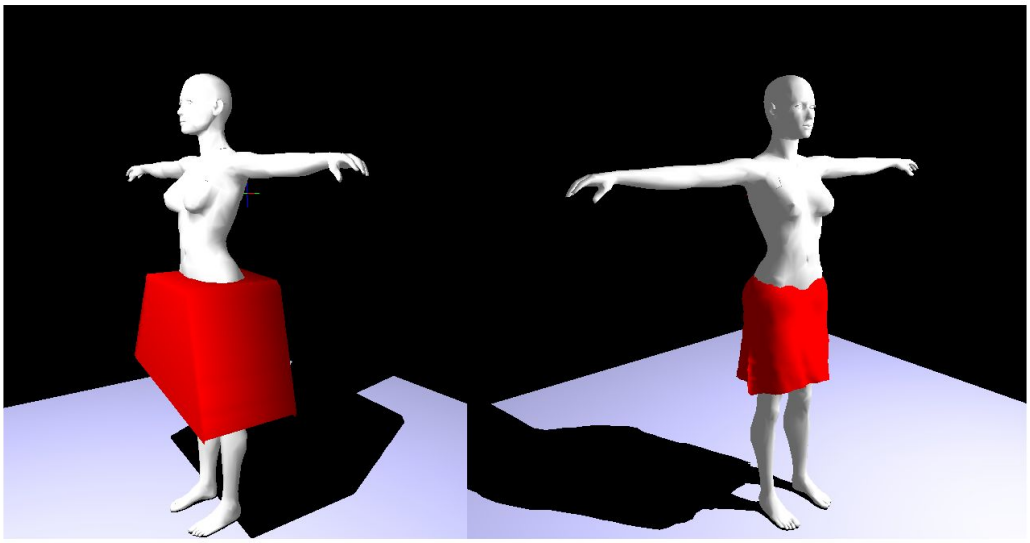

Abb. 5.6: Bekleiden mit einfachen Kleidungsstücken: Die Federn, die die Nähte repräsentieren, sind zu Beginn der Simulation stark gedehnt. Sie ziehen sich während der Simulation zusammen und bewirken ein Anpassen der Kleidung an den Kollisionskörper. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

automatische Erzeugung von Federn erreicht wird, zusätzlich eine globale Steifigkeit. Dieses Vorgehen bietet sich vor allem bei Rotationskörpern an, da diese ansonsten in sich zusammensinken würden.

Kapitel 6

Bewertung des entstandenen Verfahrens zur Gewebesimulation

Mit den im Rahmen dieser Arbeit vorgestellten Konzepten der impliziten Integration physikalisch basierter Kräfte und expliziten Integration mit der Euler-Cromer-Methode in Bezug auf Kleidungs- und Gewebesimulation bieten sich zahlreiche Möglichkeiten, Kleidungs- oder Gewebesimulation in Echtzeitumgebungen zu realisieren. Derartige Simulationen sind z.B. in heutigen Graphiktools und Computerspielen so gut wie gar nicht zu finden. Innerhalb von Anwendungen, die nicht der Beschränkungen der Echtzeitsimulation unterliegen, können sehr fein unterteilte Gewebestücke simuliert und animiert werden.

In diesem Kapitel werden zunächst die Möglichkeiten des entwickelten Konzeptes besprochen, dessen Realisierung in der Praxis mit dem im Rahmen dieser Arbeit entwickelten Progamm `ClothSimulator` gegeben ist. Anschließend erfolgt eine Darstellung und Diskussion von Performancemessungen und Vergleichen verschiedener Integrationsverfahren. An dieser Stelle erfolgt auch ein Vergleich mit bestehenden Implementierungen. Abschließend werden die Möglichkeiten des entwickelten Feder-Masse Simulators über die Kleidungssimulation hinaus behandelt.

In den folgenden Abschnitten werden unter anderem verschiedene Performancevergleiche vorgestellt und diskutiert. Damit kann eine quantitative Bewertung in verschiedenen Szenarien durchgeführt werden. Es sei jedoch bedacht, dass die Messwerte jeweils ausschließlich für die jeweilige Szene gelten. Eine Verallgemeinerung der gewonnenen Messwerte für beliebige Szenen ist nicht möglich, da die Simulation von zahlreichen Parametern und der Meshtopologie abhängt. Ein in der Topologie verändertes Mesh kann

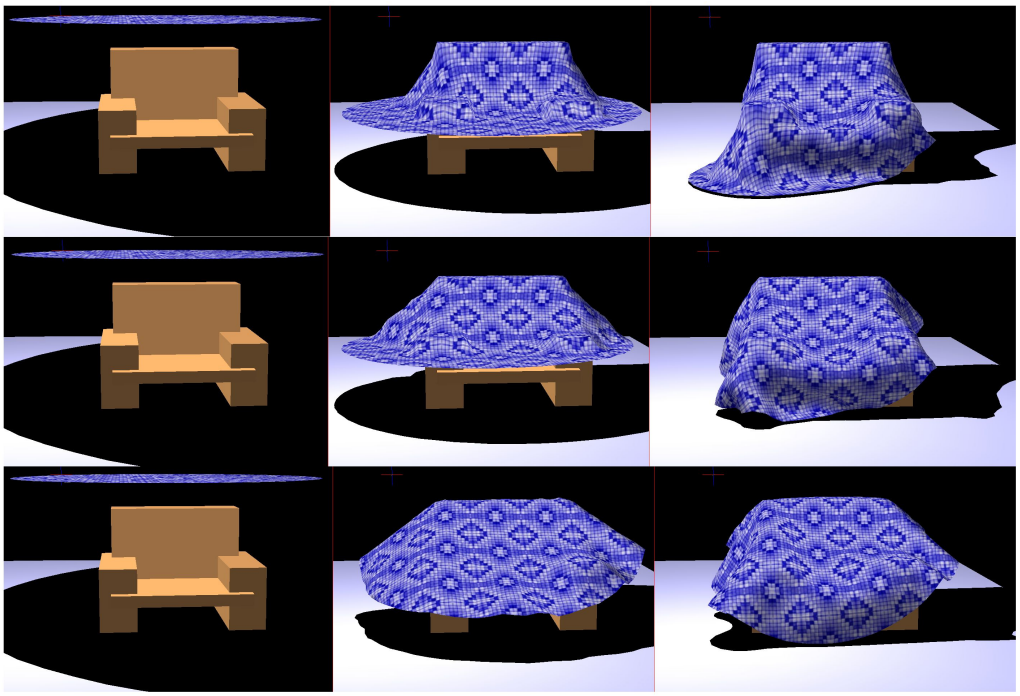

Abb. 6.1: Verschiedene Federkonstanten erzielen unterschiedliche Ergebnisse. Hier: Fallendes rundes Tuch bestehend aus einem inhomogenen Mesh (n=757). Oben: D=$1,5\frac{kg}{s^2}$, Mitte: D=$15\frac{kg}{s^2}$, Unten: D=$109,5\frac{kg}{s^2}$. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

z.B. einen anderen Maximalwert der Schrittweite h besitzen. Aus diesem Grund kann auch keine Vorhersage über optimale Parameter oder z.B. maximal mögliche Federkonstanten zur Simulation eines konkreten Meshes gegeben werden. Simulationsparameter müssen immer empirisch gewonnen werden. Die folgenden Abschnitte geben jedoch eine gute Übersicht und die gewonnen Messdaten können als Orientierungswerte angesehen werden, so dass eine stabile, überzeugende und realistisch anmutende Simulation durch die Variation der Parameter schnell erzielt werden kann.

Alle Messungen erfolgten auf einem Pentium 4 - 2400 MHz mit einer auf dem nVidia GeForce4 Ti4800 SE pro Chip basierenden Graphikkarte unter Windows XP, OpenGL 1.4, Glut 1.7 und QT 3.0.0, wobei in ein Widget der Auflösung 1024×768 gerendert wurde. Bei Zeitmessungen wurde außer zur Bestimmung der Normalenberechnungszeiten nicht gerendert, um die Simulationszeiten konkret vergleichen zu können und etwaige Messungenauigkeiten durch den Rendervorgang zu vermeiden. Aus dem gleichen Grund wurde sofern nicht anders angegeben auf Kollisionsobjekte

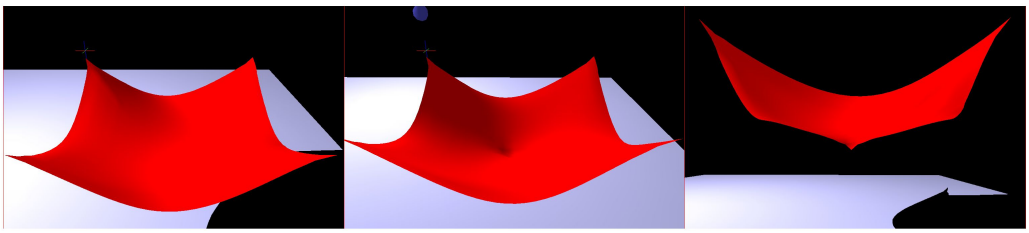

Abb. 6.2: Inhomogenitäten können durch Massenveränderung von Partikeln erzielt werden. Links: Homogenes Tuch. Mitte u. Rechts: Inhomogenes Tuch - das Partikel in der Mitte besitzt eine größere Masse als alle anderen. n=625. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

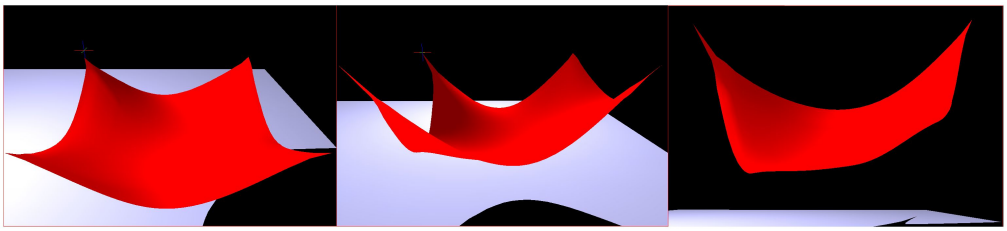

Abb. 6.3: Inhomogenitäten können durch Federkonstantenveränderung erzielt werden. Links: Homogenes Tuch. Mitte u. Rechts: Inhomogenes Tuch - die Federkonstanten wurden in x-Richtung linear vergrößert. n=625. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

verzichtet. Der Kollisionserkennungs- und -behandlungsprozess benötigt viel Rechenzeit und würde die Messungen verfälschen. Lediglich Constraint Points wurden in den Beispielszenen verwendet.

6.1 Möglichkeiten der Gewebesimulation

Die Feder-Masse Repräsentation eines Gewebes bietet zahlreiche Möglichkeiten um verschiedene Gewebe- oder Kleidungsformen zu simulieren. Prinzipiell kann jedes Gewebe- oder Kleidungsstück modelliert und simuliert werden. Die verwendete Anzahl von Partikeln und Federn ist massgeblich entscheidend für die Simulationsgeschwindigkeit aber gleichzeitig auch für die Darstellungsqualität. Gerade in Echtzeitumgebungen muss an dieser Stelle ein Kompromiss gefunden werden.

Durch die Wahl der Partikelmassen und -positionen, der Federn, der Federkonstanten und -dämpfungen können Materialeigenschaften definiert werden (vgl. Abb. 6.1). Auf diese Art können auch lokale Materialinhomogenitäten modelliert werden (siehe dazu Abb. 6.2, 6.3 und 6.4).

Anhand der Parameter der globalen Reibung, Gravitation und Windei-

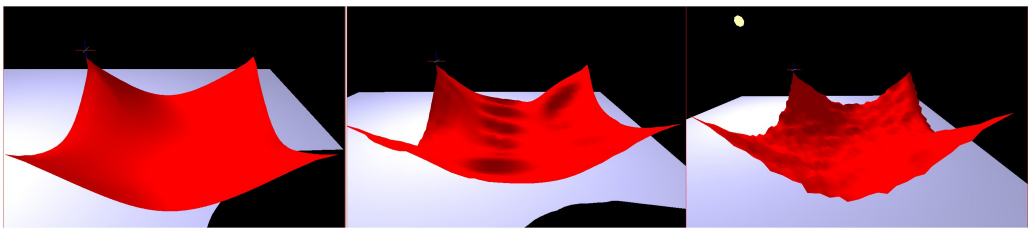

Abb. 6.4: Inhomogenitäten können durch Verändern der Ruhelänge von Federn erzielt werden. Links: Homogenes Tuch. Mitte u. Rechts: Inhomogenes Tuch - die Ruhelängen der Federn wurden variiert. So können zerknitterte Gewebe modelliert werden. n=625. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

genschaften kann das simulierte Gewebe global beeinflusst werden.

Gewebestücke bis zu einer Partikelanzahl von etwa 2500 Partikeln (explizites Euler-Cromer-Verfahren) bzw. etwa 1000 Partikeln (implizite Integration) können bei interaktiven Frameraten inklusive Normalenberechnung und Shading simuliert werden. Ohne Normalenberechnung können mit der expliziten Euler-Cromer-Methode bis zu 5000 Partikel in Echtzeit (20Hz) simuliert werden. In der Praxis werden realistische Ergebnisse erzeugt, wenn bei etwa zwei- bis fünffacher Echtzeit und einer hohen globalen Dämpfung simuliert wird. Dadurch schwingen die Federn weniger und die Modellierung des Gewebes mit Hilfe von Federn ist weniger offensichtlich. Dennoch besitzt das simulierte Gewebe oftmals noch eine ein wenig zu hoch anmutende Steifigkeit, die sich in einem Schwingungsverhalten ähnlich einem Ledertuch äußert. Dieser Effekt wird jedoch mit zunehmender Anzahl von Partikeln stark reduziert.

Die physikalischen Eigenschaften eines Gewebes, die sich z.B. im Schwingungsverhalten und der Dehnbarkeit äußern, werden mit dem Feder-Masse System realistisch anmutend und plausibel simuliert (siehe Abb. 6.5). Auch das Verhalten des Gewebes nach Schnitten durch Entfernen von Federn besitzt einen der Realität entsprechenden Anschein.

6.2 Performance

Wie in Abschnitt 5.6 erwähnt, beansprucht die im Rahmen dieser Arbeit umgesetzte Berechnung der Normalen eine große Zeitspanne. Bei zunehmender Partikelanzahl nimmt die Normalenberechnung eine die maximal erreichbare Framerate stark beeinflussende Stellung ein. In Tabelle 6.1 sind Messwerte gegeben, die sich auf die in Abbildung 6.6 dargestellte Szene beziehen. Die das Gewebe beschreibende Anzahl von Partikeln wurde verändert. Die für ein Shading notwendige Normalenberechnung beansprucht etwa ebensoviel

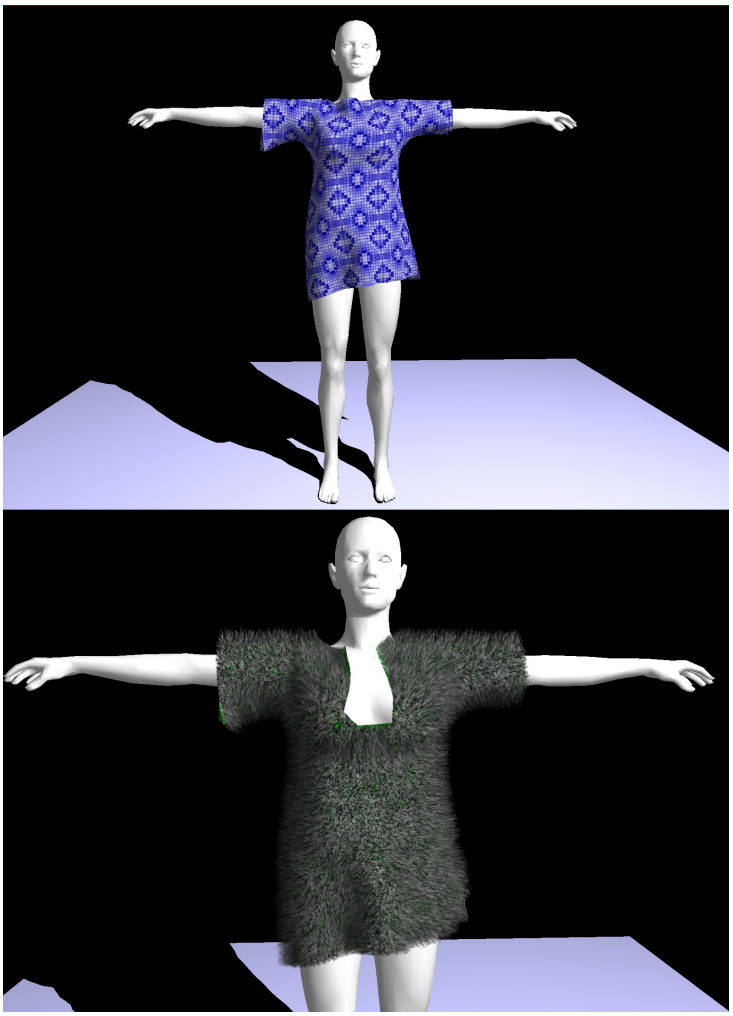

Abb. 6.5: Ein Kleid im Wind. Ohne Fell und mit Fell. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

Zeit wie ein Simulationsschritt mit der impliziten Integration, oder etwa das neunfache der Zeit wie ein Simulationsschritt mit der expliziten Integration.

Der eigentlich quadratische Aufwand der Berechnungszeiten wird aufgrund der Topologie des Meshes, welches eine Maximalanzahl von Federn pro Partikel vorgibt, auf einen linearen Aufwand verringert. Graphisch dargestellt sind die Berechnungszeiten in Abbildung 6.7. Der Aufwand der impliziten Integration mit der in Kapitel 4 entwickelten Berechnung der wirkenden Kräfte beansprucht etwa die 10-fache Zeit wie die explizite Euler-Cromer-Methode. Der mathematische und somit auch der numerische Aufwand ist bei der impliziten Integration wesentlich höher als bei der expliziten - damit ist die wesentlich höhere Berechnungszeit zu erklären.

n	Renderzeit R [s]	impl. Euler S_{iE} [s]	impl. Euler S_{iE}/R	expl. Euler-Cromer S_{eE} [s]	expl. Euler-Cromer S_{eE}/R
20^2	0,017	0,012	0,705	0,0022	0,129
30^2	0,026	0,027	1,038	0,0029	0,111
40^2	0,045	0,052	1,155	0,0054	0,120
50^2	0,071	0,080	1,126	0,0078	0,109
60^2	0,098	0,123	1,255	0,0107	0,109

Tabelle 6.1: Benötigte Simulations- und Renderzeiten pro Frame. S: benötigte Zeit für die Simulation, R: benötigte Zeit für den Rendervorgang, $h = 0.02\,s$, $D = 15\frac{kg}{s^2}$, $k_{global} = 0,01$, $k_d = 0,1$, $m = 0,05\,\text{kg}$, es wurden lediglich für die Echtzeitanimationen interessante Partikelanzahlen gemessen.

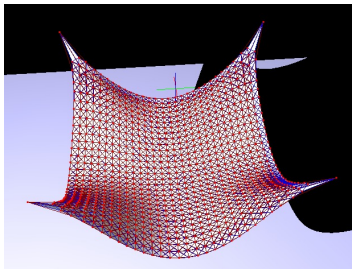

Abb. 6.6: Die zur Messung verwendete Szene 1. Die Parameter entsprechen den in Tabelle 6.1 gegebenen.

Messungen mit dem Profiler ergaben, dass etwa 60-70% der Berechnungszeit der impliziten Integration zum Lösen des Gleichungssystems mit dem Konjugierten Gradientenverfahren benötigt werden. Den größten Zeitanteil benötigt dabei die Matrizenmultiplikation während jedes Iterationsschrittes (vgl. dazu Abschnitt 2.1.6). Eine leichte Performanceverbesserung kann auch durch eine Variation der Parameter ϵ und i_{max} (max. Anzahl von Iterationen, vgl. Abschnitt 2.1.6) des Konjugierten Gradientenverfahrens erzielt werden. Die Präzision sinkt jedoch entsprechend. Zudem liegt die durchschnittliche Anzahl von Iterationsschritten bis zum Erreichen der gewünschten Genauigkeit $\epsilon = 0,0001$ bei nur etwa 2-4, so dass die Performance nur gering steigen kann. Für die in diesem Kapitel vorgestellten Messungen wurden folgende Werte benutzt: $\epsilon = 0,001$, $i_{max} = 50$.

Die großen Renderzeiten werden hauptsächlich durch die für ein Shading notwendige Normalenberechnung verursacht. So ergibt sich für eine 20 Hz Bildwiederholrate folgende maximal simulierbare Partikelanzahl (vgl. Abb. 6.7): Mit der impliziten Integration können etwa 1000 Partikel mit

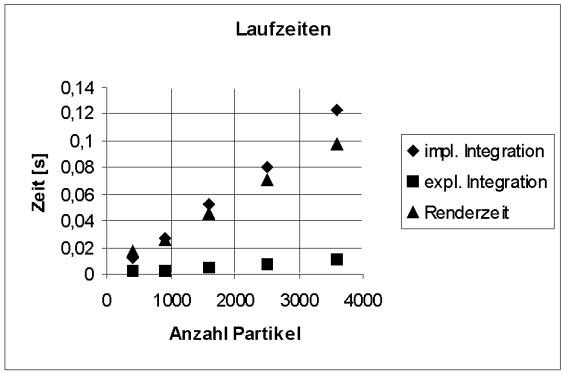

Abb. 6.7: Graphische Darstellung der Messwerte aus Tabelle 6.1.

0,025s Rechenzeit pro Zeitschritt simuliert werden. Der Rendervorgang beansprucht bei 1000 Partikeln ebenfalls etwa 0,025s. In der Summe ergeben sich 0,05s, was einer Bildwiederholrate von 20 Hz entspricht. Da Zeitschrittweiten von 0,05s mit der impliziten Integration möglich sind (vgl. die folgenden Abschnitte), können 1000 Partikel in Echtzeit mit 20 Hz simuliert werden. Wird auf ein Shading verzichtet, können bis zu etwa 1500 Partikel mit 20 Hz simuliert werden.

Die wesentlich schnellere explizite Euler-Cromer-Methode kann mit Shading bis zu etwa 1500 Partikel mit 20 Hz simulieren. Da die Renderzeit in diesem Fall wesentlich größer als die Integrationszeit ist, bietet es sich an, mehrere Simulationsschritte pro Rendervorgang durchzuführen. Dadurch kann eine Echtzeitsimulation erzielt werden. Dieses Vorgehen empfiehlt sich auch bei der impliziten Integration, falls weniger als etwa 800 Partikel simuliert werden sollen. Wird bei der expliziten Integration auf ein Shading verzichtet, können bis zu 5000 Partikel in Echtzeit simuliert werden, da die Zeitschrittweite für eine stabile Simulation nur wesentlich geringere Werte als bei der impliziten Integration annehmen darf.

Auf einen Vergleich mit dem Midway-, Runge-Kutta 2. Ordnung und Runge-Kutta 4. Ordnung wurde aus den im folgenden Abschnitt behandelten Gründen verzichtet. Sie ermöglichen kaum größere Zeitschrittweiten, benötigen jedoch die doppelte, bzw. das RK4-Verfahren die vierfache Rechenzeit wie das Euler-Cromer-Verfahren.

Für andere Szenen als die in Abb. 6.6 dargestellte ergeben sich nahezu die gleichen Werte. Es sei jedoch bedacht, dass zahlreiche Kollisionsobjekte die Maximalwerte der in Echtzeit simulierbaren Partikelanzahlen wesentlich verringern.

6.3 Möglichkeiten der verschiedenen Integrationsverfahren

In diesem Abschnitt werden die von `ClothSimulator` unterstützten Integrationsverfahren Euler, Midway, Runge-Kutta 2. und 4. Ordnung und impliztes Euler-Verfahren in der Praxis verglichen. Die Dämpfungskonstanten k_d und k_{global} sind in den folgenden Messungen konstant. Ein Vergrößern selbiger erhöht gleichzeitig die maximal möglichen Werte für die Schrittweite h oder D für alle Integrationsverfahren.

6.3.1 Maximale Schrittweiten

Die maximal mögliche Schrittweite ist die Schrittweite, bei deren Überschreitung das Partikelsystem wegen numerischer Ungenauigkeiten explodiert. Sie wird für verschiedene Szenarien bestimmt, alle anderen Parameter sind konstant.

In Tabelle 6.2 sind die Messdaten für die in Abbildung 6.6 dargestellte Szene aufgetragen.

Int.verf.	h_{max}	t/Simulationsschritt
expl. Euler	0,0030	0,030
Midway	0,0039	0,032
RK2	0,0070	0,032
RK4	0,0070	0,034
expl. Euler (Cromer)	0,0242	0,030
Midway (Cromer)	0,0242	0,032
RK2 (Cromer)	0,0242	0,032
RK4 (Cromer)	0,0260	0,034
impl. Euler	0,0570	0,050

Tabelle 6.2: Vergleich von Integrationsverfahren. $n = 30 \times 30$, $h = 0.02\,s$, $D = 15\frac{kg}{s^2}$, $k_{global} = 0,01$, $k_d = 0,1$, $m = 0,05\,\mathrm{kg}$,

Die Cromer-Methoden als stabilere Verfahren ermöglichen größere Schrittweiten bei dem gleichen numerischen Aufwand. Da die maximalen Schrittweiten des Midway-, Rk2- und RK4-Cromer-Verfahrens denen der Euler-Cromer-Methode nahezu entsprechen und dieses Ergebnis sich in der Praxis für zahlreiche Szenen ergeben hat, wird im Folgenden lediglich die Euler-Cromer-Methode als explizites Verfahren zum Vergleich herangezogen.

Der minimalen Zuwachs an Stabilität rechtfertigt kaum den zusätzlichen Aufwand der genannten Integrationsverfahren (doppelt, bzw. vierfach bei RK4) Die expliziten Verfahren scheinen an eine Stabilitätsgrenze zu stoßen, so dass höhere Ordnungen kaum noch einen Zuwachs an Stabilität erzielen.

Bei Kollisionen kann es vorkommen, dass die Integrationsverfahren höherer Ordnung ein wenig empfindlicher auf die Positionsveränderungen während der Kollisionsbehandlung reagieren als das Euler-Cromer-Verfahren.

Obwohl auf ein Shading verzichtet wurde, benötigt der Rendervorgang der Federn (dargestellt mit Linien) noch etwa 0,03s. Die benötigte Zeit der expliziten Verfahren liegt im Bereich von 0,002 bis 0,004s, die zur Simulation eines Integrationsschrittes benötigt wird.

In Tabelle 6.3 sind die Messwerte für die gleiche Szene, allerdings mit einem Mesh der Größe 50×50 aufgetragen. Das implizite Euler-Verfahren benötigt etwa die dreifache Zeit des expliziten Euler-Verfahrens, wobei es mehr als $\frac{1}{3}$ größere Zeitschritte zulässt.

Int.verf.	h_{max}	t/Simulationsschritt
expl. Euler	0,024	0,035
impl. Euler	0,039	0,106

Tabelle 6.3: Vergleich von Integrationsverfahren. $n = 50 \times 50$, $h = 0.02\,s$, $D = 15\frac{kg}{s^2}$, $k_{global} = 0,01$, $k_d = 0,1$, $m = 0,05\,\mathrm{kg}$

In Tabelle 6.4 sind die Messwerte für die in Abbildung 6.8 dargestellten Szenen aufgetragen.

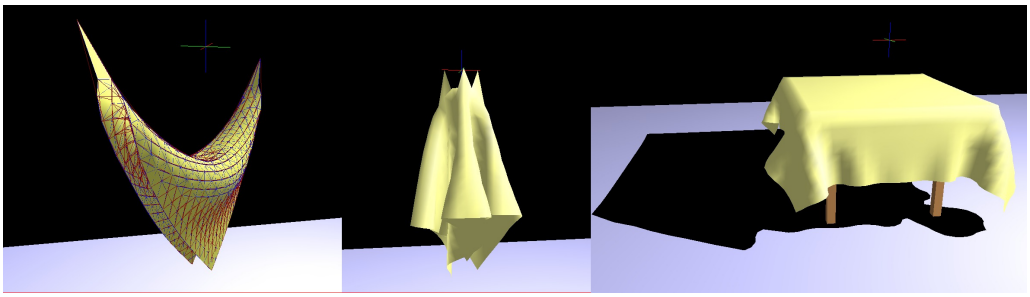

Abb. 6.8: Die zur Messung verwendeten Szenen 3-5. Szene 1 und 2 entsprechen Szene 1 des vorhergehenden Abschnittes mit den Partikelanzahlen 20×20 und 30×30. Die Parameter entsprechen den in Tabelle 6.4 gegebenen.

Das implizite Euler-Verfahren kann in einem Simulationsschritt größere Zeitschritte stabil simulieren. Wesentlich größere Zeitschritte als 0,05 s sind

Int.verf.	Szene 1 h_{max}	Szene 2 h_{max}	Szene 3 h_{max}	Szene 4 h_{max}	Szene 5 h_{max}
expl. Euler	0,0260	0,0242	0,0240	0,0342	0,0220
impl. Euler	0,0570	0,0570	0,0620	0,0472	0,0571

Tabelle 6.4: Vergleich von Integrationsverfahren. Die Parameter für Szene 1 und 2 entsprechen den in Tabelle 6.1 gegebenen der Partikelanzahlen 20×20 und 30×30. Szene 3: $n = 20 \times 20$, $D = 15\frac{kg}{s^2}$, $k_{global} = 0,01$, $k_d = 0,1$, $m = 0,05$ kg, Szene 4: $n = 30 \times 30$, $D = 15\frac{kg}{s^2}$, $k_{global} = 0,1$, $k_d = 0,1$, $m = 0,05$ kg, Szene 5: $n = 40 \times 40$, $D = 15\frac{kg}{s^2}$, $k_{global} = 0,15$, $k_d = 0,1$, $m = 0,05$ kg.

für die Echtzeitdarstellung nicht von großem Interesse, da die Framerate einer der Realität entsprechenden Simulation unter den Wert 20 fallen würde, so dass die Animation nicht mehr flüssig erscheint. Der große Nachteil der impliziten Integration im Vergleich der Berechnungszeiten mit dem expliziten Euler-Verfahren sei allerdings bedacht.

Überraschender als die hohe Stabilität der impliziten Integration[1] ist die relativ hohe Stabilität der expliziten Euler-Cromer-Methode[2], die es erlaubt relativ große Zeitschrittweiten h zu simulieren. Diese Messergebnisse decken sich nicht mit zahlreichen Veröffentlichungen die besagen, dass mit der Euler-Integration nur relativ kleine Zeitschritte stabil simuliert werden können (etwa eine Ordnung geringer - im Bereich $h \approx 0,003$). Da die explizite Euler-Cromer-Methode bis zu dem Faktor 10 schneller ist als die implizite Euler-Methode (siehe Abschnitt 6.2), können Federmassesysteme mit nicht zu steifen Federn ausgesprochen schnell simuliert werden. Damit ist die explizite Euler-Cromer-Methode attraktiv für Echtzeitumgebungen. Dieses Ergebnis wird in Abschnitt 6.3.5 genauer behandelt.

6.3.2 Maximale Federkonstanten

Die maximal möglichen Federkonstanten wurden bei einem konstanten $h = 0.02\,s$ ermittelt. Bei Überschreitung der maximalen Federkonstante explodiert das System. Die ermittelten Federkonstanten für die implizite und explizite Integration und die gleichen Szenen wie im vorhergehenden Abschnitt sind in Tabelle 6.5 gegeben.

[1] Diese ist unter anderem Motivation zur Einführung der impliziten Integration in die Kleidungssimulation gewesen.

[2] bei kleinen Federkonstanten - mit entsprechender Dämpfung kann jedoch eine überzeugende Gewebesimulation erreicht werden

Int.verf.	Szene 1 D_{max}	Szene 2 D_{max}	Szene 3 D_{max}	Szene 4 D_{max}	Szene 5 D_{max}
expl. Euler	28,5	28,5	28,5	69,0	27,0
impl. Euler	421,5	561,0	510,0	238,0	216,0

Tabelle 6.5: Vergleich der Federkonstanten. Parameter wie in Tabelle 6.3.

Welchen optischen Unterschied die verschiedenen Federkonstanten ausmachen ist beispielhaft für Szene 2 in Abbildung 6.9 dargestellt. Die implizite Euler-Integration erlaubt es, wesentlich steifere Gewebe zu simulieren, wodurch das Problem der Federhaftigkeit der Kleidung bei der expliziten Euler-Integration behoben wird. Das Gewebe wirkt realistischer. Dieses Ergebnis entspricht der Motivation zur Einführung der impliziten Integration in die Kleidungssimulation.

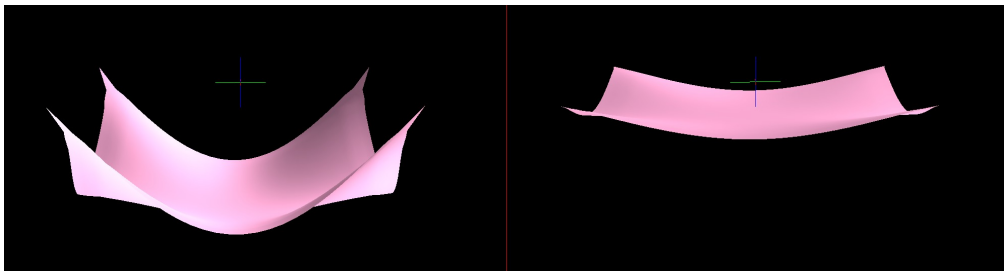

Abb. 6.9: Unterschied der expliziten und impliziten Integration in Bezug auf die maximalen Federkonstanten anhand von Szene 2. Links: Explizites Euler-Verfahren. Rechts: Impliztes Euler-Verfahren.

6.3.3 Statische Endpositionen

Aufgrund der numerischen Integration und der damit verbundenen Fehlerhaftigkeit der Integration ergeben sich für verschiedene Integrationsverfahren leicht unterschiedliche Bewegungen. Diese äußern sich z.B. in unterschiedlichen statischen Endpositionen des simulierten Gewebes. Sie liegen im sichtbaren Bereich und sind für ein Beispiel in Abbildung 6.10 dargestellt.

6.3.4 Nicht-Echtzeit Umgebungen

Ein hoher Grad von Realismus und Plausibilität in Bezug auf die Falleigenschaften eines Gewebes kann durch Verlassen der Echtzeit-Beschränkung

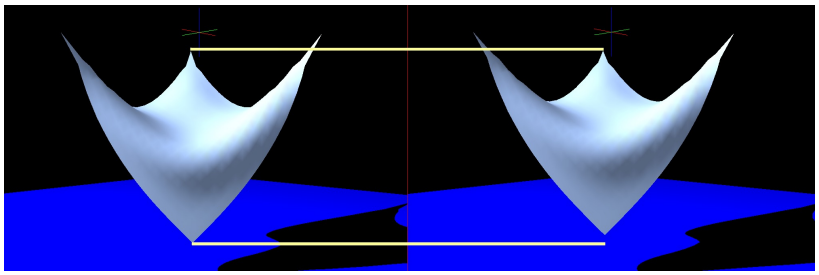

Abb. 6.10: Unterschied der expliziten und impliziten Integration in Bezug auf die statische Endposition anhand eines Beispiels. Links: Explizites Euler-Verfahren. Rechts: Impliztes Euler-Verfahren. Das implizite Euler-Verfahren simuliert bei gleichen Parametern ein steiferes Material.

erzielt werden: Der Anzahl der verwendeten Partikel und Federn ist theoretisch keine Grenze gesetzt und das Gewebe kann beliebig genau modelliert werden.

Es zeigt sich, dass das explizite Euler-Cromer-Verfahren bei sehr großen Meshes (etwa 15000 Partikel) wesentlich instabiler ist als das implizite Euler-Verfahren und nur bei sehr kleinen Simulationsschritten stabil ist. Im Bereich des Non-Realtime-Renderings lassen sich mit dem in dieser Arbeit vorgestellten Konzept der impliziten Kleidungssimulation mit physikalisch basierter Kräfteberechnung detaillierte Gewebesimulationen durchführen. In Abbildung 6.11 und Abbildung 6.12 sind Beispiele gegeben, die die erzielbare Qualität in Nicht-Echtzeitumgebungen veranschaulichen sollen. Zum Vergleich des simulierten mit dem realen Verhalten von Geweben ist Abbildung 6.13 gegeben. Obwohl das zur Simulation von Geweben benutzte Feder-Masse System nicht dem realen physikalischen Verhalten von Geweben entspricht, werden der Realität entsprechende Ergebnisse erzielt.

6.3.5 Vergleich mit bestehenden Implementierungen

Mit dem Programm `Clothy` (vgl. Abschnitt 3.2.6.1) kann ein 60×60 Partikel großes quadratisches Mesh mit $0,10 \frac{s}{\text{Frame}}$ gerendert werden, wobei eine Kugel als Kollisionsobjekt zur Verfügung steht. Dies entspricht $\frac{1}{10}$ Echtzeit (0,10 s Rechenzeit zur Simulation von 0,01 s Simulationszeit). Die Parameter sind dabei wie folgt gewählt: $h = 0,01\text{s}$, $D = 4.0 \frac{kg}{s^2}$, $k_d = 0,1$, $k_{global} = 0,04$, $10 \frac{\text{Iterationen}}{\text{Zeitschritt}}$. Während der Kollisionserkennung und -behandlung sinkt die Framerate erheblich ab. Gemessen wurde mit einem Rendern der Federn und Partikel (ein Shading wird von `Clothy` nicht unterstützt).

`ClothSimulator` benötigt für die gleiche Szene $0,04 \frac{s}{\text{Frame}}$. Dies entspricht $\frac{1}{4}$ Echtzeit mit 25 Frames pro Sekunde.

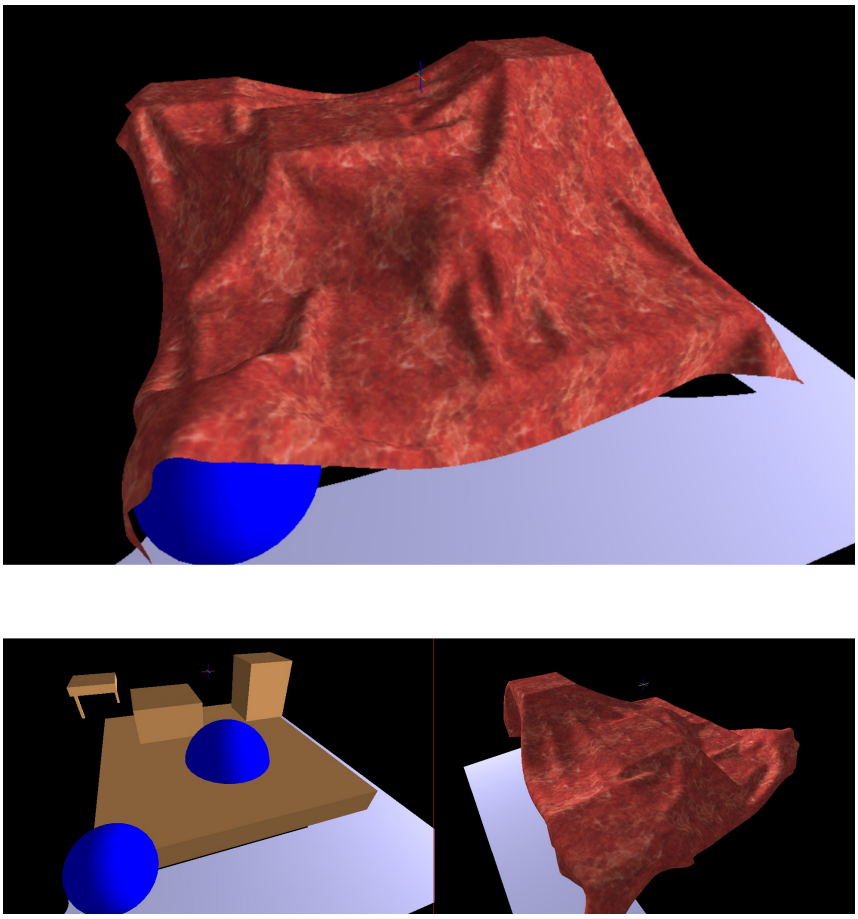

Abb. 6.11: Oben: Ein Tuch wird über die unten links dargestellte Szene gelegt. Unten rechts: Andere Perspektive. Meshgröße $n = 120 \times 120$. Berechnungszeit etwa 4 Minuten. Aufgrund der hohen Partikeldichte ergeben sich sehr feine Faltenwürfe. Gerendert mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`

Das Programm `Freecloth` ist die einzige freie Implementierung der impliziten Integration nach Baraff [BW98] (vgl. Abschnitt 3.2.6.2). Die folgenden Messungen wurden mit Texturing und Shading gemessen. Da die verwendeten Parameter - abgesehen von der Schrittweite h - wegen der unterschiedlichen Art der Kräfteberechnung nicht direkt vergleichbar sind, wurde versucht, die optischen Ergebnisse von `Freecloth` mit `ClothSimulator` nachzubilden (siehe Abb. 6.14).

Es wurden die von `Freecloth` bereitgestellten Kollisionsobjekte gemessen. Die Schrittweite beträgt bei allen Simulationen $h = 0,008$ s. Bei größeren Schrittweiten explodiert die Simulation von `Freecloth`. Die Ergebnisse sind in Tabelle 6.6 dargestellt. Diese großen Unterschiede der Be-

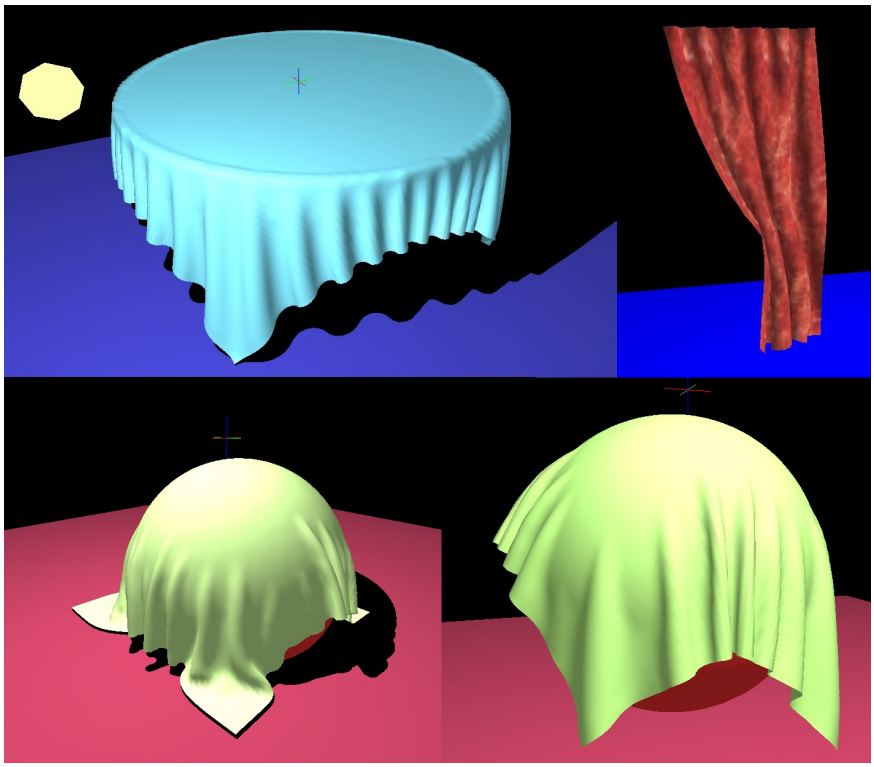

Abb. 6.12: Oben links: Tischdecke ($n = 120 \times 120$). Berechnungszeit etwa 1 Minute. Oben rechts: Gardine ($n = 50 \times 50$, ohne Shading in Echtzeit simulierbar). Unten links: Tuch über am Boden liegende Kugel (Renderzeit: etwa 1 Minute). Unten rechts: Tuch über schwebende Kugel (Renderzeit: etwa 1 Minute). Gerendert mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

rechnungszeiten können nicht erklärt werden. Die Tatsache, dass `Freecloth` bei Schrittweiten über $0,08$ s explodiert, ist ebenfalls schwerlich zu erklären: Eventuell wurde die implizite Integration nicht nach der Cromer-Methode realisiert oder die Verwendung von physikalisch basierten Reibungskräften geben den Simulationen von `ClothSimulator` eine hohe Stabilität. Die Zeitunterschiede von `Freecloth` bei den verschiedenen Kollisionsobjekten dürften eigentlich nicht auftreten, da `Freecloth` keine Kollisionserkennung durchführt, sondern die Objekte lediglich durch Constraintspunkte (alle Partikel, die auf der Oberfläche des Objektes liegen) darstellt.

Ein quantitativer Vergleich von impliziten Simulationen mit `ClothSimulator` und den von D. Baraff in [BW98] vorgestellten und den jeweilig unterschiedlichen Arten der Kräfteberechnungen ist nicht möglich, da kein Prozessor angegeben ist, mit dem die Simulationen durchgeführt wurden. Die Simulationszeiten lagen 1998 bei einer Tischdecke mit 51×51 Partikeln bei 2,23

Abb. 6.13: Ein Tuch wird über einen Sessel geworfen. Oben: Foto (Die Decke wurde der Simulation entsprechend über den Sessel geworfen und nicht mehr verändert). Unten: Mit `ClothSimulator` gerendertes Beispiel ($n = 50 \times 50$).

Programm	Szene 1 t/Frame [s]	Szene 2 t/Frame [s]	Szene 3 t/Frame [s]
`Freecloth`	0,6	1,2	1,7
`ClothSimulator`	0,046	0,04	0,042

Tabelle 6.6: Vergleich: `Freecloth` und `ClothSimulator`. Verglichen wurden die in Abbildung 6.14 dargestellten Szenen mit jeweils 30×30 Partikeln.

s/Frame und einem Shirt mit 6450 Partikeln bei 14,5 s/Frame - inklusive Cloth-Cloth-Kollisionserkennung, die 18,3% bzw. 46,1% der Rechenzeit beanspruchte. Bei Simulationen mit Menschen als Kollisionsobjekte wurde die Schrittweite h kaum über den Wert 0,02s gesetzt.

Ein qualitativer Vergleich der Ergebnisse von Baraff mit denen von `ClothSimulator` ist in Abbildung 6.15 dargestellt. Auch hier wurde versucht, die von Baraff gegebenen einfachen Beispiele mit Hilfe von `ClothSimulator` nachzumodellieren bzw. ein vergleichbares Ergebnis zu erzielen.

Die unterschiedlichen Arten der Darstellung der drei Kräfte stretch-, bend- und shear-forces, die unterschiedliche Art der Kräfte- und Differentationenberechnung und die unterschiedliche Art der Kollisionserkennung führen zu vergleichbaren Ergebnissen. `ClothSimulator` benötigt für die dar-

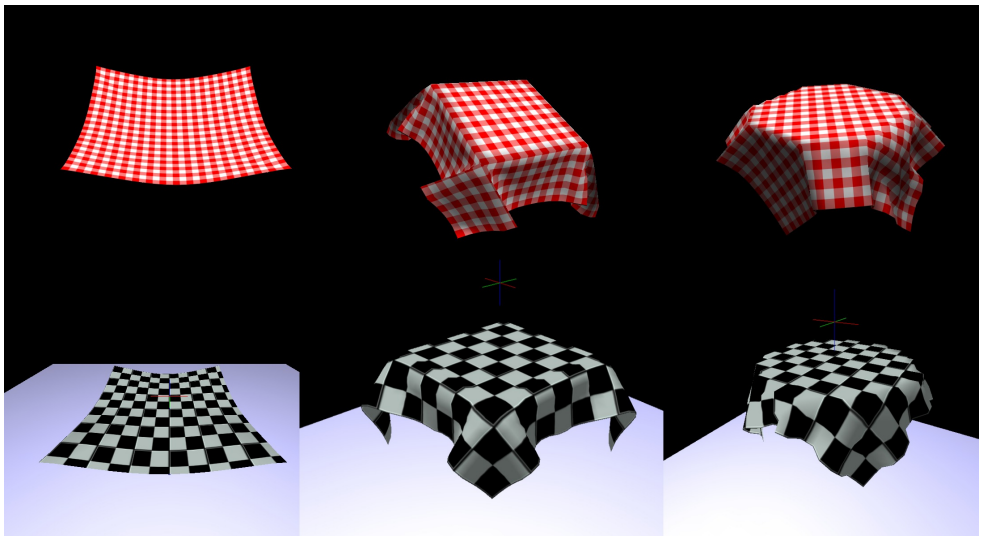

Abb. 6.14: Die für den Performancevergleich von `Freecloth` und `ClothSimulator` genutzten Szenen. Die von `Freecloth` unterstützten Constraints wurden mit Constraints und Kollisionsobjekten in `ClothSimulator` nachgestellt. Mesh-Größe: $n = 30 \times 30$. Oben: Screenshot `Freecloth`. Unten: Screenshot `ClothSimulator`.

gestellte Szenen 0,088s pro Frame und unterstützt eine maximale Schrittweite von $h = 0,042$s mit der impliziten Integration.

Die Simulationsergebnisse gehen bei ähnlichen Federkonstanten und Simulationsparametern mit denen aus [VM01] (vgl. Abschnitt 3.2.5) konform. Es sei jedoch erwähnt, dass mit der expliziten Euler-Cromer-Methode trotz verhältnismäßig kleiner Federkonstanten bei hoher Dämpfung und zahlreichen Simulationsschritten pro Frame ein ebenfalls realistisch anmutendes Ergebnis in Bezug auf das fallende Kleidungsstück erzielt wird - bei gleichzeitig etwa 10-facher Geschwindigkeit. Somit gilt das Ergebnis aus [VM01] (siehe Abschnitt 3.2.5), dass die implizite Integration mit großen Schrittweiten schneller ist als die explizite, nicht für kleine Federkonstanten in Verbindung mit einer hohen Dämpfung.

6.4 Möglichkeiten neben der Gewebesimulation

Die im Rahmen dieser Arbeit vorgestellten Konzepte zur Kleidungssimulation basieren auf der Verwendung von Feder-Masse-Systemen. Diese können für zahlreiche Anwendungen neben der Gewebesimulation genutzt werden. Einige Beispiele wurden realisiert.

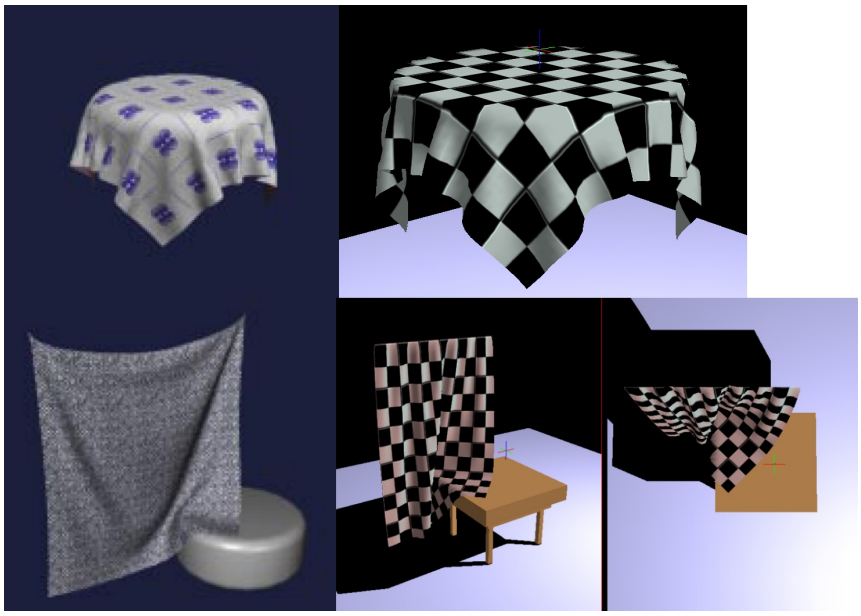

Abb. 6.15: Vergleich der Ergebnisse von Baraff und `ClothSimulator`. Links: Ergebnisse von Baraff (aus [BW98]). Rechts: Screenshot `ClothSimulator` (gerendert in Echtzeit).

Werden z.B. die Partikel nicht eben, sondern im Raum angeordnet und mit Federn verbunden, so wird dieses System gleich einem elastischen Volumenkörper simuliert. Physikalische Effekte wie Trägheit oder Drehimpulserhaltung werden durch die Kraftübertragung der Federn und Massen der Partikel angenähert. Steifere Federn erhöhen dabei die Steifigkeit des Volumenkörpers und damit den Grad der Annäherung. Die implizite Integration erhöht die maximal mögliche Steifigkeit im Vergleich zu expliziten Verfahren erheblich. So können nahezu starre Körper simuliert werden. Zwei Beispiele für Rotationskörper, deren Erzeugung von `ClothEditor` unterstützt wird, sind in Abbildung 6.16 und 6.17 gegeben. Gute Ergebnisse werden mit der Simulation elastischer Körper, etwa aus Gummi bestehend, erzielt.

Auch die Oberfläche von Fluiden kann relativ überzeugend mit Hilfe von Feder-Masse Systemen simuliert werden (siehe Abb. 6.18). Vor allem zähe Flüssigkeiten wie etwa flüssiger Wachs oder Lava können realistisch anmutend animiert werden.

Viele weitere Anwendungen sind denkbar. So könnten z.B. die Bewegung von Haut bei Bewegung eines Skelettes, hängende Seile, Pflanzen (Blumen, Gras) oder Haar im Wind, aber auch Fallschirme, Drachen oder Segel von Segelbooten mit dem in dieser Arbeit vorgestellten Feder-Masse System simuliert werden.

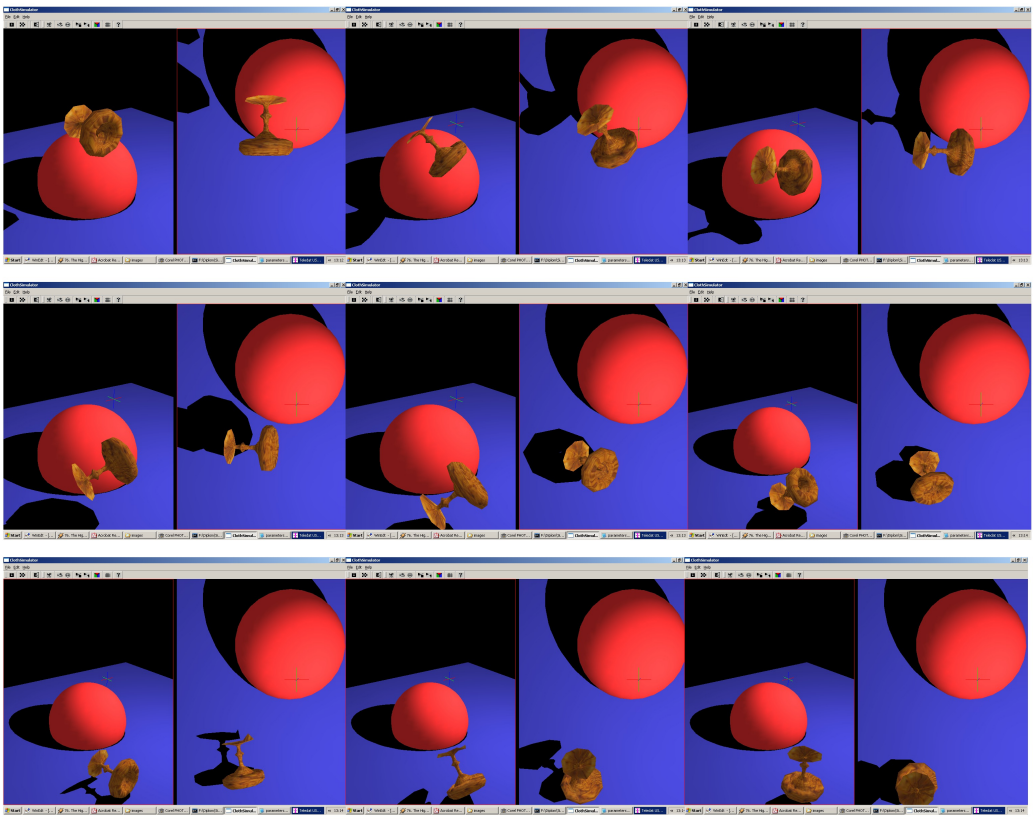

Abb. 6.16: Ein auf eine Kugel fallendes Glas als Beispiel der Möglichkeiten der Simulation von starren Körpern mit Hilfe von Feder-Masse Systemen (jeweils aus 2 Perspektiven). Gerendert in Echtzeit mit `ClothSimulator`.

6.5 Diskussion

Die im Rahmen dieser Arbeit entwickelte Art der Kraftberechnung und die Darstellung der Materialeigenschaften lediglich mit Federn in Verbindung mit einer impliziten Integration führt zu vergleichbaren Ergebnissen wie die von Baraff in [BW98] vorgestellten Methoden. Die implizite Integration ermöglicht eine Simulation bei größeren Zeitschritten und mit wesentlich größeren Federkonstanten als die explizite Integration. Mit der impliziten Integration kann das Problem der extremen Elastizität der simulierten Kleidung und das Problem des Super-Elastischen Effektes bei der expliziten Integration stark verringert werden. Darin liegt der große Vorteil der impliziten Integration gegenüber der expliziten Integration.

Bei sehr großen Zeitschritten reagiert die implizite Integration jedoch empfindlich auf Positionsveränderungen während der Kollisionsbehandlung - ansonsten wären Simulationsschritte von etwa h=0,1s möglich. Etwa 1000

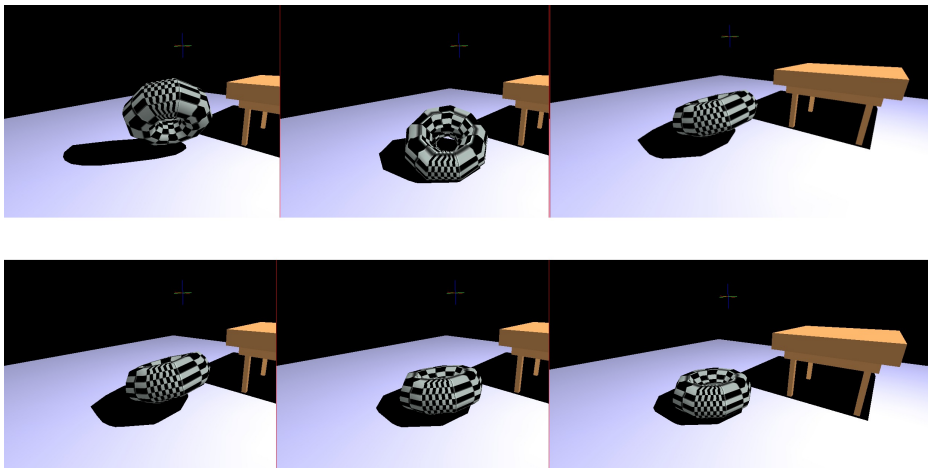

Abb. 6.17: Beispiel zur Simulation von starren Körpern mit Hilfe von Feder-Masse Systemen: Ein fallender Donut. Gerendert in Echtzeit mit `ClothSimulator`.

Partikel können inklusive Shading, Kollisionserkennung und -behandlung in Echtzeit auf einem Pentium IV/2400 Mhz simuliert werden. Mit zunehmender Prozessorgeschwindigkeit und Realisierungen von parallelen Algorithmen (vgl. Abschnitt 3.2.2) kann die Performance noch stark (bei paralleler Simulation bis zum Faktor 6) verbessert werden. Auch die Verwendung von hierarchischen Verfahren kann die Kollisionserkennung, vor allem von Cloth-Cloth-Kollisionen, noch stark beschleunigen. Die Verwendung adaptiver Verfahren kann ebenfalls eine Performancesteigerung bewirken. Durch Ausnutzen von Symmetrien (z.B. runder Tisch, rechteckiger Tisch) kann die Geschwindigkeit in Einzelfällen um den Faktor 4 oder 2 verbessert werden.

Überraschend ist die hohe Stabilität des Euler-Cromer-Verfahrens. Wegen des wesentlich geringeren numerischen Aufwandes ist die Geschwindigkeit der expliziten im Vergleich zur impliziten Integration etwa zehnfach so hoch. Bei kleinen Federkonstanten können Simulationsschritte bis in den Bereich um 0,02 s hinein gewählt werden. Bei höheren Federkonstanten fällt die maximale Schrittweite stark, aber die hohe Geschwindigkeit führt in diesem Fall in Verbindung mit zahlreichen Integrationsschritten pro gerendertem Frame zu Echtzeit-Ergebnissen. Die Motivation zur Einführung der impliziten Integration in die Kleidungssimulation war die geringe Schrittweite bei hohen Federkonstanten. Die Euler-Cromer-Methode liefert jedoch relativ gute Ergebnisse bei nicht zu steifen Systemen. An dieser Stelle bleibt die Frage offen, ob die Kritik zahlreicher Veröffentlichungen an der expliziten Euler-Methode an der Realisierung des Euler-Verfahrens und nicht des Euler-Cromer-Verfahrens liegt. Namentlich erwähnt ist das Euler-Cromer-Verfahren in [VCM98], [HB00].

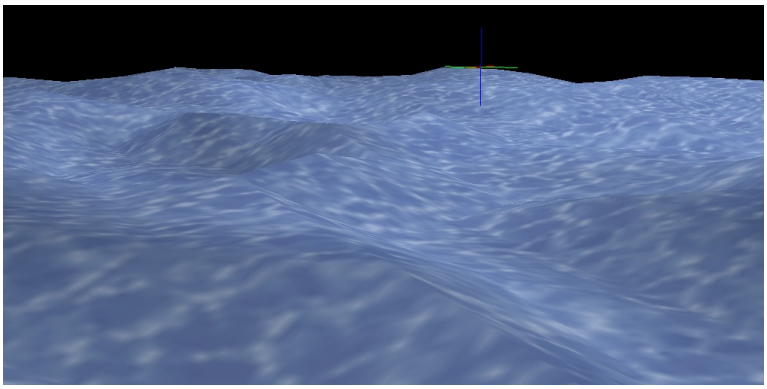

Abb. 6.18: Das Verhalten der Oberflächen von Fluiden kann mit Hilfe von Feder-Masse Systemen simuliert werden. Gerendert in Echtzeit mit dem im Rahmen dieser Arbeit entstandenen Programm `ClothSimulator`.

Für heutige Echtzeitanwendungen wie Computerspiele oder Virtual Reality ist die Qualität der Simulation mit der expliziten Euler-Cromer-Methode ausreichend und es können mit kleinen Meshes überzeugende Ergebnisse erzielt werde. Das Problem der Federhaftigkeit des simulierten Gewebes kann durch Simulation im Bereich von zwei- bis fünffacher Echtzeit und einer hohen Dämpfung verringert werden. Der große Vorteil der expliziten Integration liegt in der sehr hohen Geschwindigkeit, so dass kleine Gewebestücke ohne großen Aufwand simuliert werden können und noch ausreichend Rechenzeit für andere Funktionalitäten zur Verfügung steht.

Kapitel 7

Zusammenfassung und Ausblick

In dieser Arbeit wurden einige Möglichkeiten der Echtzeit-Kleidungssimulation basierend auf Feder-Masse Systemen vorgestellt, die mit aktueller Hardware (etwa Pentium 4/2400MHz) realisiert werden können. Es wurde ein Konzept der impliziten Kleidungssimulation mit physikalisch basierten Kräften entwickelt, welches vergleichbare Ergebnisse wie andere Konzepte basierend auf einer impliziten Integration erzielt und verhältnismäßig gute Performancewerte aufweist. In Echtzeit können mit diesem Verfahren Gewebe bestehend aus etwa 1000 Partikeln inklusive Shading simuliert werden. Die Vorteile der impliziten Integration liegen darin, dass wesentlich steifere Feder-Masse Systeme bei größeren Zeitschritten als bei der expliziten Integration simuliert werden können. Zudem ist die Stabilität der impliziten Integration bei sehr vielen Partikeln in Nicht-Echtzeitumgebungen höher als die der expliziten Integrationsmethoden. In derartigen Umgebungen können sehr realistisch anmutende Ergebnisse erzielt werden, obwohl die Darstellung von Geweben mit Feder-Masse Systemen nur bedingt der Realität entspricht.

Vor allem mit dem expliziten Euler-Cromer-Verfahren lassen sich gute Performancewerte mit der Simulation elastischer Gewebe in Echtzeitumgebungen erzielen. Die besprochenen Nachteile der Simulation von lediglich kleinen Federkonstanten können durch eine hohe Dämpfung und eine Simulation im Bereich von zwei- bis fünffacher Echtzeit verringert werden. Die Modellierung des Gewebes mit Hilfe von Federn ist im Gegensatz zur impliziten Integration aufgrund der geringen Steifigkeit der simulierbaren Systeme jedoch wesentlich stärker ersichtlich. Der entscheidende Vorteil ist die ausgesprochen hohe Geschwindigkeit des expliziten Euler-Verfahrens, die es ermöglicht, Gewebe mit sehr vielen Partikeln in Echtzeit zu simulieren (2500 Partikel mit Shading, 5000 Partikel ohne Shading). Vor allem die Simulation von kleinen Gewebe- oder Kleidungsstücken (bis etwa 1000

Partikel) benötigt mit der expliziten Euler-Cromer-Methode lediglich einen Bruchteil der zur Verfügung stehenden Rechenzeit. Damit ist die explizite Euler-Integration attraktiv für Virtual-Reality-Umgebungen und Computerspiele. Die erörterten Nachteile in Bezug auf die Darstellungsqualität liegen für diese Umgebungen in einem akzeptablen Bereich. Es wird wohl nur eine Frage der Zeit sein, bis Kleidungssimulation in aktuellen Computerspielen angewendet wird.

Sowohl das explizite als auch das implizite Euler-Verfahren kann mit Hilfe von parallelen Algorithmen oder adaptiven Ansätzen noch erheblich beschleunigt werden. Optimierungsmöglichkeiten sind auch im Bereich der Kollisionserkennung gegeben - mit entsprechenden Verfahren kann die Komplexität erheblich verringert werden. Auch die Verwendung von mehrstufigen Simulationsverfahren (siehe Kapitel 3) kann eine Beschleunigung mit sich bringen.

Der Realismus des dargestellten Gewebes kann durch Hinzufügen von Strömungseigenschaften, Gleit- und Haftreibungskräften, aber auch durch die Verwendung von komplexeren Beleuchtungsmodellen (z.B. Cook-Torrance, Ashikmin) und anderen als in dieser Arbeit vorgestellten Arten der Normalenberechnung erhöht werden. Beispielhaft wurden in dieser Arbeit ein einfaches Windmodell, Fell und Projective Shadows realisiert.

Alle in dieser Arbeit entwickelten Algorithmen wurden hinsichtlich ihrer Performance, ihrer Nutzbarkeit und ihrer Möglichkeiten diskutiert und bewertet. Die hier vorgestellten Prinzipien sind als solche allgemein gültig und nicht an die im Rahmen dieser Arbeit durchgeführte Implementierung gebunden.

Abschließend und die Vorteile des expliziten Euler-Verfahrens hervorhebend, die trotz aller in dieser Arbeit diskutierten Nachteile in der hohen Ausführgeschwindigkeit der expliziten Kleidungssimulation zu finden sind, sei ein Zitat von Andrew Witkin gegeben, dem Co-Autor der wegen der Einführung der impliziten Integration in die Kleidungssimulation entscheidenden Arbeit „Large Steps in Cloth Simulation“ [BW98]:

„Don't use explicit Euler's method (you will anyway).“
Andrew Witkin, Siggraph 2001

Literaturverzeichnis

[AB03] Ascher, U.M. und Boxerman, E.: *On the modified conjugate gradient method in cloth simulation.* Visual Computer'03, 2003.

[Ada03] Adamsson, M.: *Cloth*, 2003. Verfügbar im World Wide Web unter: `http://freespace.virgin.net/hugo.elias/models/m_cloth.htm` (07.03.2004).

[AH89] Amirbayat, J. und Hearle, J.: *The anatomy of buckling of textile fabrics: Drape and conformability.* Journal of Textile Institute 80, 1989.

[AMF03] Adabala, N., Magnenat-Thalmann, N. und Fei, G.: *Visualization of Woven Cloth.* Eurographics Symposium on Rendering, 2003.

[Bar98] Baraff, D.: *Collision Detection That Really Works.* Siggraph Courses 98, 1998.

[BFA02a] Bridson, R., Fedkiw, R. und Anderson, J.: *Robust Treatment of Collisions, Contact and Friction for Cloth Animation.* Presentation, Siggraph'02, 2002.

[BFA02b] Bridson, R., Fedkiw, R. und Anderson, J.: *Robust Treatment of Collisions, Contact and Friction for Cloth Animation.* Association for Computer Machinery, Inc. (ACM), 2002.

[BMF03] Bridson, R., Marino, S. und Fedkiw, R.: *Simulation of Clothing with Folds and Wrinkles.* Eurographics/Siggraph Symposium on Computer Animation, 2003.

[BRJ02] Bae, H.J., Ryu, K.W. und Jang, B.T.: *Procedural Approach to generate Real Time Motions of Cloth.* Proceedings of the Shape Modeling International (SMI'02), 2002.

[Bro02] Brown, K.: *Cloth*, 2002. Verfügbar im World Wide Web unter: `http://www.cs.csustan.edu/~kebrown/projects.htm` (07.03.2004).

[BSMM99] Bronstein, I.N., Semendjajew, K.A., Musiol, G. und Mühlig, H.: *Taschenbuch der Mathematik*. Harri Deutsch, Frankfurt am Main, 4. Auflage, 1999.

[BW98] Baraff, D. und Witkin. A.: *Large Steps in Cloth Simulation*. SIGGRAPH'98, 1998.

[BWK03] Baraff, D., Witkin, A. und Kass, M.: *Untangling Cloth*. Pixar Animation Studios, ACM Transactions on Graphics, Vol.22, Issue 3, 2003.

[CGC+02] Capell, S., Green, S., Curless, B., Duchamp, T. und Popović, Z.: *A Multiresolution Framework for Dynamic Deformations*. ACM Digital Library, 2002.

[CH02] Choi, K.-J. und Hyeong-Seok, K.: *Stable but Responsive Cloth*. Association for Computer Machinery, Inc. (ACM), 2002.

[CK03] Choi, K.-J. und Ko, H.-S.: *Extending the Immediate Buckling Model to Triangular Meshes for Simulationg Complex Clothes*. Eurographics'03, 2003.

[CM02] Cordier, F. und Magnenat-Thalmann, N.: *Real-time Animation of Dressed Virtual Humans*. Eurographics'02, 2002.

[Cor03] Cords, H.: *Realistische Darstellung komplexer Szenen mit hardwarebasierten Real-Time-Shadern*. Universität Rostock, Fachbereich Informatik, Studienarbeit, 2003.

[CXJS01] Cheng, C., Xu, Y.-Q., Jiaoying, S. und Shum, H.Y.: *Physically-based Real-time Animation of Draped Cloth*. Proceedings of Computer Graphics International (CGI'01), 2001.

[Dai00] Dai, X. et al: *A Particle Model based on Measured Mechanical Properties of Woven Cloth*. Proceedings of the Eight Pacific Conference on Computer Graphics and Applications (PG'00), 2000.

[DDCB01] Debunne, G., Desbrun, M., Cani, M.-P. und Barr, A.H.: *Dynamic Real-Time Deformations using Space and Time Adaptive Sampling*. Siggraph'01, 2001.

[DDCB03] Debunne, G., Desbrun, M., Cani, M.-P. und Barr, A.: *Adaptive Simulation of Soft Bodies in Real-Time.* Eurographics Association, 2003.

[Dem98] Demtröder, W. (Herausgeber): *Experimentalphysik 1 - Mechanik und Wärmelehre.* Springer Verlag, Berlin, 2. Auflage, 1998.

[DLHS01] Daubert, K., Lensch, H.P.A., Heidrich, W. und Seidel, H.-P.: *Efficient Cloth Modeling and Rendering.* Eurographics'01, 2001.

[DSB99] Desbrun, M., Schröder, P. und Barr, A.: *Interactive animation of structured deformable objects.* Caltech, 1999.

[EEH00] Eberhardt, B., Etzmuss, O. und Hauth, M.: *Implicit-Explicit Schemes for Fast Animation with Particle Systems.* ACM Digital Library, 2000. Verfügbar im World Wide Web unter: `www.gris.uni-tuebingen.de/publics/paper/Eberhardt-2000-ImplicitExplicit.pdf` (19.03.2004).

[EKS03] Etzmuss, O., Keckeisen, M. und Strasser, W.: *A Fast Finite Element Solution for Cloth Modelling.* 11th Pacific Conference on Computer Graphics and Applications (PG'03), 2003.

[Fey87] Feynman, C.: *Modeling the appearance of cloth.* Master's Thesis, Dept. of EECS, Massachusetts Inst. of Technology, 1987.

[FGL03] Fuhrmann, A., Gross, C. und Luckas, V.: *Interactive Animation of Cloth including Self Collision Detection.* Journal of WSCG, Vol.11, No.1, 2003.

[FvFH00] Foley, J.D., van Dam, A., Feiner, S.K. und Hughes, J.F.: *Computer Graphics - Principles and Practise.* Addison-Wesley, 2. Auflage, 2000.

[Gar03] Garber, M.: ***SimCloth***, 2003. Verfügbar im World Wide Web unter: `http://www.maxgarber.com/projects/cloth/#Demo` (07.03.2004).

[HB00] House, D.H. und Breen, D.E. (Herausgeber): *Cloth Modeling and Animation.* A K Peters, Natick, Massachusetts, 2000.

[HBVM02] Hadap, S., Bangerter, E., Volino, P. und Magnenat-Thalmann, N.: *Animating Wrinkles on Clothes.* MIRLab, CUI, University of Geneva, 2002.

[HEE+02] Hauth, M., Etzmuss, O., Eberhardt, B., Daubert, K. und Kautz, J.: *Cloth Animation and Rendering.* Tutorial, Eurographics'02, 2002.

[HMB01] Huh, S., Metaxas, D.N. und Badler, N.I.: *Collision Resolutions in Cloth Simulation.* Citeseer, 2001. Verfügbar im World Wide Web unter: `http://citeseer.ist.psu.edu/` (16.03.2003).

[Hou98] House, D.H. et al: *Cloth and Clothing in Computer Graphics.* Siggraph'98, Course Notes, 1998. Verfügbar im World Wide Web unter: `http://www-viz.tamu.edu/faculty/house/cloth/` (20.11.2003).

[IH03] Igarashi, T. und Hughes, J.F.: *Clothing Manipulation.* ACM Transactions on Graphics, Vol.22, Issue 3, 2003.

[Isa02] Isaak, J.: *Cloth Simulation*, 2002. Verfügbar im World Wide Web unter: `http://members.shaw.ca/jordanisaak/graphics.htm` (07.03.2004).

[Jac03] Jacobs, P.: *Cloth*, 2003. Verfügbar im World Wide Web unter: `http://home.cwru.edu/~pxj18/` (07.03.2004).

[KC02] Kang, Y.-M. und Cho, H.-G.: *Bilayered Approximate Integration for Rapid and Plausible Animation of Virtual Cloth with Realistiv Wrinkles.* Proceedings of the Computer Animation (CA 2002), 2002.

[KCC00] Kang, Y.-M., Choi, J.-H. und Cho, H.-G.: *Fast and Stable Animation of Cloth with an Approximated Implicit Method.* Proceedings of Computer Graphics International, IEEE CS Press, 2000.

[KCCL01] Kang, Y.-M., Choi, J.H., Cho, H.G. und Lee, D.-H.: *An efficient animation of wrinkled cloth with approximate implicit integration.* The Visual Computer Journal, Springer-Verlag, 2001.

[KK02] Kita, Y. und Kita, N.: *A model-driven method of estimating the state of clothes for manipulating it.* Proceedings of the Sixth IEEE Workshop on Applications of Computer Vision (WACV'02), 2002.

[KSFS03] Keckeisen, M., Stoev, S.L., Feurer, M. und Strasser, W.: *Interactive Cloth Simulation in Virtual Environments.* Virtual Reality'03 (VR), 2003.

[Kuy97] KUYPERS, F.: *Klassische Mechanik*. Wiley-VCH, Weinheim, 5., überarbeitete Auflage, 1997.

[Lah00] LAHOTI, A.: *Simulation of Highly Deformable Objects: Cloth Animation*. Indian Institute of Technology Kanpur, Department of Computer Science, 2000.

[Lan99a] LANDER, J.: *Devil in the Blue Faceted Dress: Real-Time Cloth Animation*. Graphic Content, 1999. Verfügbar im World Wide Web unter: `http://www.gdmag.com` (10.11.2003).

[Lan99b] LANDER, J.: *`Clothy`*, 1999. Verfügbar im World Wide Web unter: `http://www.darwin3d.com/gdm1999.htm` (07.03.2004).

[LGPT02] LARIO, R., GARCÍA, C., PRIETO, M. und TIRADO, F.: *A Parallel Cloth Simulator Using Multilevel Algorithms*. Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02), 2002.

[LHSM99] LUDING, S., HERRMANN, H.J., SCHWARZER, S. und MÜLLER, M.: *Physik auf dem Computer I u. II*. Skript zu den Vorlesungen Physik auf dem Computer I u. II, Universität Stuttgart, 1999.

[LMT01] LAFLEUR, B., MAGNENAT-THALMANN, N. und THALMANN, D.: *Cloth Animation with Self-Collision Detection*. Computer Graphics Lab, Swiss Federal Institute of Technology, 2001.

[Mac00] MACRI, D.P.: *Real-Time Cloth*. Game Developer Conference Proceedings, 2000.

[Mac01] MACRI, D.: *`ClothSample`*, 2001. Verfügbar im World Wide Web unter: `http://www.intel.com` (07.03.2004).

[Mac02] MACRI, D.: *Simulating Cloth for 3D Games*, 2002. Verfügbar im World Wide Web unter: `http://www.gamedev.net/reference/list.asp?categoryid=28` (16.03.2004).

[Mat] MATYKA, M.: *`Powierz`*. Verfügbar im World Wide Web unter: `http://panoramix.ift.uni.wroc.pl/~maq/eng/` (07.03.2004).

[MDDB00] MEYER, M., DEBUNNE, G., DESBRUN, M. und BARR, A.H.: *Interactive Animation of Cloth-like Objects in Virtual Reality*. Caltech, 2000.

[MDG98] Miguel, J., Dias, S. und Gamito, M.N.: *Modelling Cloth Buckling and Drape.* Eurographics 1998, 1998. Verfügbar im World Wide Web unter: `http://www.eg.org/EG/ForumEl/EG98/ShortPapers` (16.03.2004).

[Mey01] Meyer, M.: *Cloth Animation.* Lecture, CS274, 2001. Verfügbar im World Wide Web unter: `http://www.gg.caltech.edu/~cs274/Lectures/lecture09.pdf` (16.03.2003).

[MKE02] Mezger, J., Kimmerle, S. und Etzmuss, O.: *Improved Collision Detection and Response Techniques for Cloth Animation.* Universität Tübingen, WSI, 2002.

[MKE03] Mezger, J., Kimmerle, S. und Etzmuss, O.: *Hierarchical Techniques in Collision Detection for Cloth Animation.* Journal of WSCG, Vol.11, No.1, 2003.

[MVC02] Magnenat-Thalmann, N., Volino, P. und Cordier, F.: *Avenues of Research in Dynamic Clothing.* Proceedings of the Computer Animation (CA 2002), 2002.

[NG96] Ng, H.N. und Grimsdale, R.L.: *Computer Graphics Techniques for Modeling Cloth.* Computer Graphics in Textiles and Apparel, 1996.

[OM01] Oshita, M. und Makinouch, A.: *Real-Time Cloth Simulation with Sparse Particles.* Siggraph'01, 2001.

[PF02] Parks, D. und Forsyth, D.: *Improved Integration for Cloth Simulation.* Eurographics 2002, Short Presentation, 2002.

[PG00] Portnoy, W. und Grossman, D.: *Cloth Simulator 2000*, 2000. Verfügbar im World Wide Web unter: `http://www.cs.washington.edu/homes/grossman/projects/557project/results.html` (07.03.2004).

[Pow03] PowerVR: *D3DCloth*, 2003. Verfügbar im World Wide Web unter: `www.pvrdev.com/` (07.03.2004).

[Pri02] Pritchard, D.: *Implementing Baraff and Wittkin's Cloth Simulation.* Intel Corporation, 2002. Verfügbar im World Wide Web unter: `http://www.intel.com` (15.11.2003).

[Pri03] PRITCHARD, D.: ***Freecloth***, 2002-2003. Verfügbar im World Wide Web unter: `http://sourceforge.net/projects/freecloth/` (07.03.2004).

[Pro97] PROVOT, X.: *Collision and self-collision handling in cloth model dedicated to design garments.* Institut National de Recherche en Informatique et Automatique (INRIA), 1997.

[PTVF96] PRESS, W.H., TEUKOLSKY, S.A., VETTERLING, W.T. und FANNERY, B.R.: *Numerical Recipes in C.* Cambridge University Press, 2. Auflage, 1996. Verfügbar im World Wide Web unter: `http://www.library.cornell.edu/nr/bookcpdf.ht ml` (08.11.2003).

[RC02] RUDOMÍN, I. und CASTILLO J.L.: *Real-Time Clothing: Geometry and Physics.* Eurographics'02, 2002.

[RRZ00] ROMERO, S., ROMERO, L.F. und ZAPATA, E.I.: *Fast Cloth Simulation with Parallel Computers.* 6th International Euro-Par Conference (Euro-Par'2000), 2000.

[She94] SHEWCHUK, J.R.: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain.* School of Computer Science, Melon University, 1994.

[SSK03] SATTLER, M., SARLETTE, R. und KLEIN, R.: *Efficient and Realistic Visualization of Cloth.* Eurographics Symposium on Rendering, 2003.

[Stö99] STÖCKER, H.: *Taschenbuch der Physik.* Harri Deutsch, Frankfurt am Main, 1999.

[VB02] VILLARD, J. und BOROUCHAKI, H.: *Adaptive Meshing for Cloth Animation.* 11th International Meshing Roundtable'02, 2002.

[VCM98] VOLINO, P., COURCHESNE, M. und MAGNENAT-THALMANN, N.: *Versatile and Efficient Techniques for Simulating Cloth and Other Deformable Objects.* Computer Graphics Proceedings, 1998.

[VM00a] VOLINO, P. und MAGNENAT-THALMAN, N.: *Implementing Fast Cloth Simulation with Collision Response.* IEEE, 2000.

[VM00b] VOLINO, P. und MAGNENAT-THALMANN, N.: *Virtual Clothing - Theory and Practise.* Springer Verlag Berlin Heidelberg, 2000.

[VM01] Volino, P. und Magnenat-Thalmann, N.: *Comparing Efficiency of Integration Methods for Cloth Simulation*. Proceedings of Computer Graphics International (CGI'01), 2001.

[VSC01] Vassilev, B., Spanlang, B. und Chrysanthou, Y.: *Fast Cloth Animation on Walking Avatars*. Eurographics, Vol. 20, Number 3, 2001.

[Wan02] Wang, R.: *Adaptive Cloth Simulation*. School of Computer Science, Carnegie Mellon University, 2002.

[Wat00] Watt, A. (Herausgeber): *3D Computer Graphics*. Addison-Wesley, 2000.

[WB97] Wittkin, A. und Baraff, D.: *Physially Based Modeling*. Siggraph'97, Course Notes, 1997. Verfügbar im World Wide Web unter: `http://www.pixar.com/aboutpixar/research/pbm2001` (20.11.2003).

[Wei86] Weil, J.: *The Synthesis of Cloth Objects*. Proc. SIGGRAPH'86, Vol. 20, pp.49-54, 1986.

[Wlo01] Wloka, M.: *Interactive Cloth Simulation*. Nvidia Corp., 2001.

[WW92] Watt, A. und Watt, M. (Herausgeber): *Advanced Animation and Rendering Techniques - Theory and Practise*. acm Press, 1992.

[Yan01] Yang, D.: *C++ and Objectoriented Numeric Computing*. Springer-Verlag, New York, 2001.

[ZFV02] Zara, F., Faure, F. und Vincent, J.M.: *Physical cloth simulation on a PC cluster*. Fourth Eurographics Workshop on Parallel Graphics and Visualization, 2002.

[ZY00] Zhang, D. und Yuen, M.M.F.: *Collision Detection for Clothed Human Animation*. Proceedings of the Eight Pacific Conference on Computer Graphics and Applications (PG'00), 2000.

Anhang A

Überblick über die CD

Auf der beiliegenden CD sind die im Rahmen dieser Arbeit entwickelten Programme `ClothSimulator` und `ClothEditor` zu finden. Dies betrifft insbesondere auch die jeweiligen Sourcecodes, die verwendeten Texturen, die 3D-Studio Objekte und Parameterfiles verschiedener Beispiele. Zu den Programmen existieren Microsoft Visual C 6.0 Project Files. Zusätzlich ist dieses Dokument als `diplomarbeit.pdf` auf der CD-ROM abgelegt.

Thesen

1. Kleidungs- und Gewebesimulation kann mit Hilfe von Feder-Masse Systemen in Echtzeit durchgeführt werden. Die erzielten Ergebnisse entsprechen weites gehend dem Verhalten realer Textilien, obwohl eine Simulation mit Feder-Masse Systemen dem realen physikalische Verhalten von Geweben und Kleidung nur bedingt entspricht.

2. Zur Integration der wirkenden Kräfte können explizite und implizite numerische Integrationsverfahren benutzt werden.

3. Explizite Integrationsverfahren besitzen den Nachteil, dass sie lediglich kleine Federkonstanten und kleine Zeitschritte im Vergleich zur impliziten Integration simulieren können. Aufgrund der nur kleinen simulierbaren Federkonstanten besitzt das simulierte Gewebe eine hohe, nicht der Realität entsprechende Elastizität und das Problem des Super-Elastischen Effektes ist stark ersichtlich. Der Vorteil der expliziten Integrationsverfahren liegt in der hohen Ausführgeschwindigkeit.

4. Mit Hilfe einer impliziten Integration können wesentlich größere Federkonstanten bei größeren Zeitschritten simuliert werden. Die erwähnten Probleme der expliziten Integration sind kaum noch wahrnehmbar. Der benötigte Aufwand ist jedoch wesentlich höher. Diese Arbeit stellt eine implizite Integration physikalisch basierter Kräfte vor, wobei die erzielten Ergebnisse mit denen anderer Arbeiten vergleichbar sind.

5. Das in dieser Arbeit realisierte explizite Euler-Cromer Verfahren bietet sich aufgrund der hohen Ausführgeschwindigkeit für die Echtzeitsimulation von Kleidung und Geweben an. Die erwähnten Probleme können in Echtzeitumgebungen mit einer hohen Dämpfung und einer Simulation im Bereich von zwei- bis fünffacher Echtzeit verringert werden. Die implizite Integration erzielt realistischere Ergebnisse als die explizite Integration, es können jedoch nur bis zu etwa 1000 Partikel inklusive Shading in Echtzeit simuliert werden (Pentium IV/2400MHz).

6. Schnitte im Gewebe können durch Entfernen von Federn oder Partikeln simuliert werden. Ein einfaches Windmodell kann durch Addition von Geschwindigkeiten entsprechend der Orientierung der Polygone realisiert werden.

7. Mit Hilfe von Shading, Texturing, Felldarstellung und Projective Shadows kann der Realismus der dargestellten Szene wesentlich erhöht werden. Andere Echtzeitverfahren zur Verbesserung der Darstellungsqualität bieten sich ebenfalls zur Integration an.

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Hilko Cords
Rostock, 04.04.2004