Interaktive Animation textiler Materialien

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
von

Dipl.-Inform. Arnulph Fuhrmann
geboren in Wiesbaden

Referenten der Arbeit:	Prof. Dr.-Ing. José L. Encarnação
	Prof. Dr. Andreas Weber
Tag der Einreichung:	21.04.2006
Tag der mündlichen Prüfung:	09.06.2006

Darmstädter Dissertation, 2006
D 17

Danksagung

Ich danke Herrn Prof. Dr.-Ing. José L. Encarnação für die Betreuung meiner Arbeit und die Übernahme des Referats. Zudem möchte ich ihm für die Schaffung der Arbeitsumgebung am Fraunhofer Institut für Graphische Datenverarbeitung (IGD) in Darmstadt danken, an dem diese Arbeit entstanden ist.

Weiterhin danke ich Herrn Prof. Dr. Andreas Weber für die Übernahme des Koreferats und die vielen inspirierenden und motivierenden Diskussionen im Vorfeld.

Ein spezieller Dank für die gute Zusammenarbeit und die vielen Anregungen geht an meine aktuellen und ehemaligen Kollegen aus der Abteilung Echtzeitlösungen für Simulation und Visual Analytics, deren Abteilungsleiter Dr.-Ing. Jörn Kohlhammer ist. In ungefährer chronologischer Reihenfolge zählen dazu Prof. Dr.-Ing. Volker Luckas (ehemaliger Abteilungsleiter, jetzt FH-Bingen), Clemens Groß, Horst Birthelmer, Gabi Knöß, Thorsten May, Dr.-Ing. Jörg Sahm, Dr.-Ing. Eric Blechschmitt, Ingo Soetebier, Sascha Schneider, Martin Knuth, Marcus Hoffmann und Tatiana Tekusova.

Auch meinen Master- und Bachelorarbeitern und Praktikanten sowie meinen wissenschaftlichen Hilfskräften möchte ich für ihr gute Zusammenarbeit danken. Insbesondere möchte ich meinen Diplomanden Roland Sarrazin, Sebastian Gantzert, Martin Knuth und Marco Hutter für ihre sehr gute Arbeit danken.

Einen herzlichen Dank meiner Freundin Bianca Bönisch, die mir während der Erstellung dieser Arbeit verständnisvoll und geduldig zur Seite stand. Ein ganz besonderer Dank geht an meine Mutter, die mir mein Studium und somit das Erstellen dieser Arbeit ermöglicht hat.

Arnulph Fuhrmann, Darmstadt, Mai 2006

Inhaltsverzeichnis

Kapitel 1

Einleitung

Ein Ziel der Computergraphik ist die realistische Visualisierung von virtuellen Umgebungen. Dabei ist nicht nur das Rendering von hoher Bedeutung, sondern auch die Simulation des Verhaltens der einzelnen Objekte innerhalb der Umgebungen. Hierfür wurden bereits frühzeitig Methoden entwickelt, die das Objektverhalten mit Hilfe physikalischer Modelle simulieren konnten [TW88, TF88]. Diese Methoden lieferten bereits erstaunlich realitätsnahe Ergebnisse, die allerdings aufwendig berechnet werden mussten. An eine Visualisierung komplexer Objekte war damals noch nicht zu denken. Deshalb wurden im Laufe der Jahre neue Simulationsverfahren entwickelt. Diese waren nicht nur effizienter, sondern die Objekte konnten auch animiert werden.

Eine besondere Herausforderung ist die Berechnung von Animationen während der Benutzer in das Geschehen eingreift, da dann nur sehr eingeschränkt auf vorberechnete Daten zurückgegriffen werden kann. Damit die Immersion des Benutzers nicht gestört wird, benötigt man ein Simulationsmodell, das innerhalb kürzester Zeit Änderungen am Zustand des Objekts berechnen kann. Gleichzeitig muss dafür Sorge getragen werden, dass die Ergebnisse der Simulation schnell visualisiert werden können.

In den letzten Jahren wurden Verfahren zur physikalisch basierten Animation von Flüssigkeiten, Gasen, starren und deformierbaren Körpern entworfen. Textiles Material zählt zwar zu den deformierbaren Körpern, bereitet aber aufgrund seines Aufbaus ganz spezielle Probleme. Die physikalischen Eigenschaften von Stoff werden durch die einzelnen Fäden und deren Verknüpfung festgelegt. Unter den verschiedenen Textilien spielt gewebter Stoff eine sehr bedeutende Rolle, da sich dieser Herstellungsprozess besonders gut industriell umsetzen lässt. Gewebter Stoff verhält sich gegenüber Zugbelastungen annähernd wie ein starrer Körper, kann aber vergleichsweise leicht gebogen und geschert werden. Dies macht die Simulation dieses Materials außerordentlich aufwändig. Ein weiteres Problem für ein Animationssystem stellen Kollisionen zwischen dem Stoff und anderen Objekten dar, da dieser sich leicht an eine andere Oberfläche anschmiegt und somit eine Vielzahl von Kontaktpunkten hervor ruft, die alle berücksichtigt werden müssen.

Schließlich erfordert die Visualisierung von Stoff spezielle Algorithmen, da aufgrund seiner Struktur das Reflektionsverhalten hochgradig komplex ist.

1.1 Motivation

Anwendungen

Die interaktive Animation textiler Materialien ist für viele Anwendungen von Interesse. Ein Bereich ist die Animation von Kleidungsstücken, die von einer virtuellen Person – oft Avatar genannt – getragen wird. Die denkbaren Szenarien in diesem Bereich sind vielfältig. Nahe liegend sind virtuelle Menschen in Computergenerierten Filmen. Ihre Bekleidung wird dann nicht mehr von Hand animiert, sondern kann nach den Vorgaben der Bewegung der Figur simuliert werden. In diesem Fall spielt die Interaktivität jedoch eine untergeordnete Rolle. Allerdings sind effiziente Animationssysteme trotzdem bei der Produktion des Films äußert hilfreich, da schneller gearbeitet werden kann und sich besser Änderungen vornehmen lassen. Beim Einsatz in Computerspielen muss heutzutage noch auf eine physikalisch basierte Animation von kompletter Bekleidung verzichtet werden. Kleinere Details, wie z.B. ein loser Umhang, werden jedoch oft schon animiert. Durch animierte Bekleidung der Protagonisten würde sich der Realismus und somit die Immersion in ein Spiel steigern lassen.

Aus dem Bereich der Bekleidungsindustrie gab es in den letzten Jahren besonderes Interesse an der virtuellen Anprobe von Kleidungsstücken, womit sich neue Formen der Produktpräsentation und des Virtual Prototyping erschließen lassen. Ein Kunde könnte beispielsweise ein Kleidungsstück an seinem eigenen 3D-Körperscan anprobieren und sich eine virtuelle Modenschau vorführen lassen. Erst wenn ihm das virtuelle Kleidungsstück gefällt, macht er seine Bestellung. Dadurch ließen sich im Versandhandel Rücksendungen aufgrund von Nichtgefallen deutlich reduzieren.

Im Produktentwurf, d.h. dem Design neuer Kleidungsstücke, eröffnet die virtuelle Anprobe gänzlich neue Vorgehensweisen. Der Textildesigner könnte sein bekanntes Terrain von 2D-CAD-System und Anprobe der Muster an Kleiderbüsten verlassen. Er würde einfach ein System zum 3D-Design von Kleidungsstücken verwenden, das es ihm erlaubt direkt an der virtuellen Kleiderpuppe Änderungen am Kleidungsstück vorzunehmen, wenn ihm etwas nicht gefällt. Ein solches System stellt sehr hohe Anforderungen an die Interaktivität, da die durchgeführten Änderungen sofort Auswirkungen auf die simulierte Kleidung haben müssen. Gleichzeitig muss das Verhalten des Stoffes so real wie möglich durch Modelle auf dem Computer nachgebildet werden, damit eine genaue Aussage über die Passform des Kleidungsstücks gemacht werden kann.

Neben der Bekleidungsindustrie profitieren auch andere Bereiche der Textilindustrie von interaktiver Textilanimation. In der Automobilindustrie ließen sich Faltdächer simulieren. Heutzutage kann nur die Funktionsweise der Gestänge simuliert werden, nicht jedoch die der Bespannung. Die Bestimmung der Form

Abbildung 1.1: Virtuelle Anprobe einer Hose und eines Pullovers.

der einzelnen Schnittmuster des Faltdachs ist ein aufwendiger iterativer Prozess. Die einzelnen Schnittmuster müssen zunächst miteinander und dann am Gestänge vernäht werden. Darauf folgen der Funktionstest und die Anpassung der Form der Schnittmuster. Dies wird solange wiederholt bis die gewünschte Qualität erreicht ist. Ganz ähnlich ist die Vorgehensweise beim Erstellen der Schnittmuster für Autositze, die ebenfalls iterativ verbessert werden. Zusätzlich muss aber in diesem Fall die Deformation der Polsterung berücksichtigt werden. Beide Bereiche würden von einem System zum Virtual Prototyping profitieren.

Problemstellung

Die Animation von Textilien wird im Bereich der Computergraphik schon sehr lange untersucht. Erste Modelle Versuchten gewebten Stoff geometrisch zu modellieren [Wei86]. Der Stoff wurde als Fläche dargestellt. Die Simulation des Materials erfolgte dann mit Hilfe von geometrischen Funktionen, die die Fläche deformierten. Später ging man zur physikalisch basierten Simulation über [TW88, TF88], die eine genauere Abbildung realen Verhaltens anhand einstellbarer Materialparame-

ter ermöglicht. Einige Jahre später erlaubte die Verwendung von Partikelsystemen [BHW94, EWS96] schnellere Simulationen.

Der nächste Schritt war die Simulation von Bekleidung an virtuellen Menschen. Dies umfasst den Übergang von kleinen Stoffstücken zu komplexer Bekleidung, die aus mehreren miteinander vernähten Schnittteilen besteht. Jedes Schnittteil kann aus einem anderen Material bestehen. Einige bestehen sogar aus unterschiedlichen Materialien. So sorgt erst die richtige Kombination von Einlagen und Fütterungen bei einem realen Jackett für die richtige Passform. Dieses komplexe Zusammenspiel verschiedenster Stoffe muss von der Simulation berücksichtigt werden, damit aussagekräftige Resultate erzielt werden können. Bei der Animation von Bekleidung kommt erschwerend hinzu, dass auch noch Kollisionen mit dem Avatar erkannt und behandelt werden müssen.

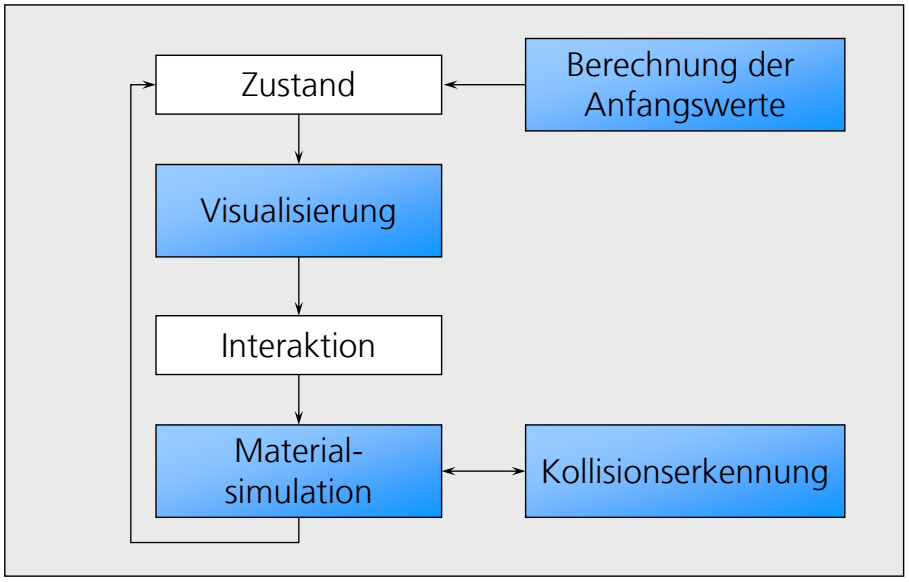


Abbildung 1.2: Überblick über die Systemarchitektur.

Für ein System zur interaktiven Animation von textilen Materialien wird eine Reihe von unterschiedlichen Algorithmen benötigt, die an definierten Schnittstellen ineinander greifen. In Abbildung 1.2 ist die grobe Struktur eines solchen Systems dargestellt. Beim Start des Systems muss ein Initialzustand der Bekleidung zur Verfügung gestellt werden. Dies ist im Allgemeinen die Position eines oder mehrere Stücke Stoffes im Raum. Dieser Zustand lässt sich visualisieren und der Anwender kann daraufhin mit dem virtuellen Stoff interagieren. Der Zustand des Stoffes und die Benutzereingaben fließen gemeinsam in die Simulation des Materialverhaltens ein. Dabei werden Kollisionen behandelt und auf die Benutzereingabe reagiert. Als Ergebnis erhält man den nächsten Zustand — ein Simulationsschritt ist berechnet.

Damit der Benutzer sinnvoll arbeiten kann und die Immersion nicht gestört wird, ergibt sich als erste Anforderung an das System, dass ein Simulationsschritt in weniger als einer Sekunde durchgeführt werden muss. Optimales Arbeiten erfordert eine Simulationsrate von 10Hz oder besser sogar 30Hz. Für bestimmte Anwendungen kann sogar eine Simulation in Echtzeit gefordert sein. In diesem Fall muss die Dauer zur Berechnung eines Simulationsschrittes kleiner als dessen Schrittweite sein. Ansonsten hinkt die Simulation hinterher, was sich besonders bei der Animation von Stoff durch sehr unrealistisches Verhalten bemerkbar macht: Der Stoff scheint, anstatt von Luft, von einer Flüssigkeit hoher Viskosität umgeben zu sein und die Bewegung sieht extrem gedämpft aus.

Die weiteren Anforderungen ergeben sich aus den drei hervorgehobenen Bereichen in Abbildung 1.2 und deren speziellen Problematiken. Da die Simulation von textilem Material äußerst zeitaufwendig ist, sucht man nach effizienteren Methoden der Berechnung. Innerhalb der Simulation stellt die Berechnung der Verformungen und Bewegungen des Stoffes als zeitliche *Integration der Bewegungsgleichung* $\ddot{\mathbf{x}} = \mathbf{M}^{-1}\mathbf{f}$ den größten Anteil an benötigter Rechenzeit dar. Die Bewegungsgleichung besteht aus einem System gewöhnlicher Differentialgleichungen, die aufgrund der Materialeigenschaften von Stoff steif sind. Dies rührt daher, dass Stoff sich unter seinem Eigengewicht nur geringfügig dehnt, aber sehr leicht gebogen werden kann. Es besteht großes Interesse an effizienten Methoden zur Integration dieser Gleichung, da explizite Integrationsverfahren nur bei kleinen Schrittweiten stabil bleiben [EWS96]. Implizite Integrationsverfahren wurden von Baraff und Witkin [BW98] in die Textilsimulation eingeführt und liefern stabile Lösungen. Dies wird aber mit erhöhtem Rechenaufwand und künstlicher Dämpfung des Materials erkauft.

Ein weiteres offenes Problem stellt das zu verwendende *Simulationsmodell* dar. Die Literatur bietet eine breite Palette von verschiedenen Modellen — jedes mit speziellen Vor- und Nachteilen. In den meisten Modellen geht man zunächst von einer Diskretisierung der Geometrie des Stoffes aus. Hierzu kann beispielsweise eine geeignete Triangulierung dienen. Alle Berechnungen der Simulation erfolgen auf den einzelnen Elementen des entstandenen Dreiecksnetzes. Eine häufig getroffene Annahme ist die Anhäufung der Masse in den Vertices. Somit stellt jeder Vertex einen so genannten Massepunkt oder auch Partikel dar. Dies führt auch direkt zu einem sehr einfachen Simulationsmodell, dem Masse-Feder Netzwerk, das einzelne Massepunkte mit Hook'schen Federn verknüpft. Verallgemeinert man dieses Modell, kommt man zum Partikelsystem, in dem Massepunkte auf verschiedenste Weise miteinander verknüpft werden können. Partikelsysteme haben den großen Vorteil, dass sie sich sehr effizient animieren lassen. Leider lassen sich gemessene Materialparameter immer noch nicht optimal übertragen, daher wurde kürzlich ein Simulationsmodell auf der Basis co-rotierter finiter Elemente vorgestellt [EKS03]. Die Übertragung gemessener Materialparameter auf das Modell funktioniert zwar sehr gut, aber leider geht dies mit Einbußen an der Performance einher.

Während die *Kollisionserkennung* für Starrkörper sehr gut erforscht ist, stellt die Kollisionserkennung während der Simulation textiler Materialien neue Heraus-

forderungen. Zunächst ergeben sich eine Vielzahl von Kontaktpunkten zwischen dem Stoff und einem Kollisionsobjekt. Diese müssen alle erkannt werden und danach muss eine entsprechende Kollisionsantwort gefunden werden. Klassische hierarchische Algorithmen zur Kollisionserkennung [KHM+98, Zac02] machen sich die Tatsache zu nutze, dass meist nur wenige Kontaktpunkte existieren. Diese können mit Hilfe der Hierarchie schnell identifiziert werden. Bei der Simulation von Bekleidung geht dieser Vorteil verloren. Ein weiteres Problem stellt die Berechnung der Kollisionsantwort dar. Damit diese korrekt und vor allem robust berechnet werden kann, muss klar sein, welcher Teil des Raumes innerhalb und welcher außerhalb des Objektes ist. Hierarchische Ansätze sind an diesem Punkt anfällig, es sei denn man verwendet kontinuierliche Kollisionserkennung, die aber für interaktive Systeme bei Weitem zu langsam ist [BFA02]. Moderne Ansätze [GKJ+05] erlauben die Berechnung einiger weniger Frames pro Sekunde. Für interaktives Arbeiten ist dies jedoch zu wenig.

Die *Visualisierung* der simulierten Materialien sollte möglichst realistisch sein. Insbesondere Details wie Nähte, Knöpfe und Aufdrucke sollten beim Rendering von Bekleidung berücksichtigt werden. Weiterhin ist die Berücksichtigung von Selbstabschattung äußerst wichtig. Speziell, wenn der Stoff Falten wirft, erhöhen Schatten den plastischen Eindruck. Liegen mehrere Lagen Stoff übereinander kann auf den Schattenwurf gar nicht mehr verzichtet werden, da sich ansonsten die einzelnen Lagen nicht mehr optisch auseinander halten lassen. Zum Rendern von Schatten, die von Punktlichtquellen erzeugt werden, gibt es zwei grundlegend verschiedene Techniken, die interaktive Frameraten erlauben: Shadow Maps [Wil78] und Shadow Volumes [Cro77]. Bei Verwendung dieser Algorithmen entstehen aber leider harte Schatten, die zwar besser sind als gar keine, jedoch nicht realistisch sind.

1.2 Zusammenfassung der wichtigsten Ergebnisse

In dieser Arbeit wird ein System zur Animation textiler Materialien vorgestellt. Hierzu werden verschiedene neue Algorithmen präsentiert, die zusammengenommen je nach Komplexität des verwendeten Materials eine Animation in Echtzeit bzw. eine interaktive Animation erlauben.

- Berechnung der Anfangswerte für die Simulation von Bekleidung. Dazu werden Ontologien für Kleidungsstücke und ein Verfahren zum interaktionsfreien Einkleiden virtueller Menschen vorgestellt.

- Schnelle Kollisionserkennung. Es wird ein Verfahren zur Kollisionserkennung mit Distanzfeldern vorgestellt, das sich insbesondere für die Kollisionserkennung zwischen stark deformierbaren Körpern und weitgehend starren Körpern eignet. Die Speicherung eines Distanzfeldes erfolgt in einer kompakten Datenstruktur. Des Weiteren wird eine effiziente Methode zur Vermeidung von Selbstkollisionen präsentiert.

- Effiziente Materialsimulation. Ein spezielles Simulationsmodell erlaubt die schnelle physikalisch-basierte Simulation von Stoff (robust, große Zeitschritte, flexibles Modell). Weiterhin wird ein Verfahren vorgestellt, mit dem zur Laufzeit die Passform eines Kleidungsstücks optimiert werden kann.

- Echtzeit Visualisierung. Es wird eine Architektur für einen Szenegraphen vorgestellt, die es erlaubt parallel zur Simulation das textile Material in Echtzeit zu rendern. Dabei werden Details wie Nähte, Knöpfe und Säume am Stoff visualisiert. Die Geometrie für die Säume und die Texturkoordinaten für Nähte werden automatisch ausgehend von den Eingabedaten erzeugt. Zudem erfolgt eine realistische Beleuchtung inklusive Schattenwurf.

1.3 Gliederung

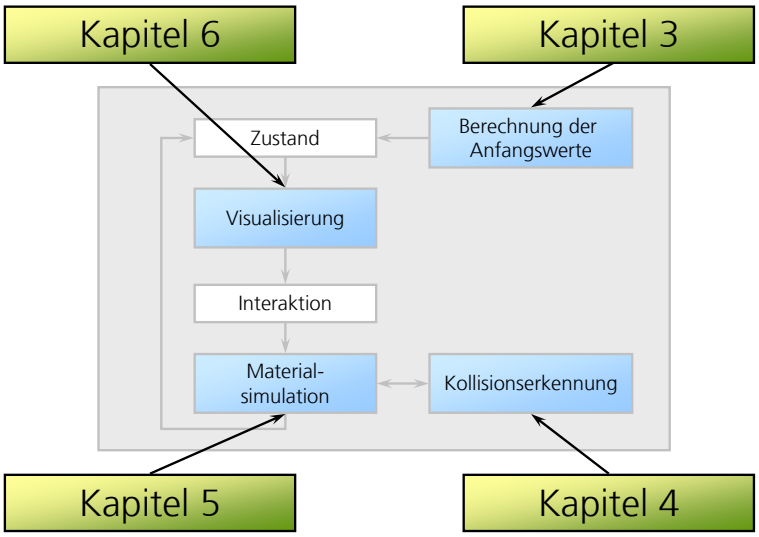


Abbildung 1.3: Überblick über die Gliederung dieser Arbeit.

1.4 Mathematische Notation

In dieser Arbeit wird die folgende Notation für mathematische Ausdrücke verwendet:

Typ	**Notation**	**Beispiel**
Winkel	Griechischer Kleinbuchstabe	α, ω
Skalar	Kleinbuchstabe kursiv	a, b, s_j, t_{ik}
Vektor oder Punkt	Kleinbuchstabe fett	$\mathbf{p}, \mathbf{s}_j, \mathbf{t}_{ik}$
Matrix	Großbuchstabe fett	$\mathbf{T}, \mathbf{M}$
skalarwertige Funktion	fett	$f(x), s(\mathbf{p})$
vektorwertige Funktion	kursiv	$\mathbf{S}(x), \mathbf{N}(\mathbf{p})$

Tabelle 1.1: Mathematische Notation in dieser Arbeit.

Kapitel 2

Wissenschaftliche Einordnung

2.1 Simulation textiler Materialien

In den letzten zwei Jahrzehnten sind eine ganze Reihe von Simulationsmodellen für textile Materialien vorgestellt worden. Einige von ihnen zielen darauf ab das physikalische Verhalten von Stoff zu reproduzieren. [BHW94, EWS96, VCMT95, BW98, VMT01]. Andere behandeln die Besonderheiten von gestrickten Materialien [EW99]. Ein Überblick über die verschiedenen Verfahren findet sich in [NG96]. Eine etwas neuere Veröffentlichung [VMT01] vergleicht den Rechenaufwand verschiedener Integrationsverfahren. Einen weiteren Überblick über das Themengebiet geben zwei Bücher [HB00, VMT00b].

In den letzten Jahren wurden weitere Verfahren, die exakter arbeiten, entwickelt [EEH00, HE01, CK02]. Andere Verfahren sind in der Lage das Verhalten zu approximieren und trotzdem plausible Animationen zu generieren. Sie werden im Kontext von interaktiven VR-Anwendungen oder Spielen eingesetzt [MDDB01, KCCL01, Jak01, KC02]. In [CMT02] wird ein Algorithmus vorgestellt, der schnelle geometrische Methoden mit physikalische basierten verbindet um interaktive Animationen bekleideter Menschen zeigen zu können.

Trotz der Fortschritte in den letzten Jahren sind genaue Simulationen immer noch zu aufwendig für Echtzeitanwendungen, da der Rechenaufwand immer noch sehr hoch ist. Insbesondere, wenn Stoff simuliert werden soll, der sich frei im Raum bewegen kann. In diesem Fall können keine Annahmen über das Verhalten des Stoffes getroffen werden, die z.B. möglich sind, wenn Kleidung an animierten Menschen gezeigt werden soll [CMT02, CMT04]. Diese Annahmen erlauben zwar recht hübsche Animationen, aber es können keine Aussagen über die Passform von Kleidungsstücken gemacht werden.

Ein Problem der physikalisch basierten Simulation von Textilien sind die auftretenden steifen Differentialgleichungen. Diese Gleichungen resultieren aus den sehr hohen internen Kräften innerhalb des Materials. Da die meisten gewebten Stoffe kaum gedehnt, aber leicht gebogen werden können, sind auch die Größenordnungen der Kräfte ebenfalls hoch. Verwendet man explizite Integrati-

onsverfahren, wie z.B. in [EWS96], dann müssen die Zeitschritte sehr klein sein, damit das System nicht divergiert. Daraus folgt ein sehr hoher Berechnungsaufwand.

Größere Zeitschritte können mit impliziten Verfahren genommen werden. Diese Technik wurde bei der Stoffanimation von [BW98] eingeführt und daraufhin immer weiter verbessert [HE01, VMT01, AB03]. Allerdings sind diese Verfahren nicht in der Lage Echtzeitanimationen mit ausreichender Partikelanzahl zu erzeugen, da in jedem Zeitschritt ein lineares Gleichungssystem gelöst werden muss. Obwohl dieses Gleichungssystem für gewöhnlich dünn besetzt ist, benötigt die Lösung mit der Methode der konjugierten Gradienten immer noch $O(n^{1.5})$, wobei n für die Anzahl der Partikel steht [BW98].

2.1.1 Animationsverfahren

Deshalb wurde in [MDDB01] eine vereinfachte implizite Methode vorgestellt, die in der Lage ist Systeme mit mehreren hundert Partikeln zu animieren. Das Verfahren verwendet eine vorberechnete Filtermatrix der Größe $O(n^2)$, die in jedem Zeitschritt angewendet wird. Das Verfahren ist zwar äußerst schnell, wenn nur wenige Partikel vorhanden sind, verliert aber seine Vorteile, wenn die Anzahl der Partikel ansteigt, da die Größe der Filtermatrix zu schnell wächst. Wenn man diese Methode testet, stellt man beim Bewegen von Stoff fest, dass dieser mehrere Sekunden benötigt, bis er seine endgültige Position erreicht. Das bedeutet, dass diese Methode, obwohl sie numerisch sehr stabil ist, zu langsame Bewegungen erzeugt. Diese liegt möglicherweise an den Eigenschaften des vereinfachten impliziten Integrationsschemas. Ähnliche Beobachtungen wurden von [KCL+00, KCCL01] gemacht. Aus diesem Grund, entwickelte diese Forschergruppe ein Verfahren, das die Methode von [MDDB01] verbessert. Die Filtermatrix wird durch eine Updateformel ersetzt, die nur Nachbarpartikel berücksichtigt. Dies löst das Problem der $O(n^2)$ Filtermatrix, aber da nur direkte Nachbarn berücksichtigt werden können nur wenige Massepunkte verwendet werden. Deshalb fügen die Autoren Falten und Details durch sogenannte 'Wrinkled cubic spline curves' hinzu. Die somit erzeugten Falten sind demnach nicht physikalisch basiert, sondern eher geometrischer Natur.

Andere Ansätze verfolgen einen Level-of-Detail Ansatz der einen oder anderen Art. In [PF94, JGW04] werden Mehrgitterverfahren verwendet, um in Kombination mit expliziten Integrationsverfahren stabilere Lösungen zu gewinnen. Die Idee hinter diesen Verfahren ist der Einsatz von Simulationsnetzen mit unterschiedlicher Auflösung. Auch andere Autoren habe diese Idee verfolgt [ZY01, KC02]. Wobei die Methode von [ZY01] sich auf das Drapieren von Stoff beschränkt. [KC02] hingegen präsentieren ein Verfahren, dass zwei unterschiedlich aufgelöste Meshes verwendet. Das grobe Mesh repräsentiert globale Bewegung und das feine Mesh fügt Details hinzu. Diese Methode ist in der Lage tausende von Partikeln in Echtzeit zu animieren. Aber da das Verfahren eine Verbesserung von [KCCL01] ist, können im groben Mesh nur wenige hundert Partikel vorhanden sein. Obwohl die Behandlung von Selbstkollisionen einen wichtigen Aspekt der Stoffsimulation darstellt, ist die-

se nicht vorgesehen. Da das grobe Mesh auch nur aus wenigen Partikeln besteht, ist es schwierig stark gekrümmte Stoffstücke zu simulieren. Dies ist z.B. nötig, wenn der Stoff auf den Boden fällt und sich auftürmt oder wenn der Stoff an nur einem Punkt festgehalten wird.

Eine andere Art der Optimierung ist möglich, wenn man bekleidete Menschen animiert, da die meisten Teile eines Kleidungsstücks ihre Position in Relation zum Körper kaum verändern [CMT02, RC02]. In [CMT02] wird ein System zur Animation komplexer Bekleidung in Echtzeit beschrieben. Dazu werden die einzelnen Partikel in drei Ebenen eingeteilt. Diejenigen, die zur ersten und zur zweiten Ebene gehören werden hauptsächlich mit geometrischen Verfahren bewegt. Nur Partikel der dritten Ebene werden physikalisch basiert animiert. Da kaum Falten entstehen, müssen auch keine Selbstkollisionen behandelt werden. Die Methode zur Bekleidungsanimation, die in [RC02] beschrieben wird, fußt ebenfalls auf dem Vorhandensein eines menschlichen Körpers. Die Partikel werden ebenfalls hauptsächlich von der Bewegung des Körpers beeinflusst. Zusätzlich wird ein explizites Euler Integrationsschema benutzt, um die Partikel untereinander agieren zu lassen. Da der Hauptteil der Bewegung durch den Körper festgelegt ist, bleibt das Verfahren auch für große Zeitschritte stabil und es kann Kleidungsstücke in Echtzeit animieren. Auch hier werden keinerlei Selbstkollisionen behandelt. Für ein allgemeines System zur Animation von textilen Materialien können die beiden Verfahren nicht verwendet werden, da beide in ganz erheblichem Maße ein Objekt auf dem der Stoff aufliegt benötigen.

2.1.2 Virtuelle Bekleidung

Der Schritt von einzelnen Stoffstücken auf virtuelle Bekleidung erfordert einigen Aufwand. Als Grundlage dienen die Schnittteile, die in einem herkömmlichen CAD-System erstellt werden. Fügt man den Schnittteilen Informationen über die Nähte hinzu, können sie an einem virtuellen Menschen positioniert und dort vernäht werden. Hierfür existieren interaktive Methoden [VMT97a, VMT00b], in denen der Anwender die Schnittteile manuell vorpositioniert. Danach können sie vernäht werden, was automatisch abläuft. Für computeranimierte Filme ist diese Vorgehensweise durchaus tragbar [Vis01].

Möchte man aber eine virtuelle Anprobe an seinem virtuellen Gegenüber durchführen, bei der mehrere Kleidungsstücke schnell hintereinander verarbeitet werden müssen, wird ein effizientes automatisches Verfahren benötigt [FGLW03]. Dieses Verfahren positioniert, ohne dass der Benutzer eingreifen muss, die einzelnen Schnittteile eines Kleidungsstücks mit Hilfe einer geometrischen Methode an einem 3D-Scan einer Person. Das Verfahren nutzt abwickelbare Flächen, wie z.B. Zylinder oder Kegel, die die Körperteile des Scans umgeben, um die Schnittteile darauf zu positionieren. Durch die Verwendung abwickelbarer Flächen wird gewährleistet, dass sich die Schnittteile nur minimal dehnen und somit gute Anfangswerte für die Simulation liefern. Größere Dehnung kann zu Problemen bei der numerischen Simulation führen. Das Verfahren wurde in [GFL03] erweitert,

um mehrere Kleidungsstücke gleichzeitig vorpositionieren zu können. Damit lässt sich beispielsweise ein Pullover über eine Hose ziehen.

Nach der Positionierung werden die Schnittteile noch vernäht. Hierfür kann man sogenannte virtuelle Gummifäden verwenden. Diese sorgen für Kräfte zwischen Paaren von Partikeln, die entlang einer Nahtkurve miteinander verschmolzen werden sollen.

2.2 Kollisionserkennung

Ein weiteres wichtiges Forschungsgebiet ist die Kollisionserkennung zwischen dem Stoff selbst und zwischen Stoff und anderen Objekten. Im Bereich der Stoff-Objekt Kollisionserkennung sind in den letzten Jahren einige Verfahren vorgestellt worden [MKE03, BFA02, VCMT95, Pro97, VMT00b]. Die vielversprechendsten für interaktive Anwendungen sind Bild-basierte und Voxel-basierte Techniken [ZY00, VSC01, MDDB01]. Diese Methoden tendieren dazu wesentlich schneller zu sein als klassische Ansätze, die Hüllkörperhierarchien benutzten [BW98].

2.2.1 Selbstkollisionen

Ganz wesentlich für die realistische Simulationen ist die Behandlung von Selbstkollisionen. Ein sehr robuster Algorithmus für die Behandlung von Kollisionen im Allgemeinen und Selbstkollisionen im Speziellen wurde von [BFA02] vorgestellt. Diese Methode modelliert auch unterschiedlich dicke Stoffe und erzeugt sehr beeindruckende Animationen. Allerdings erkauft man diese sehr exakte Kollisionsbehandlung mit hohem Rechenaufwand. Kürzlich wurden Methoden vorgestellt, die den Vorgang der exakten und kontinuierlichen Kollisionserkennung beschleunigen [GKJ+05, WB05]. Trotzdem sind diese Ansätze für interaktive Anwendungen nicht geeignet. Auch andere Verfahren, die die Krümmung von Stoff ausnützen [Pro97, VMT00b], werden häufig nicht verwendet, um nicht zu langsam zu sein.

In [BWK03] wird ein Algorithmus zur Behandlung von Kollisionen beschrieben, der auf einer globalen Analyse aller Überschneidung der Meshes basiert. Dadurch wird es möglich illegale Zustände, d.h. entstandene Selbstdurchdringungen aufzulösen. Da der Algorithmus recht aufwendig ist, eignet er sich hauptsächlich für die Produktion computeranimierter Filme. Zudem sind in diesem Anwendungsgebiet schon kleinste Fehler in den Meshes nicht tolerierbar.

2.2.2 Kollisionen

Für die Kollisionserkennung zwischen dem Stoff und anderen Objekten können klassische Verfahren verwendet werden. In diesem Bereich der Kollisionserkennung sind eine Reihe von Methoden vorgestellt worden. Für konvexe Polyeder sind Algorithmen entwickelt worden, die in erwarteter linearer Zeit arbeiten [GHZ99, Mir98]. Da die meisten relevanten Objekte konkav sind, wurden verschiedene Ansätze mit Hüllkörperhierarchien entwickelt. Dazu zählen

Axis-aligned Bounding Box Trees [vdB97], orientierte Bounding Boxen (OBB-Trees) [GLM96], Sphere-Trees [Hub96, PG95], diskret orientierte Polytope (k-DOPs) [KHM+98] und dynamisch ausgerichtete DOP-Trees [Zac98]. Eine neuere Hüllkörperhierarchie wurde von [Zac02] vorgeschlagen. Sie ist so schnell wie OBBTrees und DOP-Trees, benötigt aber weniger Speicher. Eine andere Repräsentation wird in [FUF06] gewählt. Ein Objekt wird mit einem hierarchischen sphärischen Distanzfeld repräsentiert. Das Verfahren eignet sich aber nur für die Kollisionserkennung zwischen starren Körpern. Im Kontext der physikalisch basierten Simulation von Stoff wurden spezialisierte hierarchische Algorithmen entwickelt [Pro97, VMT00b, BFA02, MKE03].

Am besten für interaktive Anwendungen geeignet erscheinen Bild-basierte bzw. Voxel-basierte Methoden. In [MPT99] wird z.B. jeder Voxel als Innen, Außen oder Oberfläche markiert. Da keine Hierarchie traversiert werden muss und keine Überlappungstests mit Primitiven (z.B. Dreiecken) erfolgen, ist der Algorithmus extrem schnell. Als Konsequenz wird in diesem Verfahren Genauigkeit gegen Geschwindigkeit abgewogen. So fehlen Information über die Oberflächennormale und den genauen Schnittpunkt. Um die Genauigkeit zu erhöhen haben [ZY00] vorgeschlagen zusätzlich die Primitiven in den Voxel-Gittern abzuspeichern. Damit lassen sich genaue Überlappungstest durchführen und man spart die Traversierungszeit.

Um das Überprüfen von Primitiven zu umgehen haben [MDDB01] vorgeschlagen, nur die Normale und den nächsten Punkte der Oberfläche in den Voxeln zu speichern. Diese Information kann für die Kollisionsbehandlung verwendet werden. Da die Oberfläche innerhalb der Voxel linearisiert wird, entstehen Ungenauigkeiten und insbesondere Unstetigkeiten zwischen den Voxeln. Schließlich haben Vassilev et al. [VSC01] eine Methode beschrieben, die die Rendering Hardware benutzt, um zwei Tiefenbilder und Geschwindigkeitsbilder des Objektes zu erzeugen. Es wird jeweils ein Bild für die Vorderseite und ein Bild für die Rückseite erzeugt. Diese Bilder werden dann für die Distanzberechnungen und die Kollisionsantwort verwendet. Der Algorithmus ist allerdings auf konvexe Objekte oder – im Fall von virtuellen Menschen – auf entsprechende Abbildungsrichtungen beschränkt. Zudem ist die Genauigkeit an Silhouetten der Objekte sehr gering.

2.3 Echtzeit Visualisierung

Ein echtzeitfähiges Simulations- und Visualisierungssystem erfordert eine spezielle Architektur des Szenegraphen um dynamische Szenen mit hoher Darstellungsqualität effizient rendern zu können. Bei vielen Architekturen beeinflussen sich Darstellungsqualität und Simulationsgeschwindigkeit sehr stark, d.h. je besser die Darstellung desto langsamer läuft die Simulation. Im schlimmsten Fall wird die Simulation so langsam, dass kein interaktives Arbeiten mit dem System mehr möglich ist.

Da die Simulation und das Rendering jeweils sehr viel Rechenzeit beanspruchen aber auch zwei getrennte Einheiten sind, liegt es nahe beides zu parallelisieren. Dies ist insbesondere vorteilhaft, da aktuelle Grafikkarten in der Lage sind, fast alle Berechnungen zur Erzeugung eines Bildes eigenständig durchzuführen. Dabei entsteht das Problem, dass jetzt mehrere Threads synchronisiert werden müssen. OpenGL Performer [RH94] war das erste System, welches bereits sehr früh threadsichere Datenhaltung unterstützte. Allerdings sind in diesem System das Rendering und der eigentliche Szenegraph sehr eng miteinander verknüpft.

In [RVR04] wird ein Konzept für einen threadsicheren Szenegraphen vorgeschlagen, das in OpenSG umgesetzt wurde. Daten im Szenegraph werden selektiv repliziert, wenn von verschiedenen Threads auf sie zugegriffen wird. Der Vorteil dabei ist, dass Daten nur gepuffert werden, wenn sie auch wirklich von mehreren Threads benötigt werden. Die Synchronisation der verschiedenen Threads erfolgt mit Hilfe sog. ChangeLists. Für die Anwendungsentwicklung nachteilig ist die Tatsache, dass bei diesem Konzept der Anwender bei jedem Zugriff auf den Szenegraph die ChangeLists selbst aktualisieren muss.

Zum Rendern von Schatten, die von Punktlichtquellen erzeugt werden, gibt es zwei grundlegend verschiedene Techniken, die interaktive Frameraten erlauben: Shadow Maps [Wil78] und Shadow Volumes [Cro77]. Shadow Maps arbeiten im Bildraum, daher entstehen häufig Artefakte aufgrund von Aliasing. Zur Lösung dieses Problems wurde kürzlich ein verbesserter Algorithmus vorgestellt [SCH03]. Schadow Volumes arbeiten hingegen im Objektraum und erzeugen präzise Schatten. Der Algorithmus lässt sich gut auf Graphik-Hardware umsetzen [EK02] und erlaubt somit sehr hohe Frameraten. Aber da in der Realität quasi nur Flächenlichtquellen existieren, sieht man selten scharf abgegrenzte Schatten. Deshalb werden neue Methoden benötigt, um Schatten mit weichen Kanten zu erzeugen. Einen Überblick über diese meist rechenaufwändigen Methoden gibt [HLHS03].

Kapitel 3

Ontologien für Bekleidung

Die Simulation von Bekleidung an virtuellen Charakteren bietet in vielen Bereichen neue Möglichkeiten. Für die Bekleidungsindustrie erschließen sich damit neue Formen der Produktanprobe und Produktpräsentation. So könnte der Kunde in digitalen Verkaufsräumen Textilien zunächst in einer virtuellen Modenschau begutachten und später an seinem eigenen Körperscan anprobieren. Im Bereich der Computeranimation virtueller Charaktere wird die Bekleidung der Protagonisten nicht mehr von Hand animiert, sondern nach den Vorgaben der Bewegung der Figur simuliert.

Existierende Simulationssysteme haben allerdings noch einen gravierenden Mangel, der die beschriebenen Szenarien noch nicht bzw. nur unter Einschränkungen ermöglicht: Der virtuelle Charakter muss manuell eingekleidet werden, da eine Simulation des „Anziehens“ viel zu aufwändig wäre. Daher werden zunächst die noch nicht vernähten Schnittteile des Kleidungsstückes mittels Benutzereingabe um den Avatar herum platziert. Erst jetzt kann die eigentliche Simulation beginnen, die die Schnittteile zusammen fügt und die Passform bestimmt.

Auch die geometrische Position der Schnittteile nach der manuellen Platzierung ist nicht optimal, da die gängigen Systeme die Schnittteile nur als ebene Polygone behandeln. Sie können nur verschoben oder gedreht werden und lassen sich deshalb nur sehr grob platzieren. Deshalb sind die Schnittteile noch vergleichsweise weit vom Körper entfernt. Je weiter aber die Schnittteile von ihrer endgültigen Lage entfernt sind, desto länger dauert aber die Simulation.

Zur Lösung dieser Problematik wird in diesem Kapitel ein neues Verfahren vorgestellt, dass ausgehend von einer Ontologie über den Kleidungsstücken ein automatisches Einkleiden von Avataren ermöglicht. Die Ontologie wird zunächst für die Schnittteile spezifiziert und danach auf Kleidungsstücke erweitert. Diese Ontologien stellen semantische Informationen bereit, die es zusammen mit einer Methode zur geometrischen Vorpositionierung erlauben, Schnittteile ohne Nutzereingaben an virtuellen Menschen zu platzieren.

3.1 Interaktionsfreies Einkleiden virtueller Menschen

In diesem Abschnitt wird ein neues Verfahren zum interaktionsfreien Einkleiden von virtuellen Menschen beschrieben. Hierzu werden zunächst die einzelnen Schnittteile mit Hilfe einer geometrischen Abbildung um die Person gewickelt. Zusammen mit einem anschließenden Vernähen durch eine physikalisch basierte Simulation, wird dadurch erstmals ein automatisches Anziehen von Avataren ermöglicht (siehe dazu Abbildung 3.1).

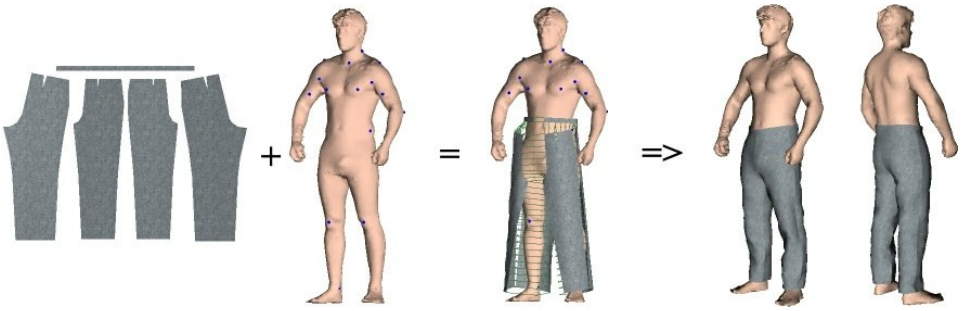


Abbildung 3.1: Links: Der 3D Laserscan einer Person mit Merkmalspunkten und Schnittteile einer Hose. — Mitte: Die Schnittteile nach der geometrischen Vorpositionierung und virtuellen Gummifäden. — Rechts: Das Kleidungsstück vernäht und fertig simuliert.

Als Basis dient eine abstrakte Beschreibung der Schnittteile aus denen das Kleidungsstück besteht. Dies erlaubt das Speichern in einer Bekleidungsdatenbank und eine einheitliche Handhabung für Kleidung verschiedener Größe und Art. In Kapitel 3.3 wird diese abstrakte Beschreibung zu einer Ontologie erweitert.

Dadurch, dass sich die Zeit, die für das interaktive Platzieren der Schnittteile in herkömmlichen Verfahren [VMT97a, VMT00b] benötigt wird, eliminieren lässt, kann eine Simulation von Bekleidung in wesentlich kürzerer Zeit durchgeführt werden. Daher erlaubt das neue Verfahren eine praktikable Bekleidungssimulation an individuellen 3D-Laserscans zur virtuellen Anprobe. Eine Implementierung des Verfahrens hat gezeigt, dass die Vorpositionierung in weniger als einer Sekunde durchgeführt werden kann. Die anschließende Simulation konvergiert sehr schnell und das fertige virtuelle Kleidungsstück kann nach wenigen Sekunden visualisiert werden.

Weiterhin ist das Verfahren so flexibel, dass es in jedem System zur physikalisch basierten Simulation von Bekleidung verwendet werden kann. Es liefert im Prinzip nur nicht-triviale Anfangswerte für die Bewegungsgleichung, die ja ein Anfangswertproblem formuliert. Diese Differentialgleichung kann dann mit einer Methode nach Wahl gelöst werden kann.

3.1.1 Ausgangssituation

Manuelles Einkleiden

Die Simulation von Stoff und ganzer Bekleidung wird schon seit geraumer Zeit im Bereich der Computergraphik erforscht. Es wurden sogar mehrere Systeme zur Modellierung und Simulation von Kleidungsstücken vorgeschlagen. Jedoch unterstützt keines eine automatische Vorpositionierung von Schnittteilen. Diese müssen per Benutzereingabe manuell an bzw. vor dem Avatar platziert werden.

Während die manuelle Vorpositionierung der Schnittteile in einigen Anwendungen kein Problem darstellt, ist diese Vorgehensweise für eine virtuelle Anprobe, bei der der Kunde virtuelle Kleidung an seinem digitalen Zwilling anprobiert, unakzeptabel. In diesem Szenario fehlt sowohl die Zeit als auch das Know-How um per Hand Schnittteile in 3D zu platzieren.

Das MIRACloth System, das am MIRALab an der Universität von Genf entwickelt wurde, stellt ein solches System dar [MIR05]. Es ermöglicht zum einen die klassische Konstruktion von Schnittteilen in 2D. Zum anderen können die Schnittteile danach in 3D um einen virtuellen Charakter platziert werden. Allerdings werden die Schnittteile nur auf zueinander senkrechten Ebenen vor oder hinter dem Avatar im Raum platziert. Die Schnittteile sind dann immer noch recht weit entfernt. Nach dieser manuellen Vorpositionierung werden die Schnittteile mit Hilfe von „virtuellen Gummifäden“ zusammengefügt. Dazu werden Gummifäden zwischen zwei korrespondierenden Dreiecken auf den Schnittteilen eingefügt. Dann beginnt die physikalisch basierte Simulation, die die Gummifäden solange verkürzt, bis die Nahtkurven zusammenstoßen [VCMT95, VMT97a, VMT00b].

In [MDDB01, DMB00] wird ein Simulationssystem beschrieben, in dem der Benutzer die Vorpositionierung in Echtzeit und direkt am Avatar vornehmen kann. Dazu wickelt er das Schnittteil mit einem geeigneten 6DOF Eingabegerät von Hand um den Avatar. Die Bewegung des Schnittteils wird durch ein vereinfachtes physikalisches Modell berechnet. Liegt das Schnittteil in der richtigen Position, muss der Benutzer noch Punkte auf dem Schnittteil definieren, die miteinander vernäht werden sollen. Insgesamt ein sehr aufwändiger Prozess.

In [Vas00] wird ebenfalls ein System vorgestellt, mit dem virtuelle Charaktere bekleidet werden können. Das Hauptaugenmerk der Autoren liegt auf der Vorstellung ihres Simulationsmodells und der Kollisionserkennung zwischen dem Kleidungsstück und dem Avatar. Über das eigentliche Anziehen wird nur gesagt, dass die Schnittteile am Körper positioniert werden. Wie dies geschehen soll, bleibt ungeklärt.

Das Anfangswertproblem

Bei der Simulation von Stoff, der als Partikelsystem modelliert wird, erhält man ein System gekoppelter gewöhnlicher Differentialgleichungen. Die Bahnen der Partikel werden nach den Gesetzen der Newton’schen Mechanik berechnet, indem man das Anfangswertproblem löst, dass durch die Differentialgleichungen (DGL) und

den *Anfangswerten* für die Positionen und Geschwindigkeiten der Partikel gegeben ist.

Das Hauptproblem bei der numerischen Lösung dieses Anfangswertproblems ist, dass die DGLen ein *steifes System* bilden [PTVF92]. Dies resultiert aus der Tatsache, dass bei textilen Materialien, die Kräfte, die bei Zug auftreten, wesentlich höher sind als die beim Biegen oder Scheren. Zusätzlich steigen die Kräfte bei Zug zunächst nur langsam, nach Erreichen einer gewissen Schwelle aber sehr schnell sehr stark an. Diese Schwelle ist relativ klein für gewebte Stoffe und wesentlich höher für Gestricke [EW99].

Beim Errechnen von Anfangswerten, sollten daher die Positionen der Partikel so gewählt werden, dass die internen Kräfte im Material vergleichsweise klein sind. Daraus folgt, dass die Abstände von mit Strukturfedern verbundenen Partikeln ungefähr genauso groß sein müssen, wie in Ruhelage. Da Stoff leicht gebogen werden kann, entstehen dabei auch nur geringe Kräfte, die keine numerischen Probleme bereiten.

Das Errechnen von Anfangswerten, die obigen Anforderungen genügen, ist im Allgemeinen schwierig. Daher verwenden die in Abschnitt 3.1.1 beschriebenen Systeme zur Simulation von Bekleidung nur planare und undeformierte Schnittteile als Anfangswerte. In diesem Fall gibt es keine internen Kräfte, die behandelt werden müssen.

3.1.2 Anforderungen an den Avatar

Für das Einkleiden wird ein geeigneter Avatar benötigt, da das Verfahren nicht beliebige Avatare verarbeiten kann. Zunächst wird davon ausgegangen, dass der Laserscan einer Person vorhanden ist. Dieser muss in einer Körperhaltung stehen, die es erlaubt die Schnittteile am Körper zu vernähen. Diese Haltung ist ähnlich wie bei einem echten Schneider, der Maß nimmt oder einen fertigen Anzug absteckt. Zusätzlich müssen markante Punkte, wie das Knie oder die Hüfte, am Körper identifiziert sein, damit die Kleidung korrekt platziert werden kann.

Körperhaltung

Das vorgeschlagene Verfahren zur Vorpositionierung stellt einige Anforderungen an die Haltung des Avatars. Die Arme dürfen nicht am Körper anliegen, da dann die Hüllflächen von Arm und Oberkörper besser aneinander passen. Dies ist wichtig für das Vernähen von Schnittteilen, die auf unterschiedlichen Hüllflächen liegen. Außerdem erleichtert diese die Segmentierung des Avatars.

Die Arme und Beine müssen gestreckt sein und die Person muss aufrecht stehen, damit eng anliegende Hüllflächen generiert werden können. Da die Hüllflächen abwickelbare Flächen sein müssen, liegen diese nur eng an, wenn die Körperteile gestreckt sind. So lässt sich beispielsweise um einen angewinkelten Arm kein Zylinder legen, der auch nur annähernd den Durchmesser des Oberarms hat.

Schließlich dürfen sich die Oberschenkel unterhalb des Schrittes nicht berühren, da ansonsten keine Schnittteile mehr an diese Stelle gebracht werden können, bzw. das Vernähen mit Vorder- und Rückteil unmöglich wird. Dies wird am besten erreicht, wenn die Beine leicht gespreizt sind.

Diese Haltung muss von der jeweiligen Person beim Scannen eingenommen werden, wenn eine nachträgliche Animation nicht durchführbar ist. Ermöglicht hingegen die Repräsentation des Avatars das Bewegen der Gliedmaßen, kann die Haltung im Nachhinein verändert werden.

Merkmalspunkte

Merkmalspunkte sind ausgezeichnete Punkte auf einer Oberfläche, die deren Form charakterisieren bzw. besondere Stellen markieren. Damit erhält man eine Oberfläche, die um symbolische Information erweitert wurde. Anhand dieser symbolischen Informationen lässt sich z.B. eine Segmentierung vornehmen. Eine Methode zum Auffinden von Merkmalspunkten an gescannten Avataren wird in [DDBT99] beschrieben. Dabei wird davon ausgegangen, dass nicht nur eine Punktwolke sondern die Oberfläche des Avatars gegeben ist. Anhand verschiedener Metriken und Heuristiken werden die Merkmalspunkte auf dieser Oberfläche bestimmt. Weitere Methoden zur Bestimmung von Merkmalspunkten finden sich in [SMT03, ACP03].

Für die korrekte Platzierung von Schnittteilen mit Hilfe des vorgeschlagenen Verfahrens sind Merkmalspunkte am Avatar unentbehrlich. Anhand der Merkmalspunkte werden Positionen der Schnittteile relativ zum Körper festgelegt. Anhand dieser Korrespondenzen kann beispielsweise ein Rock richtig herum und in der korrekten Höhe angezogen werden.

Bezeichner	**Lage an der Figurine**
armpit	vordere Achselhöhle
elbow (l,r)	Ellenbogen
biceps (l,r)	höchster Punkt des Bizeps
neck front	vorderer Übergang Nacken Hals
neck back	hinterer Übergang Nacken Hals
wrist (l,r)	Handgelenk Innenseite
nipple (l,r)	Brustwarze
knee (l,r)	Knie (vordere Mitte)
ankle inseam (l,r)	Innenseite Fußknöchel
waist to hip point (l,r)	seitlich zwischen Hüfte und Taille
hip girth point	Punkte am Rücken auf Höhe der Hüfte

Tabelle 3.1: Eine Auswahl der verwendeten Merkmalspunkte. Linke und rechte Punkte sind zusammengefasst.

Die in dieser Arbeit verwendeten Laserscans, wurden von der Human Solutions GmbH [Hum06] im Rahmen des Virtual Try-On Projektes [DTE+04] erstellt und mit einem proprietären Verfahren mit Merkmalspunkten versehen. In Tabelle 3.1 sind einige der verwendeten Merkmalspunkte aufgeführt.

3.1.3 Digitale Repräsentation von Kleidung

Für das automatische Einkleiden wird eine digitale Repräsentation eines Kleidungsstückes benötigt. Ein Kleidungsstück besteht im Wesentlichen aus mehreren miteinander vernähten Schnittteilen. Zusätzlich sind eventuell noch Knöpfe, Reißverschlüsse u.ä. vorhanden. Die Modellierung letzterer wird im Folgenden nicht betrachtet. Es genügt, die Nähte und Schnittteile eines Kleidungsstückes zu beschreiben.

Schnittteile

Der Rand eines Schnittteils lässt sich durch Angabe einer Folge von Punkten beschreiben. Man könnte ihn auch mit Hilfe von Splines darstellen, um eine glatte Kurve zu erhalten. In gängigen Textil-CAD Systeme, wie z.B. cad.assyst [ASS06], werden Schnittteile auch als parametrische Kurve modelliert. Beim Export aus dem System wird jedoch meist ein Polygonzug ausgegeben. In Abbildung 3.2 sind alle Schnittteile eines Godetrockes dargestellt. Die Randkurven liegen bereits als Polygonzug vor und wurden im Inneren texturiert.

Abbildung 3.2: Alle Schnittteile eines Godetrockes mit Textur und Randkurven.

Für eine konsistente Definition der Nähte wird eine Orientierung des Polygonzugs benötigt. Hierzu kann angenommen werden, dass die Schnittteile beim Export nach folgender Regel orientiert werden:

- Betrachtet man den Polygonzugs des Schnittteils von oben, so muss diese Seite des Schnittteils am Avatar ebenfalls sichtbar sein. Des Weiteren muss die Folge der Punkte im Polygonzug im Uhrzeigersinn durchlaufen werden.

Die Orientierung ergibt sich somit aus der Reihenfolge der Punkte auf dem Polygonzug.

Nähte

Jede einzelne Naht wird durch die Angabe von zwei Teilstücken von Randkurven, die im folgenden *Nahtkurven* genannt werden, beschrieben. Die zwei Nahtkurven werden später kompatibel miteinander vernäht. Sie gehören entweder beide zu einem Schnittteil oder liegen auf zwei verschiedenen Schnittteilen. Des Weiteren können sie sich durchaus in ihrer Länge unterscheiden. Darauf muss später beim Erzeugen der virtuellen Gummifäden Rücksicht genommen werden.

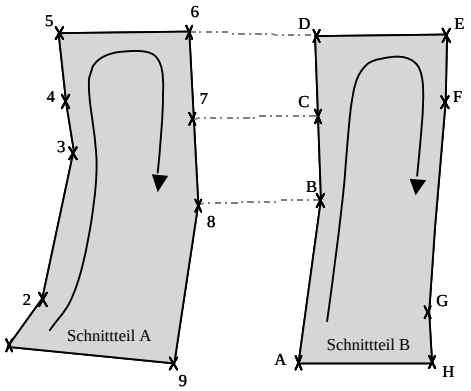


Abbildung 3.3: Vernähen zweier Schnittteile: Die Pfeile deuten die Orientierung der Schnittteile an. In Schnittteil A verläuft die Nahtkurve von Punkt 6 über 7 nach Punkt 8. In Schnittteil B ist D der Startpunkt und B der Endpunkt. Man erkennt, dass die Nahtkurven bezüglich der Orientierung der Schnittteile in unterschiedliche Richtung verlaufen.

Die Nahtkurven werden durch ihre Anfangs- und Endpunkte definiert. Diese müssen Eckpunkte der Randkurve des zugehörigen Schnittteils sein. Hierbei ist darauf zu achten, dass der Anfangspunkt der Nahtkurve des ersten Schnittteils in der geordneten Folge der Punkte der Randkurve vor dem Endpunkt auftaucht. Daraus ergibt sich, dass im zweiten Schnittteil der Endpunkt vor dem Anfangspunkt steht. Dieser Vorgang wird in Abbildung 3.3 nochmals verdeutlicht. Geht man auf diese Weise vor, sind die Nahtkurven eindeutig festgelegt. Außerdem verhindert man, dass der Algorithmus zum Vernähen die Punkte $6, 5, \ldots, 9, 8$ mit den Punkten $D, E, \ldots, A, B$ verbindet.

Kompatible Triangulierung von Schnittteilen

Beim Vernähen von zwei Schnittteilen werden unterschiedlich lange Nahtkurven miteinander verbunden. Die Schnittteile sind bereits trianguliert. Um die korrespondieren Punkte auf den Nahtkurven zu finden, wird die Länge der beiden Kurven normiert. Danach können Punkte, die den gleichen normierten Abstand vom Startpunkt aus haben, später mit einem Gummifaden miteinander verbunden werden. Betrachtet man die Nahtkurven als Parameterkurven über dem Einheitsinter-

vall, kann man auch sagen, dass man Punkte gleicher Bogenlänge miteinander verbindet. Der Algorithmus beginnt an den beiden Startpunkten und läuft dann auf beiden Seiten die Nahtkurve ab. Dabei fügt er, sobald er auf einen Eckpunkt der Triangulierung stößt, einen Gummifaden ein. Zuletzt werden die beiden Endpunkte der Kurven miteinander verbunden.

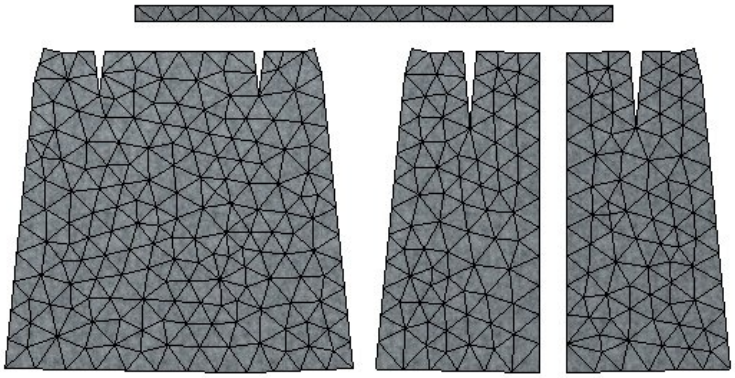


Abbildung 3.4: Schnittteile eines Rocks. Die Triangulierung ist kompatibel mit den Nahtkurven, daher müssen beim Vernähen der Schnittteile keine zusätzlichen Vertices eingefügt werden.

Es ist ausreichend, nur Punkte aus der Triangulierung miteinander zu verbinden, da die Schnittteile sehr fein und gleichmäßig trianguliert sind und damit die maximale Kantenlänge beschränkt ist. Daraus folgt, dass genügend Gummifäden generiert werden. Findet der Algorithmus keinen Punkt auf der gegenüberliegenden Seite, so wird dort ein neuer Punkt in das Netz eingefügt und das entstehende Viereck geteilt. Bei der Überprüfung, ob auf der anderen Seite ein passender Punkt liegt, wird eine gewisse Abweichung akzeptiert. Ansonsten würden allein aus numerischen Gründen zu viele Gummifäden eingefügt werden. Ist auf der anderen Seite ein Punkt in der Nähe vorhanden, so wird dieser verbunden und kein neuer Punkt erzeugt.

3.1.4 Zweistufiges Einkleiden von virtuellen Menschen

Im Folgenden wird ein neues Verfahren zum Einkleiden von virtuellen Menschen beschrieben, das in zwei Schritten arbeitet. Zuerst erfolgt eine rein geometrische Vorpositionierung und danach die Endpositionierung mit Hilfe einer Textilsimulation.

Da jeder Mensch eine andere Körperform hat, besteht das Problem darin, dass für jeden neuen Avatar, die Kleidung von neuem angezogen werden muss. Daher verwenden bisherige Systeme eine manuelle Positionierung der Schnittteil und überlassen das Problem dem Anwender.

Zur automatisierten Lösung des Problems, wird eine geeignete abstrakte Beschreibung eines Kleidungsstückes benötigt. Einige dieser Informationen sind bereits in den digitalen Kleidungsstücken vorhanden (Schnittteile mit ihren Randkurven, Nahtinformationen und Orientierung). Allerdings reicht dies zum Positionieren der Schnittteile noch nicht aus, da noch nicht klar ist, wohin die Schnittteile bewegt werden sollen. Ein Lösungsansatz wäre es, die Kleidungsstücke zu klassifizieren (Rock, Hose, etc.) und dann für jede Klasse ein eigenes Verfahren zu entwickeln. Problematisch sind dann aber Kleidungsstücke, die zu einer neuen Klasse gehören.

Um unabhängig von der Art der Kleidungsstücke zu sein, sieht das neue Verfahren einen allgemeineren Ansatz vor. Es werden *relative Positionen* der Schnittteile zum Körper des Avatars gespeichert. Hierzu wird der Avatar segmentiert und die Schnittteile den einzelnen Segmenten zugeordnet. Innerhalb der Segmente liefern die Merkmalspunkte die nötigen Korrespondenzen.

Geometrische Vorpositionierung

Die geometrische Vorpositionierung liefert die Anfangswerte, sprich die Positionen der Schnittteile, für die Simulation. Damit die Schnittteile sinnvoll weiterverarbeitet, d.h. vernäht und simuliert werden können, wird eine Reihe von Anforderungen an diese Anfangswerte gestellt.

Es muss darauf geachtet werden, dass die Simulation durchgeführt werden kann und dass sie konvergiert. Dazu sind im Wesentlichen vier Aspekte zu beachten:

1. Keine allzu starke Dehnung der Strukturfedern.

2. Schnittteile müssen außerhalb des Avatars liegen.

3. Keine Durchdringung zweier Schnittteile.

4. Richtige Position am Avatar.

Die obigen Punkte sind für einen korrekten Ablauf erforderlich. Zusätzlich lassen sich aber noch weitere Anforderungen definieren, die die Simulation schneller konvergieren lassen. Die Simulation sorgt für das Vernähen und schließt die Lücke zwischen dem Schnittteil und der Oberfläche des virtuellen Menschen. Je näher das Schnittteil an dieser Oberfläche, bzw. an der Position, die es endgültig einnehmen soll, liegt, desto weniger Arbeit muss die Simulation leisten. Da die Simulation im Vergleich zur Vorpositionierung langsamer verläuft, ergibt sich daraus, dass die Schnittteile nach der Vorpositionierung geometrisch möglichst nah am Avatar liegen müssen.

Ein Verfahren, dass diese Anforderungen erfüllt wird in Kapitel 3.2. Zunächst wird jedoch erst einmal der zweite Schritt beim Einkleiden beschrieben.

Physikalisch basierte Endpositionierung

Nach ihrer Vorpositionierung werden die Schnittteile mittels einer physikalisch basierten Simulation in ihre endgültige Lage gebracht. Um die Schnittteile miteinander zu verbinden werden virtuelle Gummifäden verwendet [VCMT95]. Diese Gummifäden erzeugen Kräfte zwischen den verbundenen Partikeln, die dafür sorgen, dass sich die Partikel annähern und schließlich verschmolzen werden können. Während der Simulation werden auch Selbstkollisionen und Kollisionen behandelt. Dadurch lassen sich anfängliche Durchdringungen auflösen, die in manchen Fällen vorkommen können.

Besonders effizient kann das Vernähen mit denen in dieser Arbeit vorgestellten Algorithmen zur Simulation und Kollisionserkennung durchgeführt werden. Es sei aber an dieser Stelle nochmals betont, dass das Verfahren auch mit herkömmlichen Methoden kompatibel ist. Man kann also einen bestehenden Simulator um ein Modul zum Einkleiden erweitern.

3.2 Geometrische Vorpositionierung

In diesem Kapitel wird ein Verfahren zur geometrischen Vorpositionierung beschrieben, das die Anforderungen aus Abschnitt 3.1.4 gerecht wird.

Hierzu werden abwickelbare Flächen zur Abbildung der Schnittteile aus der Ebene in den Raum verwendet. Da abwickelbare Flächen längentreu in die Ebene abgewickelt werden können, haben benachbarte Partikel des Schnittteils in der Ebene und auf der Fläche die gleiche Entfernung voneinander. Daraus folgt, dass die Strukturfedern nicht gedehnt sind.

Durch die Verwendung geeigneter Hüllflächen, die außerhalb des Avatars liegen, lässt sich schließlich auch der zweite Punkt erfüllen. Die Hüllflächen lassen sich leicht zu Bounding Volumes erweitern, die einen bestimmten Körperteil enthalten. In Abbildung 3.5 rechts ist ein Querschnitt durch ein solche Hüllfläche dargestellt.

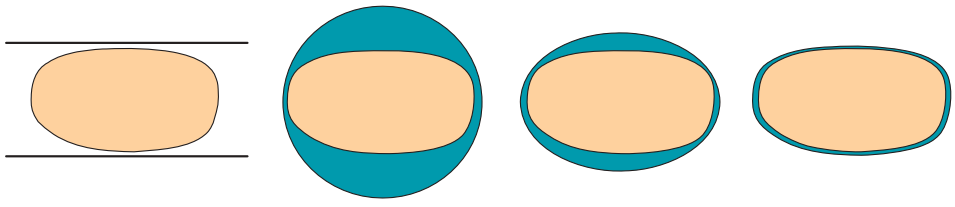


Abbildung 3.5: Querschnitte eines Körpersegments und mehrere umgebende Hüllflächen. — Links: Ebene Flächen, die in herkömmlichen Verfahren verwendet werden. — Mitte links: Kreiszylinder. — Mitte rechts: Elliptischer Zylinder. — Rechts: Das neue Verfahren, mit einer Hüllfläche um die konvexe Hülle des Körpersegments.

Wie die Anforderung der letzten beide Punkte eingehalten werden kann, wird in Kapitel 3.2.2 besprochen.

3.2.1 Segmentierung und Hüllflächen

Segmentierung des Avatars

Für die Vorpositionierung wird der Avatar in mehrere zum Teil überlappende Segmente unterteilt. Für jedes Segment wird später eine eigene Hüllfläche erzeugt und die Schnittteile um diese herum gewickelt. Die Segmente sind so gewählt, dass sich einzelne Schnittteile einem Segment zuordnen lassen. Folgende Segmente werden benötigt:

- linkes, rechtes Bein
- linker, rechter Arm
- unterer, oberer Torso
- Hals
- Unterkörper (unterer Torso und Beine)
- Torso (unterer und oberer Torso)
- linker Unterkörper (linkes Bein und linke Seite des unteren Torsos)
- rechter Unterkörper (rechtes Bein und rechte Seite des unteren Torsos)

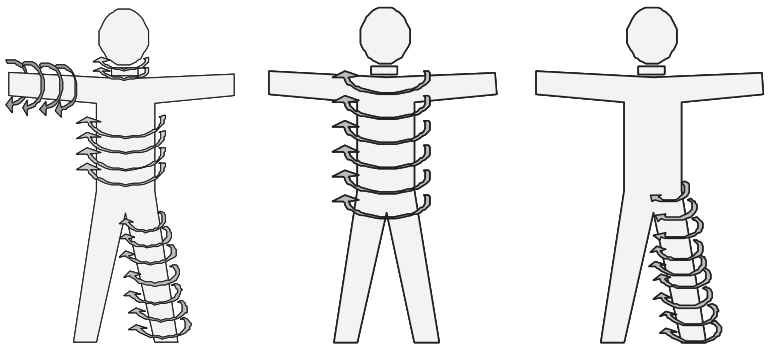


Abbildung 3.6: Links: Hüllflächen um den Hals, Arm, unteren Torso und ein Bein. Mitte: Hüllfläche des Torsos. — Rechts: Hüllfläche um den rechten Unterkörper.

In Abbildung 3.6 ist die Lage der einzelnen Hüllflächen skizziert. Für einige spezielle Kleidungsstücke kann die Unterteilung noch verfeinert werden. Beispielsweise kann man für Handschuhe zusätzlich noch Segmente für die einzelnen

Finger und die Hand selbst definieren. Das Verfahren ist so allgemein, dass es hierfür einfach erweitert werden kann.

Die Information welches Schnittteil zu welchem Segmente gehört, wird vorab in der Bekleidungsdatenbank gespeichert.

Abwickelbare Flächen

Für die Vorpositionierung werden Flächen gesucht, die längentreu in die Ebene abgebildet werden können. Längentreu bedeutet, dass alle Linien nach der Abbildung die gleiche Länge wie vorher haben. Die Vorpositionierung macht sich die Umkehrung dieser Abbildung zu Nutze und bildet Schnittteile aus der Ebene längentreu auf diese dreidimensionale Fläche ab. Dabei bleibt der Abstand aller Partikel gleich, d.h. die Strukturfedern zwischen den einzelnen Partikel werden nicht gedehnt.

Die *Gaußsche Krümmung* beschreibt die Krümmung einer Fläche in einem Punkt. Zur Erläuterung der Gaußschen Krümmung werden zunächst kurz einige andere Begriffe eingeführt. Die Krümmung einer ebenen Kurve in einem Punkt ist definiert als das Reziproke des Radius des Krümmungskreises, der an diesem Punkt anliegt. Je nachdem ob dieser Kreis rechts oder links der Kurve liegt, ist die Krümmung positiv oder negativ.

Die *Normalkrümmung* ist die Verallgemeinerung des Krümmungsbegriffs auf Flächen. Sei **F** eine Fläche und **v** ein Richtungsvektor parallel zur Tangentialebene eines Punktes $\mathbf{p} \in \mathbf{F}$. Schneidet man **F** mit der Fläche, die durch **v** und **p** und der Normalen der Tangentialebene von **F** aufgespannt wird, erhält man eine Kurve. Die Krümmung dieser Kurve entspricht der Normalkrümmung von **F** im Punkt **p** in Richtung **v**.

Betrachtet man die Normalkrümmungen in allen Richtungen, gibt es einen maximalen und einen minimalen Wert. Diese beiden Krümmungen heißen Hauptkrümmungen. Die Gaußsche Krümmung ist das Produkt der beiden *Hauptkrümmungen*. Ist die Gaußsche Krümmung 0, bedeutet dies, dass die Fläche in einer Richtung eben ist.

Flächen mit unterschiedlicher Gaußscher Krümmung können nicht längentreu ineinander übergeführt werden [Lip80]. So kann z.B. eine Kugel mit Radius r und damit $K = \frac{1}{r^2}$ nicht in die Ebene, für die $K = 0$ gilt, abgewickelt werden. Daraus folgt, dass die gesuchten Flächen verschwindende Gaußsche Krümmung besitzen müssen, da für die Ebene $K = 0$ gilt.

Eine Fläche, die an allen Punkten ein Gaußsche Krümmung von 0 hat, wird *abwickelbare Flächen* genannt. Einfache Beispiele für abwickelbare Flächen sind Zylinder und Kegel. In der Computergraphik werden abwickelbare Flächen häufig verwendet, beispielsweise zum Texture Mapping und für die geometrische Modellierung [BVIG91, SF96].

Bestimmung von Hüllflächen

Für jedes Körpersegment wird eine eng umschließende abwickelbare Hüllfläche generiert. Hierfür wird zunächst die Hauptachse des Segments bestimmt. Danach wird das Segment entlang der Hauptachse auf eine Ebene projiziert. Die extrudierte konvexe Hülle der Projektion ergibt dann die minimale Hüllfläche um das Körpersegment.

Im ersten Schritt wird die Oberfläche uniform abgetastet. Dazu werden alle Vertices der Dreiecke verwendet und zusätzlich weitere Vertices mittels Subdivision von großen Dreiecken erzeugt. Dies liefert eine Punktwolke $\mathbf{P}$, deren Hauptachse als Lösung eines Least-Square-Fit berechnet wird. Sei hierzu $\mathbf{l}(\lambda) = \mathbf{s} + \lambda \cdot \mathbf{t}$ mit $\mathbf{s}, \mathbf{t} \in \mathbb{R}^3$ und $\lambda \in \mathbb{R}$ die Achse die bestimmt werden soll und $\mathbf{p}_i \in \mathbb{R}^3, i \in 1, \ldots, N$ die gesampelten Punkte. Um $\mathbf{l}$ an die Punkte $\mathbf{p}_i$ anzupassen wird

$$E(\mathbf{s}, \mathbf{t}) = \sum_{i=1}^{N} \left(\mathbf{s} + \left(\frac{\langle \mathbf{t}, \mathbf{p}_i - \mathbf{s} \rangle}{\langle \mathbf{t}, \mathbf{t} \rangle} \right) \mathbf{t} - \mathbf{p}_i \right)^2 \tag{3.1}$$

minimiert.

Danach wird $\mathbf{l}$ noch durch zwei Ebenen begrenzt, um später die Hüllfläche nach oben und unten abzuschließen. Diese beiden Ebenen sollen möglichst dicht an der Punktwolke liegen. Unter den Punkten von $\mathbf{P}$ gibt es einen Punkt $\mathbf{p}_{min}$, für den λ mit

$$\lambda = \frac{\mathbf{t} \cdot (\mathbf{p} - \mathbf{s})}{\mathbf{t} \cdot \mathbf{t}} \quad (\forall \mathbf{p} \in \mathbf{P}) \tag{3.2}$$

minimal ist. Analog gibt es einen Punkt $\mathbf{p}_{max}$ der 3.2 maximiert. Begrenzt man $\mathbf{l}$ durch zwei zur Geraden senkrechte Ebenen die jeweils durch $\mathbf{p}_{min}$ und durch $\mathbf{p}_{max}$ verlaufen, erhält man die gesuchte Gerade $\mathbf{l}'$.

Danach werden die $\mathbf{p}_i$ entlang $\mathbf{l}'$ in eine Ebene projiziert und dort die zweidimensionale konvexe Hülle der Punkte berechnet. Das Polygon, das sich am Rand der konvexen Hülle ergibt, dient als Basis für die Hüllfläche. Hierzu wird es an die beiden Enden $\mathbf{p}_{min}$, $\mathbf{p}_{max}$ von $\mathbf{l}'$ verschoben und man erhält die Polygone $\mathbf{b}_1$ und $\mathbf{b}_2$. Interpretiert man die Polygone als parametrisierte Kurven vom Grad 1 und verbindet man Punkte mit gleichem Parameterwert u auf $\mathbf{b}_1(u)$ und $\mathbf{b}_2(u)$ so ergibt sich die Hüllfläche als verallgemeinerter Zylinder.

Vergrößern der Hüllflächen: Meist ist es notwendig, dass die Hüllfläche vergrößert werden muss, damit die Schnittteile auf ihr platziert werden können. Hierzu werden die beiden Polygone $\mathbf{b}_1(u)$ und $\mathbf{b}_2(u)$ entsprechend expandiert. Durch die voneinander unabhängige Vergrößerung der Polygone kann die zylindrische Hüllfläche in einen Kegel transformiert werden.

Die Veränderung einer der beiden Polygone $\mathbf{b}(u)$ erfolgt anhand einer zentrischen Streckung. Hierzu wird der Schnittpunkt $\mathbf{c}$ zwischen der Hauptachse $\mathbf{l}'$ und der Ebene, in der $\mathbf{b}(u)$ liegt, gebildet. Sei σ der Streckungsfaktor, dann erhält man die gestreckte Kurve $\mathbf{b}_t(u)$ als

$$\mathbf{b}_t(u) = \sigma \cdot (\mathbf{b}(u) - \mathbf{c}) + \mathbf{c} \tag{3.3}$$

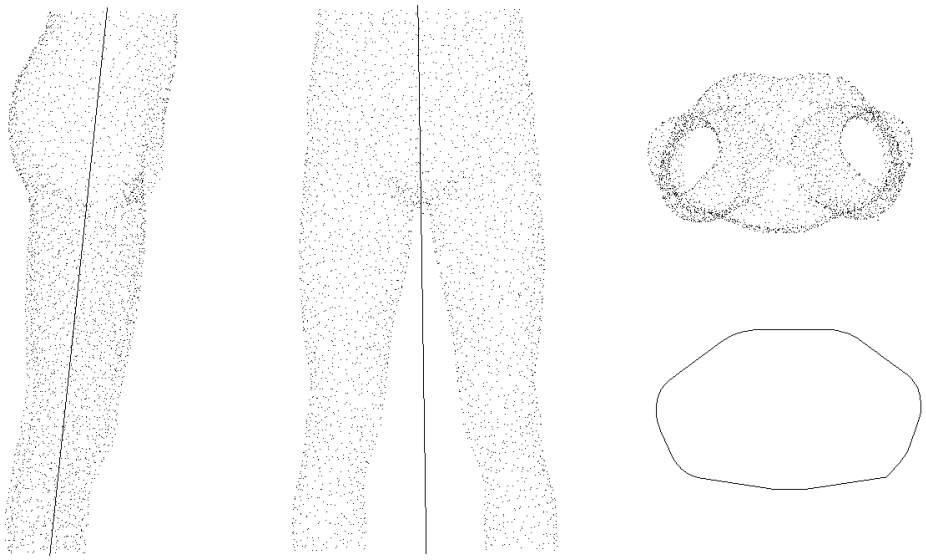


Abbildung 3.7: Links: Punktwolke eines Körperteils mit Hauptachse aus zwei Richtungen. — Rechts: Die Projektion der Punkte entlang der Hauptachse und der Rand der konvexen Hülle.

Dadurch, dass Punkte mit gleichem Parameterwert u immer noch durch eine Gerade miteinander verbunden sind, bleibt die Gaußsche Krümmung an allen Punkten der Hüllfläche 0 und somit ist $\mathbf{b}_t(u)$ ebenfalls eine abwickelbare Fläche.

Abbildung von $\mathbb{R}^2$ nach $\mathbb{R}^3$

Bei der geometrischen Vorpositionierung sollen Schnittteile so wenig wie möglich gedehnt werden, da ansonsten sehr hohe interne Kräfte im Material entstehen bevor der erste Simulationsschritt begonnen hat. Daher sind abwickelbare Flächen für die Vorpositionierung besonders geeignet, da die Schnittteile bei der Abbildung auf die Hüllfläche nicht gedehnt oder geschert werden. Auch der Flächeninhalt bleibt gleicht Die entstehenden Biegekräfte sind verhältnismäßig gering solange die Krümmung der Hüllfläche ebenfalls klein bleibt, was einfach zu erreichen ist. Daher verursacht die leichte Biegung auch keine numerischen Probleme für die anschließende physikalisch basierte Endpositionierung.

Herkömmliche Techniken zur Abbildung von abwickelbaren Flächen nach $\mathbb{R}^2$ verwenden Differentialgleichungen, die aufwendig gelöst werden müssen [SF96]. Da die Effizienz des gesamten Verfahrens stark von der Geschwindigkeit dieser Abbildung abhängt, wurde nicht diese Methode verwenden, sondern die Hüllfläche in ein Dreiecksnetz umgewandelt. Ausgehend von den einzelnen Dreiecken wird

jetzt die Hüllfläche abgewickelt. Dazu wird ausgenutzt, dass abwickelbare Flächen die Eigenschaft haben Winkel und Längen zu erhalten.

Beim Abwickeln startet man mit einem Dreieck und bildet es beliebig nach $\mathbb{R}^2$ ab. Danach wird eine Kante eines benachbarten Dreiecks t ebenfalls nach $\mathbb{R}^2$ abgebildet. Der neue Endpunkt des Dreiecks wird berechnet, indem der Winkel im Dreieck und die Länge der Kante erhalten bleiben (siehe hierzu Abbildung 3.8 (a)).

Abbildung 3.8: (a) Abwickeln von Dreiecken: Die Kante e wird nach $\mathbb{R}^2$ abgebildet, indem der Winkel zwischen ihr und der diagonalen Kante und auch ihre Länge erhalten bleiben ($\alpha = \beta$ und $|e| = |e'|$). — (b) Ein elliptischer Kegel und die Abwicklung in die Ebene.

Die letzte Kante von t ist somit schon festgelegt und man kann mit dem nächsten Dreieck fortfahren. Der Fehler, der durch die Diskretisierung in ein Dreiecksnetz entsteht, kann durch die Anzahl der verwendeten Dreiecke gut gesteuert werden. In der Praxis genügen einige hundert Dreiecke für einen ausreichend geringen Fehler. In Abbildung 3.8 (b) ist ein Beispiel für Abwicklung eines Kegels dargestellt.

Nachdem die Hüllfläche abgewickelt ist, kann die Abbildung $\psi : \mathbb{R}^2 \to \mathbb{R}^3$ definiert werden. Sie ist nur für Punkte innerhalb der abgewickelten Hüllfläche definiert. Um einen Punkt $\mathbf{p}$ abzubilden, wird zunächst das Dreieck t in dem er liegt bestimmt. Danach werden die baryzentrischen Koordinaten w_1, w_2, w_3 von $\mathbf{p}$ bezüglich t bestimmt. Wendet man diese Koordinaten auf die drei Eckpunkte $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ des korrespondierenden Dreiecks t' im $\mathbb{R}^3$ an, so ergibt sich der neue Punkte als

$$\mathbf{p}' = w_1\mathbf{p}_1 + w_2\mathbf{p}_1 + w_3\mathbf{p}_1 \ . \tag{3.4}$$

Die Lage von $\mathbf{p}'$ innerhalb von t' entspricht der Lage von $\mathbf{p}$ innerhalb von t. Somit wurde der Punkt aus der Ebene auf die Hüllfläche abgebildet.

Genauso kann man vorgehen, um Punkte von der Hüllfläche in die Ebene abzubilden. Allerdings ist dann der Test, ob der Punkt im Dreieck liegt und die Berechnung der baryzentrischen Koordinaten ein wenig aufwändiger.

Natürlich treten dadurch, dass der Stoff als triangulierte Fläche vorliegt und daher nur Vertices abgebildet werden, gewisse Approximationsfehler auf: Bei der beschriebenen Abbildung ψ ändert sich die Länge der Kanten, wenn die Hüllfläche

gekrümmt ist. Die Größe dieser Änderung bzw. des Fehlers hängt von den Kantenlängen der Dreiecke und dem Verlauf der Hüllfläche zwischen den Eckpunkten der Dreiecke ab. Aber diese Fehler verursachen nur geringe Dehnungen des Materials und sind daher kein Problem für die Simulation.

Weiterhin können einige Dreiecke den Körperteil durchdringen, da ja nur Punkte abgebildet werden. Dieses Problem kann jedoch einfach gelöst werden, indem die Hüllfläche ein wenig vergrößert wird, bevor die Abbildung ψ erfolgt.

Hierzu können die Kurven $\mathbf{b}_1(u)$ und $\mathbf{b}_2(u)$ entsprechend Gleichung 3.3 gestreckt werden. Der Faktor σ hängt dabei von der längsten Kante im Schnittteil und von der Krümmung der Hüllfläche ab. Eine genaue Bestimmung von σ ist allerdings nicht notwendig, da kleinere Durchdringungen von der Kollisionserkennung während der Simulation behandelt werden können. Außerdem würden die Hüllflächen bei einem größeren σ weiter vom Körper entfernt sein, was wiederum eine längere Rechenzeit in der Simulation verursacht.

3.2.2 Positionierung der Schnittteile

Im Folgenden wird ein Verfahren beschrieben, das die Schnittteile innerhalb der Hüllfläche in $\mathbb{R}^2$ so platziert, dass sie nach der Anwendung von ψ in korrekten Positionen um den Avatar herum liegen.

Dazu werden die Schnittteile zunächst mit Zusatzinformationen versehen. Diese legen beispielsweise die Orientierung der Schnittteile auf den Hüllflächen fest. Hierfür werden bestimmte Merkmalspunkte verwendet. Danach werden ausgehend von einem Schnittteil, alle anderen um dieses herum platziert, indem die Nahtinformationen ausgewertet werden. Abschließend werden die Abstände zwischen den Schnittteilen angepasst, so dass diese korrekt an Avataren unterschiedlicher Größe platziert werden können.

Annotieren der Schnittteile

Wie oben beschrieben, wird jedes Schnittteil einem Körperteil und damit der dafür erzeugten Hüllfläche zugeordnet. Es verbleibt die Aufgabe alle Schnittteile, die zu einer Hüllfläche gehören, korrekt auf dieser zu platzieren. Natürlich kann hierfür die zwei-dimensionale Hüllfläche verwendet werden. Danach kann die Abbildung ψ verwendet werden, um alle Schnittteile um das Körperteil zu wickeln.

Zunächst werden noch weitere Informationen zu den Schnittteilen hinzugefügt, damit diese automatisch auf der Hüllfläche platziert werden können. Folgende Informationen werden benötigt:

- Eine Orientierung für jedes Schnittteil (d.h. welche Seite ist sichtbar nach dem Anziehen).
- Das zugehörige Körpersegment.
- Ein Vektor, der in Richtung der Hauptachse des Körpersegments zeigt.

- Für ein Schnittteil: Die Position eines Merkmalspunktes in Relation zum Schnittteil.

Die letzten beiden Informationen lassen sich nur sinnvoll hinzufügen, wenn bekannt ist, wie das Schnittteil am Körper liegen soll. Dann geben diese Informationen die Lage auf der Hüllfläche vor. Die Daten können manuell zur Bekleidungsdatenbank hinzugefügt werden oder durch ein geeignetes Bekleidungs-CAD Programm erstellt werden. Dies muss nur einmal erfolgen, da die Informationen unabhängig von der Größe des Avatars sind. Daher stellt das manuelle Hinzufügen kein Hindernis beim interaktionsfreien Bekleiden dar.

Einige der Informationen erscheinen zunächst sehr spezifisch für die in diesem Kapitel beschriebene Methode zur geometrischen Vorpositionierung. Allerdings werden auch in der realen Welt oft Zusatzinformationen benötigt. Beispielsweise kann ein Rock um die Hüfte rotiert werden, was die Notwendigkeit einer Korrespondenz zu einem Merkmalspunkt unterstreicht. Diese Korrespondenz kann ein Mensch einfach erstellen. Auch der Vektor, der in Richtung der Hauptachse zeigt, wird beim realen Rock benötigt, ansonsten könnte man ihn nämlich einfach verkehrt herum anziehen. Allerdings wird hier die Information nur für ein Schnittteil benötigt, der Rest ist durch die Vernähung dann festgelegt.

Merkmalspunkte

Mit Hilfe der Merkmalspunkte, die auf der Oberfläche des Avatars liegen, können die Schnittteile auf den abgewickelten Hüllflächen platziert werden. Hierzu werden die Merkmalspunkte zunächst auf die Hüllfläche projiziert und danach kann mit Hilfe von ψ^{-1} die Lage der Merkmalspunkte in $\mathbb{R}^2$ bestimmt werden. Für gewöhnlich genügt hierfür ein einzelner Merkmalspunkt pro Hüllfläche. In Tabelle 3.2 sind die verwendeten Merkmalspunkte und die Zuordnung zu den Körperteilen, und somit auch zu den Hüllflächen, aufgeführt.

Merkmalspunkt	**Körperteil**
neck front	Hals
neck back	Hals, oberer Torso, Torso
biceps (l,r)	Arm
elbow (l,r)	Arm
wrist (l,r)	Arm
waist girth point	Unterkörper, Torso
hip girth point (l,r)	Unterkörper, Torso
nipple (l,r)	Oberkörper
ankle inseam (l)	linker Unterkörper, linkes Bein
ankle inseam (r)	rechter Unterkörper, rechtes Bein
knee (l,r)	Beine

Tabelle 3.2: Verwendete Merkmalspunkte und die Zuordnung zu den Körperteilen.

Die *Projektion eines Merkmalspunktes* $\mathbf{p}_f$ erfolgt, indem eingangs der nächste Punkt $\mathbf{p}_n$ zu $\mathbf{p}_f$ auf der Hauptachse des Körpersegments bestimmt wird. Dann wird der Strahl, der in $\mathbf{p}_n$ startet und auf den Merkmalspunkt zeigt, mit der Hüllfläche geschnitten. Da die Hauptachse in der Mitte des Körpersegments liegt, wird $\mathbf{p}_f$ auf die richtige Seite der Hüllfläche projiziert.

Die Anzahl der benötigten Korrespondenzen zwischen Schnittteilen und Merkmalspunkten hängt von der Anzahl der verwendeten Hüllflächen ab und somit von Art des Kleidungsstückes ab. Je mehr Körperteile vom Kleidungsstück überdeckt werden, desto mehr Hüllflächen müssen betrachtet werden. Pro Hüllfläche wird nur ein einziger Merkmalspunkt verwendet, wobei je nach Schnitt des Kleidungsstückes jeweils andere Merkmalspunkte besser geeignet sind.

Beispielsweise wird ein *Rock* um zwei verschiedene Hüllflächen gewickelt und daher zwei Korrespondenzen benötigt. Der Bund wird auf den unteren Torso abgebildet und die restlichen Schnittteile auf den Unterkörper. Diese Aufteilung ist sinnvoll, denn durch die Verwendung des unteren Torsos anstelle des Unterkörpers für den Bund, wird eine Vorpositionierung erreicht, die enger am Körper liegt. Für beide Korrespondenzen wird der *waist girth point*, der in der Mitte des Rückens liegt, benutzt. Für Hosen kann der Bund genauso behandelt werden, die Beinteile liegen aber auf dem linken und rechten Unterkörper, wobei das *knee* als Merkmalspunkt dient.

Platzieren der Schnittteile in $\mathbb{R}^2$

Beim Platzieren der Schnittteile wird mit dem Schnittteil angefangen, für das die relative Position zu einem Merkmalspunkt in der Datenbank gespeichert ist. Danach werden die anderen Schnittteile um dieses herum gelegt.

Eine allgemeine Lösung für dieses Problem zu finden, erscheint sehr schwierig, da es Kleidungsstücke gibt, deren Schnittteile einzelnen betrachtet höchst komplexe Formen haben. Daher wird an dieser Stelle ein vereinfachtes Verfahren vorgeschlagen, das einen weiten Bereich gängiger Kleidungsstücke abdeckt.

Zuerst werden die Schnittteile anhand ihrer Nahtinformationen sortiert. Dabei werden Schnittteile die von den Nähten aus gesehen adjazent zum bereits platzierten Schnittteil liegen, als Erste einsortiert. Danach die Schnittteile die zu diesen adjazent sind, usw. Danach wird die sortierte Liste abgearbeitet, indem Schnittteil für Schnittteil an die bereits platzieren angelegt wird, bis keine Schnittteile mehr in der Liste stehen. Geht man auf diese Weise vor, kann die Platzierung fehlschlagen, wenn die Schnittteile eine ungünstige Form besitzen. Allerdings arbeitet das Verfahren sehr effizient.

Das Anlegen der Schnittteile erfolgt dadurch, dass korrespondierende Teile der Randkurven zweier Schnittteile so nah wie möglich aneinander platziert werden. Nach dem Vernähen wird aus diesen beiden Teilen eine Naht, daher ist es wichtig diese nah beieinander zu platzieren. Man beachte, dass die zwei Ränder stark unterschiedliche Geometrien haben können. Dies liegt darin begründet, dass nur so

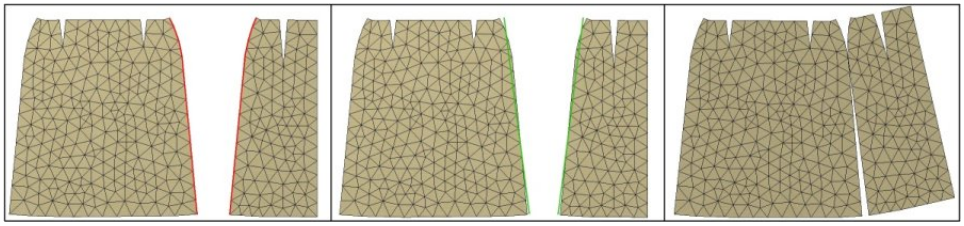


Abbildung 3.9: Links: Zwei Schnittteile eines Rocks. Die roten Linien werden später vernäht. — Mitte: Die grüne Regressionsgerade der Vertices am Rand. — Rechts: Die Schnittteile rotiert und aneinander gelegt.

eine gute Passform für ein Kleidungsstück erreicht werden kann. Die Nahtkurven passen im Dreidimensionalen gut aneinander, jedoch nicht in der Ebene.

Um die zwei Ränder aneinander zu legen, wird für jeden Rand die Regressionsgerade bestimmt (siehe dazu Abbildung 3.9 links). Das neue Schnittteil wird so rotiert, dass die beiden Geraden parallel liegen. Danach werden die beiden Geraden durch Translation des Schnittteils in Deckung gebracht. Schließlich wird das neue Schnittteile orthogonal zur Geraden hin- bzw. wegbewegt, bis keine Kollision der Schnittteile mehr erkannt wird (Abbildung 3.9 rechts).

3.2.3 Anpassen der Hüllflächen

In vielen Fällen wird die initiale Hüllfläche noch verändert, da ansonsten nicht alle Schnittteile auf ihr Platz finden. Dieses Problem tritt für gewöhnlich bei Kleidungsstücken wie Röcken auf, die viele Falten werfen. Wie in Abschnitt 3.2.1 beschrieben, werden die Hüllflächen durch zwei Kurven $\mathbf{b}_1(u)$ und $\mathbf{b}_2(u)$ gebildet. Diese können mit Gleichung 3.3 unabhängig voneinander vergrößert werden, wodurch einen verallgemeinerter Kegel entsteht. Aus der Vergrößerung der Kurven folgt auch eine Vergrößerung der abgewickelten Hüllfläche. Daher kann somit leicht die Größe iterativ geändert werden, bis alle Schnittteile hinein passen (siehe auch Abbildung 3.10).

Der Algorithmus startet mit den nach Abschnitt 3.2.2 platzierten Schnittteilen, die dicht nebeneinander liegen. Das linke Schnittteil in Abbildung 3.10 links, wird nach dem Aufwickeln mit dem rechten Schnittteil vernäht. Diese Naht wird hier nicht weiter berücksichtigt. Im Folgenden werden zwei Fälle unterschieden:

1. Die Schnittteile schneiden den Rand der Hüllfläche.
   In diesem Fall wird die Hüllfläche vergrößert. Dazu wird die ungefähre Ausdehnung der Schnittteile am oberen Rand und am unteren Rand berechnet, indem die Abstände der Nähte addiert werden. Diese Ausdehnungen werden mit den Bogenlängen von $\mathbf{b}_1(u)$ und $\mathbf{b}_2(u)$ verglichen, woraufhin die beiden Kurven entsprechend vergrößert werden.

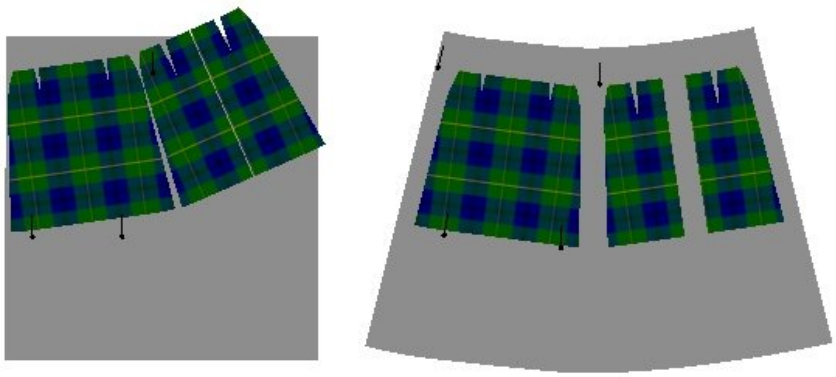


Abbildung 3.10: Links: Die initiale Hüllfläche und die platzierten Schnittteile. — Rechts: Die angepasste Hüllfläche.

2. Die Schnittteile sind weit vom rechten Rand der Hüllfläche entfernt.
   In diesem Fall wird der Abstand zwischen den Schnittteilen erhöht, indem die Regressiongeraden aus Abschnitt 3.2.2 nicht so dicht aneinander gelegt werden.

Die Anzahl der benötigten Iterationen bis die Hüllfläche angepasst ist, hängt stark von ihrer späteren Form ab. Je kegelförmiger, desto mehr Iterationen werden durchlaufen. Die Hüllfläche für den Rock wurde in sieben Iterationen angepasst, wohingegen die Schnittteile eines Hemds schon nach vier Iterationen hinein passen.

Begrenzung der Ausdehnung der Hüllflächen

Die Vergrößerung der Hüllflächen verursacht in einigen Fällen Probleme für die anschließende Simulation. Zum einen erhöht sich der Abstand der Schnittteile zum Körper und zum anderen können Kollisionen mit anderen Körperteilen entstehen. Die angepasste Hüllfläche für einen Godetrock [1] wäre beispielsweise ein weit geöffneter Kegel.

Um die Ausdehnung der Hüllflächen zu begrenzen, werden die erzeugenden Kurven $\mathbf{b}_1(u)$ und $\mathbf{b}_2(u)$ der Hüllfläche mit einer Sinusfunktion moduliert. Die Amplitude und die Anzahl der Schwingungen hängen dabei von der Bogenlänge der Kurve ab. Abbildung 3.11 zeigt eine Hüllfläche vor und nach der Modulation.

Die Hüllfläche, die sich durch die Modulation ergibt, ist wesentlich kompakter, besitzt aber ungefähr den gleichen Flächeninhalt wie die ursprüngliche Hüllfläche. Ein kleiner Nachteil dieser Methode ist die Tatsache, dass die Hüllfläche keine abwickelbare Fläche mehr ist. Allerdings bereitet dies keine Probleme für den Algo-

[1] Ein Godetrock ist ein sehr weiter Rock, der dementsprechend viele Falten wirft (siehe Abbildung 3.2 für dessen Schnittteile).

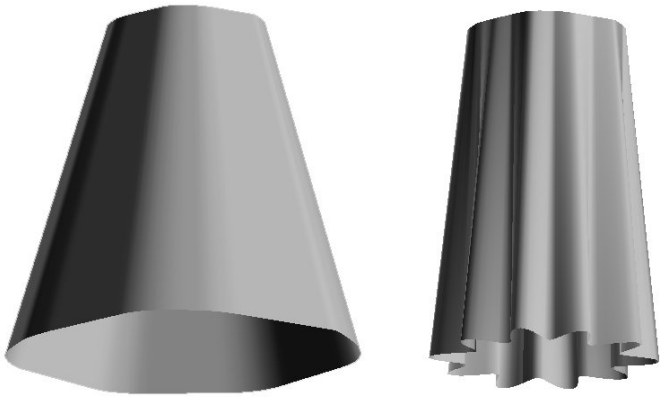


Abbildung 3.11: Zwei Hüllflächen mit gleicher Bogenlänge der erzeugenden Kurven $\mathbf{b}_1(u)$ und $\mathbf{b}_2(u)$. Die rechte Hüllfläche ist wesentlich kompakter als die Linke.

rithmus zum Abwickeln und die entstehenden Dehnungen der Schnittteile bleiben gering. Ein Ziel der geometrischen Vorpositionierung ist es, die Abstände der Vertices der Schnittteile möglichst konstant zu halten. Die Abstände ändern sich auch mit den modulierten Hüllflächen kaum. Ein Beispiel von modulierten Schnittteilen anhand eines Faltenrocks ist in Abbildung 3.12 dargestellt.

In Algorithmus 1 sind die Schritte der Anpassung der Hüllfläche übersichtlich zusammengefasst.

3.2.4 Virtuelle Gummifäden

Wird Bekleidung mit einem Partikelsystem simuliert, so können die Partikel auf unterschiedlichen Schnittteilen durch virtuelle Gummifäden miteinander verbunden werden. Korrespondierende Partikel ergeben sich anhand der Nahtinformation. Während der Simulation werden die Gummifäden verkürzt und die Schnittteile können virtuell vernäht werden. Für weitere Details sei auf [VMT00b, Kapitel 4.3] verwiesen. Die Idee der virtuellen Gummifäden wurde in der vorliegenden Arbeit ebenfalls verwendet. Zusätzlich wurde sie aber um die Verwendung von zusätzlichen Partikeln entlang der Gummifäden erweitert. Diese erlauben auch in schwierigen Fällen eine korrekte Simulation.

Die Gummifäden erzeugen Kräfte, die zwei Partikel zusammenführen. In einigen Fällen kann diese Methode jedoch versagen. Beispielsweise, wenn ein Schnittteil während der Vorpositionierung nicht ganz um das Körperteil gewickelt werden konnte (siehe hierzu Abbildung 3.13 oben). Daher wird die ursprüngliche Idee um Hilfspartikel erweitert, die die Gummifäden unterteilen. Wenn während der Simulation Kollisionen zwischen dem Hilfspartikel und dem Avatar überprüft werden, kann ein korrektes Vernähen garantiert werden (siehe Abbildung 3.13 unten). Hier-

Abbildung 3.12: Vorpositionierung eines Godetrockes mit modulierter Hüllfläche. Der Rock nach der physikalisch basierten Endpositionierung ist in Abbildung 4.24 dargestellt.

bei ist eine gute Platzierung der Hilfspartikel entscheidend. Dies wird im nächsten Abschnitt erläutert.

Während der Simulation wird die Distanz zwischen den Hilfspartikeln und den Partikeln der Schnittteile verringert. Fallen die Distanzen unter einen Schwellwert, kann der Hilfspartikel entfernt und die beiden anderen Partikel zusammengefasst werden. Sollte der Abstand einiger Partikel niemals unter dem Schwellwert liegen, so ist dies ein Zeichen dafür, dass das Kleidungsstück nicht passt. In diesem Fall kann der Anwender entscheiden, ob eine Vernähung erzwungen werden soll, d.h. die Kräfte zum Vernähen werden erhöht, um die internen Kräfte im Stoff zu überwinden. Allerdings werden dadurch die Schnittteile unnatürlich gedehnt, aber die Partikel könnten zusammengefasst werden. Eine bessere Wahl für den Anwender wäre es, ein größeres Kleidungsstück zu verwenden.

Erzeugung von Hilfspartikeln

Hilfspartikel müssen immer dann berechnet werden, wenn die direkte Verbindung zweier Partikel mit einem Gummifaden durch den Avatar läuft oder falsche Ergebnisse liefern würde. Um diese Überprüfung nicht durchführen zu müssen, wird

Algorithmus 1 : ANPASSEN DER HÜLLFLÄCHE UND DER ABSTÄNDE DER SCHNITTTEILE

Eingabe : Hüllfläche, platzierte Schnittteile
Ausgabe : Schnittteile, die innerhalb der Hüllfläche liegen

B = Hüllfläche
$\mathbf{B}_f = \psi^{-1}(\mathbf{B})$ (siehe Abschnitt 3.2.1)
repeat
  if *Schnittteile schneiden* $\mathbf{B}_f$ **then**
    Vergrößere $\mathbf{b}_1(u)$ und $\mathbf{b}_2(u)$ (siehe Formel 3.3)
    if *der Umfang von* **B** *übersteigt Schwellwert* **then**
      Moduliere $\mathbf{b}_1(u)$ und $\mathbf{b}_2(u)$ (siehe Abschnitt 3.2.3)
    end
    $\mathbf{B}_f = \psi^{-1}(\mathbf{B})$
  end
  else if *Ränder der Schnittteile weit weg vom Rand von* $\mathbf{B}_f$ **then**
    Vergrößere Abstand der Regressionsgeraden
    Platziere Schnittteile neu
  end
until *Schnittteile gleichmäßig verteilt in* $\mathbf{B}_f$

einfach für jeden Gummifaden ein Hilfspartikel erzeugt. Bei der Erzeugung lassen sich drei Fälle unterscheiden:

1. Die Schnittteile, die vernäht werden sollen, liegen auf der gleichen Hüllfläche.
   In diesem Fall wird die Position des Hilfspartikels zunächst auf der abgewickelten Hüllfläche bestimmt, indem er in die Mitte der direkten Verbindung der beiden zu vernähenden Partikel gesetzt wird. Die endgültigen Koordinaten ergeben sich dann durch Anwenden von ψ.

2. Die Naht verläuft entlang der Schulter.
   In diesem Fall kann nicht die Hüllfläche verwendet werden, da sie an dieser Stelle offen ist. Verbindet man zwei korrespondierende Partikel, liegt diese Linie im Avatar, wenn die Schnittteile etwas tiefer am Oberkörper liegen. Zur Platzierung der Hilfspartikel wird eine Schulterlinie erzeugt, die oberhalb des Avatars liegt. Die Hilfspartikel können dann entsprechend auf diese Linie projiziert werden.

3. Die Naht verbindet Schnittteile auf unterschiedlichen Hüllflächen.
   Hier genügt es einfach den Mittelpunkt der direkten Verbindungsline zu verwenden. Beispielsweise werde so Schnittteile des Arms mit denen des Oberkörpers verbunden.

In der Praxis wird selten mehr als ein Hilfspartikel benötigt, obwohl die Verwendung mehrerer Hilfspartikel möglich ist und auch robuster erscheint. Das Ver-

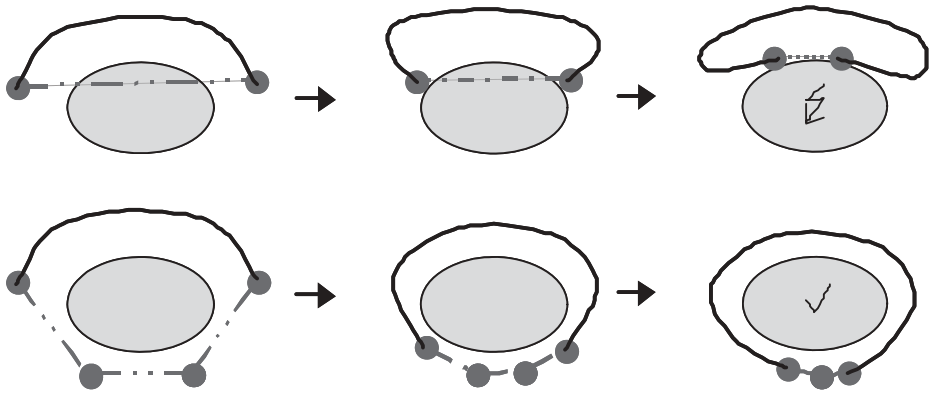


Abbildung 3.13: Oben: Direktes Vernähen der Partikel. — Unten: Die Verwendung von Hilfspartikeln ermöglicht ein robustes Vernähen.

fahren zur Berechnung der Koordinaten des Hilfspartikels kann hierfür gegebenenfalls erweitert werden.

3.2.5 Vorpositionierung eines Kleidungsstücks

Die bisher beschrieben Verfahren behandelten nur einzelne Schnittteile oder die Schnittteile einer Hüllfläche. Daher und zum besseren Überblick werden in Algorithmus 2 die Schritte des Verfahren zur geometrischen Vorpositionierung in algorithmischem Pseudocode zusammengefasst. Dabei berücksichtigt der Algorithmus alle Schnittteile eines Kleidungsstückes. Einige der Schritte sind in Abbildung 3.14 mit verschiedenen Kleidungsstücken veranschaulicht.

3.2.6 Ergebnisse

In Abbildung 3.14 sind einige Beispiele zum interaktionsfreien Einkleiden von virtuellen Menschen dargestellt. Die verschiedenen Schritte (Hüllflächen, geometrische Vorpositionierung und physikalisch basierte Endpositionierung) sind auf die Spalten verteilt. Abbildung 3.14 (a) zeigt links die initiale Hüllfläche und rechts die auf den Rock angepasste Hüllfläche.

In den Zeilen wurden jeweils andere 3D-Laserscans einer realen Person und andere Kleidungsstücke verwendet. Die Avatar bestehen aus ungefähr 34 K Dreiecken. Da diese Auflösung vergleichsweise hoch ist, können die Vertices der Dreiecke als Punktwolke für die Berechnung der Hauptachsen der Körperteile verwendet werden. Die Haltung der Avatare musste nicht verändert werden (siehe Abschnitt 3.1.2), da die verwendeten Kleidungsstücke auch so korrekt vorpositioniert werden konnten. Für andere Kleidungsstücke, wie beispielsweise ein Hemd mit langen Ärmeln, wäre die Änderung der Körperhaltung aber vorteilhaft.

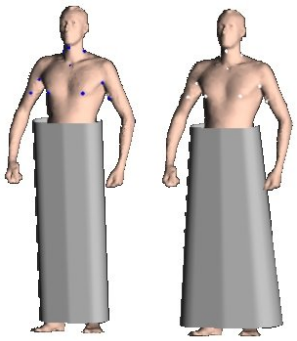
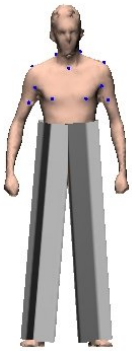
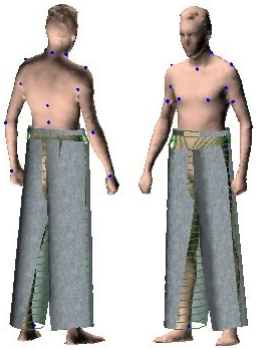
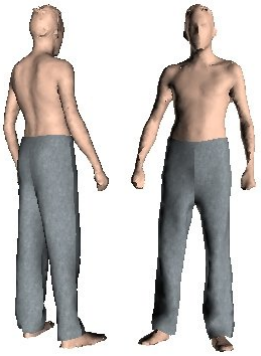
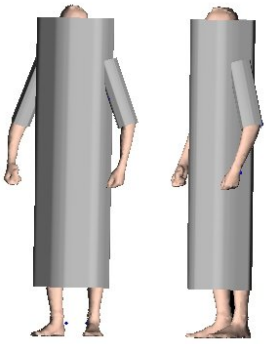
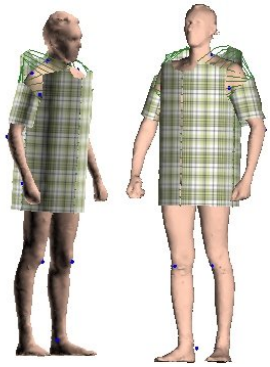
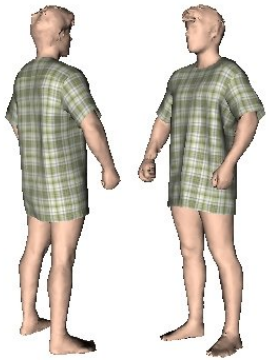
Hüllflächen um Körperteile	**Vorpositionierte Schnittteile und Gummifäden für das Vernähen**	**Kleidungsstück nach der physikalisch basierten Endpositionierung**
 (a)	 (b)	 (c)
 (d)	 (e)	 (f)
 (g)	 (h)	 (i)

Abbildung 3.14: Einige Beispiele zum interaktionsfreien Einkleiden von virtuellen Menschen. In jeder Zeile wurde eine andere Person eingekleidet.

Algorithmus 2 : GEOMETRISCHE VORPOSITIONIERUNG

Eingabe : Avatar mit Merkmalspunkten, CAD-Schnittteile eines Kleidungsstücks mit Annotationen und Nahtinformationen
Ausgabe : am Avatar vorpositionierte Schnittteile

Unterteile den Avatar in Segmente s_i
forall s_i **do**
  berechne Hüllflächen $\mathbf{B}_i$ (siehe Abschnitt 3.2.1)
end
weise jedes Schnittteil p_j einem s_i zu
forall s_i **do**
  repeat
    Projiziere Merkmalspunkte auf $\mathbf{B}_i$
    abwickeln der Hüllfläche: $\mathbf{B}_i^f = \psi^{-1}(\mathbf{B}_i)$
    abwickeln der Merkmalspunkte mit ψ^{-1}
    platziere p_j^i in $\mathbf{B}_i^f$ (siehe Abschnitt 3.2.2)
    passe $\mathbf{B}_i^f$ an (siehe Abschnitt 3.2.3)
  until *alle Schnittteile* p_j^i *liegen in* $\mathbf{B}_i^f$
  Berechne $\psi(p_j)$
end
Erzeuge Gummifäden mit Hilfspartikeln (siehe Abschnitt 3.2.4)

Das Verfahren ist nicht auf die gezeigten Kleidungsstücke beschränkt. Das System wurde mit weiten Röcken, Hemden mit langen Ärmeln, Unterwäsche und einem Sakko validiert. Für eine Messung der Dehnung der Schnittteile bei der Vorpositionierung sei auf [Fuh01] verwiesen.

Tabelle 3.3 enthält Zeitmessungen zur geometrischen Vorpositionierung, die auf einem AMD Athlon mit $1,3$ *GHz* durchgeführt wurde. Implementiert wurde das Verfahren in C++. Man erkennt, dass die Komplexität des Avatars nur geringen Einfluss auf die Laufzeit hat. Da die Geometrie des Avatars nur am Anfang für die Berechnung der initialen Hüllflächen um die Körperteile benötigt wird, ist dies auch nicht verwunderlich. Die Zeit, die benötigt wird, um einen 3D Laserscan zu segmentieren, ist in der Tabelle nicht mit eingerechnet, da diese Segementierung als Eingabe vorausgesetzt wird. Zudem ist dieser Vorgang unabhängig von der eigentlichen Vorpositionierung. Verwendet man beispielsweise synthetische Avatare so entfällt je nach Datenmodell dieser Schritt sogar ganz. Die Segmentierung der Laserscan mit 34 *K* Dreiecken dauerte 260 *ms*, wohingegen beim Scan mit 120 *K* Dreiecken 600 *ms* benötigt werden.

Die Zeit, die der Algorithmus zur Vorpositionierung benötigt, liegt in allen Fällen unter $0,3$ *sec*. Dies belegt, dass das vorgestellte Verfahren wesentlich schneller arbeitet, als eine manuelle Vorpositionierung jemals sein könnte. Die anschließende Simulation konvergiert ebenfalls sehr schnell innerhalb weniger Sekunden,

Vertices der Kleidung	**Avatar mit** 34 *K* **Dreiecken**	**Avatar mit** 120 *K* **Dreiecken**
1000	180 ms	190 ms
5000	210 ms	220 ms
10000	250 ms	260 ms

Tabelle 3.3: Zeitmessung zur geometrischen Vorpositionierung.

je nachdem wie hoch die Auflösung der Schnittteile ist. Details zu diesem Schritt werden in Kapitel 5 erörtert.

Bewertung

Das vorgestellte Verfahren zum interaktionsfreien Einkleiden von virtuellen Menschen arbeitet für eine Reihe von Kleidungsstücken zuverlässig. Einige Beispiele wurden im vorigen Kapitel in Abbildung 3.14 gezeigt. Für viele andere wurde die Funktionsweise ebenfalls getestet. Allerdings können die Schnittteile mancher Kleidung äußerst komplexe Formen annehmen. Um diese Fälle abzudecken, muss dass Verfahren noch robuster gestaltet werden, was allerdings ein Thema weiterer Forschungsarbeiten sein wird.

Eine wichtige Anwendung des Verfahrens, die auch Forschung in diesem Bereich stark vorangetrieben hat, ist eine virtuelle Boutique, in der der Kunde Kleidungsstücke an seinem eigenen 3D-Laserscan anprobieren kann. Hierfür ist ein automatisierter Ablauf zwangsläufig notwendig, da es dem Kunden nicht zugemutet werden kann, die Schnittteile per Hand zu platzieren.

Weiterhin profitiert auch der Entwurf von Konfektionsbekleidung von diesem Verfahren. In diesem Kontext, wird mit Hilfe der virtuellen Anprobe überprüft, ob die entworfenen Schnittteile eine gute Passform besitzen. Als Avatare können hierbei 3D-Laserscans von Models der Firma verwendet werden, die eine Konfektionsgröße repräsentieren. Der gesamte Prozess wird durch das interaktionsfreie Einkleiden enorm beschleunigt, da auf Knopfdruck aus den zweidimensionalen Schnittteilen ein fertig simuliertes Kleidungsstück erstellt werden kann.

Das Verfahren benötigt Schnittteile, die zusätzlich mit vielen Informationen annotiert sind. Anhand dieser Daten erfolgt die Aufteilung auf die verschiedenen Hüllflächen und es werden die relativen Positionen auf den Hüllflächen festgelegt. Wünschenswert wäre ein Verfahren, dass die Schnittteile nur mit Hilfe der Nahtinformationen vorpositionieren kann.

Ein weiteres Problem stellen Durchdringungen zwischen Schnittteilen dar, die auf unterschiedlichen Hüllflächen liegen, da die Hüllflächen unabhängig voneinander sind. Daher müssen diese Durchdringungen später bei der physikalisch basierten Endpositionierung aufgelöst werden. Hierfür kann ein Verfahren von Volino et al. [VMT97b, VCMT95] verwendet werden, das solche Durchdringungen

auflöst. Diese Methode zur Kollisionserkennung erlaubt die Korrektur, indem begrenzte Regionen auf der Oberfläche des Stoffs bestimmt werden, die miteinander in Kontakt sind. Jede Region wird entweder als auf der richtigen oder als auf der falschen Seite liegend markiert. Die jeweils größeren Regionen werden dabei als auf der richtigen Seite erkannt. Die Durchdringungen werden dann anhand einer geeigneten Kollisionsantwort aufgelöst. In dem hier vorliegenden Fall durchdringen sich nur kleine Regionen der Schnittteile und können daher einfach als auf der falschen Seite liegend erkannt werden.

Erweiterungen

Das Verfahren zur geometrischen Vorpositionierung kann in mehrere Richtungen erweitert werden. Ein Problem stellen mehrlagige bzw. mehrere Kleidungsstücke dar, die man übereinander anprobieren möchte. Hierzu muss zunächst eine geeignete Anziehreihenfolge definiert werden. Eine Information, die leicht hinzugefügt werden kann. Wendet man dann die vorgestellte Methode zur Vorpositionierung iterativ auf die einzelnen Kleidungsstücke an, könnten die einzelnen Schnittteile korrekt platziert werden. Hierzu könnten die Hüllflächen nach jedem Schritt in der Vorpositionierung entsprechend vergrößert werden. Für Details zur Vorpositionierung mehrerer Kleidungsstücke, die auch übereinander liegen können, sei auf die Arbeit von Clemens Groß et al. [GFL03] verwiesen.

Ein anderes Forschungsthema ist die Erweiterung der geometrischen Vorpositionierung auf weitere, kompliziertere Kleidungsstücke. Derzeit lassen sich nur Schnittteile verarbeiten, die zu der Topologie der Hüllflächen passen. Damit wird zwar ein weiter Bereich gängiger Bekleidung abgedeckt. Um aber die Kreativität der Designer nicht einzuschränken, ist eine Erweiterung sinnvoll. Beispielsweise lassen sich Schnittteile, die über mehrere Hüllflächen verlaufen nicht behandeln. So existieren Pullover, in denen ein einzelnes Schnittteil vom linken Arm über den Rücken zum rechten Arm verläuft. Eine mögliche Lösung für dieses Problem wäre eine geeignete Unterteilung solcher Schnittteile.

Die Grundidee des Verfahrens ist das zweistufige Einkleiden: Zuerst eine geometrische Vorpositionierung und danach die physikalisch basierte Endpositionierung. Hier wäre auch ein völlig anderer Ansatz denkbar. Angenommen, ein Avatar mit durchschnittlichen Maßen wäre bereits bekleidet. Dann könnte sowohl der Avatar, als auch die Kleidung an die individuellen Maße des Kunden angepasst werden. Voraussetzung für die Anpassung des Avatars wäre ein auf Körpermaße parametrisierbares Menschmodell. Die Kleidung könnte mit den Techniken aus Kapitel 5.5 verändert werden, d.h. man würde kontinuierlich mit Hilfe der Simulation von einer Kleidergröße auf die neue übergehen. Wichtig ist hierbei, dass das Menschmodell ebenfalls kontinuierlich verändert werden kann.

Diese Vorgehensweise bietet eine Reihe von Vorteilen. Zunächst ist sie sehr robust, da das erste Ankleiden nicht zwangsläufig automatisch ablaufen muss. Damit steht zu erwarten, dass beliebige Kleidungsstücke behandelt werden können. Des

Weiteren ist die benötigte Rechenzeit sehr gering, da weder vorpositioniert noch vernäht werden muss. Allerdings stellt das Verfahren sehr hohe Anforderungen an die Kollisionserkennung und es können keine völlig neuen Kleidungsstücke auf Knopfdruck angezogen werden.

3.3 Erweiterung zur Ontologie

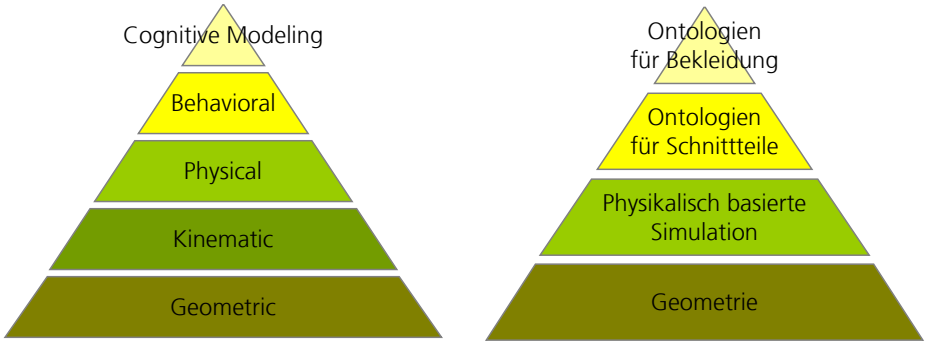
Die Modellierung von virtueller Bekleidung unter Verwendung der physikalisch basierten Animation als Ebene über der geometrischen Ebene hat sich in den letzten Jahren als äußerst erfolgreich erwiesen. Im letzten Jahrzehnt wurde die physikalisch basierte Ebene für virtuelle Bekleidung gründlich erforscht. Es wurden spezielle Simulationsmodelle für textiles Material vorgeschlagen [HB00, BW98, EW99] und dedizierte Lösungsverfahren entwickelt [MDDB01, EKS03]. Die wichtigen Aspekte der Kollisionserkennung [LMTT91, VMT00a, BMF03, BWK03, KNF04] und der Kollisionsantwort [VMT00b, BFA02] wurden ebenfalls ausführlich behandelt.

Allerdings wurden bisher kaum höhere Ebenen zur Modellierung im Bereich der virtuellen Bekleidung verwendet. In anderen Bereichen der Computergraphik wurden hingegen schon Ebenen über der physikalisch basierten Ebene vorgeschlagen. Beispielsweise wird in [FTT99, Fun99] eine Hierarchie zur Modellierung im Kontext der Charakteranimation vorgeschlagen, an deren Spitze die kognitive Modellierung steht (siehe Abbildung 3.15 (a)). Hierbei wurde über die bereits bekannte Ebene zur Modellierung des Verhaltens ein kognitives Modell gelegt. Diese oberste Ebene dient zur gezielten Steuerung der Aktionen der Charaktere. Die Ebenen über der physikalischen Ebene sind für das Animieren von autonomen Charakteren ausgelegt und bilden in gewisser Weise reale Denkprozesse nach. Da Bekleidung kein Eigenleben führt, lassen sich diese Methoden zur Modellierung natürlich nicht direkt übertragen. Die Grundidee, mit Hilfe einer Hierarchie zu modellieren, kann jedoch übernommen werden.

Im Bereich der Simulation von Kleidung stellt das in Kapitel 3.1 vorgestellte Verfahren zum interaktionsfreien Bekleiden von virtuellen Menschen eine Ausnahme dar, da dort bereits die geometrische Vorpositionierung ansatzweise als Ebene vor bzw. über der physikalisch basierten Simulation gesehen werden kann. Ein wesentlicher Aspekt dieses Verfahrens ist die abstrakte Repräsentation von Schnittteilen bzw. der Kleidung und des Avatars.

In dieser abstrakten Repräsentation ist eine Ontologie implizit enthalten. Im Folgenden werden aufbauend auf [FGW05] explizite Ontologien für Schnittteile und für Bekleidung vorgestellt. Diese können in gängigen Sprachen wie der „Web Ontology Language“ (OWL) [OWL04] formal beschrieben werden.

Ausgehend von der Ontologie für Bekleidung (dies schließt Ontologien für Schnittteile, Kleidungsstücke und virtuelle Menschen ein) kann eine semantische Modellierungsebene definiert werden. Diese Ebene liegt über der physikalisch basierten Ebene, die wiederum über der geometrischen Ebene liegt (siehe auch Ab-



(a) (b)

Abbildung 3.15: (a) Die kognitive Modellierung als höchste Ebene in der Modellierung (Bild nach [FTT99]) — (b) Eine Hierarchie zur Ontologie-basierten Modellierung.

bildung 3.15 (b)). Letztere gibt die Ausgangssituation vor und anhand dieser lassen sich beispielsweise die Materialkonstanten ableiten. Mit diesen kann die physikalische Ebene das dynamische Verhalten errechnen. Gleichzeitig erlaubt die oberste, semantische Ebene eine abstrakte Modellierung unabhängig von konkreten geometrischen Ausprägungen. Diese sind nur Ergebnis des Ineinanderwirkens der einzelnen Ebenen.

Die Informationen, die sich aus der Ontologie ergeben, stellen eine Vergröberung der geometrischen Information dar. Mit diesen wird eine Modellierung auf einem abstrakteren Level möglich.

Weiterhin kann darauf aufbauend ein Algorithmus zur semantischen Kollisionserkennung entworfen werden. Dieser verwendet die zusätzlichen Informationen, die die Schnittteile neben ihrer Geometrie mit sich führen, um robust und effizient Selbstkollisionen zu erkennen. Der vorgestellte Algorithmus erlaubt schwierige bzw. fehlerhafte Anfangswerte für die Positionen der Schnittteile und ermöglicht eine robuste Kollisionserkennung zwischen mehrlagigen Kleidungsstücken.

3.3.1 Eine Ontologie für Schnittteile

Abstrakte Beschreibung eines menschlichen Körpers

Die Ontologie für Bekleidung erfordert semantische Informationen für alle beteiligten Geometrien. Daher wird hier zunächst eine abstrakte Beschreibung des menschlichen Körpers vorgenommen, die diesen in mehrere teilweise überlappende Segmente unterteilt und markante Punkte auf der Oberfläche klassifiziert. Diese Informationen sind ganz ähnlich wie die in Abschnitt 3.1.2 definierten. An dieser Stelle wird diese Information zusätzlich in Klassen eingeteilt und

mit einer Relation in Beziehung zueinander gesetzt, um eine Ontologie über dem menschlichen Körper zu bilden.

Es wird ein Klasse

$$\texttt{bfp} \tag{3.5}$$

von Merkmalspunkten (engl. body feature points) bestehend aus elf Elementen definiert. Darunter fallen beispielsweise `girthPoint`, `rightNipple`, `neck front` etc. Die Merkmalspunkte sind in Tabelle enthalten 3.4.

Weiterhin wird eine Klasse

$$\texttt{bsh} \tag{3.6}$$

von Hüllflächen (engl. body segment hulls) mit 8 Elementen definiert, die ebenfalls in Tabelle 3.4 aufgelistet sind.

Aus diesen beiden Klassen kann eine binäre Relation

$$\texttt{featurePointsOnSegment} \subseteq \texttt{bsh} \times \texttt{bfp} \tag{3.7}$$

gebildet werden, die zu jeder Hüllfläche eine Menge von korrespondierenden Merkmalspunkten zur Verfügung stellt. Diese Relation ist in Tabelle 3.4 aufgelistet. Verständlicherweise stehen einige Merkmalspunkte in Relation zu mehreren Hüllflächen. Beispielsweise liegt der `waistToHipPointRight` sowohl auf `rightLeg` als auch auf `bothLegs` und ebenfalls auf `torsoAndLegs`.

Hüllfläche	**Merkmalspunkte**
neck	neck back, neck front, right neck, left neck
leftArm	left upper arm
rightArm	right upper arm
torso	neck back, neck front, left nipple, right nipple, waist girth point, waist to hip point right, waist to hip point left
leftLeg	waist to hip point left
rightLeg	waist to hip point right
bothLegs	waist to hip point left, waist to hip point right, waist girth point
torsoAndLegs	neck back, neck front, left nipple, right nipple, waist girth point, waist to hip point right, waist to hip point left

Tabelle 3.4: Die Hüllflächen, die die Körperteile umschließen, und die korrespondierenden Merkmalspunkte (die Relation `featurePointsOnSegment`).

Abbildung 3.16 zeigt einen menschlichen Körper mit einigen der Hüllflächen und den verwendeten Merkmalspunkte. Die zylindrischen Hüllflächen wurden mit der Methode aus Abschnitt 3.2.1 berechnet. Alle weiteren Berechnungen zur geometrischen Vorpositionierung erfolgen anhand dieser Hüllflächen und der Merkmalspunkte, die auf die Hüllflächen projiziert werden. Die eigentliche Geometrie

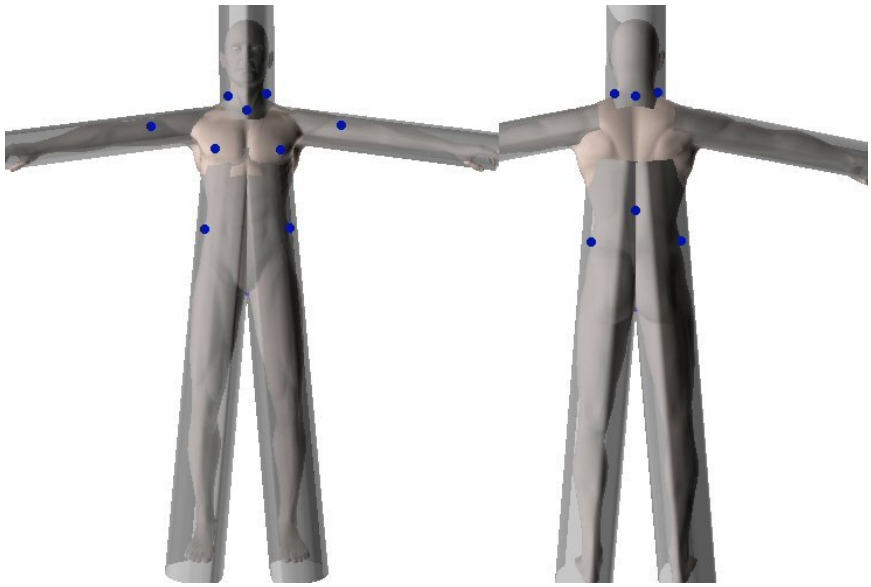


Abbildung 3.16: Virtueller Mensch mit Merkmalspunkten und Hüllflächen für den Nacken, die Arme und die Beine.

des Körpers wird erst wieder auf der tieferen Ebene der physikalisch basierten Simulation für die Kollisionserkennung benötigt.

Abstrakte Beschreibung für Schnittteile

Für die Schnittteile (engl. cloth patterns) wird die Klasse

$$\texttt{cp} \tag{3.8}$$

definiert, zu der die drei folgenden Relationen spezifiziert werden.

Die binäre Relation

$$\texttt{isDirectlySewedWith} \subseteq \texttt{cp} \times \texttt{cp} \tag{3.9}$$

liefert alle Paare von Schnittteilen, die direkt miteinander vernäht sind. Daraus lässt sich die transitive Eigenschaft `isSewedWith` ableiten, die die transitive Hülle der Relation `isDirectlySewedWith` darstellt. Es sei angemerkt, dass in OWL solche transitiven Eigenschaften definiert werden können.

Die nächste binäre Relation

$$\texttt{liesOn} \subseteq \texttt{bfp} \times \texttt{cp} \tag{3.10}$$

ordnet Merkmalspunkte zu Schnittteilen zu. Diese Relation drückt aus, dass die Projektion eines Merkmalspunktes auf einem Schnittteil liegt. Da nicht für jedes

Schnittteil ein Merkmalspunkt gegeben ist und auch nicht benötigt wird, stehen nicht alle Schnittteile in Relation zu einem Merkmalspunkt.

Die Relation `liesOn` ist ganz wesentlich für das interaktionsfreie Einkleiden. Da ohne solche Information beispielsweise ein Rock einfach um die Hüfte rotiert werden kann und man nicht in der Lage ist seine genaue Position am Körper zu bestimmen. Daher wird durch die Merkmalspunkte die relative Position der Schnittteile zum Körper festgelegt. Eine genauere Festlegung der Lage der Schnittteile wäre prinzipiell möglich, hätte aber den Nachteil, dass mehr Informationen spezifiziert werden müssten, als nötig sind.

Schließlich wird noch die binäre Relation

$$\texttt{isLyingOn} \subseteq \texttt{cp} \times \texttt{bsh} \tag{3.11}$$

definiert, die Hüllflächen und Schnittteile in Beziehung zueinander setzt. Informell bedeutet dies, dass Schnittteile auf bestimmte Hüllflächen projiziert werden. Die Auswirkungen auf die konkrete Geometrie der Schnittteile ergeben sich aus den Algorithmen zur geometrischen Vorpositionierung, die im vorigen Kapitel beschrieben wurden.

Im folgenden Abschnitt wird erläutert, wie die hier beschriebenen Relationen festgelegt werden können. Hierzu werden die Informationen, die in den CAD Eingangsdaten vorhanden sind analysiert und gegebenenfalls erweitert.

Annotation der CAD Daten

Innerhalb eines CAD System werden die Schnittteile durch ihre Randkurven repräsentiert. Zusätzlich werden Nahtinformationen benötigt, um die Relation `isDirectlySewedWith` zu erstellen. Diese Informationen liegen in CAD Systemen für gewöhnlich nicht vor und müssen noch hinzugefügt werden. In Abbildung 3.17 sind die Randkurven und die Nahtinformationen für eine Jacke dargestellt. Hierbei definieren die Nummern, welche Teile der Randkurven vernäht werden sollen. Es ist allerdings nur grob zu erkennen wie die Nahtkurven genau verlaufen. Da die Relation `isDirectlySewedWith` nur einen Teil der Nahtinformationen abbildet, ist dies soweit ausreichend. Für das eigentliche Vernähen werden natürlich noch die exakten Start- und Endpunkte der Naht benötigt.

Die Relation `isLyingOn` wird einfach erstellt, indem jedem Schnittteile die entsprechende Hüllfläche zugewiesen wird.

Für die Vorpositionierung und die Relation `liesOn` wird für jeweils eines der Schnittteile auf einer Hüllfläche die relative Position eines Merkmalspunktes benötigt. Für die Ausrichtung des Schnittteils auf der Hüllfläche wird noch ein Obenvektor verwendet, der nach der Abbildung des Schnittteils auf die Hüllfläche entlang der Hauptachse des Körpersegments verlaufen muss. Da diese Information unabhängig von der Körpergröße des Avatars ist, muss sie nur einmal eingegeben und gespeichert werden.

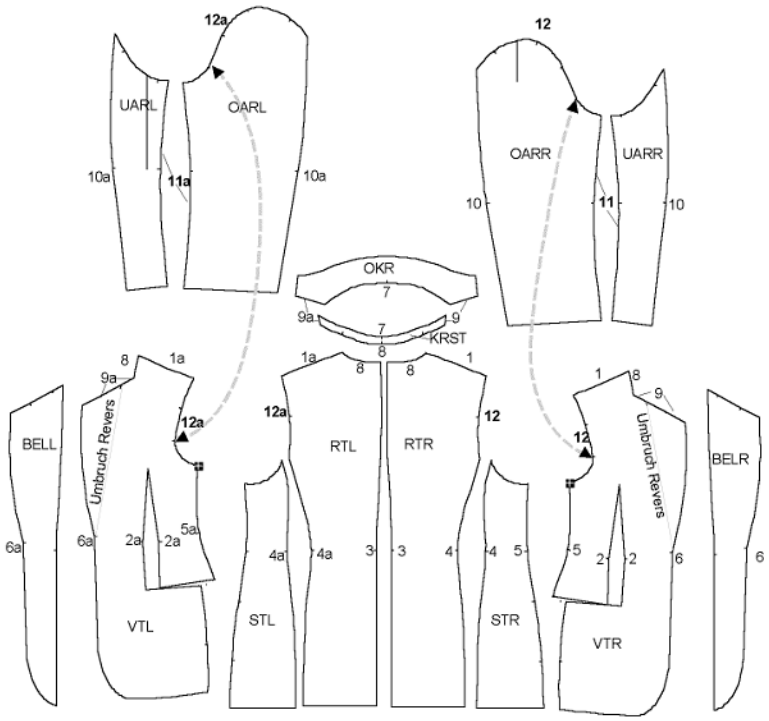


Abbildung 3.17: Schnittteile einer Jacke mit Nahtinformationen.

3.3.2 Eine Ontologie für Kleidung

Ausgehend von der Ontologie für Schnittteile kann die Klasse

$$\texttt{garments} \tag{3.12}$$

für einzelne Kleidungsstücke eingeführt werden. Solch ein Kleidungsstück ist definiert als die Gesamtheit aller Schnittteile, die miteinander vernäht wurden. Gebräuchliche Klassifizierungen von Kleidung wie z.B. `jackets, trousers, skirts` und `dresses` können damit als Subklassen von `garments` eingeführt werden. Die Eigenschaft

$$\texttt{isPartOf} \subseteq \texttt{cp} \times \texttt{garments} \tag{3.13}$$

gibt an, dass ein bestimmtes Schnittteil Teil eines Kleidungsstücks ist.

Die formale Definition dieser Subklasssen geht einher mit den üblichen Bezeichnungen für verschiedene Typen von Kleidungsstücken, wie Hosen, Rock, etc.. Zudem lassen sich die Subklassen über die Ontologie selbst definieren. Beispielsweise könnte die Subklasse Hose definiert sein als ein Kleidungsstück, für das ein Schnittteil X existiert mit

$$\texttt{isLyingOn}(X, \texttt{leftLeg}) \tag{3.14}$$

und ein weiteres Schnittteil Y existiert mit

$$\texttt{isLyingOn}(Y, \texttt{rightLeg}). \tag{3.15}$$

Im Gegensatz dazu ließe sich die Subklasse Rock spezifizieren als ein Kleidungsstück mit einem Schnittteil X mit

$$\texttt{isLyingOn}(X, \texttt{bothLegs}). \tag{3.16}$$

Damit eine Hose nicht fälschlicherweise als Rock klassifiziert werden kann (der Bund einer Hose liegt oft auf `bothLegs`), muss noch eine weitere Bedingung spezifiziert werden. Für alle Schnittteile Z muss gelten

$$\neg\texttt{isLyingOn}(Z, \texttt{rightLeg}). \tag{3.17}$$

Das Beispiel des Rocks demonstriert, dass die Angabe von konkreten Regeln nicht trivial ist. Daher erscheint es sinnvoll diese formalen Definitionen lernen zu lassen. Durch einen überwachten Lernprozess, in dem zahlreiche Kleidungsstücke definiert und entsprechend klassifiziert werden, könnten ausgehend von diesen Daten die entsprechenden Regeln abgeleitet werden.

Mehrere Kleidungsstücke

Virtuelle Bekleidung besteht in den meisten Fällen nicht aus einem einzelnen Kleidungsstück, sondern aus mehreren Einzelteilen, beispielsweise einer Hose, einem Hemd und einem Jackett. Das Wissen, in welcher Reihenfolge die Kleidungsstücke angezogen werden, kann durch die binäre Relation

$$\texttt{isDressedAfter} \subseteq \texttt{garments} \times \texttt{garments} \tag{3.18}$$

formalisiert werden. So wird in dem obigen Beispiel das Jackett als letztes und das Hemd als erstes angezogen. Weiterhin kann die ähnliche Eigenschaft

$$\texttt{isDressedAfterP} \subseteq \texttt{cp} \times \texttt{garments} \tag{3.19}$$

für die Schnittteile definiert werden. Als standardmäßige Implikation ergibt sich aus der Eigenschaft $\texttt{isDressedAfter}(A,B)$ für zwei Kleidungsstücke, dass die Eigenschaft $\texttt{isDressedAfterP}(X,Y)$ für alle Schnittteile $X \subseteq A$ und alle Schnittteile $Y \subseteq B$ gilt.

In einigen Fällen wird die Eigenschaft `isDressedAfterP` jedoch anders definiert: Wenn einzelne Schnittteile über einem anderen Kleidungsstück liegen. So kann für einen Pullover, der über einem Hemd getragen wird, der Kragen des Hemds über den Pullover gelegt werden (siehe auch Abbildung 3.19 (a)). Damit mehrere Kleidungsstücke von der geometrischen Vorpositionierung behandelt werden können, muss das Verfahren aus Kapitel 3.2 erweitert werden. Hierzu sei auf [GFL03] verwiesen.

3.3.3 Abstrakte Modellierung

Mit Hilfe der vorgestellten Ontologien lässt sich eine abstrakte Modellierung virtueller Bekleidung durchführen. Durch einfaches Ändern einiger Werte der Eigenschaften in den Ontologien, können ein oder mehrere Kleidungsstücke manipuliert werden, ohne dass auf die physikalisch basierte oder geometrische Ebene hinabgestiegen werden muss. Anders ausgedrückt, ergeben sich aus kleinen Veränderungen große Wirkungen. Letztere werden mittels der zugrunde liegenden Ebenen automatisch berechnet. Unter der Voraussetzung, dass die Kleidung und der virtuelle Mensch von ihren Proportionen zusammen passen, erhält man die gewünschten Resultate. Im Folgenden werden einige Beispiele für die abstrakte Modellierung virtueller Bekleidung vorgestellt.

Die *Reihenfolge*, in der *mehrere Kleidungsstücke* angezogen werden, hat verständlicherweise erhebliche Auswirkungen auf die resultierende Geometrie. Durch die zuvor entworfene Ontologie kann eine solche Änderung sehr einfach vollzogen werde, indem die Eigenschaft `isDressedAfter` angepasst wird.

In Abbildung 3.18 (a) ist das Ergebnis der Vorpositionierung einer Hose, der der Name `trousersA` zugewiesen wurde, unter einem Hemd mit dem Namen `shirtB` dargestellt. In diesem Fall lautet die Eigenschaft

$$\texttt{isDressedAfter}(\texttt{shirtB}, \texttt{trousersA}). \tag{3.20}$$

Um die Reihenfolge um zukehren muss nur diese Eigenschaft zu

$$\texttt{isDressedAfter}(\texttt{trousersA}, \texttt{shirtB}) \tag{3.21}$$

verändert werden. Abbildung 3.18 (b) zeigt das Ergebnis dieser Veränderung. Dazu muss natürlich die Vorpositionierung und die Simulation von neuem gestartet werden.

Im folgenden Beispiel wird von der Relation `isDressedAfter` Gebrauch gemacht, um die *Lage eines Hemdkragens* zu modellieren. In Abbildung 3.19 (a) ist ein Pullover `pulloverC` dargestellt, der über dem Hemd und der Hose getragen wird. In der Ontologie wurde hierzu spezifiziert, dass der Kragen über allen anderen Schnittteilen getragen wird. Daher liegt der Kragen ganz oben und der Rest des Hemds unter dem Pullover. Sei `collarP` der Name des Kragens und `X` die Schnittteile des Pullovers, dann kann dies mit

$$\texttt{isDressedAfter}(\texttt{trousersA}, \texttt{shirtB}),$$

$$\texttt{isDressedAfter}(\texttt{pulloverC}, \texttt{trousersA}),$$

$$\texttt{isDressedAfterP}(\texttt{collarP}, \texttt{X})$$

formalisiert werden. Hierzu sei angemerkt, dass in dem Beispiel das gesamte Hemd simuliert wurde und nicht nur die sichtbaren Teile. Erkennen kann man dies im

(a)

(b)

Abbildung 3.18: Vorpositionierung und Simulation mehrerer Kleidungsstücke übereinander. (a) Hose unter dem Hemd — (b) Hemd in der Hose.

(a)

(b)

Abbildung 3.19: (a) Beispiel für mehrlagige Bekleidung: Hemd, Hose, Pullover und der Kragen des Hemds. Obwohl das Hemd als erstes angezogen wurde, wurde dem Kragen die Eigenschaft zugewiesen über allen anderen Schnittteile zu liegen. — (b) Beispiel eines Kleides, dass verkehrt herum angezogen wurde. Zum Vergleich ist unten rechts die korrekte Lage abgebildet.

Bereich der Beine, wo die Hose weit vom Körper entfernt ist, da dort zu viel textiles Material des Hemds vorhanden ist.

Um die beiden Beispiele robust simulieren zu können, wird aber auch eine Verbesserung in der physikalisch basierten Ebene benötigt. Dies wird im nächsten Kapitel näher erläutert.

Im letzten Beispiel wird demonstriert, dass ein Kleidungsstück prinzipiell auch falsch herum angezogen werden kann. In Abbildung 3.19 (b) ist ein einfaches Kleid dargestellt, dass aus Schnittteilen besteht, die nur auf der Hüllfläche `torsoAndLegs` liegen. Die relative Lage der Schnittteile war zunächst durch den Merkmalspunkt `waist to hip point left` gegeben. Ändert man diesen zu `waist to hip point right` erhält man das gezeigte Ergebnis. Das Kleid in der Abbildung wurde ohne weitere Interaktion des Anwenders durch die geometrische Vorpositionierung und die physikalisch basierte Endpositionierung erzeugt.

3.3.4 Semantische Selbstkollisionserkennung

Eine der großen Herausforderungen in der Animation von Bekleidung ist die effiziente und robuste Kollisionserkennung zwischen mehrlagigen Textilien. In bisherigen Ansätzen wurden unterschiedliche Algorithmen zur Lösung vorgeschlagen, die alle auf der geometrischen bzw. physikalischen Ebene arbeiten. Im folgenden wird ein neuer Ansatz vorgestellt, der zwar auch geometrische Ansätze verwendet, aber zusätzlich die semantischen Informationen, die durch die Ontologie gegeben sind, auswertet. Hierzu wird die Relation `isDressedAfterP` verwendet. Dieser Ansatz ist durch die Tatsache motiviert, dass die meisten Kleidungsstücke einer wohl definierten Schicht zugeordnet werden können und dass sich die gegenseitige Lage während der Animation nicht ändert. Wenn also schon bekannt ist, das ein bestimmtes Kleidungsstück unter einem anderen liegt, dann sollte diese Information auch dem Modul zur Kollisionserkennung bekannt sein.

Bisherige Methoden führen die Kollisionserkennung auf der geometrischen Ebene durch. Einige Algorithmen verwenden die Historie des Kleidungsstück, d.h. Zustände in vergangenen Zeitschritten [BFA02, VMT00a]. Diese Verfahren versagen, sobald ein kleiner Fehler im System auftaucht, sei es durch numerische Ungenauigkeiten oder durch unsaubere Anfangswerte. Nachdem ein solcher Fehler im System ist, kann dieser vom Algorithmus nicht mehr behoben werden bzw. führt sogar zur Explosion des Systems.

Ein Verfahren, dass ohne Historie auskommt, wird in [BWK03] beschrieben. Es wird für das Mesh der Bekleidung eine „Global Intersection Analysis“ durchgeführt, die Selbstkollisionen erkennen und auflösen kann. Der Algorithmus kann bestehende Fehler aus vergangen Zeitschritten erkennen und richtig aufheben. Das Verfahren arbeitet sehr gut für begrenzte Regionen auf dem Mesh. Durchdringungen an den Rändern des Meshes können zwar erkannt aber nicht aufgelöst werden.

Einen ganz anderen Weg beschreiten Cordier und Thalmann [CMT02]. Sie lösen das Problem mehrerer Kleidungsschichten dadurch, dass die nicht sichtbaren Teile der Kleidungsstücke vor der Simulation entfernt werden. Dies wird mit einer aufwändigen Simulation in einem Vorverarbeitungsschritt durchgeführt. Danach lassen sich die sichtbaren Teile in Echtzeit animieren. Als Konsequenz ergibt sich, dass keinerlei Interaktionen zwischen den Schichten mehr möglich sind und daher das Ergebnis zwar hübsch aussieht, aber nur wenig Realtitätsbezug hat.

Algorithmus zur semantischen Kollisionserkennung

Um die Schwierigkeiten in den bisherigen Verfahren zur Kollisionserkennung zu umgehen, verwendet der Algorithmus zur semantischen Kollisionserkennung die Information über die einzelnen Lagen der Kleidung, die sich aus der beschriebenen Ontologie ableiten lassen. In einem Vorverarbeitungsschritt werden den Partikeln, Kanten und Dreiecken des Meshes die Nummern der Schicht, in der sie liegen, zugeteilt. Sollten die Nummern adjazenter Partikel unterschiedlich sein, werden die sich daraus ergebenden Kanten oder Dreiecke von der semantischen Kollisionserkennung nicht weiter behandelt. Die Nummern werden aus der Relation `isDressedAfterP` abgeleitet. Für gewöhnlich ändern sich diese Nummern während der Simulation nicht. Weiterhin wird angenommen, dass der Avatar nicht von der Kleidung deformiert wird, d.h. er ist zwischen den Simulationsschritten ein Starrkörper. Daher kann seine Geometrie als statisch angenommen werden (am Ende eines Zeitschritts) und als Referenzfläche im Folgenden dienen.

Während eines Simulationsschrittes werden zunächst alle Kollisionen mit den in Kapitel 4 vorgestellten Methoden erkannt und die entsprechenden Kollisionsantworten erzeugt. Danach wird der sich ergebende Zustand des Partikelsystems auf korrekte Reihenfolge der Schichten überprüft. Hierzu werden die Distanzen von Elementen des Meshes (Kanten, Dreiecke und Partikel) zur Referenzfläche verglichen: Wenn eines der Elemente mit kleinerer Nummer weiter von der Referenzfläche weg ist, als ein Element mit höherer Nummer, ist die Reihenfolge nicht korrekt. Dies kann vereinfacht werden, indem nur die Relationen von Partikeln zu anderen Elementen des Meshes überprüft werden. Dadurch entfallen die Tests von Kanten gegeneinander.

Sei $\mathbf{M} = \mathbf{M}(\mathbf{V}, \mathbf{T})$ das Mesh der Kleidung und $\mathbf{M}_i \subseteq \mathbf{M}$ Teile des Meshes mit gleicher Nummer *i*. Jetzt wird für jeden Partikel $\mathbf{p}_j \in \mathbf{M}_i$ das „closest feature“ $\mathbf{c}_j$ zu $\mathbf{M} \backslash \mathbf{M}_i$, d.h. zu dem Teil von $\mathbf{M}$ berechnet, der auf einer anderen Schicht als $\mathbf{p}_j$ liegt. Dann wird die Distanz von $\mathbf{p}_j$ und die Distanz von $\mathbf{c}_j$ zur Referenzfläche miteinander verglichen. Entsprechen die Distanzen nicht der Reihenfolge der Schichten, d.h. wenn $\mathbf{c}_j$ dichter am Körper des Avatars ist, obwohl $\mathbf{c}_j$ auf einer höheren Schicht liegt, muss eine Korrektur erfolgen.

Für die *Kollisionsantwort* gibt es drei unterschiedliche Möglichkeiten: Man kann die Position des Partikels $\mathbf{p}_j$, die Position des „closest feature“ $\mathbf{c}_j$ oder beide Positionen ändern. Da eine Bewegung von $\mathbf{c}_j$ dazu führen würde, dass sich die Bekleidung vom Körper wegbewegt, ist es sinnvoller nur die Koordinaten von $\mathbf{p}_j$

Abbildung 3.20: Ein Hemd, das über der Hose angezogen wurde (links oben). Während der Simulation wird die Reihenfolge der Kleidung mittels `isDressedAfter` geändert. Die semantische Kollisionserkennung vollzieht diese Änderung äußerst robust (links nach rechts).

zu verändern. Dazu wird der Partikel ein wenig in Richtung der Normalen der Oberfläche des Avatars auf den Körper zu bewegt. Der Rest wird im nächsten Simulationsschritt von der herkömmlichen Kollisionserkennung aufgelöst. Für die Berechnung der Abstände zum Avatar wird ein Distanzfeld verwendet, dass sehr effizient die Auswertung von Distanzen und Normalen erlaubt (siehe auch Kapitel 4.4).

Der beschriebene Algorithmus zur semantischen Kollisionserkennung toleriert schwierige anfängliche Überschneidungen und sogar eine bestehende Reihenfolge der Kleidungsstücke kann einfach verändert werden. In Abbildung 3.20 wird beispielsweise gezeigt, wie durch Ändern von `isDressedAfter(shirtA, trouserB` zu `isDressedAfter(trouserB, shirtA)` die Reihenfolge zur Laufzeit umgekehrt werden kann. Bisherige Algorithmen die ähnliches leisten sind dem Autor dieser Arbeit nicht bekannt.

Allerdings besitzt das Verfahren noch ein paar Nachteile. Zunächst können keine Kleidungsstücke behandelt werden, die teilweise von einem anderen Klei-

dungsstück verdeckt sind und gleichzeitig dasselbe Kleidungsstück verdecken. Beispielsweise kann ein Hemd, das unter einer Hose getragen wird nicht auf einer Seite über der Hose liegen, nachdem es ein Stück herausgezogen worden ist. Weiterhin ist die Methode nur so genau, wie die zugrunde liegende Kollisionserkennung. Man erkennt in Abbildung 3.20 unten rechts, dass die Hose weiter vom Körper absteht als durch die Stoffdicke des Hemds vorgegeben ist. Damit die Hose enger anliegen kann, sind noch Erweiterungen an der Selbstkollisionserkennung nötig, die in diesem Verfahren verwendet wurde und in Kapitel 4.5 beschrieben wird.

3.4 Zusammenfassung

In existierenden Simulationssystemen werden virtuelle Charaktere noch per Hand eingekleidet. Neue Formen der Produktpräsentation und Produktanprobe verlangen aber nach automatisierten Verfahren, damit virtuelle Kleidung schnell und mit wenig Aufwand dargestellt werden kann. Daher werden in diesem Kapitel Ontologien für Bekleidung vorgestellt. Mit Hilfe der Ontologien und einem neuen Verfahren zum interaktionsfreien Einkleiden virtueller Menschen können die Anfangswerte für die Simulation von Bekleidung effizient berechnet werden.

Das *interaktionsfreie Einkleiden* erfolgt als zweistufiger Prozess. Zunächst werden die Schnittteile durch eine geometrische Vorpositionierung um den Avatar herum platziert. Die Endpositionierung erfolgt dann mittels physikalisch basierter Simulation unter der Verwendung von virtuellen Gummifäden, die die Schnittteile miteinander vernähen.

Die geometrische Vorpositionierung benötigt nur wenige Vorbedingungen um korrekt arbeiten zu können. Der Avatar muss in einer definierten Körperhaltung stehen. Ferner müssen die Kleidungsstücke auf eine Art und Weise repräsentiert sein, die es ermöglicht, sie eindeutig zu platzieren. Dazu genügen einige wenige Parameter pro Kleidungsstück, die auf abstrakte Weise die gewünschte Lage der Schnittteile am Körper festlegen.

Ausgehend von den Körperteilen des Avatars werden unterschiedliche Hüllflächen, die die Körperteile umschließen definiert. Die Schnittteile der Kleidungsstücke werden den Hüllflächen zugeordnet und auf diesen anhand der definierten Parameter positioniert. Da die Schnittteile um die einzelnen Körperteile gewickelt werden, liegen sie wesentlich näher am Avatar als bei herkömmlichen manuellen Verfahren und die Simulation kann die Schnittteile sehr schnell vernähen.

Es werden *Ontologien* für Schnittteile und Kleidungsstücke vorgestellt. Diese semantischen Informationen lassen sich verwenden, um virtuelle Bekleidung auf einer höheren Ebene als der physikalisch basierten Ebene zu modellieren. Weiterhin erlauben die enthaltenen semantischen Informationen das interaktionsfreie Einkleiden von virtuellen Menschen.

Die Ontologien können eingesetzt werden um auf einer hohen bzw. abstrakten Ebene intuitiv die Eigenschaften von mehreren gleichzeitig getragenen Klei-

dungsstücken zu ändern. Dabei ergibt sich die dreidimensionale Geometrie der Kleidung an einem speziellen Avatar als Resultat der Anwendung der Semantik auf die tiefer gelegenen Schichten der Modellierung. Hierbei sind keinerlei Eingriffe des Anwenders in diese Schichten nötig. Er spezifiziert nur anfangs einige semantische Eigenschaften und dies ergibt die gewünschten Änderungen an der Geometrie.

Auch der für die Simulation von Bekleidung fundamentale Vorgang der Kollisionserkennung zwischen unterschiedlichen Kleidungsstücken bzw. Lagen von Stoff kann mit Hilfe der vorgestellten Ontologien verbessert werden. Obwohl es bekannt ist, dass die Kollisionserkennung verbessert werden kann, wenn auf der geometrischen Ebene auch Informationen der physikalisch basierten Ebene (Geschwindigkeit, maximale Krümmung des Material, etc.) in Betracht gezogen werden, wurde bisher in keiner Arbeit die Information aus höheren Ebenen der Modellierung verwendet.

Kapitel 4

Schnelle Kollisionserkennung

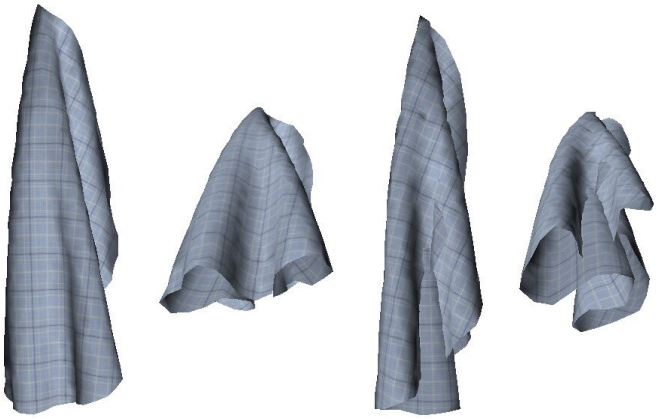
4.1 Kollisionserkennung für deformierbare Objekte

Der Bereich der Kollisionserkennung wird seit vielen Jahren in der Computergraphik intensiv erforscht. Zu Anfang lag das Hauptaugenmerk auf Verfahren, die Kollisionen zwischen starren Körpern erkennen können. Mit diesen Methoden eröffneten sich Anwendungen in der Robotik, der physikalisch basierten Animation von Festkörpern und natürlich für immersive virtuelle Umgebungen wie Einbausimulationen im Automobilbereich und Computerspiele. Erst später wurden Verfahren entwickelt, die das Problem der Kollisionserkennung von deformierbaren Objekten lösten. Dadurch ließen sich neue Szenarien realisieren, unter anderem Stoffsimulationen (siehe Abbildung 4.1) und die Simulation von Weichgewebe im Kontext von Chirurgiesimulation.

Die physikalisch basierte Simulation und Animation hat sich in den letzten Jahren zu einem immer wichtigeren Forschungsbereich der Computergraphik entwickelt, da sich durch diese Methoden der Realismus einer virtuellen Umgebung wesentlich verbessern lässt und sich somit eine Vielzahl neuer Anwendungen ergeben. Sieht man von Festkörpern ab, so sind die meisten zu simulierenden Materialien mehr oder weniger stark deformierbar. Gase, Flüssigkeiten und Stoff zählen zu den sehr stark deformierbaren Objekten. Andere Materialien wie Plastik, Holz oder auch Metall lassen sich meist nur begrenzt deformieren.

Im Gegensatz zu Gasen und Flüssigkeiten bereitet aber die *Kollisionserkennung von Stoff* besondere Schwierigkeiten, da trotz der starken Deformierbarkeit die Topologie stets erhalten bleibt und sich mehrere Lagen übereinander schichten können. Dieses Problem der Selbstkollisionen ist in Abbildung 4.2 illustriert. Behandelt man keine Selbstdurchdringungen, so sind die Simulationsergebnisse höchst unbefriedigend. Geht man über zur virtuellen Bekleidung, so müssen zusätzlich Verfahren zur Kollisionserkennung mit deformierbaren bzw. animierten Avataren mit den Algorithmen zur Selbstkollisionserkennung kombiniert werden. Hierbei kann die Tatsache ausgenutzt werden, dass der Avatar sich im Verhältnis zum Stoff nur wenig deformiert.

Abbildung 4.1: Für eine interaktive Stoffsimulation, die in Echtzeit abläuft, ist eine effiziente Kollisionserkennung für deformierbare Objekte unabdingbar.



(a) (b) (c) (d)

Abbildung 4.2: Ein rundes Tischtuch, das an einem Ende festgehalten wird. In (a) und (b) ist die Selbstkollisionserkennung aktiviert. In (c) und (d) ist sie ausgeschaltet. Selbstdurchdringungen sind störend erkennbar. Die Abbildungen (b) und (d) sind von schräg unten aufgenommen.

Die *Simulation von Weichgewebe* stellt etwas andere Anforderungen an die Kollisionserkennung, da die zu betrachtenden Objekte ein Volumen besitzen, wohingegen Stoff meist flächig ist. Beispielsweise müssen bei einer Chirurgiesimulation Kollisionen zwischen dem Gewebe der einzelnen Organe erkannt und aufgelöst werden. Greift der Operateur in die Szene ein, so müssen zusätzlich Kollisionen zwischen seinen Instrumenten und dem Gewebe behandelt werde. Erschwerend kommt hinzu, dass sich dabei die Topologie der Objekte ändern kann, wenn Schnitte durchgeführt werden. Damit ein sinnvolles virtuelles Training durchgeführt werden kann, darf die Immersion nicht gestört werden und der Algorithmus muss echtzeitfähig sein.

Ein weiteres Problem stellt die Kollisionserkennung zwischen einer Vielzahl von *eingeschränkt deformierbaren Objekten* dar. In diesem Fall muss zunächst

erkannt werden, welche Objekte überhaupt einen geringen Abstand zueinander haben und in einem zweiten Schritt dann die eigentliche Kollision zwischen den Objekten erkannt werden. Besondere Schwierigkeiten bereitet die Tatsache, dass eine passende Kollisionsantwort generiert werden muss, damit Objekte korrekt übereinander liegen können.

Dieses Kapitel behandelt das Thema der schnellen Kollisionserkennung für deformierbare Objekte, wobei ein Schwerpunkt die Kollisionserkennung bei der Simulation von textilen Materialien ist. Die vorgestellten und neu entwickelten Algorithmen werden an Beispielen zur Stoff- und Bekleidungssimulation validiert. Beschränkt man sich auf starre Körper, können die Algorithmen natürlich auch eingesetzt werden. Allerdings ist die Verallgemeinerung der Algorithmen auf beliebige deformierbare Körper nicht immer möglich bzw. praktikabel.

Der restliche Abschnitt gliedert sich folgendermaßen: Zunächst wird auf die speziellen Probleme bei der Kollisionserkennung mit deformierbaren Objekten eingegangen. Danach werden bestehende Algorithmen zur Kollisionserkennung klassifiziert, kurz vorgestellt und danach bewertet. Einige davon wurden zwar für die Kollisionserkennung mit Starrkörpern entworfen, lassen sich aber so anpassen, dass auch deformierbare Objekte behandelt werden können.

Anschließend werden in den folgenden Abschnitten dieses Kapitels neue Verfahren zur Kollisionserkennung vorgestellt. Hierzu wird zunächst auf Distanzfelder (4.2) eingegangen, die direkt Informationen über Eindringtiefe und Normale liefern. Beides wird für eine realistische Kollisionsantwort im Rahmen einer Simulation benötigt. Da Distanzfelder sehr komplexe Informationen über ein Objekt liefern, ist die Berechnung für einen als Dreiecksnetz gegebenen Körper alles andere als einfach. Daher wird die effiziente Erzeugung von Distanzfeldern in Abschnitt 4.3 näher betrachtet.

Darauf aufbauend wird eine Methode zur Kollisionserkennung mit Distanzfeldern präsentiert (4.4). Abschließend wird ein Verfahren zur effizienten Vermeidung von Selbstdurchdringungen beschrieben (4.5). Der Algorithmus nutzt die Tatsache aus, dass sich die Topologie von Stoff während der Simulation nicht ändert. Daher können geeignete hierarchische Strukturen aufgebaut werden.

4.1.1 Spezielle Probleme

Im Gegensatz zur Kollisionserkennung für starre Körper treten im Falle von deformierbaren Objekten eine Reihe zusätzlicher Schwierigkeiten auf. Daher können bestehende Algorithmen, die für starre Körper entwickelt wurden, nicht einfach übernommen werden, sondern es müssen angepasste Verfahren entwickelt werden. Die folgende Auflistung gibt einen Überblick über die speziellen Probleme bei der Kollisionserkennung für deformierbare Objekte:

- **Mehrfache Kontaktpunkte:** In vielen Anwendungen mit Starrkörpern genügt es, einen einzigen Schnittpunkt zu berechnen. Danach ist die Kolli-

sion erkannt und eventuell wird noch eine einfache Kollisionsantwort generiert. Bei deformierbaren Objekten treten aber meist eine Vielzahl von Kontaktpunkten auf, die alle korrekt behandelt werden müssen.

- **Selbstkollisionen:** Solche Kollisionen können bei Starrkörpern nicht auftreten, da sie sich nicht verformen können. Kann ein Objekt sich jedoch so verformen, dass es sich selbst durchringt, muss dies entsprechend verhindert werden. Bei der Erkennung tritt allerdings das Problem auf, dass adjazente Dreiecke von klassischen Algorithmen *immer* als kollidierend erkannt werden.

- **Eingeschränkte Vorberechnungen:** Für die meisten Verfahren zur Kollisionserkennung werden in einem Vorverarbeitungsschritt entsprechende Datenstrukturen angelegt, die zur Laufzeit beschleunigte Kollisionsabfragen ermöglichen. Bei starren Körpern bleiben diese Datenstrukturen während der ganzen Simulation gültig, da affine Transformationen leicht integriert werden können. Bei deformierbaren Objekten müssen die Datenstrukturen jedoch nach jedem Zeitschritt upgedated werden, was einen enormen Overhead bedeutet.

- **Kollisionsantwort:** Bei deformierbaren Objekten muss eine konsistente Kollisionsantwort berechnet werden, die alle Teile des Objektes in einen kollisionsfreien Zustand bringt. Dies kann sich sehr komplex gestalten, da die einzelnen Teile abhängig voneinander sind, d.h. eine Kollisionsantwort an einer Stelle beeinflusst die Antwort an einer anderen und kann sie evtl. aufheben. Weiterhin sind Informationen über Eindringtiefe und Flächennormale notwendig, die bei der Kollisionserkennung von starren Körpern oft nicht berücksichtigt werden müssen, um plausible Simulationen zu erhalten.

- **Zeitpunkt der Kollision:** Bei der Simulation von starren Körpern in diskreten Zeitschritten wird häufig der Ansatz verfolgt den genauen Kollisionszeitpunkt zu berechnen und dann die Simulation zu diesem Zeitpunkt zurückzusetzen. Da bei deformierbaren Objekten mehrfache Kollisionen zu unterschiedlichen Zeiten auftreten, würde dieser Ansatz nur extrem kleine Zeitschritte erlauben.

- **Proximity:** Bei starren Körpern genügt es meist eine Überschneidung zweier Objekte festzustellen und dann entsprechend darauf zu reagieren. Da aber, wie im vorigen Punkt dargelegt, bei deformierbaren Körpern kein eindeutiger Kollisionszeitpunkt existiert, muss nicht nur auf Überschneidung, sondern auf Proximity getestet werden.

- **Konsistenz:** Insbesondere bei Stoff, der im Verhältnis zur Fläche sehr dünn ist, treten Konsistenzprobleme auf, wenn mehrere Lagen übereinander liegen.

In interaktiven Anwendungen wie der Stoffsimulation steht nur ein Teil der Rechenzeit für die Kollisionserkennung zur Verfügung. Daher ist die größte Herausforderung, die sich aus den oben skizzierten Problemen ergibt, die Entwicklung von Verfahren, die nicht nur diese Schwierigkeiten lösen, sondern dies auch höchst effizient erledigen.

4.1.2 Verfahren zur Kollisionserkennung

Die wichtigsten Methoden zur Kollisionserkennung für deformierbare Objekte werden in [TKH+05, TKH+04] ausführlich vorgestellt. Darunter fallen *hierarchische Methoden*, die für die zu untersuchenden Primitiven eine sogenannte Bounding Volume Hierarchie erstellen, um schnell bestimmen zu können, ob zwei Primitive sich überschneiden. Kollisionen zwischen zwei Objekten oder Selbstkollisionen werden durch den gegenseitigen Überschneidungstest der beiden Hierarchien erkannt. Dabei wird die Hierarchie entsprechend traversiert. *Stochastische Methoden* sind für interaktive Anwendungen sehr interessant, da bei diesen Genauigkeit und Geschwindigkeit gegeneinander abgewogen werden können. Oft wird eine hierarchische Datenstruktur verwendet und entsprechend um stochastische Methoden erweitert [KNF04]. Andere Ansätze bauen keine Hierarchie auf, sondern *unterteilen den Raum* und sortieren die Primitiven in einzelne Gitterzellen ein. Primitive, die in gleichen bzw. benachbarten Gitterzellen liegen, müssen anschließend auf Überschneidung getestet werden. Für textile Materialien gibt es verfeinerte Methoden, die der Tatsache Rechnung tragen, dass die zu simulierenden Objekte äußert dünn sind.

Spezielle Verfahren für textile Materialien

In den letzten Jahren sind einige spezielle Verfahren zur Kollisionserkennung für textile Materialien entwickelt worden, die besonders robust sind. In [Pro97] wird ein Verfahren zur *kontinuierlichen Kollisionserkennung* vorgestellt, das in [BFA02] aufgegriffen und verbessert wird. Der wesentliche Unterschied zu anderen Verfahren ist, dass nicht nur die neuen Positionen der Primitiven untersucht werden, sondern ebenfalls ihr Weg dorthin. Für einen Partikel und ein Dreieck bedeutet dies beispielsweise, dass der Zeitpunkt der Kollision für die vier beteiligten sich bewegenden Eckpunkte bestimmt werden muss. Hierfür ist die Koplanarität der vier Punkte zu einem bestimmten Zeitpunkt eine notwendige Bedingung, die geprüft werden kann. Danach muss noch verifiziert werden, ob tatsächlich eine Kollision vorliegt. Um bei diesem Ansatz die Primitiven zu bestimmen, die auf Koplanarität getestet werden müssen, kann eine Bounding Volume Hierarchie über den bewegten Primitiven aufgebaut werden. Insgesamt ist dieser Vorgang äußerst rechenintensiv.

Kürzlich wurden Methoden vorgestellt, die diesen Vorgang beschleunigen [GKJ+05, WB05]. In [GKJ+05] wird mit Hilfe einer chromatischen Zerlegung des Dreiecksnetzes die Anzahl der fälschlicherweise als kollidierend angenomme-

nen Paare von Primitiven stark reduziert. In Kombination mit einem GPU-basierten Verfahren zum Überlappungstest von bewegten Dreiecken, erreicht man eine starke Beschleunigung bei der Kollisionserkennung. Allerdings ist das Verfahren noch weit von Echtzeit entfernt. Für komplexere Dreiecksnetze dauert allein die Kollisionserkennung annähernd zwischen $100\,ms$ und $500\,ms$. Des Weiteren scheint das Verfahren nicht für mehrlagige Bekleidung geeignet zu sein, da in diesem Fall die chromatische Zerlegung keine besonderen Vorteile mehr bietet.

In [WB05] wird beschrieben, wie man eine deformierbare topologische 2-Mannigfaltigkeit in sogenannte $(\pi, \beta, \mathbf{I}) - surfaces$ aufteilen kann. Das sind genau die Bereiche, für die sichergestellt ist, dass in ihnen während eines Zeitschritts keine Selbstkollisionen auftreten werden. Somit stellt das Verfahren eine Erweiterung der Normal-Cones dar [Pro97], die nur Aussagen zu diskreten Zeitpunkten zulassen. Die Geschwindigkeit des Verfahrens ist ähnlich einzustufen, wie die der chromatischen Zerlegung [GKJ+05]. Der Algorithmus arbeitet schneller als bisherige Verfahren, ist aber für interaktive Anwendungen immer noch bei weitem zu langsam, da alleine die Kollisionserkennung für einen einzelnen Zeitschritt in etwa eine halbe Sekunde benötigt.

4.1.3 Verfahren zur Kollisionsbehandlung

Physikalische Kollisionsbehandlung

Bei der physikalischen Kollisionsbehandlung wird jedes Objekt von einem abstoßenden *Kraftfeld* umhüllt, welches andere Objekte im Abstand r mit einer Kraft $f \approx r^{-12}$ abstößt. Dadurch werden Durchdringungen der Objekte verhindert. Das Kraftfeld wirkt natürlich nur in sehr geringer Entfernung der beiden Objekte, dafür steigt die Kraft sehr schnell an, wenn sich die Objekte näher kommen. Mit dieser Methode wird der physikalische Vorgang von Kollisionen sehr genau modelliert. Dies bringt viele Vorteile bei der numerischen Integration der Bewegungsgleichung mit sich, da die Kollisionsbehandlung über Kräfte modelliert wird und somit direkt in die dynamische Analyse mit einfließt.

Allerdings treten auch eine Reihe von Nachteilen auf. Zunächst müssen die Kräfte sehr hoch sein, damit Objekte sich nicht durchdringen, was die Performanz bei der numerischen Integration verschlechtert oder gar zu Instabilitäten führen kann [Etz02]. Ein weiteres Problem tritt auf, wenn die Simulation in großen Zeitschritten $10^{-4}\,s$ bis $10^{-2}\,s$ durchgeführt wird. In diesem Fall kann ein Objekt in einem Zeitschritt völlig außerhalb der Wirkung des Kraftfelds liegen und im nächsten schon das Objekt durchdrungen haben. Um dieses Problem zu umgehen könnte man den Einflussbereich des Kraftfelds vergrößern. Allerdings kann man dann nicht mehr von einer physikalischen Kollisionsbehandlung sprechen, da Objekte in unrealistisch großer Entfernung von einander bleiben.

Geometrische Änderungen

Mittels geometrischer Änderungen kann auf einfache Weise eine Kollisionsantwort erzeugt werden, die den kollidierenden Partikel in einen neuen Zustand versetzt, der kollisionsfrei ist. Je nach gewähltem Verfahren kann dabei die Beschleunigung, die Geschwindigkeit und/oder die Position des Partikels geändert werden. Durch die Veränderung des Zustands der Partikel entsteht aus numerischer Sicht eine Singularität, die im nächsten Zeitschritt behandelt werden muss. Je nach verwendetem Integrationsverfahren kann dies eine verschlechterte Performanz oder sogar Instabilität bedeuten.

Der große Vorteil dieser Methode ist die einfache Umsetzbarkeit, da Kollisionsbehandlung und numerische Integration voneinander getrennt durchgeführt werden können. Weiterhin benötigt man keine Kraftfelder mit unrealistisch großem Einflussbereich, wie sie bei der physikalischen Kollisionsbehandlung verwendet werden müssten.

Eine weitere Möglichkeit besteht darin den Zustand der Partikel mit *Constraints* einzuschränken. Diese Methode wurde in [BW98] für die Stoffsimulation eingeführt und wird auch in [EEHS00, EKK+01] verwendet. Die Constraints definieren zusätzliche algebraische Gleichungen, die bei der Lösung der Gleichungssysteme, die bei impliziten Integrationsverfahren auftreten, berücksichtigt werden können. Dadurch wird die Kollisionsantwort mit der numerischen Integration eng gekoppelt. Dies wirkt sich positiv auf die Performanz und Stabilität aus.

Constraints reduzieren die Dimension des Raums, in dem sich ein Partikel bewegen kann. Beispielsweise kann erzwungen werden, dass ein Partikel sich nur noch orthogonal zur kollidierenden Fläche bewegt, wobei natürlich dafür gesorgt werden muss, dass der Constraint aufgehoben wird, wenn sich der Partikel wieder entfernt. Genau an dieser Stelle liegt das größte Problem dieses Verfahrens [Kim05]. Bei komplizierten Selbstkollisionen, wenn beispielsweise mehrere Stofflagen aufeinander liegen, kann es vorkommen, dass die Constraints nicht wieder aufgehoben werden. Dadurch erscheint der Stoff verklebt und verhält sich äußerst unrealistisch.

Sehr viel versprechend ist ein *hybrider Ansatz*, der geometrische Änderungen und Kraftfelder miteinander verbindet [BFA02]. Die Kräfte der Kraftfelder sind begrenzt und wirken nur in der Nähe der Oberflächen der Objekte. Damit Objekte sich nicht durchdringen, falls sie sich zu schnell aufeinander zu bewegen, wird zusätzlich in diesem Fall eine geometrische Änderung der Geschwindigkeiten vorgenommen. Dabei werden Kraftstöße zwischen den Objekten ausgetauscht, um eine realistische Kollisionsantwort zu erhalten.

Die Anwendung von Kraftstößen geht zurück auf Mirtich und Canny [MC94], die diese für die Kollisionsantwort bei der Simulation von starren Körpern verwendeten. Diese Methode modelliert die physikalischen Gegebenheiten sehr plausibel und erlaubt eine effiziente Berechnung der Kollisionsantwort. Die wichtigste Eigenschaft der Methode ist die Impulserhaltung bei einer Kollision. Durch

die Kraftstöße werden die Bewegungsimpulse der beiden Objekte so ausgetauscht, dass die Kollision aufgelöst wird, aber kein Impuls verloren geht.

4.2 Distanzfelder

4.2.1 Definition

Distanzfelder liefern für alle Punkte innerhalb des Feldes die minimale Distanz zu einer oder mehreren geschlossenen Flächen. Die Distanz kann mit einem Vorzeichen versehen werden, um zwischen Innen und Außen zu unterscheiden. Formal ist ein Distanzfeld D einer Fläche $\mathbf{S}$ definiert als ein Skalarfeld $D : \mathbb{R}^3 \rightarrow \mathbb{R}$, wobei für jeden Punkt $\mathbf{p} \in \mathbb{R}^3$ gilt

$$D(\mathbf{p}) = sign(\mathbf{p}) \cdot min\{|\mathbf{p} - \mathbf{q}| : \mathbf{q} \in \mathbf{S}\} \tag{4.1}$$

mit

$$sign(\mathbf{p}) = \begin{cases} 1 & \text{if } \mathbf{p} \text{ außen} \\ -1 & \text{if } \mathbf{p} \text{ innen} \end{cases} \tag{4.2}$$

Zur Bestimmung der Distanz können unterschiedliche Normen verwendet werden, wobei für die Kollisionserkennung die euklidische Norm gebräuchlich ist. Abkürzend wird daher in den folgenden Kapiteln wahlweise Distanz oder Abstand synonym für die euklidische Distanz benutzt. Zur Veranschaulichung des Konzeptes von Distanzfeldern ist in Abbildung 4.3 ein geometrisches Objekt und dessen partielles Distanzfeld dargestellt. Das Distanzfeld ist nur in einer Umgebung der Oberfläche gültig.

Die Darstellung einer geschlossenen Fläche durch ein Distanzfeld ist vorteilhaft, da keinerlei topologische Einschränkungen vorliegen. Außerdem kann die Auswertung von Distanzen und Normalen, die man für die Kollisionserkennung benötigt sehr schnell erfolgen und ist unabhängig von der Komplexität des Objektes.

Anwendungsbereiche

Distanzfelder haben ein weites Anwendungsspektrum. Sie wurden für Morphing [BMWM01, COSL98], volumetrische Modellierung [FPRJ00, BPK+02], Motion Planning [HKL+99] und für die Animation von Feuer [ZWF+03] verwendet. Distanzfelder werden manchmal auch als Distance Volumes [BMWM01] oder Distanzfunktionen [BMF03] bezeichnet.

Distanzfelder definieren eine Fläche als Isofläche zum Isowert Null. Aber im Gegensatz zu anderen impliziten Darstellungen erhält man durch eine einfache Funktionsauswertung die euklidische Distanz zu einer Oberfläche. Da Distanzfelder recht viele Informationen über eine Fläche speichern, ist die effiziente Berechnung eines Distanzfeldes zu einer gegebenen Darstellung einer Fläche immer noch ein offenes Forschungsgebiet [WK03, FPRJ00, JS01, SPG03, SOM04, BA05].

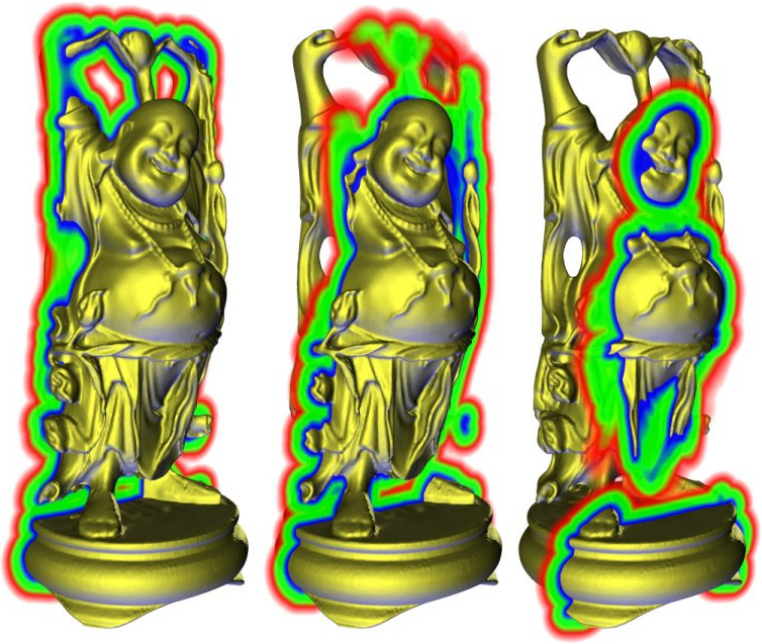


Abbildung 4.3: Die Happy Buddha Figur mit drei farbkodierten Schnitten durch das Distanzfeld. Blau entspricht kleinen und Rot mittleren Distanzen.

4.2.2 Datenstrukturen für Distanzfelder

Distanzfelder eignen sich sehr gut zur Kollisionserkennung bei der Animation textiler Materialien, da mit ihnen schnell Distanzen und Normalen, die für die Kollisionsantwort benötigt werden, ausgewertet werden können. Die Effizienz dieser Auswertung hängt stark von der Wahl der richtigen Datenstruktur zur Repräsentation eines Distanzfeldes ab. Dazu werden in diesem Kapitel verschiedene Datenstrukturen analysiert und auf ihre Eignung für die Kollisionserkennung bewertet.

Im Folgenden werden zunächst *kartesische Gitter* vorgestellt, die höchst effizient abgefragt werden können, aber auch sehr speicherintensiv sind. Danach wird auf *adaptive Distanzfelder* eingegangen, die durch einen Octree repräsentiert werden und somit mit wesentlich weniger Speicher auskommen. Durch diese Hierarchie verschlechtert sich aber auch die Abfragezeit. Deshalb werden noch weitere Hierarchien vorgestellt und schließlich eine neue Datenstruktur, das *Sparse Integer Distance Grid*, die sowohl kompakt als auch effizient ist.

Kartesische Gitter

Ein Rechteckgitter im $\mathbb{R}^3$ wird meist auf ein bestimmtes quaderförmiges Gebiet D beschränkt und es unterteilt seinen Definitionsbereich entlang der Achsen mit Trennebenen in wiederum quaderförmige Zellen. Die Abstände der Ebenen zuein-

ander können frei gewählt werden. Da für die Kollisionserkennung eine uniforme Abtastung von D sinnvoll erscheint, wird im Folgenden von *kartesischen Gittern* und somit von würfelförmigen Zellen ausgegangen. Kartesische Gitter lassen sich einfach implementieren und die Abfrage von Distanzen erfolgt in konstanter Zeit, sprich in $O(1)$. Dies ist insbesondere für Echtzeit-Anwendungen von großer Bedeutung, da unterschiedlich lange Abfragezeiten starke Schwankungen in der Framerate hervorrufen können. Dies stört die Immersion des Anwenders erheblich.

Die Distanzen $D(\mathbf{p})$ werden bei Verwendung von kartesischen Gittern nur für die Gitterpunkte berechnet. Werte innerhalb der Zellen werden mit Hilfe der trilinearen Interpolation rekonstruiert. Dabei stellt sich sofort die Frage: Wie gut können Objekte bzw. 3D-Flächen mit dieser Datenstruktur repräsentiert werden? Zur Beantwortung dieser Frage muss man unterschiedliche geformte Objekte unterscheiden. Da die trilineare Interpolation innerhalb der Gitterzelle eine glatte gekrümmte Fläche rekonstruiert, lassen sich glatte Flächen mit einem geringen Approximationsfehler in einem kartesischen Gitter repräsentieren, auch wenn die Auflösung des Gitters nicht allzu hoch ist. Zwischen den Zellen ist die Fläche hingegen nur C^0 stetig. Möchte man also den menschlichen Körper darstellen, der ja sehr glatt ist, eignen sich die kartesischen Gitter sehr gut. Im Gegensatz dazu lassen sich Objekte mit scharfen Kanten eher schlecht repräsentieren.

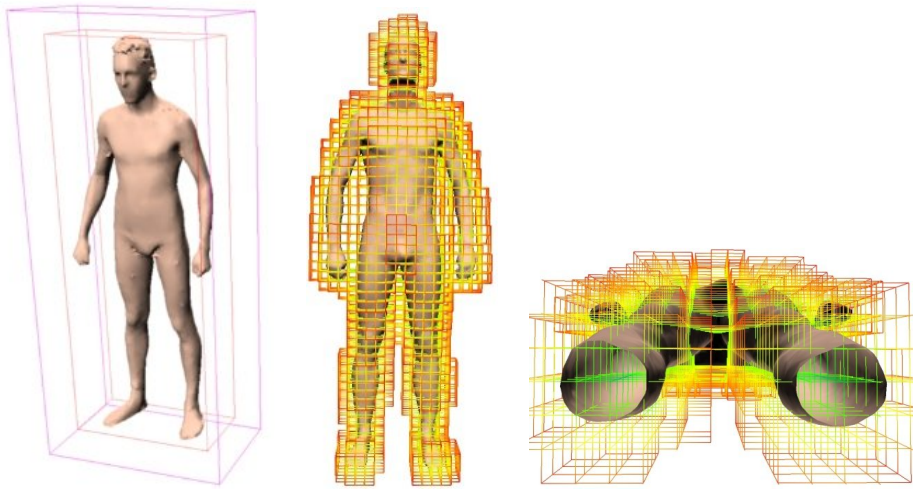
Für die Kollisionsantwort wird neben der Distanz $D(\mathbf{p})$ auch noch die Normale $\mathbf{n} = \mathbf{N}(\mathbf{p})$ benötigt, die für $\mathbf{p} \in \mathbf{S}$ der Oberflächennormale entspricht. Ansonsten soll $\mathbf{n}$ von der Fläche weg nach außen gerichtet sein (im Inneren des Objektes zur Fläche hin). Diese Information lässt sich durch die Berechnung des Gradienten gewinnen. Da der Gradient eines skalaren Feldes in Richtung der Normale der Isofläche zum aktuellen Isowert zeigt, muss der Gradient nur noch normalisiert werden und man erhält die Normale mit den gewünschten Eigenschaften:

$$\mathbf{N}(\mathbf{p}) = \frac{\nabla D(\mathbf{p})}{|\nabla D(\mathbf{p})|} \tag{4.3}$$

Für die Berechnung von $\nabla D(\mathbf{p})$ hat man mehrere Möglichkeiten. Sehr effizient ist es den analytischen Gradienten der trilinearen Interpolation zu bestimmen [FPRJ00]. Würde man eine Normale an jedem Gitterpunkt speichern, so steigt der Speicherverbrauch an und die Interpolation der acht Normalen wird dreimal so aufwändig, wie die Berechnung der Distanz.

Das Distanzfeld ist nur dort interessant, wo sich das repräsentierte Objekt befindet. Da die meisten Kollisionsobjekte nur eine beschränkte Ausdehnung haben, kann der Definitionsraum begrenzt werden. Dazu wird eine Bounding Box um das Objekt gebildet, die den Definitionsraum vorgibt. Die Bounding Box wird je nach Anwendung dann noch entsprechend vergrößert, so dass, für eine genügend große Umgebung um das Objekt herum, Distanzen korrekt abgefragt werden können (siehe auch Abbildung 4.4 (a)).

Die Nachteile von kartesischen Gittern sind die hohen Speicheranforderungen und die begrenzte Auflösung, wenn Objekte mit starker Struktur repräsentiert wer-



(a) (b) (c)

Abbildung 4.4: (a) Bounding Boxen um den Avatar. — (b) Distanzfeld eines Avatar. Als Datenstruktur wird ein kartesisches Gitter verwendet. — (c) Querschnitt durch die Beine. Die Distanzen sind farbkodiert.

den sollen. Beispielsweise werden für die Repräsentation eines Avatars mit den ungefähren Dimensionen $1m \times 0,5m \times 2m$ bei einer Breite der Gitterzellen von $0,5cm$ schon $200 \cdot 100 \cdot 400 = 8.000.000$ Gitterpunkte benötigt. Dies ergibt bei Verwendung von Gleitpunktzahlen mit einfacher Genauigkeit 4 Bytes pro Punkt und damit insgesamt $32MB$ an Daten.

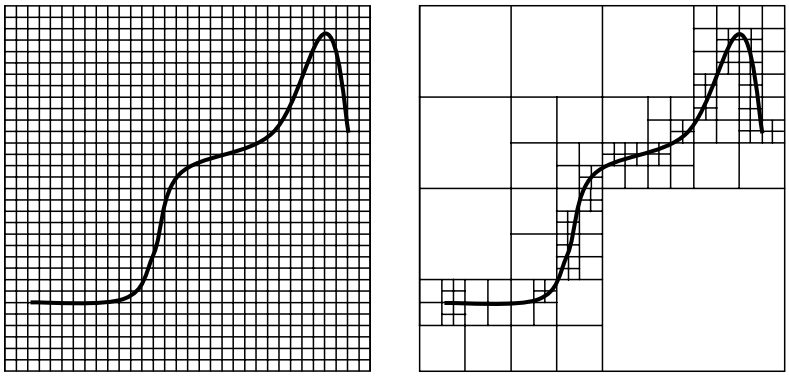
Da beim Auswerten des Distanzfeldes eine Interpolation stattfindet, erhält man bei niedriger Auflösung eine schlechtere Repräsentation und es entstehen die üblichen Abtastprobleme. Feine Details lassen sich somit nur durch eine hohe Auflösung darstellen. Unglücklicherweise impliziert aber eine Halbierung der Breite einer Gitterzelle einen acht Mal höheren Speicherverbrauch, womit schnell die Grenze bei der Erhöhung der Auflösung erreicht ist. Handelt es sich bei den zu repräsentierenden Objekten um virtuelle Menschen, werden erfreulicherweise keine allzu hohen Auflösungen benötigt, da der Körper bis auf wenige Ausnahmen relativ glatt ist.

In Abbildung 4.4 ist das Gitter des Distanzfeldes für einen virtuellen Menschen abgebildet. Nur Zellen, die nahe am Avatar liegen, sind dargestellt. Die Distanzen, die nur an den Eckpunkten gespeichert sind, sind farbkodiert und werden entlang der Kanten interpoliert.

Adaptive Distanzfelder

Um die Nachteile der hohen Speicheranforderung und der begrenzten Auflösung von kartesischen Gittern zu umgehen, wurden von Frisken et al. [FPRJ00] soge-

nannte *Adaptively Sampled Distance Fields* (ADFs) vorgeschlagen. In einem ADF werden Distanzen in einer Hierarchie gespeichert, die es ermöglicht die Abtastrate lokal zu erhöhen, wenn dies wegen feinen Details im Objekt nötig ist. Obwohl hierfür verschiedenste räumliche Datenstrukturen denkbar sind, werden ADFs für gewöhnlich in einem speziellen Octree gespeichert (siehe Abbildung 4.5 für einen Vergleich zwischen kartesischen Gittern und einem ADF).



(a) (b)

Abbildung 4.5: (a) Ein kartesisches Gitter für ein Distanzfeld einer Kurve. — (b) Ein adaptives Gitter, welches wesentlich weniger Zellen als das kartesische Gitter enthält.

Eine Octree Datenstruktur erlaubt einen effizienten Top-Down Aufbau eines ADFs. Jede Zelle des Octrees wird solange unterteilt, bis ein bestimmtes Gütekriterium erfüllt ist. Beispielsweise kann das Ergebnis der trilinearen Interpolation an ausgewählten Punkten mit den originalen Distanzen verglichen werden. Stimmen die beiden Werte bis auf einen frei wählbaren Approximationsfehler überein, muss nicht unterteilt werden. Als Punkte kommen die Mittelpunkte der Zelle, der Kanten und der Flächen in Frage (insgesamt 19 Vergleiche). Diese Unterteilungsregel weicht von einem herkömmlichen Octree [ESK97, Sam90] ab, bei der jede Zelle unterteilt wird, die nicht vollständig außer- oder innerhalb des Objektes liegt. Die Unterteilung des herkömmlichen Octrees wird bei einer bestimmten Tiefe des Baumes angehalten und bietet nur ein grobes Gütekriterium, das durch die Diagonale der kleinsten Gitterzelle festgelegt wird.

Verglichen mit kartesischen Gittern bieten ADFs eine sehr gute Kompressionsrate. In Abbildung [FPRJ00] werden ADFs und herkömmliche Quadtrees verglichen. Das ADF hat $1,7K$ Zellen. Nimmt man an, dass eine Zelle vier Werte speichert und vier Zeiger auf die nachfolgenden Zellen enthält, benötigt eine Zelle 32 Byte (jeweils 4 Byte pro Wert und pro Zeiger). Dies ergibt einen Speicherbedarf von $54,4kB$. Ein kartesisches Gitter mit der vollen Auflösung des ADFs hätte demnach $262K$ Zellen. Hier werden nur jeweils 4 Byte für den Wert in jeder Zel-

le benötigt, also insgesamt $1048\,kB$. Somit erhält man einen ca. 20-fach höheren Speicherbedarf. Als Nachteil eines ADFs ist die erhöhte Abfragezeit zu nennen, die durch die Traversierung des Baums entsteht.

Verwendet man ADFs zur Kollisionserkennung muss noch darauf geachtet werden, dass benachbarte Zellen auf unterschiedlichen Stufen des Baumes stetig ineinander übergehen [BMF03]. Hierfür kann ein Verfahren von Westermann et al. [WKE99] verwendet werden: Immer wenn eine Zelle zu einer gröberen Zelle benachbart ist, werden die Werte ihrer Gitterpunkte so geändert, dass sie den interpolierten Werten der gröberen Zelle entsprechen.

Andere Hierarchien

Benutzt man einen *Binary Space Partitioning Tree* (BSP-Baum), so kann der Speicherbedarf weiter gesenkt werden [WK03]. Dies wird dadurch erreicht, dass ein BSP-Baum flexiblere Unterteilungen erlaubt und dass eine stückweise lineare Approximation des originalen Distanzfeldes in den Zellen benutzt wird. Diese Approximation ist nicht C^0 stetig, was im Sinne einer Approximation einer Fläche ja auch nicht nötig ist. Wu und Kobbelt [WK03] stellen mehrere Algorithmen zur Auswahl von geeigneten Trennebenen des Baums vor und sie zeigen, dass die Repräsentation mit einem BSP-Baum sehr kompakt ist und die von ADFs sogar übertrifft.

Leider benötigt der Aufbau des Baumes aber eine ziemlich lange Zeit, was den Einsatz des Verfahrens in interaktiven Systemen erschwert, da hier nur eine kurze Zeit für Vorberechnungen zur Verfügung steht. Ein anderes Problem während der Kollisionserkennung kann durch die Unstetigkeiten zwischen den Zellen entstehen, da diese Risse in der Fläche nicht so einfach aufgelöst werden können wie bei den ADFs. Man benötigt daher eine sehr genaue Approximation der originalen Fläche und verliert dabei den Vorteil des geringen Speicherverbrauchs. Ähnlich wie bei ADFs muss auch hier eine Hierarchie traversiert werden, um Distanzen zu erhalten.

In [EHK$^+$00] wird vorgeschlagen den hohen Speicherbedarf durch Verwendung einer *Hashtabelle* zu reduzieren. Da für die Kollisionserkennung nur Gitterpunkte in einer Umgebung der zu repräsentierenden Oberfläche benötigt werden, können die meisten dieser Punkte einfach als weit entfernt oder weit im Objekt liegend markiert werden. Dadurch haben viele Punkte den gleichen Hashwert und der Speicherverbrauch sinkt drastisch. Eine Hashtabelle erlaubt Abfragen von Distanzen ähnlich wie ein kartesisches Gitter in $O(1)$. In der Implementierung dieser Methode zeigt sich aber, dass zwar die theoretische Zeitkomplexität der Abfragen gleich ist, aber in der Praxis die Hashtabelle um den Faktor 10 langsamer ist als ein kartesisches Gitter, welches als Array implementiert werden kann. Somit scheint diese Methode für interaktive Anwendungen weniger geeignet zu sein. Zusätzlich wird noch ein Level-of-Detail Ansatz vorgeschlagen, bei dem ein grobes reguläres Gitter in Bereichen feiner Details höher aufgelöst wird. Ergebnisse zu dieser Methode wurden aber nicht präsentiert.

Ein Ansatz um unterschiedlich lange Laufzeiten bei der Traversierung von Octree-basierten Datenstrukturen zu umgehen, wird in [MPT99] vorgestellt. Die Tiefe des Baums wird auf drei beschränkt. Dieser Ansatz stellt einen Kompromiss zwischen Baumtiefe und Speicherverbrauch dar und *verallgemeinert* die Idee des *Octrees*.

Jede Zelle im Baum bekommt 2^{3n} Nachfolger, wobei n eine ganze Zahl ist und die Zelle in 2^{3n} gleichgroße Würfel unterteilt wird. Setzt man $n = 1$ erhält mein einen Octree. Das Verfahren wurde im Rahmen von Haptic-Rendering eingesetzt und im Baum wurde ein Voxelmodell [1] eines Flugzeuges gespeichert. In diesem Kontext haben sich Bäume mit $n = 3$ als sehr speichersparend erweisen. Der optimale Wert hängt von den darzustellenden Daten ab und wie sich diese im Raum verteilen.

Durch die fixe Tiefe ist die Genauigkeit stark beschränkt. Für den vorgeschlagenen 512-Baum erhält man aber durchaus eine sehr feine Auflösung. Nimmt man ein Gebiet von $10\,m^3$ erhält man eine Breite einer Gitterzelle von ca. $2\,mm$. Insgesamt ist das Verfahren zwar eine Verallgemeinerung von Octrees, aber streng genommen wenig flexibel, da die Einschränkung auf einen quaderförmigen Wurzelknoten für die meisten zu repräsentierenden Objekte zu unflexibel ist.

Eine andere hierarchische Datenstruktur für Distanzfelder wird in [FUF06] vorgestellt. Ein Objekt wird mit einem *hierarchischen sphärischen Distanzfeld* repräsentiert. Zunächst wird für ein gegebenes polygonales Objekt ein sphärisches Distanzfeld aufgebaut, dessen Mittelpunkt in etwa im Schwerpunkt des Objekts liegt. Ausgehend vom Mittelpunkt wird ein sphärisches Höhenfeld erzeugt, wobei an jedem Abtastpunkt die minimale und maximale Distanz des Objektes vom Mittelpunkt gespeichert wird.

Für die Abtastung schlagen die Autoren ein Verfahren vor, dass eine Einheitskugel sehr gleichmäßig unterteilt, indem diese in sechs kongruente Regionen zerlegt wird. Jede Region wird dann mittels einer winkelbasierten Parametrisierung nochmals unterteilt. Werte zwischen den Gitterpunkten können dann ähnlich wie auch bei kartesischen Gittern rekonstruiert werden. Über dem sphärischen Distanzfeld wird dann eine Hierarchie aus Spherical Shells [KPLM98] aufgebaut, die für eine effiziente Kollisionserkennung traversiert werden kann.

Der Vorteil dieser Methode liegt in der effizienten Rotation von Objekten und der schnellen Bestimmung von Kollisionen zwischen zwei starren Körpern. Das Verfahren eignet sich nach Aussage der Autoren jedoch nicht für die Kollisionserkennung zwischen deformierbaren Körpern, da Updates der Dreiecke in den Blättern der Datenstrukturen sehr aufwändig sind. Ein weiterer Nachteil des Verfahrens ist, dass es nicht die exakte Topologie des Objektes speichert. Insbesondere Objekte mit Selbstverdeckung aus Sicht des Mittelpunktes können nicht akkurat

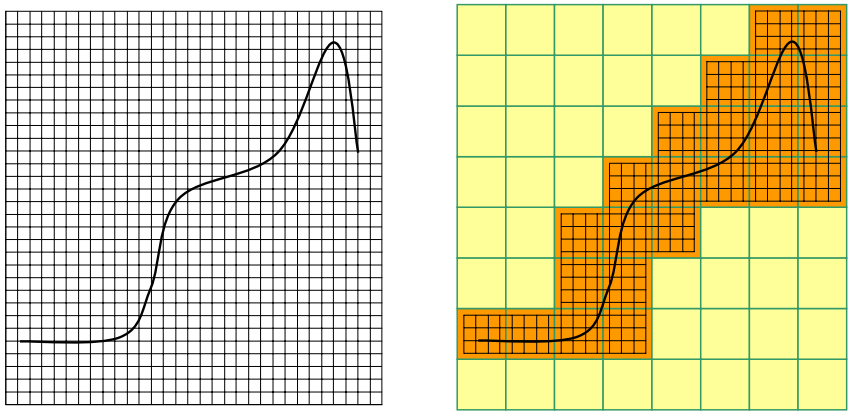
[1] In einem Voxelmodell werden Skalarwerte in der Mitte einer Zelle gespeichert und als konstant angenommen. Oft wird auch nur zwischen Innen, Außen und dem Rand eines Objektes unterschieden.

repräsentiert werden. Daher eignet sich diese Datenstruktur nicht für die Kollisionserkennung zwischen Avataren, die Selbstverdeckung besitzen, und deren Kleidung.

4.2.3 Sparse-Integer-Distance Grids

An dieser Stelle wird eine neue Gitterstruktur vorgestellt, das *Sparse-Integer-Distance Grid* (SID Grid). Es stellt eine Erweiterung des Sparse Block Grids [Bri03] dar. Wie oben beschrieben, genügt es für die Kollisionserkennung, wenn Distanzen nur in der Nähe der Objekte zur Verfügung stehen. Alle anderen Zellen außerhalb des Objektes werden mit *OUT* markiert, innere Zellen mit *IN*. In einem kartesischen Gitter bedeutet dies, dass sehr viele Zellen mit einem dieser Werte belegt sind. Diese Redundanz lässt sich verringern, wenn eine geeignete Hierarchie verwendet wird.

Unterteilt man den Definitionsbereich des Gitters in Blöcke, kann man in einem groben Gitter jeden Block mit einem Zeiger versehen. Ist der Block leer, verweist der Zeiger auf *IN* oder *OUT*, je nachdem wo der Block liegt. Ansonsten werden feinere Subgitter referenziert. Ein wesentlicher Unterschied zu ADFs besteht jetzt darin, dass sich die Ränder der Subgitter nicht überlappen. Dies ist in Abbildung 4.6 (b) dargestellt. Um die Auflösung zu erhöhen können weitere Hierarchiestufen eingeführt werden. Weitere Stufen können je nach Notwendigkeit und gewünschter Genauigkeit der Darstellung der Originalfläche auch adaptiv hinzugefügt werden.



(a) (b)

Abbildung 4.6: (a) Ein kartesisches Gitter für eine Kurve. — (b) Konzept des Sparse-Integer-Distance Grid: Ein grobes Gitter wird in mehrere Blöcke unterteilt. In der Nähe der zu repräsentierenden Kurve enthalten die Blöcke jeweils feinere Gitter.

Im Vergleich zu kartesischen Gittern ist die Evaluierung von Distanzen komplizierter und somit auch weniger effizient. Der aufwändigste Fall tritt dann auf, wenn für die bilineare Interpolation Werte in vier verschiedenen Blöcken benötigt werden. Bei geschickter Implementierung ist diese Abfrage aber nur unwesentlich langsamer als bei einem kartesischen Gitter.

Zum Vergleich des Speicherverbrauchs wurde ein Distanzfeld für einen Avatar (siehe Abbildung 4.7) mit einem kartesischen Gitter und einer Auflösung von $378 \times 396 \times 81$ erstellt. Dies entspricht $48,5\,MB$, wenn Gleitkommazahlen einfacher Genauigkeit verwendet werden (4 Byte pro Gitterpunkt). Das Sparse Block Grid mit der gleichen Auflösung und feinen $3 \times 3 \times 3$ Gittern benötigt hingegen nur $6,7\,MB$, wobei die Verweise auf die feinen Gitter $1,8\,MB$ belegen. Dies entspricht einer Ersparnis von 86%. Feine Gitter mit den Auflösungen 2^3 bzw. 4^3 benötigten jeweils mehr Speicher. Fairerweise muss erwähnt werden, dass der verwendete Avatar in einer für das kartesische Gitter nachteiligen Haltung steht.

Abbildung 4.7: Das Objekt, das für den Vergleich des Speicherverbrauchs verwendet wurde.

Eine weitere Stelle, an der der Speicherverbrauch verringert werden kann, ist die Darstellung der Distanzen in den Gitterpunkten. Da das Distanzfeld nur in der Nähe der Oberfläche gültige Werte liefern muss, genügt für die Distanzen ein Wertebereich von $[-\Delta, \Delta]$. Für menschliche Avatare kann $\Delta = 3\,cm$ gesetzt werden. Anstatt an jedem Gitterpunkt eine Gleitkommazahl zu speichern, können die Werte mit 8 oder $16\,Bit$ quantisiert werden. Bei $8\,Bit$ ergibt dies für den Avatar eine genügend hohe Auflösung von ca. $0,2\,mm$ pro Wert. Das SID Grid benötigt dann nur noch $3,0\,MB$, was einer Ersparnis von $93,8$% gegenüber einem kartesischem Gitter mit Gleitpunktzahlen entspricht.

In den Gitterzellen werden dann Integers gespeichert, die vor einer Abfrage entsprechend umgewandelt werden müssen. Da diese Umwandlung viele Taktzyklen benötigt, kann eine Lookup Table verwendet werden, die für jeden Integer die entsprechende Distanz als Gleitpunktzahl enthält. Durch die Verwendung einer solchen Lookup Table lässt sich die Abfrage von Distanzen nochmals deutlich beschleunigen.

Ein SID Grid ist wesentlich kompakter als ein kartesisches Gitter und es lässt sich sehr einfach erzeugen (im Gegensatz zu BSP-Bäumen). Es bietet wie auch kartesische Gitter Abfragen in $O(1)$ und ist im Gegensatz zu Hashtabellen auch in der Praxis sehr effizient. Der einzige Nachteil ist der etwas höhere Speicherbedarf als für ADFs bzw. BSP-Bäume.

Mit Hilfe eines kompakten SID Grids können problemlos mehrere Frames einer Animation eines Avatars im Speicher gehalten werden, wobei für jeden Frame ein eigenes SID Grid in einem Vorverarbeitungsschritt erzeugt wird. Hiermit lässt sich eine sehr effiziente Kollisionserkennung für Avatare mit vorgegebener Bewegung ermöglichen. Dies wird in Kapitel 4.4.6 präzisiert.

4.3 Berechnung eines Distanzfeldes

In vielen Anwendungen ist die Oberfläche eines Objektes durch ein Dreiecksnetz gegeben, das sich gegebenenfalls mit der Zeit deformiert (z.B. ein skelettanimierter Avatar). Daher wurde das Problem der Berechnung eines Distanzfeldes für ein gegebenes Dreiecksnetz besonders intensiv untersucht. Liegt ein Objekt in einer anderen Flächenrepräsentation vor, können die Methoden von Breen et al. [BMWM01] verwendet werden, um eine Konvertierung durchzuführen.

Ausgehend von den Gitterpunkten kann man unabhängig für jeden Punkt die Distanz zum Dreiecksnetz berechnen. Um diesen Vorgang zu beschleunigen können Baumdatenstrukturen verwendet werden, um weit entfernt liegende Dreiecke schnell auszuschließen. In einer älteren Arbeit werden hierfür Bounding Box Trees verwendet [PT92]. Später ging man zu Octrees über [JS01]. Allerdings sind diese Verfahren nicht konkurrenzfähig zu moderneren Verfahren, da die Laufzeit in der Größenordnung von Minuten ist. Der Vorteil dieser Methoden für einige Anwendungen ist, dass sie Distanzfelder für den gesamten Definitionsbereich berechnen. Für die Kollisionserkennung genügt jedoch die Bestimmung von Distanzen in einer kleinen Umgebung der Oberfläche. Dadurch kann der Rechenaufwand deutlich verringert werden und es lassen sich Algorithmen für diesen speziellen Fall entwickeln.

In [BA05] wird ein Verfahren zur Bestimmung der Vorzeichen eines Distanzfeldes auf der Basis einer Pseudonormale vorgestellt. Die Autoren zeigen, dass die nach Winkeln gewichtete Pseudonormale verwendet werden kann, um zwischen dem Inneren und Äußeren eines geschlossenen Meshes zu unterscheiden. Insbesondere weisen sie darauf hin, dass andere Pseudonormalen zur Bestimmung der Vorzeichen nicht geeignet sind, da man Gegenbeispiele konstruieren kann, in denen falsche Vorzeichen bestimmt werden. Eine Alternative zur Berechnung der Vorzeichen ist das Raycasting, wofür ein Verfahren in Abschnitt 4.3.5 vorgestellt wird.

Im Folgenden werden einige wichtige aktuelle Verfahren zur Erzeugung von Distanzfeldern vorgestellt und hinsichtlich der Eignung für die Kollisionserken-

nung bewertet. Zunächst wird ein naiver Algorithmus vorgestellt um die Problematik darzulegen. Danach werden einige bestehende Algorithmen präsentiert, die sich in folgende Klassen einteilen lassen:

- Propagationsverfahren
- Bildbasierte Ansätze
- Methoden basierend auf Voronoi-Diagrammen

Danach werden zwei neue Verfahren vorgestellt. Das erste ähnelt Methoden, die auf Voronoi-Diagrammen aufbauen. Es eignet sich besonders für sehr große Dreiecksnetze, wobei die einzelnen Dreiecke jedoch orientiert sein müssen. Es werden nur geringe Vorberechnungen durchgeführt und keine Hilfsdatenstrukturen benötigt. Daher arbeitet die Methode sehr effizient, allerdings gibt es einige Vorzeichenfehler. Das zweite Verfahren benutzt eine hierarchische Datenstruktur zur Berechnung der Distanzen und ist demnach ein Baum-basierter Ansatz. Zusätzlich wird ein gesonderter Algorithmus auf der Basis von Raycasting verwendet um die Vorzeichen (sprich Innen und Außen) zu bestimmen.

Der Naive Ansatz

Die Eingabedaten umfassen die Dreiecke, die das Objekt repräsentieren, und die einzelnen Gitterpunkte – zunächst erst einmal unabhängig von der verwendeten Datenstruktur. Es liegt nahe einfach für jeden Gitterpunkt den Abstand zu allen Dreiecken zu berechnen und dann den kürzesten davon zu behalten. Die Berechnung des Vorzeichens der Distanz sei an dieser Stelle außen vor gelassen.

Diese Methode ist äußerst einfach zu implementieren. Allerdings hat sie den großen Nachteil, dass der Aufwand proportional zu der Anzahl p der Gitterpunkte und der Anzahl t der Dreiecke wächst. Eine Berechnung eines Distanzfeldes für einen Avatar mit $20\,k$ Dreiecken und $200\,k$ Gitterpunkten, was einer Auflösung von nur $1,5\,cm$ entspricht, dauert mit dieser Methode ungefähr 8 Stunden. Eine solche Laufzeit ist nicht einmal in einer Vorberechnung akzeptabel, zumal die Auflösung viel zu gering ist.

4.3.1 Propagationsverfahren

Propagationsverfahren starten mit einem schmalen Band von Distanzen, die in der Nähe der Oberfläche des Dreiecksnetzes bestimmt werden. Diese initialen Informationen werden danach über benachbarte Gitterpunkte auf das gesamte Volumen verteilt. Dies kann entweder durch mehrere Sweeps entlang der Achsen erfolgen oder durch das Propagieren einer Front. Zwei Beispiele hierfür sind *Fast marching methods* [Set96] und *Distance transforms* [COSL98].

Sethian [Set99] hat die so genannten Level-Set Methoden in den Bereich der Computergraphik eingeführt. In diesem mathematischen Framework wird eine sich

ausbreitende Front durch die Eikonalgleichung $|\nabla u| = 1/f$ beschrieben. Nachdem ein schmales Band von Distanzen berechnet wurde, dienen diese Werte als Anfangswerte für einen Fast-Marching Algorithmus, der die Eikonalgleichung in $O(NlogN)$ löst. Man erhält ein vorzeichenbehaftetes Distanzfeld, wenn man für f eine konstante Funktion wählt.

In [JS01] werden verschiedene Distance Transforms verglichen. Bei einer Distance Transform wird der eigentlich globale Vorgang zur Berechnung von Distanzen durch einen lokalen Vorgang approximiert. Eine einfache *Champfer Distance Transform* ist sehr ungenau, die sie anstelle der euklidischen Distanzen diskrete Chamfer Distanzen verwendet. Speichert man an jedem Gitterpunkt noch einen Vektor, der zur Oberfläche zeigt, oder den nächsten Punkt auf der Oberfläche, erhält man genauere Lösungen. Generell werden mit diesen Methoden zumeist nur Teile eines Distanzfeldes in der Nähe der zu repräsentierenden Oberfläche berechnet. Je weiter man sich von der Oberfläche entfernt, desto höher wird auch der Approximationsfehler.

4.3.2 Bildbasierte Ansätze

Im Gegensatz zu den Propagationsverfahren berechnen bildbasierte Ansätze das volle Distanzfeld in seinem Definitionsbereich, der wie bei solchen Verfahren üblich dann quaderförmig ist. Hoff et al. [HKL$^+$99] schlagen vor die Graphik-Hardware zu benutzen um verallgemeinerte *Voronoi Diagramme* in zwei und drei Dimensionen zu berechnen. Es wird für jedes Voronoi Feature (Vertices, Kanten und Facetten) ein Distanzmesh aufgebaut. Für einen Punkt $\mathbf{p} \in \mathbb{R}^3$ ist dies beispielsweise ein zweischaliges Hyperboloid und für $\mathbf{p} \in \mathbb{R}^2$ ein Kegel. In 2D erhält man durch einfaches Rendern aller Meshes ein vorzeichenloses Distanzfeld im Z-Buffer der Graphik-Hardware. In 3D werden die Meshes wesentlich komplizierter und der Algorithmus geht schrittweise von Ebene zu Ebene vor. Daher muss eine gewaltige Menge von Dreiecken gerendert werden, was diese Methode sehr stark verlangsamt. Zudem ist die Konstruktion der Distanzmeshes schwierig zu implementieren.

Die Methode von Hoff et al. [HKL$^+$99] wurde etwas später noch erweitert, um auch das Vorzeichen von 2D Distanzfeldern mit der Graphik Hardware zu erzeugen [IZLM01]. Dazu wird in einem zweiten Renderingpass das Innere des Polygons in den Framebuffer gerendert um negative Gebiete zu markieren. Danach kann die Distanz aus dem Z-Buffer mit dem Vorzeichen aus dem Framebuffer kombiniert werden. Dieses 2D Distanzfeld kann beispielsweise für die Kollisionserkennung in der Ebene verwendet werden. Die Autoren berichten, dass das Verfahren interaktive Frameraten sowohl mit konkaven Starrkörpern, als auch mit deformierbaren Körpern erlaubt. Allerdings ist diese Methode nur auf Probleme anwendbar, die sich in zwei Dimensionen formulieren lassen.

In einem weiteren Verfahren wurden die oben beschriebenen Methoden noch verbessert [SOM04]. Durch Ausnutzung von räumlicher Kohärenz zwischen den einzelnen Schichten und inkrementeller Berechnungen, konnte die Renderge-

schwindigkeit und somit die gesamte Performanz deutlich erhöht werden. Die Autoren demonstrieren die Berechnung von hoch aufgelösten Distanzfeldern für große Modelle, die aus zehntausenden Primitiven bestehen, wobei sich die Rechenzeit um eine Größenordnung gegenüber [IZLM01] verbessert hat.

Ein anderer bildbasierter Ansatz beschleunigt die Berechnung der Distanzfelder durch eine Approximation der korrekten Distanz [VSC01]. Hier wird die Graphik Hardware verwendet, um zwei Tiefen-, Normalen- und Geschwindigkeitsbilder des Objektes zu konstruieren. Die Bilder werden jeweils für die Vorder- und Rückseite des Objektes aufgenommen. Aus diesen Bildern können dann alle Informationen für die Kollisionserkennung und -behandlung erzeugt werden. Die Tiefenbilder können sehr schnell erzeugt werden. Allerdings ist das Verfahren auf konvexe Objekte oder – im Fall von Avataren – auf geeignete Projektionsrichtungen beschränkt. Weiterhin liefert es beispielsweise an den Silhouetten keine korrekten Distanzen.

4.3.3 Beschränkte Voronoi Regionen

Ein Algorithmus, der lineare Komplexität in der Anzahl der Dreiecke eines Meshes besitzt, wurde von Breen et al. [BMWM01] vorgestellt. Die Methode verwendet ein *Voronoi Diagramm* der Facetten, Kanten und Vertices eines Dreiecksnetzes. Jede Voronoi Region wird nicht komplett dargestellt, sondern durch einen Polyeder repräsentiert. Dieser beschränkt die Voronoi Region auf ein Gebiet in der Nähe des jeweiligen Voronoi Features.

Die Polyeder werden entlang einer Achse des Gitters in Schichten zerteilt und die entstehenden Polygone werden scan-konvertiert um die Gitterpunkte zu bestimmen, die innerhalb liegen. Für diese Gitterpunkte können dann sehr einfach die Distanzen zum Voronoi Feature berechnet werden (siehe auch Abbildung 4.8). Die Orientierung der Dreiecke des Meshes liefert das korrekte Vorzeichen für jede Region. Sollte ein Gitterpunkt mehrfach scan-konvertiert werden, so wird die vom Betrag kleinste Distanz übernommen. Ein solcher Fall tritt beispielsweise auf, wenn ein Teil Oberfläche sehr nah an einem anderen Teil liegt.

Der Algorithmus arbeitet in der Praxis sehr effizient. Auch seine Zeitkomplexität von $O(t)$ mit t als Anzahl von Dreiecken, Kanten und Vertices kann als ziemlich gut angesehen werden.

Trotzdem wurde dieser Algorithmus von Sigg et al. [SPG03] beschleunigt, indem die Graphik Hardware benutzt wurde, um die Scan-Konvertierung der Polyeder vorzunehmen. Weiterhin beschreiben die Autoren, wie die Polyeder für Facetten, Kanten und Vertices durch einen einzigen Polyeder für jede Facette ersetzt werden können. Dadurch reduziert sich die Anzahl an Polyedern, die scan-konvertiert werden müssen, um den Faktor Drei. Es wird gezeigt, dass es möglich ist das Distanzfeld für den Stanford Bunny, der aus $69K$ Dreiecken besteht, in $3,7$ Sekunden zu erzeugen. Die Auflösung des Gitters betrug dabei 256^3 und die Bandbreite war auf 10% der Ausdehnung des Objektes beschränkt.

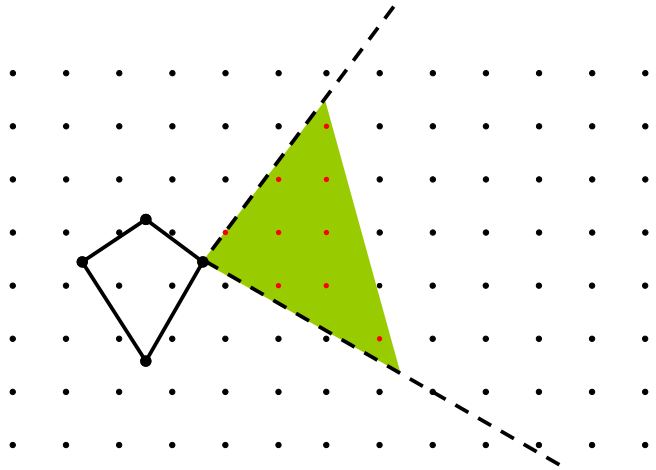


Abbildung 4.8: Scan-Konvertierung einer beschränkten Voronoi-Region. Nur für die roten Punkte müssen Distanzen ausgewertet werden.

4.3.4 Topologielose Berechnung

Dreiecksnetze werden häufig in einer indizierten Datenstruktur repräsentiert, da diese sich kompakt speichern lässt. Aus einem indizierten Dreiecksnetz lassen sich sehr effizient alle nötigen topologischen Informationen gewinnen. Aber oft werden die Dreiecke auch einzeln ohne jegliche Nachbarschaftsinformationen gespeichert. Sind die einzelnen Dreiecke nicht einmal orientiert und einige von ihnen degeneriert, spricht man von einer *Polygon Soup*. In diesem Fall kann natürlich kein Innen und auch kein Außen sinnvoll definiert werden.

Nimmt man aber an, dass bei Erstellung des Modells die Dreiecke korrekt orientiert wurden, lässt sich ein Distanzfeld definieren. Die Rekonstruktion der Topologie ist in diesem Fall zwar möglich, aber zeitaufwändig, da eine spezielle Datenstruktur erzeugt werden muss und geeignete Schwellwerte festgelegt werden müssen, um beispielsweise nahe beieinander liegende Vertices zusammenzufassen.

Da diese Nachbarschaftsinformationen aber benötigt werden um Voronoi Regionen zu bestimmen, können die sehr effizienten Ansätze aus Abschnitt 4.3.3 nicht verwendet werden. Daher wird in [FSG03] ein Algorithmus vorgeschlagen, der Distanzen unabhängig für jedes Dreieck berechnet. Außer einigen Vorzeichenfehlern, die aber nur an spitzen Dreiecken auftreten und das Resultat der fehlenden Information über adjazente Dreiecke sind, kann mit dieser Technik ein Distanzfeld für sehr fein tesselierte Objekte in kurzer Zeit berechnet werden. Das Verfahren benötigt beispielsweise nur 14 Sekunden, um das Distanzfeld für den Happy Buddha, der aus $1,1M$ Dreiecken besteht, zu berechnen (auf einem $167 \times 167 \times 400$ Gitter).

4.3.5 Shrinking Spheres

In diesem Abschnitt wird ein neues baumbasiertes Verfahren beschrieben. Es basiert auf der Methode von Payne und Toga [PT92]. Das ursprüngliche Verfahren wird an mehreren Stellen optimiert und es wird das Konzept der Shrinking Spheres eingeführt, das die Kernidee des Verfahrens anschaulich charakterisiert. Das Verfahren berechnet unabhängig voneinander die Distanzen und deren Vorzeichen, wobei letzteres im nächsten Abschnitt vorgestellt wird. Dem Autor ist derzeit kein schnelleres Verfahren bekannt, um ein Distanzfeld basierend auf einer Baumdatenstruktur zu generieren.

Sei eine Fläche $\mathbf{S}$ durch ein Dreiecksnetz $\mathbf{M} = \mathbf{M}(\mathbf{V}, \mathbf{T})$ beschrieben. Zunächst werden die Dreiecke von $\mathbf{M}$ in einen AABB-Baum eingefügt. Hierfür wird ein normaler Top-Down Ansatz verwendet. Anstatt der Dreiecke werden beim Aufbau der Hierarchie deren Schwerpunkte für die Berechnungen verwendet. In jedem Schritt wird jeweils die Achse mit der größten Varianz in den Schwerpunkten der Dreiecke geteilt. Diese Teilung erfolgt durch den Mittelpunkt zwischen maximalen und minimalen Schwerpunkt. Ergibt eine Teilung ein einzelnes Dreieck, so wird dieses in ein Blatt des Baumes einsortiert.

Ausgehend von einem Gitterpunkt $\mathbf{p}$ wird dessen Distanz bestimmt. Hierzu wird zunächst eine Kugel (Sphere) mit dem Radius ∞ um den Gitterpunkt gelegt, da die Distanz noch unbekannt ist. Im nächsten Schritt wird die Wurzel des AABB-Baum mit dieser Kugel auf Überschneidung geprüft. Dabei ergeben sich durch die Punkte der Bounding Box mit der größten und kleinsten Distanz zu $\mathbf{p}$ eine minimale d_{min} und maximale d_{max} mögliche Distanz zu $\mathbf{S}$. Daher kann der Radius der Kugel auf $r = d_{max}$ verkleinert werden. Jetzt wird auf dem AABB-Baum eine Breitensuche durchgeführt. Eine Bounding Box deren d_{min} kleiner als r ist kann verworfen werden. Trifft man auf einen Blattknoten wird die korrekte Distanz d zum Dreieck berechnet und ebenfalls $r = d$ gesetzt.

Die Breitensuche führt dabei zu einer möglichst schnellen Verringerung von d_{max} und damit des Radius der Kugel oder anders ausgedrückt zu einer *Shrinking Sphere*. Dies ist wichtig, da desto mehr Bounding Boxen ausgeschlossen werden können, je kleiner die Kugel ist. Die Überlappung von Kugel und Box kann effizient berechnet werden [Arv90, AMH02]. Bis zu dieser Stelle entspricht der Algorithmus bis auf die Terminologie dem Verfahren von [PT92].

Um schneller Bounding Boxen ausschließen zu können, wird in jedem Knoten des Baumes ein Punkt $\mathbf{p}$, der auf $\mathbf{S}$ liegt, gespeichert. Dieser wird so gewählt, dass er möglichst nahe am Mittelpunkt der Box liegt. Somit stellt $\mathbf{p}$ einen guten Repräsentanten der Oberfläche dar, der im Folgenden *Proxy* genannt wird. Dieser Proxy liegt, aus allen Richtungen betrachtet, ungefähr gleich weit vom Rand der Box entfernt. Da der Proxy $\mathbf{p}$ auf $\mathbf{S}$ liegt, kann er beim Berechnen der maximalen Distanz anstelle einer Ecke der Box verwendet werden. Dadurch kann der Radius der Kugel schneller verkleinert werden. In der Implementierung genügt es eine Referenz auf einen Vertex $\mathbf{v} \in \mathbf{V}$ zu speichern. Dies ist speichersparend und $\mathbf{v}$ lässt

Verfahren	Zeit	getestete Dreiecke	Bandbreite
Shrinking Spheres	$34,7s$	$11,8K$	$3cm$
	$111,5s$	$57,6K$	$10cm$
ohne Nachbarn	$38,0s$	$12,4K$	$3cm$
	$124,5s$	$58,6K$	$10cm$
ohne Proxy	$36,7s$	$12,1K$	$3cm$
	$111,2s$	$59,6K$	$10cm$
Payne und Toga [PT92]	$45,7s$	$14,5K$	$3cm$
(ohne Proxy und ohne Nachbarn)	$197,7s$	$70,9K$	$10cm$

Tabelle 4.1: Zeitmessungen zur Erzeugung eines Distanzfeldes für das Dreiecksnetz eines Avatars mit $19K$ Dreiecken in einer Auflösung von $186 \times 392 \times 78$ Gitterpunkten.

sich schnell bestimmen. Dazu benötigt man nur die Dreiecke, die die aktuelle Box umschließt.

Weiterhin können Kohärenzen für benachbarte Gitterpunkte ausgenutzt werden. Anstatt den Radius mit ∞ zu initialisieren können Distanzen von den benachbarten Gitterpunkten, für die schon ein korrekter Wert vorliegt, verwendet werden. Werden die Gitterpunkte Schicht für Schicht durchlaufen, so liegen an allen inneren Gitterpunkten drei bereits berechnete Distanzen $d_{\{1,2,3\}}$ an Nachbarpunkten vor. Davon wird die kleinste Distanz d_i gewählt. Der Radius wird dann auf $r = d_i + b$ gesetzt, wobei b die Breite einer Gitterzelle bezeichnet.

Wird kein volles Distanzfeld benötigt, sondern nur in einer Umgebung von $\mathbf{S}$, so kann der Radius mit dem entsprechenden Abstand initialisiert werden. Dies erlaubt ebenfalls den sehr schnellen Ausschluss von Boxen. Wird keine Überlappung festgestellt, kann der Gitterpunkt geeignet markiert werden.

Zur Validierung wurde das Verfahren in Java 1.4 implementiert. Danach wurden Zeitmessungen für die Erzeugung der Distanzfelder für das Dreiecksnetz eines Poser-Avatars durchgeführt. Die Messung wurde auf einem $2,0GHz$ Pentium IV erstellt. Die Ergebnisse sind in Tabelle 4.1 zusammengefasst. Man erkennt, dass das neue Verfahren für größere Bandbreiten 50% schneller ist, als das ursprüngliche Verfahren von [PT92]. Weiterhin stellt man fest, dass vor allem die Einführung eines Proxy's die Anzahl der zu testenden Dreiecke enorm senkt. Wenn das Verfahren mit anderen Algorithmen zur Erzeugung von Distanzfeldern verglichen werden soll, muss natürlich die Gesamtdauer des Verfahrens berücksichtigt werden. Für den Poser-Avatar muss hierzu noch die Zeit für den Aufbau der Hierarchie (ca. eine Sekunde) und für die Vorzeichen (ebenfalls ca. eine Sekunde) hinzugerechnet werden.

Vorzeichen durch Raycasting

Im vorigen Kapitel wurden die Distanzen ohne Vorzeichen berechnet. In diesem Abschnitt wird gezeigt, wie sich für das Distanzfeld D des Meshes $\mathbf{M} = \mathbf{M}(\mathbf{V}, \mathbf{T})$

die Vorzeichen berechnen lassen. Dafür wird davon ausgegangen, dass **M** ein mannigfaltiges Mesh und wasserdicht ist. Wasserdicht bedeutet, dass das Mesh keine Löcher hat und somit ein geschlossenes Objekt darstellt.

In Abbildung 4.9 wird das Prinzip der Raycasting-basierten Methode zur Berechnung der Vorzeichen der Gitterpunkte in einem Distanzfeld dargestellt. Es werden parallele Strahlen durch den Definitionsbereich von D geschickt und Schnittpunkte mit den Dreiecken gespeichert. Man kann annehmen, dass man sich am Anfang des Strahls außerhalb befindet, indem der Definitionsbereich initial geeignet vergrößert wird. An jedem Schnittpunkt wechselt der Strahl sein Vorzeichen und man ist abwechselnd innerhalb und außerhalb des Objekts. Das Vorzeichen kann an den entsprechenden Gitterpunkten zugewiesen werden.

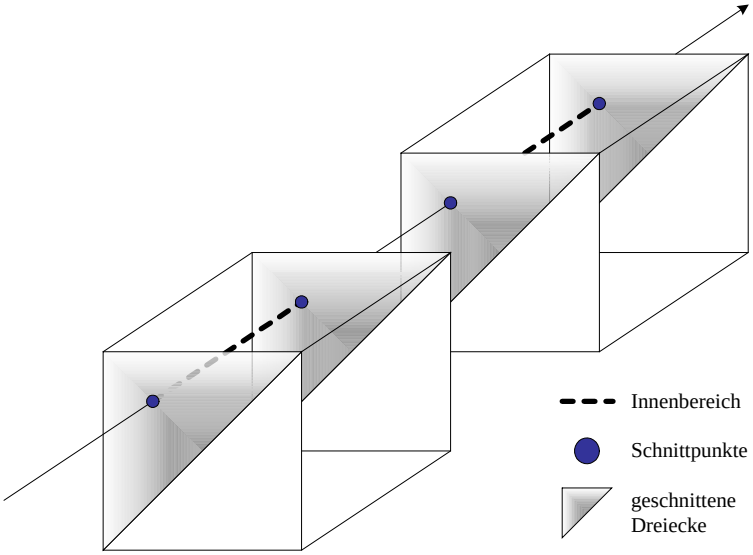


Abbildung 4.9: Raycasting-basierte Berechnung von Vorzeichen: An jedem Schnittpunkt wechselt das Vorzeichens

Der Algorithmus zur Bestimmung der Vorzeichen läuft in folgenden vier Schritten ab:

1. Bestimmung potenziell geschnittener Dreiecke
2. Bestimmung geschnittener Dreiecke
3. Berechnung der Schnittpunkte
4. Setzen der Vorzeichen entlang des Strahls

Die *Strahlen* laufen entlang einer beliebigen Achse durch das Gitter. Im Weiteren wird hierfür die z-Achse verwendet und das Gitter in der xy-Ebene abgearbeitet.

Für die Bestimmung der Dreiecke, die einen Strahl schneiden, kann dann deren *z*-Koordinate vernachlässigt werden. Mit $\mathbf{p} \in \mathbb{R}^2$ wird die Projektion des Strahls auf die *xy*-Ebene bezeichnet.

Um nicht jedes Dreieck mit dem Strahl zu schneiden, kann die AABB-Hierarchie aus Abschnitt 4.3.5 verwendet werden, um effizient die potentiellen geschnittenen Dreiecke zu bestimmen. Da die *z*-Koordinate unerheblich ist, kann ein Überlappungstest zwischen Strahl und AABB vermieden werden. Die Bounding Boxen werden ebenfalls in die *xy*-Ebene projiziert. Dann kann einfach überprüft werden, ob der Punkt **p** innerhalb der 2D Box liegt. Falls ja, wird wie üblich die Suche im Baum fortgesetzt bis man auf einen Blattknoten trifft.

In den Blattknoten wird überprüft, ob ein Dreieck tatsächlich vom Strahl geschnitten wird. Auch hier kann wieder in der Projektion gerechnet werden, d.h. man prüft ob der Punkt **p** innerhalb eines Dreiecks $\mathbf{T} \in \mathbb{R}^2$ liegt. Es sind unterschiedliche Herangehensweisen zur Lösung dieses Problems denkbar. Ein gängiger Ansatz ist es, den Flächeninhalt der drei Dreiecke zu berechnen, die der Punkt und die Eckpunkte von **T** bilden. Diesen vergleicht man dann mit dem Flächeninhalt von **T**. Der Punkt **p** liegt außerhalb, wenn der Flächeninhalt der Summe größer ist (siehe auch Abbildung 4.10).

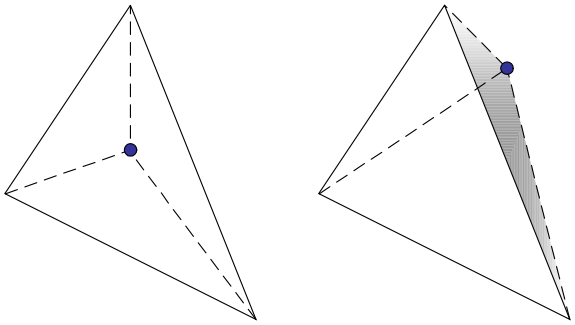


Abbildung 4.10: Test ob ein Punkt **p** innerhalb eines Dreiecks **T** liegt: Es wird der Flächeninhalt der drei Dreiecke, die von **p** aufgespannt werden, und der von **T** miteinander verglichen. **p** liegt innerhalb, wenn der Flächeninhalt gleich ist.

Da bei diesem Verfahren zwei Gleitkommazahlen verglichen werden, können dabei numerische Probleme auftreten, da ja mit begrenzter Genauigkeit gerechnet wird. Daher kann es vorkommen, dass ein Strahl der auf der gemeinsamen Kante zweier benachbarter Dreiecke liegt, als in beiden Dreiecken liegend erkannt wird. Solche mehrfachen Erkennungen müssen später korrigiert werden, da ansonsten das Vorzeichen fälschlicherweise gewechselt wird.

Nachdem man herausgefunden hat, das **p** innerhalb von **T** liegt, muss nur noch der exakte Schnittpunkt berechnet werden. Dazu genügt es den Schnittpunkt zwischen der Ebene, in der **T** liegt, und dem Strahl zu bestimmen. Sei $\mathbf{n} = (a,b,c)$ die

Normale des Dreiecks. Die Ebene kann dann durch die Normalform

$$\mathbf{n} \cdot \mathbf{q} = -d \tag{4.4}$$

beschrieben werden. d ergibt sich, wenn man für $\mathbf{q}$ einen Punkt der Ebene einsetzt. Da der Strahl parallel zur z-Achse verläuft, kann der Schnittpunkt in 2D durch einfache orthogonale Projektion bestimmt werden. Damit sind für den Schnittpunkt $\mathbf{s} = (x,y,z)$ die Werte x und y schon festgelegt, da es die Koordinaten des aktuellen Strahls sind.

Setzt man $\mathbf{q} = \mathbf{s}$ in Gleichung 4.4 und löst nach z auf ergibt sich

$$ax + by + cz = -d \tag{4.5}$$

$$z = \frac{-d - x - by}{c}. \tag{4.6}$$

Sollte $c = 0$ sein, dann steht $\mathbf{T}$ senkrecht zur xy-Ebene und wird auch nicht vom Strahl geschnitten. Dieses Dreieck spielt damit auch in der Berechnung des Vorzeichens keine Rolle mehr.

Im letzten Schritt werden die Vorzeichen entlang des Strahls gesetzt. Gegeben ist die Liste von Schnittpunkten bzw. Ein- und Austrittspunkten aus dem Mesh. Diese müssen noch nach z sortiert werden. Danach können doppelte Schnittpunkte, die durch numerische Ungenauigkeiten entstanden sind, entfernt werden. Die Vorzeichen erhält man, wenn man die Gitterzellen entlang des Strahls durchläuft. Dabei wird an jedem Gitterpunkt das Vorzeichen gesetzt und bei jedem Auftreffen auf einen Schnittpunkt aus der Liste das Vorzeichen gewechselt.

Der Vorteil dieses Verfahrens liegt in der Unabhängigkeit von der Berechnung der Distanzen. Zudem arbeitet es effizient und toleriert in gewissen Grenzen Löcher im Mesh. Die numerischen Probleme bei der Bestimmung, ob ein Dreieck geschnitten wird oder nicht, führen aber auch bei einigen Meshes zu Fehlern in den Vorzeichen, da der Übergang von Außen nach Innen nicht korrekt erkannt wird. Meist erhält man dann im Inneren eines Objektes positive Vorzeichen.

Des Weiteren nimmt der Algorithmus an, dass die Objekte geschlossen sind und nicht mehrere Schichten besitzen. Objekte mit mehreren dicht auf einander folgenden Schichten führen zu Problemen. Betrachtet man beispielsweise zwei Kugeln mit annähernd gleichem Radius, so stellt man fest, dass das gesamte Innere der Kugel positive Vorzeichen besitzt. Negative Vorzeichen tauchen nur zwischen der ersten und der zweiten Schicht auf.

Eine Lösung für die numerischen Probleme wäre die Scan-Konvertierung der Dreiecke bzw. von Polygonen, die entstehen, wenn man nur eine Schicht des Gitters betrachtet. Allerdings würde dies nicht das Problem mit offenen oder mehrschichtigen Meshes lösen. Offene Meshes lassen sich nur behandeln, wenn man die Definition von Innen und Außen nicht mehr so strikt nimmt. Danach ließe sich das Vorzeichen anhand der Orientierung der Oberfläche des Meshes bestimmen. Bei Objekten mit mehreren Schichten wäre auch ein Flood-Fill Algorithmus interessant. Zunächst würde man alle Gitterpunkte mit negativem Vorzeichen versehen.

Dann könnten positive Vorzeichen vom Rand des Definitionsbereichs nach Innen propagiert werden. An der Oberfläche würde der Vorgang gestoppt und man erhält korrekte Vorzeichen.

Distanz Punkt-Dreieck

Während der Berechnung eines Distanzfeldes wird sehr häufig die Distanz zwischen einem Punkt und einer Fläche im dreidimensionalen Raum ausgewertet. Meist ist die Fläche als Dreiecksnetz gegeben und daher wird ein Algorithmus zur Bestimmung des minimalen Abstandes eines Punktes $\mathbf{p}$ zu einem Dreieck $\mathbf{T}$ benötigt. Im Folgenden wird ein Algorithmus zu dieser Berechnung skizziert, der in [SE03] zu finden ist.

Sei das Dreieck als

$$\mathbf{T}(s,t) = \mathbf{p}_0 + s\mathbf{v}_1 + t\mathbf{v}_2 \quad \text{mit} \quad (s,t) \in [0,1]^2, s+t \leq 1$$

gegeben, wobei $\mathbf{p}_0$, $\mathbf{p}_1$ und $\mathbf{p}_2$ die Eckpunkte des Dreiecks und $\mathbf{v}_1 = \mathbf{p}_1 - \mathbf{p}_0$ und $\mathbf{v}_2 = \mathbf{p}_2 - \mathbf{p}_0$ sind. Die minimale Distanz wird berechnet, indem man die Werte (s',t') bestimmt, die den Punkt $\mathbf{p}'$ auf dem Dreieck beschreiben, der $\mathbf{p}$ am nächsten ist. Dazu wird zunächst die quadrierte Distanzfunktion zwischen $\mathbf{p}$ und $\mathbf{T}$ gebildet

$$\mathbf{Q}(s,t) = |\mathbf{T}(s,t) - \mathbf{p}|^2 \quad \text{mit} \quad (s,t) \in D.$$

Minimiert man $\mathbf{Q}(s,t)$ über D ergibt sich $\mathbf{p}'$ auf dem Dreieck. Da $\mathbf{Q}$ stetig differenzierbar ist, tritt das Minimum entweder innerhalb des Dreiecks auf (wenn $\nabla\mathbf{Q} = (0,0)$) oder an dessen Rand. Der Algorithmus muss im zweiten Fall herausfinden wo $\mathbf{p}'$ auf dem Rand des Dreiecks liegt. Dazu werden sieben Fälle unterschieden, da die Kanten des Dreiecks die Ebene in der es liegt, in ebendiese Anzahl von Regionen unterteilen. Eine vollständige Beschreibung des Algorithmus und dessen Implementierung sind in [SE03] zu finden.

4.4 Kollisionserkennung mit Distanzfeldern

In den letzten Jahren wurde das Thema Kollisionserkennung intensiv erforscht und es sind eine Reihe von effizienten Algorithmen entwickelt worden, die auf spezielle Anwendungen hin optimiert wurden. Beispielsweise bestehen 3D-Szenen häufig aus starren Körpern, die sich in einer statischen Umgebung bewegen. In diesem Fall liefern Algorithmen zur hierarchischen Kollisionserkennung schnelle und genaue Lösungen. Dafür wird eine Bounding Volume Hierarchie (BVH) für jeden Körper erstellt und während der Animation können Kollisionen erkannt werden, indem die BVHs rekursiv gegeneinander auf Durchdringungen getestet werden. Da diese Tests für die jeweiligen Bounding Volumes sehr effizient durchgeführt werden können und die Anzahl der etwas aufwändigeren Tests mit geometrischen Primitiven für gewöhnlich sehr gering ist, können somit Echtzeit-Anwendungen realisiert werden.

Sobald aber der Fall eintritt, dass eines der Objekte nicht mehr starr ist, verkomplizieren sich die Algorithmen. Man denke etwa an ein Stück Stoff, das durch eine virtuelle Umgebung gezogen wird. Da textiles Material sich leicht biegen lässt, kann es passieren, dass der Stoff mit seiner ganzen Fläche im Kontakt mit anderen Objekten ist. In diesem Fall verschlechtert sich die Performanz der Kollisionserkennung mit einer BVH enorm. Obwohl die Hierarchie immer noch sehr effizient nahe beieinander liegende Dreiecke erkennen kann, müssen eine Vielzahl von Durchdringungstest mit Primitiven durchgeführt werden, da alle Primitiven des deformierbaren Objektes gegen die des anderen Objektes getestet werden müssen. In vielen Anwendungen bedeutet dies, dass sehr viele Dreiecke gegeneinander getestet werden müssen. Dieser Ansatz ist für interaktive und Echtzeitanwendungen nicht durchführbar. Insbesondere, wenn die Anwendung nicht nur Kollisionen behandeln muss, sondern auch noch Differentialgleichungen für die dynamische Bewegung der Objekte lösen muss.

Zur Überwindung dieser Performanz-Probleme wird in diesem Kapitel ein Algorithmus zur schnellen Kollisionserkennung zwischen deformierbaren und starren Körpern beschrieben, der die zuvor beschriebenen Distanzfelder (siehe Abschnitt 4.2) als Basis für die nötigen Tests auf Nähe und Durchdringung verwendet. Dabei werden die starren Körper durch ihre Distanzfelder repräsentiert. Das Verfahren ist nicht nur effizient, sondern auch exakter als bisherige Methoden, die Voxelmodelle verwendet haben [MPT99, DMB00].

Im Folgenden wird zunächst beschrieben, wie Kollisionen an sich erkannt werden können. Danach wird auf die Behandlung der erkannten Kollisionen im Rahmen der physikalisch basierten Simulation eingegangen. In diesem Kontext wird auch ein neues Reibungsmodell vorgestellt, um das Verhalten, wenn Stoff über eine Oberfläche gezogen wird, akkurat zu simulieren. Danach wird beschrieben wie mit konkaven Regionen des starren Objektes umgegangen werden kann und wie mehrere starre Körper, die eine kinematische Kette bilden, gemeinsam behandelt werden können. Weiterhin wird dargelegt, wie mit Hilfe von Distanzfeldern Kollisionen zwischen animierten Objekten und deformierbaren Objekten erkannt werden können. Abschließend werden die Ergebnisse der Implementierung des Verfahrens präsentiert und gezeigt, wie das Verfahren im Bereich der interaktiven Textilanimation eingesetzt werden kann.

4.4.1 Kollisionen mit Starrkörpern

In diesem Abschnitt wird beschrieben, wie Kollisionen zwischen Starrkörpern und deformierbaren Objekten wie beispielsweise Stoff oder Haar effizient erkannt werden können. Es wird angenommen, dass sowohl das deformierbare Objekt, als auch der starre Körper zunächst als Dreiecksnetze vorliegen. Für den starren Körper wird dann sein Distanzfeld D berechnet. Um die Kollisionen zu überprüfen, wird eine Approximation des deformierbaren Objektes durchgeführt: Anstatt jedes Dreieck des deformierbaren Objektes zu testen, werden nur die Vertices betrachtet.

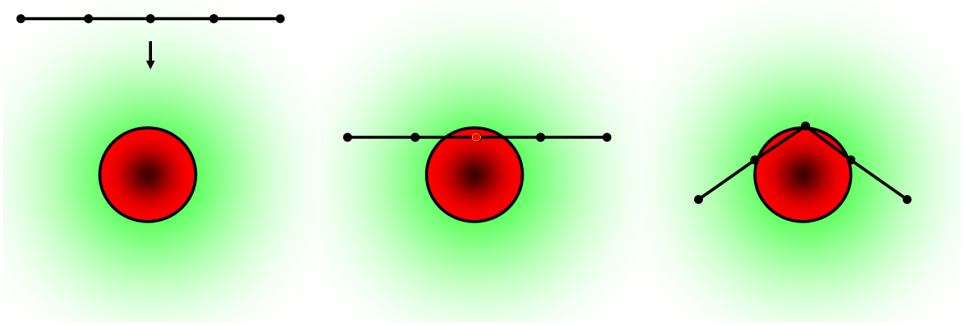


Abbildung 4.11: Links: Ein deformierbares Objekt bewegt sich auf eine Kugel zu. — Mitte: Ein Vertex ist in das Innere der Kugel eingedrungen. — Rechts: Der Vertex wurde nach oben verschoben, um einen kollisionsfreien Zustand zu erhalten. Dabei deformiert sich das Objekt.

Eine Kollision hat stattgefunden, wenn ein Vertex $\mathbf{p}$ eine negative Distanz hat. Dazu muss nur noch die Distanzfunktion des starren Körpers ausgewertet werden und überprüft werden, ob

$$D(\mathbf{p}) < 0 \tag{4.7}$$

gilt. In Abbildung 4.11 ist dies dargestellt.

Dieser Ansatz ist durchaus gerechtfertigt, wenn man bedenkt, dass die Oberflächen von deformierbaren Objekten oft sehr fein diskretisiert sind. Besonders im Bereich der Animation von textilem Material sind hoch aufgelöste Meshes unabdingbar um das komplexe Biegeverhalten dieser Materialklasse simulieren zu können. Weiterhin erlaubt diese Approximation eine interaktive Anwendung wie später noch gezeigt wird. Dies wäre nicht möglich, wenn eine vollständige Kollisionserkennung durchgeführt würde. Das punktweise Abfragen von Distanzen kann effizient mit den bereits vorgestellten Datenstrukturen für Distanzfelder durchgeführt werden (siehe Abschnitt 4.2.2). Die Bestimmung des minimalen Abstandes zwischen einem Dreieck und einer Oberfläche, die durch ein Distanzfeld repräsentiert ist, wäre hingegen äußerst komplex.

Sind *zwei deformierbare Objekte* gegeben, so kann das Verfahren erweitert werden, indem die Vertices auf den Oberflächen der beiden Objekte gegen das jeweils andere Distanzfeld geprüft werden. Allerdings müssen in diesem Fall die Distanzfelder in jedem Schritt neu aufgebaut werden, was recht rechenintensiv ist. Daher eignen sich Distanzfelder insbesondere für die Animation von deformierbaren Flächen über starren Körpern, für die keine physikalisch basierte Simulation durchgeführt wird. In diesem Fall müssen nur die Vertices der Fläche auf Kollision überprüft werden.

Um Artefakte bei der Kollisionsbehandlung, wie sie in Abbildung 4.11 rechts zu sehen sind, zu vermeiden, müssen die Vertices durch einen Offset von der

Isofläche zum Isowert Null verschoben werden. Hierzu kann ein vordefinierter Offset ε verwendet werden (siehe auch Abbildung 4.12). Dazu wird Gleichung 4.7 entsprechend angepasst und ein Vertex kollidiert, wenn

$$D(\mathbf{p}) < \varepsilon \tag{4.8}$$

gilt. Der ε Offset hängt von der Auflösung des Meshes des deformierbaren Objektes ab und davon, ob es hohe Krümmungen im starren Körper gibt. Je feiner das deformierbare Mesh aufgelöst ist, desto kleiner kann ε gewählt werden. Andererseits bedürfen hohe Krümmungen ein größeres ε.

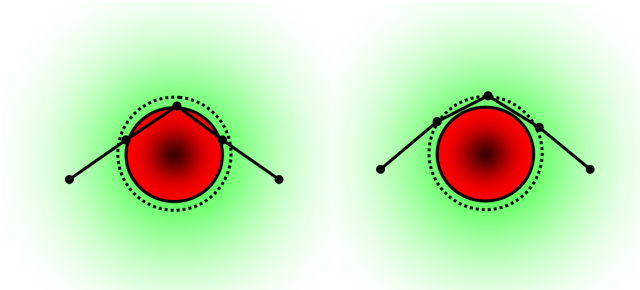


Abbildung 4.12: Links: Testet man Kollisionen nur an Vertices, treten Durchdringungen auf. — Rechts: Die Einführung eines ε-Offsets löst das Problem.

Ein großer Vorteil eines Distanzfeldes ist es, dass man mit dieser Datenstruktur nicht nur effizient Kollisionen erkennen kann, sondern dass man auch sofort die Eindringtiefe (engl. *penetration depth*) und mit wenig mehr Aufwand die Normale der Oberfläche erhält. Diese beiden Informationen werden später für die Kollisionsbehandlung benötigt.

Während der Kollisionserkennung muss die Distanzfunktion an einem bestimmten Punkt $\mathbf{p}$ ausgewertet werden. Verwendet man ein kartesisches Gitter zur Repräsentation des Distanzfeldes, muss man hierfür die Gitterzelle bestimmen, in der $\mathbf{p}$ liegt. Dann kann aus den acht Eckpunkten mit trilinearer Interpolation die Distanz rekonstruiert werden. Die Normalen lassen sich ebenfalls nur mit den Eckpunkten bestimmen, indem man den analytischen Gradienten berechnet (siehe dazu Abschnitt 4.2.2). Dieses Auswerten der Distanzfunktion kann sehr effizient durchgeführt werden und erlaubt somit das Verarbeiten einer sehr großen Anzahl von Vertices.

4.4.2 Kollisionsbehandlung

Da bei der physikalisch basierten Simulation die Kollisionserkennung und das Simulationsmodell eng ineinander greifen, werden zunächst kurz die einzelnen Schritte einer Iteration in der Simulation zusammengefasst. Danach wird der Algorithmus zur Kollisionserkennung vorgestellt. Es wird von einem Partikelsystem

ausgegangen, in dem jeder Vertex einer diskretisierten Fläche (oder auch Volumens) einen Partikel $\mathbf{p}_i$ darstellt. Die Masse, die in einer Fläche normalerweise homogen verteilt ist, wird in den Partikeln als m_i zusammengefasst.

Ein neuer Simulationsschritt startet mit den alten Positionen $\mathbf{x}_i^n \in \mathbb{R}^3$ und Geschwindigkeiten $\mathbf{v}_i^n \in \mathbb{R}^3$ aller Partikel. Die Bewegung der Partikel wird durch die Newton'sche Bewegungsgleichung

$$\mathbf{f}_i = m_i \cdot \mathbf{a}_i = m_i \cdot \frac{d^2\mathbf{x}_i^n}{dt^2} \tag{4.9}$$

bestimmt, wobei $\mathbf{f}_i \in \mathbb{R}^3$ die Kräfte und $\mathbf{a}_i \in \mathbb{R}^3$ die Beschleunigungen, die auf den Partikel wirken bezeichnen. Bevor die Gleichung gelöst werden kann, müssen die internen und externen Kräfte bestimmt werden, die auf die Partikel wirken. Danach kann über die Zeit integriert werden und man erhält die neuen Positionen $\mathbf{x}_i^{n+1}$und Geschwindigkeiten $\mathbf{v}_i^{n+1}$der Partikel. Der Ablauf eines Simulationsschrittes wird in Kapitel 5 ausführlicher beschrieben.

Kollisionen werden einfach dadurch aufgelöst, dass ein Partikel zurück zur Oberfläche bewegt wird, falls er das andere Objekt durchdrungen hat. Da diese Verschiebung die Trajektorie des Partikels ändert, müssen nach der Kollisionsantwort noch die Geschwindigkeiten korrigiert werden. Danach ist das Partikelsystem in einem Zustand, in dem keine Durchdringungen sichtbar sind. Jetzt wird der Zeitschritt beendet und die neuen Positionen der Partikel können an die Visualisierung weitergereicht werden.

Es sei angemerkt, dass bei dieser Art der Behandlung von Kollisionen die Berechnung der Kräfte der Partikel nicht beeinflusst wird. Daher wird das Gesamtsystem nicht steifer und die Bewegungsgleichung kann so gelöst werden, als ob keine Kollisionen auftreten würden.

Andere Techniken, wie beispielsweise das Constraint-based-Modeling [BW98], fügen zwar auch keine zusätzlichen Kräfte ein, aber erhöhen den Rechenaufwand und die Bewegung der Partikel verläuft weniger glatt und in einigen Fällen ruckartig. Techniken, wie z.B. *Penalty-Methoden* und *Lagrange-Multiplikatoren* zur Behandlung von Kollisionen sind gegenüber der hier vorgestellten Methode ebenfalls im Nachteil. Bei ersteren können die Kollisionen durch Einführung großer Energien (steife Federn) hergestellt werden, was die Steifigkeit des Systems verschlechtert. Zudem führt diese Methode nicht zur exakten Vermeidung der Kollisionen. Lagrange-Multiplikatoren führen ebenfalls zusätzliche Kräfte zur Kollisionsbehandlung ein. Dadurch fließen jedoch weitere Variablen (die Multiplikatoren) und Gleichungen (die Kräfte) in die Bewegungsgleichung mit ein, was die gesamte Performanz stark senkt.

Kollisionsantwort

An dieser Stelle wird der Algorithmus zur Berechnung der Kollisionsantwort im Detail vorgestellt, wobei zunächst angenommen wird, dass sich das Kollisionsobjekt nicht bewegt. Dieser Fall wird später in Abschnitt 4.4.5 erläutert.

Wenn ein Partikel als kollidierend erkannt wird, d.h. $\mathbf{x}_i^{n+1}$ ist dichter am Objekt als der Schwellwert ε erlaubt, wird er in Richtung der Normalen $\mathbf{n}$ des Distanzfeldes an der Stelle $\mathbf{x}_i^{n+1}$ zurückgesetzt.

Sei d der Abstand des Vertex von der Oberfläche des Objektes, dann berechnet

$$\mathbf{r}_n = (\varepsilon - d)\mathbf{n} \tag{4.10}$$

den Anteil der Kollisionsantwort in Normalenrichtung. Um die Reibung zu approximieren, wird zusätzlich ein Anteil $\mathbf{r}_t$ in tangentialer Richtung berechnet, indem der tangentiale Teil der Bewegung des Vertex $\Delta\mathbf{x}_t$nach

$$\Delta\mathbf{x}_t = \Delta\mathbf{x} - \mathbf{n}\langle\Delta\mathbf{x} \mid \mathbf{n}\rangle \tag{4.11}$$

$$\mathbf{r}_t = -c_f\Delta\mathbf{x}_t \tag{4.12}$$

skaliert wird, wobei $\Delta\mathbf{x} = \mathbf{x}_i^{n+1} - \mathbf{x}_i^n$ und c_f der Reibungsparameter ist. Setzt man $c_f = 1{,}0$ wird die gesamte tangentiale Bewegung aufgehoben. Die endgültige Position des Vertex wird durch

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^{n+1} + \mathbf{r}_n + \mathbf{r}_t \tag{4.13}$$

berechnet.

Es sei noch erwähnt, dass der Algorithmus eine völlig unelastische Kollision modelliert. Daher geht die gesamte Energie des Aufpralls verloren. Durch geeignete Skalierung von $\mathbf{r}_n$ könnte auch eine elastische Kollision modelliert werden, falls dieser Effekt für die Anwendung von Interesse ist. Für textiles Material, das eine hohe interne Dämpfung aufweist, ist dieses unelastische Verhalten realistisch. Die später gezeigten Beispiele verwenden alle das unelastische Modell.

Obwohl dieser Algorithmus nicht den exakten Schnittpunkt $\mathbf{p}_{is}$ zwischen der Trajektorie des Partikels und der Oberfläche des Objektes berechnet, erhält man in der Praxis sehr gute Resultate. Dies liegt daran, dass der Fehler, der bei dieser Annahme gemacht wird, nur gering ist. Die Oberflächennormale an der Stelle $\mathbf{x}_i^{n+1}$ zeigt nur in eine geringfügig andere Richtung, als sie an $\mathbf{p}_{is}$ zeigen würde. Des Weiteren ist die Skalierung der tangentialen Komponente nur ein klein wenig zu groß. Dies hilft aber sogar die Stabilität des Systems zu Erhöhen, da die Partikel etwas stärker gebremst werden.

Für die genauere Lösung könnte man die Distanz bestimmen, die der Partikel zurücklegen darf, bevor er das Objekt trifft. Dann könnte man $\mathbf{x}_i^n = \mathbf{p}_{is}$ setzen und den zuvor beschriebenen Algorithmus anwenden. Dies hätte aber zur Folge, dass zwei zusätzliche Distanzen abgefragt werden müssen: Eine an $\mathbf{x}_i^n$ um den Schnittpunkt $\mathbf{p}_{is}$ zu finden und eine weitere an $\mathbf{p}_{is}$ selbst. Da das Auswerten von Distanzen der rechenintensivste Teil des Algorithmus ist, verdoppelt sich beinahe die Dauer pro Partikel. Aus diesem Grund und weil sich in der Praxis keine besseren Ergebnisse mit der genaueren Auswertung erzielen lassen, kann der einfache Algorithmus, der nicht den genauen Schnittpunkt berechnet, verwendet werden.

Wichtig ist, dass der Algorithmus Partikel, die aufliegen, korrekt behandelt, damit ein Stück Stoff, das auf eine Fläche fällt, zur Ruhe kommt und nicht durch falsche Kollisionsbehandlung anfängt sich wie von Geisterhand zu bewegen. Im hier beschriebenen Algorithmus konvergiert ein Partikel gegen diese Ruhelage, je näher er der Fläche kommt, da dann $\mathbf{x}_i^{n+1} = \mathbf{p}_{is}$ gilt.

Einige Probleme verbleiben jedoch noch und werden in den folgenden Kapiteln behandelt. Zunächst können einzelne Partikel in konkaven Regionen hängen bleiben. Des Weiteren wird das einfache Reibungsmodell, das oben beschrieben wurde und welches kein realistisches Verhalten nachbildet, durch ein verbessertes Modell ersetzt.

4.4.3 Hängende Partikel

Der im vorigen Abschnitt beschriebene Algorithmus arbeitet sehr gut für konvexe Objekte. Aber sollte das Objekt konkave Regionen besitzen, so kann ein Partikel hängen bleiben. Dies ist in Abbildung 4.13 dargestellt. Die Kraft, die durch einen verbunden Partikel verursacht wird, versucht den Partikel weiter zu ziehen, während die Kollisionsantwort den Partikel immer wieder zurück an die Oberfläche schiebt. Dies resultiert in einem Deadlock und lässt das System explodieren, wenn die Kraft, die an dem einen Partikel zieht, immer weiter erhöht wird.

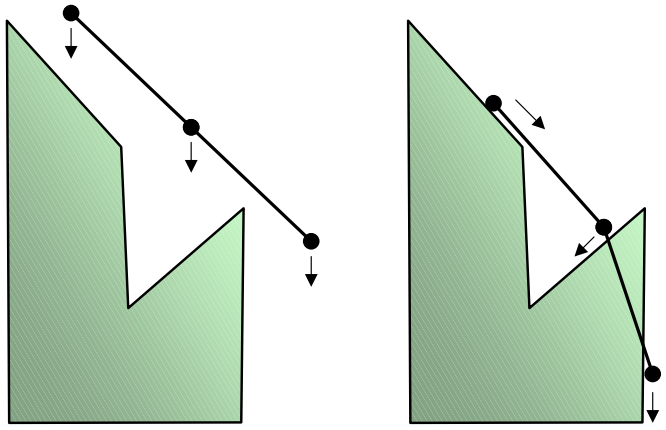


Abbildung 4.13: Die Kollisionsantwort kann fehlerhaft arbeiten, wenn nur einzelne Partikel betrachtet werden: Der Partikel in der Mitte wird durch die Kollisionsantwort in eine konkave Region gezogen, aus der er nicht mehr hinauskommt.

Um dieses Problem zu lösen, kann ein zusätzlicher Test an den Kanten in die Kollisionsbehandlung mit aufgenommen werden. Die Mitte $\mathbf{p}_{ce}$ einer jeden Kante des Partikelsystems wird auf den Abstand zum Objekt hin überprüft. Falls nötig, wird die gleiche Methode wie in Abschnitt 4.4.2 angewandt. Der resultierende Korrekturvektor $\mathbf{r}_n + \mathbf{r}_t$ wird dann zu jedem der beiden Partikeln, die die Kante bilden, hinzu addiert. In den meisten Fällen müssen nicht alle Kanten überprüft werden.

Man muss nur Kanten überprüfen, von denen einer (oder beide) der adjazenten Partikel einen geringen Abstand zur Oberfläche hat.

Als angenehmer Nebeneffekt, kann man auf diese Weise nicht nur Partikel aus der konkaven Region befreien, sondern es wird ermöglicht, dass man ε stark verkleinern kann, ohne dass Artefakte sichtbar werden. Durch das Überprüfen der Kanten in ihrer Mitte wird die Auflösung des Meshes virtuell erhöht. Dies erlaubt wiederum, dass man sich dem Objekt stärker nähern darf.

Allerdings gibt es auch einen Nachteil, da im schlimmsten Fall die Anzahl der Kollisionsbehandlungen um den Faktor vier steigt. Aber da die fest hängenden Partikel in vielen Szenarien auftreten und die gesamte Genauigkeit erhöht wird, kann eine Implementierung des Kantentests sehr empfohlen werden.

4.4.4 Verbesserte Modellierung von Reibung

In diesem Abschnitt wird ein neues Verfahren zur schnellen Modellierung von Reibung auf der Basis des Coulomb'schen Reibungsmodells vorgestellt. Das in Abschnitt 4.4.2 beschriebene Modell für Reibung ist äußerst simpel und man erhält damit kein realistisches Verhalten des Stoffs, da die tangentiale Bewegung nur mit einem fixen Wert skaliert wird. Das Coulomb'sche Reibungsmodell würde bessere Ergebnisse erlauben, da es die Reibungskräfte zwischen sich berührenden Körpern berücksichtigt. Diese Eigenschaft fehlt im Modell aus Abschnitt 4.4.2. In der Praxis bedeutet dies, dass Partikel eine schiefe Ebene immer mit der gleichen Geschwindigkeit hinab gleiten, egal wie steil man die Ebene auch stellt. Weiterhin können Partikel sogar langsam über vertikale Flächen gleiten. Diese Effekte sind mehr als unerwünscht. Deshalb wird hier ein genaueres Reibungsmodell auf Basis Coulomb'scher Reibung entwickelt.

Das Reibungsmodell nach Coulomb

Sobald sich zwei Körper berühren treten Reibungskräfte auf, die weiterer Bewegung entgegen wirken. Wird beispielsweise ein Körper auf einer Unterlage bewegt, wirkt die Reibung als Reaktionskraft dieser Bewegung entgegen. Zwei Faktoren beeinflussen die Höhe dieser Kraft. Zunächst hängt die Reibung von der Oberflächenbeschaffenheit der beiden Körper ab. Rauere Oberflächen erzeugen oft eine höhere Reibung, dies ist jedoch nicht zwangsläufig der Fall. Der zweite Faktor ist die Normalkraft, die die beiden Körper zusammen drückt. Weiterhin kann man zwischen Haft- und Gleitreibung unterscheiden, da beim Gleiten meist eine geringere Reibungskraft überwunden werden muss, als wenn ein Körper zunächst noch angeschoben werden muss.

Der französische Physiker und Ingenieur Charles Coulomb stellte fest, dass die Kraft aufgrund der Reibung proportional zur Kraft in der Richtung der Normale der Fläche ist:

$$F_{f0} = \mu_0 F_N \qquad \text{bzw.} \qquad F_f = \mu F_N \tag{4.14}$$

Hierbei bezeichnet μ_0 den Haftreibungskoeffizienten und μ den Gleitreibungskoeffizienten. Diese beiden Koeffizienten hängen von der Oberflächenbeschaffenheit der beiden Körper ab und müssen empirisch bestimmt werden. Insbesondere hängen die Koeffizienten von der Paarung der beiden Körper ab. Daher muss für jede Paarung ein eigener Versuch durchgeführt werden. Dieses Verhalten lässt sich dadurch veranschaulichen, dass die Reibung von der Mikrostruktur der Oberfläche abhängt. Je nach Paarung können sich die beiden Oberflächen mehr oder weniger stark ineinander verzahnen und daher ist Reibung jeweils völlig unterschiedlich. Weiterhin lässt sich mittels Versuch zeigen, dass die Größe der Reibungsfläche keine Rolle spielt.

Die Reibungskoeffizienten sind meist kleiner Eins. Viele Materialien haben einen Gleitreibungskoeffizienten von ca. $0,5$. Dieser Wert wird als Standard im Simulationssystem verwendet, wenn kein anderer gemessener Wert zur Verfügung steht. Einige weitere Koeffizienten finden sich beispielsweise in [Her02] auf Seite 103. Besonders geringe Reibung besitzt Teflon-Stahl; diese Kombination hat einen Haftreibungskoeffizienten von $0,04$. Eine recht hohe Gleitreibung besitzt beispielsweise Aluminium-Aluminium mit $\mu = 1,05$.

Kollisionsantwort unter Berücksichtigung der Reibung

Das hier vorgestellte Modell zur Berücksichtigung der Reibung hat zum Ziel die Performance nicht allzu sehr zu beeinträchtigen, daher werden wie auch bei der bisherigen Kollisionsbehandlung nur die Positionen der Partikel verändert. Auf die Unterscheidung zwischen Haft- und Gleitreibung wird zunächst verzichtet. Dieses Modell lehnt sich an das Reibungsmodell von [BFA02] an, verwendet jedoch keine Information über die Geschwindigkeit der Objekte.

Zunächst wird die Kraft approximiert, die für die weitere Berechnung der Reibung benötigt wird. Dazu wird angenommen, dass die Kraft proportional zur Positionsänderung des Partikels ist. Daher kann $\Delta\mathbf{x}_n$ als eine Annäherung der Kraft in der Richtung der Normale der Fläche verwendet werden und $\Delta\mathbf{x}_t$ als Kraft in tangentialer Richtung. Man erhält

$$\beta = \max\left(\frac{\|\Delta\mathbf{x}_t\| - \mu\|\Delta\mathbf{x}_n\|}{\|\Delta\mathbf{x}_t\|}, 0\right) \tag{4.15}$$

als Skalierungsfaktor für die tangential Bewegung des Partikel, wobei $\beta = 0$ keine Bewegung bedeutet und $\beta = 1$ keine Reibung. Für den Fall, dass keine tangentiale Bewegung stattgefunden hat, d.h. $\|\Delta\mathbf{x}_t\| = 0$, kann die Berechnung der obigen Gleichung übersprungen werden und $\beta = 0$ gesetzt werden.

Für den Korrekturvektor in tangentialer Richtung folgt damit

$$\mathbf{r}_t = (\beta - 1)\Delta\mathbf{x}_t. \tag{4.16}$$

Dieses verbesserte Reibungsmodell erhöht den Realismus von simulierten Objekten, ohne allzu viele zusätzliche Berechnungen einzuführen. Außerdem sind keine Änderungen an vorhergehenden Schritten der Simulation nötig, was die Implementierung stark erleichtert.

4.4.5 Erweiterung auf bewegte Starrkörper

In diesem Abschnitt wird beschrieben, wie das Grundverfahren erweitert werden kann, so dass auch Kollisionen zwischen textilem Material und bewegten Starrkörpern korrekt erkannt werden können. Besonders problematisch sind *rotierende Starrkörper*, da in diesem Fall ein Distanzfeld keine Information über die Rotation enthält. So ändert sich beispielsweise das Distanzfeld einer Kugel, die um ihren Mittelpunkt rotiert, nicht. Ein weiteres Problem stellt die korrekte Behandlung der *Reibung* dar, da die Kraft in Richtung der Normalen von der Relativbewegung des Körpers und des Stoffs abhängt. Diese Relativbewegung wird in Gleichung 4.15 noch nicht berücksichtigt.

Es ergeben sich mehrere Anforderungen an das Verfahren zur Kollisionsbehandlung, damit ein realistisches Stoffverhalten simuliert werden kann. Zunächst muss beachtet werden, dass zwischen Objekten, die sich auseinander bewegen, keine Reibung mehr wirken darf. Weiterhin muss es möglich sein zwischen Haft- und Gleitreibung zu unterscheiden. Viele bisherige Simulationsmodelle berücksichtigen diesen Unterschied nicht. In [BMF03, Kim05] wird für beide Arten von Reibung der jeweils gleiche Koeffizient verwendet. In [WB05] wird eine Unterscheidung vorgenommen, allerdings ist das Verfahren für interaktive Anwendungen nicht effizient genug. Jedoch ist besonders bei Stoff die Unterscheidung zwischen Gleit- und Haftreibung recht ausgeprägt, da sich beobachten lässt, dass viele textile Materialien eine verhältnismäßig hohe Haftreibung besitzen.

Schließlich muss die Masseträgheit des Stoffs auch bei bewegten Starrkörpern wirken. Dazu ein Beispiel: Liegt ein Tuch auf einer rotierenden Scheibe, die langsam beschleunigt wurde, wirkt die ganze Zeit über die Haftreibung und Tuch und Scheibe bewegen sich gemeinsam. Wird jetzt die Rotation der Fläche abrupt angehalten, so rotiert das Tuch aufgrund der Masseträgheit weiter.

Bewegung von Starrkörpers und Distanzfeld

Eine Bewegung eines Starrkörpers besteht nur aus einer Rotation und einer Translation. Daher kann diese affine Transformation gut als homogene 4*x*4-Matrix dargestellt werden. Der Körper selbst wird nicht verformt und der Abstand von jeweils zwei Punkten des Körpers bleibt invariant. Deshalb wird auch keine aufwändige Neuberechnung des Distanzfeldes nach jeder Bewegung des Starrkörpers benötigt. Es genügt eine affine Transformation des Distanzfeldes durchzuführen, da sich dann Objekt und Distanzfeld gleichermaßen bewegen lassen.

Anstatt aber alle Gitterpunkte [2] zu transformieren, wird die inverse Transformation auf die zu behandelnden Partikel angewandt. Sei im Folgenden $\mathbf{T}$ die Transformation des Körpers K.

Unter Verwendung des Distanzfeldes D und dessen Gradienten $\mathbf{N}$ (siehe 4.2) ergibt sich für einen Partikel $\mathbf{p}$ im Weltkoordinatensystem der Abstand d als

$$d = D(\mathbf{T}^{-1}\mathbf{p}). \tag{4.17}$$

[2] Bei Verwendung eines kartesischen Gitters zur Speicherung der Distanzen.

Für die Normale **n** gilt dann

$$\mathbf{n} \quad = \quad \mathbf{T}^t\mathbf{N}(\mathbf{T}^{-1}\mathbf{p}). \tag{4.18}$$

In dieser Gleichung wird die transponierte Matrix **T** verwendet, um die Normale korrekt in das Koordinatensystem von **p** zu transformieren (für eine Erläuterung siehe [ESK96]).

Kollisionsbehandlung

Zwei wesentliche Ideen ermöglichen die Kollisionsbehandlung mit bewegten Starrkörpern auf der Basis von Distanzfeldern. Zunächst wird für jeden Partikel die Information gespeichert, ob er im letzten Zeitschritt mit dem Starrkörper kollidiert ist. Falls ja, weiß man dass der Partikel im Kontakt mit dem Starrkörper war, da eine unelastische Kollision modelliert wurde. Zudem wird die Normale **n** der Oberfläche des Körpers gespeichert, die für die Kollisionsantwort bereits berechnet wurde.

Damit die Relativbewegung berechnet werden kann, wendet man die Änderung an der affinen Transformation $\mathbf{T}_\Delta$ des Starrkörpers auf den alten Kollisionspunkt an, der der Position des Partikels $\mathbf{x}_i^n$ im letzten Zeitschritt entspricht.

Die Änderung an der Transformation $\mathbf{T}_\Delta$ lässt sich bequem berechnen, wenn man die alte Transformation $\mathbf{T}_n$ und die aktuelle Transformation $\mathbf{T}_{n+1}$ zur Verfügung hat: $\mathbf{T}_\Delta = \mathbf{T}_{n+1}\mathbf{T}_n^{-1}$. Der Vorteil dieser Methode ist die Unabhängigkeit von der Implementierung des Animationssystems und man muss nur auf bereits berechnete Transformationen zurückgreifen. Die Invertierung von $\mathbf{T}_n$ stellt ebenfalls kein Problem dar, da diese nur einmal pro Zeitschritt durchgeführt werden muss.

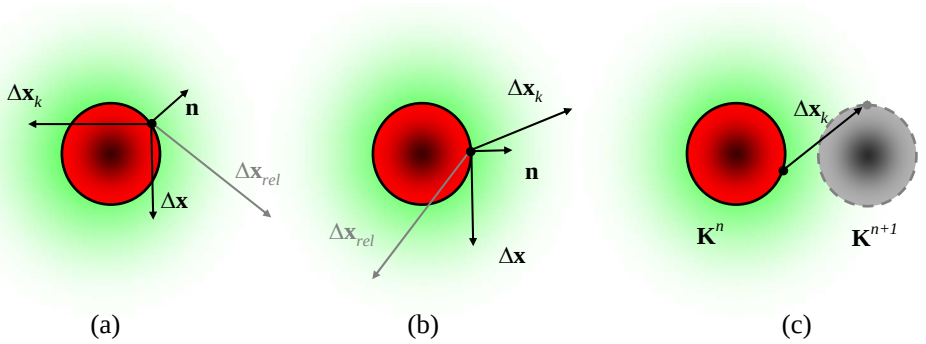
Mit Hilfe von $\mathbf{T}_\Delta$ ergibt sich der Kollisionspunkt des Starrkörpers im neuen Zeitschritt. Somit erhält man mit

$$\Delta\mathbf{x}_k \quad = \quad \mathbf{T}_\Delta\mathbf{x}_i^n - \mathbf{x}_i^n \tag{4.19}$$

die linearisierte Bewegung des Starrkörpers am Kollisionspunkt des Partikels $\mathbf{p}_i$ (siehe auch Abbildung 4.14 (c)). Durch die Verwendung der affinen Transformation $\mathbf{T}_\Delta$ zur Bestimmung von $\Delta\mathbf{x}_k$ erhält man den erwünschten Effekt, dass die Geschwindigkeit unterschiedlich hoch ist, je nach Abstand zur Rotationsachse. Allerdings ist die Geschwindigkeit aufgrund der Linearisierung der Bewegung geringfügig zu niedrig, was sich in der Praxis jedoch nicht bemerkbar macht.

Bemerkung: Teilt man die Relativbewegung, die hier beschrieben wird, durch die Dauer eines Zeitschritts, erhält man die Relativgeschwindigkeit, die in anderen Verfahren meist verwendet wird [BFA02, BWAK03]. Die Relativgeschwindigkeit muss allerdings nach Berechnung der Reibungskräfte noch geeignet integriert werden. Dieser Schritt ist bei dem hier beschriebenen Verfahren nicht mehr notwendig.

Anhand der Bewegung des Körpers $\Delta\mathbf{x}_k$ und der Bewegung des Partikels $\Delta\mathbf{x}_i = \mathbf{x}_i^{n+1} - \mathbf{x}_i^n$ kann die Relativbewegung zueinander berechnet werden. Die Relativbewegung allein lässt aber noch keine Aussage über das Kollisionsverhalten



(a) (b) (c)

Abbildung 4.14: (a) Die Relativbewegung $\Delta\mathbf{x}_{rel}$ von Objekt und Partikel zeigt, dass sich die beiden Objekte voneinander wegbewegen. Die Bewegung des Partikels allein genommen suggeriert eine Kollision. — (b) Aufgrund der Relativbewegung ergibt sich eine Kollision. — (c) Berechnung der Bewegung eines Starrkörpers anhand der alten und neuen Position eines fixen Punktes auf dem Körper. Im Beispiel wurde die Kugel verschoben und gedreht.

der beiden Objekte zu. Hierfür wird zusätzlich die Oberflächennormale $\mathbf{n}$ benötigt, mit der entschieden werden kann, ob die Objekte sich voneinander entfernen oder ob sie kollidieren. Sei

$$x_{rel} = (\Delta\mathbf{x}_i - \Delta\mathbf{x}_k)\mathbf{n} \tag{4.20}$$

die Relativbewegung bezüglich $\mathbf{n}$. Wenn $x_{rel} \leq 0$ ist, kollidieren die Objekte bzw. sind sie in Kontakt. Andernfalls entfernen sich die Objekte voneinander. Diese Sachverhalte sind in den Abbildungen 4.14 (a) und (b) illustriert. In (b) schiebt die Kugel den Partikel vor sich her. Dies zeigt insbesondere, dass unter Berücksichtigung der Relativbewegung ein hoher Anteil der Bewegung in Normalenrichtung vorhanden ist. Diese Bewegung muss bei der Kollisionsbehandlung zu einer entsprechend hohen Reibung führen. Das Zusammenkleben von Flächen wird hierbei nicht modelliert, könnte aber durch eine entsprechende Klebekraft (analog zur Reibungskraft) in das Verfahren integriert werden.

Damit die Reibungskräfte korrekt berechnet werden, genügt es in Gleichung 4.15 die Werte für $\Delta\mathbf{x}_n$ und $\Delta\mathbf{x}_n$ durch die Relativbewegung in tangentialer und normaler Richtung zu ersetzen. Man sieht leicht ein, dass durch diese Vorgehensweise die Anforderungen an die Masseträgheit erfüllt sind.

Anhand der resultierenden Tangentialbewegung nach Berücksichtigung der Reibung, lässt sich die Unterscheidung zwischen Haft- und Gleitreibung durchführen. Findet keine relative Tangentialbewegung statt, so gleiten die Objekte nicht aneinander vorbei und im *nächsten* Zeitschritt wirkt Haftreibung, d.h. μ wird in Gleichung 4.15 auf den dem Material entsprechenden Koeffizienten μ_0 gesetzt. Andernfalls rutschen die Objekte und es wirkt Gleitreibung.

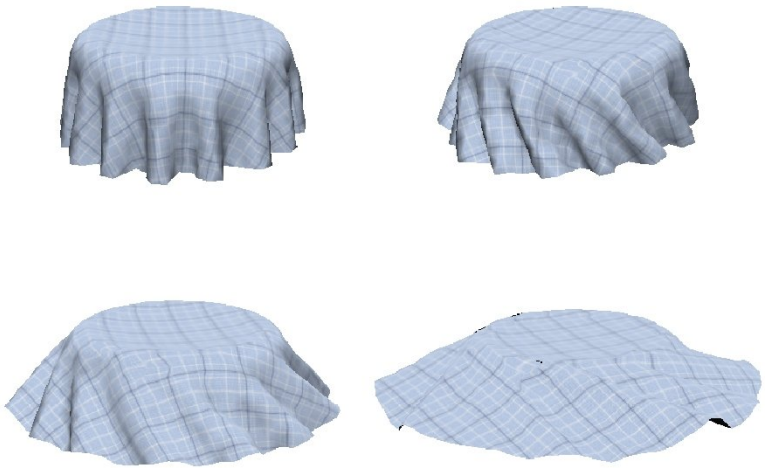


Abbildung 4.15: Vier Bilder aus einer Animation eines Tischtuchs, das auf einem Torus liegt. Der Torus wird um seinen Mittelpunkt rotiert und die Beschleunigung aufgrund der Reibung auf das Tuch übertragen. Würden die Reibungskräfte nicht behandelt, bliebe das Tuch einfach liegen, wenn man den Torus dreht.

Das vorgestellte Verfahren erfüllt die eingangs gestellten Anforderungen an Masseträgheit, Unterscheidung zwischen Gleit- und Haftreibung und Aufhebung der Reibung, wenn sich Objekte voneinander entfernen. Somit steht zu erwarten, dass eine Umsetzung des Verfahrens realistisches Verhalten des simulierten Materials ergibt. Dazu mehr in Abschnitt 4.4.7.

Mehrere Distanzfelder

Die meisten virtuellen Umgebungen enthalten nicht nur ein einzelnes Objekt, sondern viele Objekte – einige davon sogar in Bewegung. Für dieses Problem kann eine zweistufige Kollisionserkennung verwendet werden. Zuerst wird mit Hilfe einer Raumunterteilung oder einer BVH herausgefunden, welche Objekte einen geringen Abstand zum deformierbaren Objekt haben. Danach wird der Distanzfeld Algorithmus auf die erkannten Objekte angewandt und eine genaue Kollisionserkennung durchgeführt.

Ein weiteres Problem stellen alle Objekte dar, die nicht vollständig starr sind und deshalb nicht mit einer affinen Transformation animiert werden können. Darunter fallen alle teilweise deformierbaren Körper wie menschliche Avatare oder kinematisch Ketten (Roboterarm). In diesen Fall kann ein Distanzfeld pro Starrkörper bzw. Glied der Kette verwendet werden. Während der Kollisionserkennung werden dann zunächst mit einer AABB alle betroffenen Starrkörper bestimmt und danach alle Distanzen d_i berechnet. Für alle weiteren Berechnungen wird nur noch das Minimum der d_i berücksichtigt, da alle anderen Objekte weiter entfernt sind.

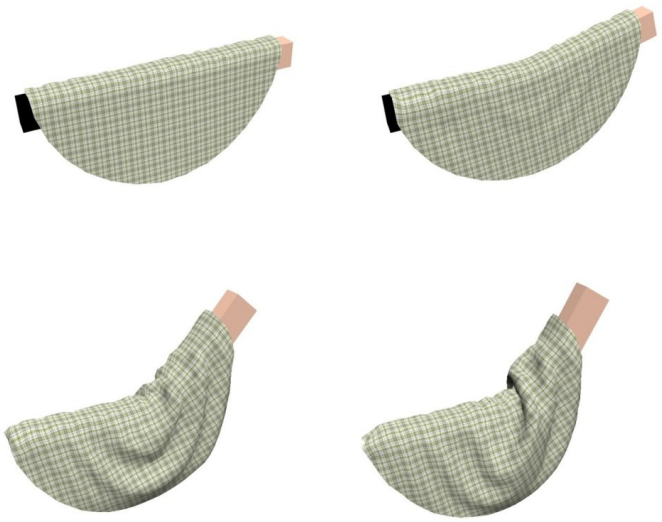


Abbildung 4.16: Vier Bilder aus einer Animation eines Tischtuchs, das über einem Stab liegt, der in der Mitte gebogen wird.

In Abbildung 4.16 sind mehrere Bilder aus einer Animation eines Tischtuchs dargestellt. Der Stab besteht aus zwei Gliedern, wovon das rechte bewegt wird. Mit diesem Verfahren können allerdings keine weichen Übergänge zwischen den Gliedern berechnet werden, wie sie für die Animation von Menschen benötigt würden.

4.4.6 Erweiterung auf animierte Körper

In diesem Abschnitt wird beschrieben, wie das Grundverfahren erweitert werden kann, so dass auch Kollisionen zwischen textilem Material und animierten Körpern korrekt erkannt werden können. Insbesondere die Kollisionserkennung zwischen Bekleidung und virtuellen Menschen wird hier beleuchtet.

Mit Hilfe des in Kapitel 4.2.3 beschriebenen kompakten SID Grids können problemlos mehrere Frames einer Animation eines Körpers im Speicher gehalten werden. Wie auch bisher wird angenommen, dass der Körper vom Stoff nicht deformiert wird. Zunächst wird für jeden Frame das zugehörige Distanzfeld berechnet. Wie im vorigen Kapitel beschrieben, ist allein die Speicherung der Distanzen nicht ausreichend, um das Verhalten eines sich bewegenden Objektes repräsentieren zu können. Allerdings lässt sich jetzt die Bewegung des Körpers nicht mit einer affinen Transformation beschreiben.

Daher wird zusätzlich zum Distanzfeld D ein Geschwindigkeitsfeld $\mathbf{V}$ erstellt. Analog zum Distanzfeld ist ein Geschwindigkeitsfeld $\mathbf{V}$ ein Vektorfeld mit $\mathbf{V}: \mathbb{R}^3 \rightarrow \mathbb{R}^3$, wobei $\mathbf{V}$ für jeden Punkt $\mathbf{p} \in \mathbb{R}^3$ eine Geschwindigkeit liefert.

Zur Bestimmung der Geschwindigkeiten werden diese zunächst für die Vertices des Körpers berechnet. Dabei wird angenommen, dass die Bewegung zwischen zwei Keyframes linear ist. Somit ergibt sich die Geschwindigkeit eines Vertex als skalierte Differenz zwischen seinen Koordinaten in den zwei Keyframes. Durch Interpolation in den Dreiecken erhält man alle Geschwindigkeiten auf der Oberfläche des Körpers. Um die Geschwindigkeiten in $\mathbf{V}$ zu berechnen, werden diese ausgehend von der Oberfläche extrapoliert. Mit Hilfe von $\mathbf{V}$ kann jetzt analog zum vorigen Abschnitt die Reibung berechnet werden.

Weiterhin kann das Geschwindigkeitsfeld für die Interpolation von Distanzen zwischen zwei Keyframes verwendet werden. Eine Interpolation ist nötig, da die Simulation meist in kleineren Zeitschritten abläuft als die Keyframes. Interpoliert man aber die Distanzen direkt, so ergeben sich bei größeren Abständen zwischen zwei Keyframes erhebliche Fehler bei der Berechnung, da die, durch die Distanzfelder definierten, impliziten Flächen sich beim Interpolieren zusammen ziehen. Dieser Effekt ist auch vom Morphing her bekannt [COSL98]. Angenommen man interpoliert Distanzen zwischen zwei Keyframes einer Kugel mit Durchmesser d, die um $2 \cdot d$ verschoben wird. Zum Zeitpunkt $t = 1/2$ liefert die Interpolation der beiden Distanzfelder für jeden $\mathbf{p} \in \mathbb{R}^3$ positive Werte. Dies ist sicherlich nicht das gewünschte Resultat. Durch die Verwendung von mehr Keyframes lässt sich dieser Effekt abmildern. Je weniger Keyframes aber benötigt werden, desto kompakter kann die Animation gespeichert werden und es müssen auch weniger Distanzfelder berechnet werden. Aufpassen muss man, wenn die Bewegung zwischen zwei Keyframes nicht linear ist, da dann ebenfalls Artefakte entstehen können.

Mit Hilfe des Geschwindigkeitsfeldes kann das oben beschriebene Problem gelöst werden. Durch die Verwendung von $\mathbf{V}$ können bei der Distanzabfrage Verschiebungen im Distanzfeld realisiert werden. Die Distanz für einen Punkt $\mathbf{p} \in \mathbb{R}^3$ zu einem Zeitpunkt t, der zwischen den Keyframes n und $n+1$ liegt, kann dann als

$$D(\mathbf{p}, t) = D_n(\mathbf{p} - (t_{n+1} - t)\mathbf{V}_n(\mathbf{p})) \tag{4.21}$$

bestimmt werden, wobei t_{n+1} das Ende des Keyframes $n+1$ markiert. Diese Methode ist durchaus ähnlich zu einem Verfahren, welches auf der Basis von Distanzfeldern Morphing von verschiedenen Objekten ermöglicht [COSL98].

Das Geschwindigkeitsfeld erlaubt somit nicht nur die korrekte Berechnung der Reibung, sondern es kann gleichzeitig dafür verwendet werden, zwischen Keyframes zu interpolieren.

4.4.7 Ergebnisse

Das Verfahren zur Kollisionserkennung wurde in einem System zur Animation textiler Materialien validiert. Es ermöglicht die Animation komplexer Stoffstücke, die

aus $2K$ Dreiecken bestehen, in Echtzeit. Während der Animation werden die Dynamik des Stoffs, Selbstkollisionen und Kollisionen zwischen Stoff und anderen Objekten berücksichtigt. Der Zeitschritt beträgt $0,01s$. Abbildung 4.17 zeigt mehrere Bilder aus einer Animation eines Tischtuchs, das über einen Kopf gezogen wird. Der Kopf besteht aus $293K$ Dreiecken und das Distanzfeld wird in einem kartesischen Gitter mit einer Auflösung von $168 \times 272 \times 199$ Punkten repräsentiert.

Abbildung 4.17: Mehrere Bilder aus einer Sequenz, in der ein Tischtuch interaktiv über einen 3D-Scan der Büste von Joseph von Fraunhofer gezogen wird.

Setzt man den Reibungskoeffizienten auf Null, beginnt das Tuch langsam über den Kopf zu rutschen. Dabei treten keinerlei springende Partikel oder Jitter-Effekte auf. Da die Distanzfunktion stetig ist, können die Partikel glatt herunter gleiten. Die Unstetigkeiten des Gradienten zwischen den Gitterzellen können nicht wahrgenommen werden.

In Tabelle 4.18 erkennt man, dass nur die Hälfte eines Zeitschritts für die Kollisionsbehandlung verwendet wird. Die restliche Zeit wird für die Lösung der Differentialgleichungen und die Selbstkollisionserkennung benötigt. Die Tabelle zeigt außerdem, dass die Berechnung der Normalen nicht zeitkritisch ist. Auch das verbesserte Reibungsmodell verursacht nur einen geringen Overhead.

Eine weitere Animation eines Tischtuchs ist in Abbildung 4.19 zu sehen. Der Anwender bewegt das Tischtuch mit der Maus und er bekommt den Eindruck ein reales Stück Stoff festzuhalten. Die 3D Interaktion wird durchgeführt durch Abbilden der Maus-Bewegung auf die *xy*-Ebene und durch die Verwendung des Mausrades zum Ändern der *z* Werte. Das Dreiecksnetz des Tuchs besteht aus $2,6K$ Dreiecken. Da das originale Buddha-Model zu komplex zum Rendern ist [3], wurde das Mesh durch eine Marching Cubes Rekonstruktion ersetzt, die $390K$ Dreiecke besitzt. Da der Algorithmus zur Kollisionserkennung unabhängig von der Anzahl der Dreiecke ist, kann immer noch eine Simulation in Echtzeit durchgeführt werden.

Die Implementierung wurde in Java durchgeführt und Java3D [Sun03] wurde zur Visualisierung benutzt. Das System lief auf einen Dual Intel Xeon^{TM} Prozessor mit $2,0GHz$.

[3] Es wurde Java3D zum Rendern verwendet.

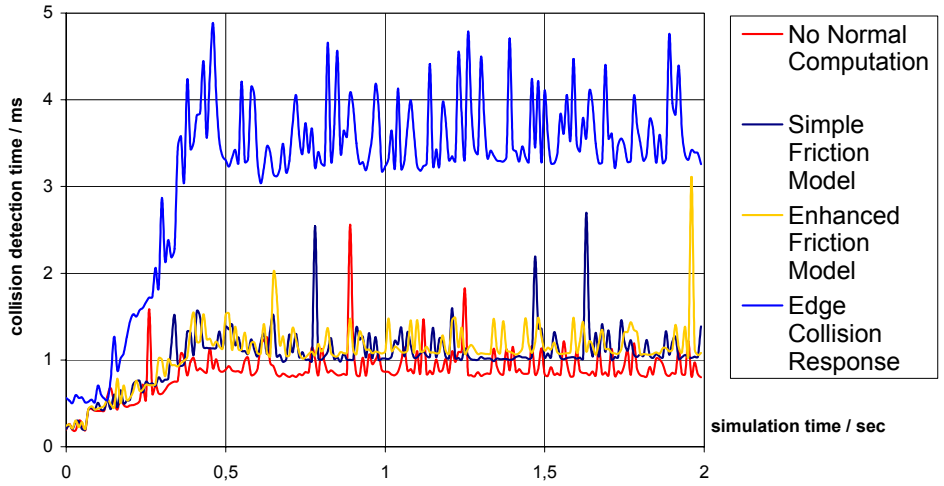


Abbildung 4.18: Vergleich der Zeiten zur Kollisionsbehandlung während einer Simulation von zwei Sekunden Dauer (Tischtuch über Kopf, siehe Abbildung 4.17). Wenn die Kanten getestet werden, steigt die Zeit wie vermutet um den Faktor vier. Lässt man die Berechnung der Normalen für die Kollisionsantwort weg, gewinnt man nicht viel Zeit.

Beispiele mit animierten Starrkörpern

In Abbildung 4.20 sind mehrere Bilder aus einer Animation nebeneinander gestellt. Der Stoff wurde mit dem in dieser Arbeit vorgestellten Verfahren zur Kollisionsbehandlung mit animierten Starrkörpern simuliert. Die Ergebnisse zeigen, dass der Algorithmus ein realistisches Stoffverhalten berechnet. Man erkennt sogar anhand der Bewegung des Stoffes in welche Richtung die Frau sich dreht, obwohl nur wenige Einzelbilder der gesamten Animation abgebildet sind. Ein Versuch, der ohne korrekte Behandlung der Reibung mit Hilfe der Relativbewegung arbeitet, hat gezeigt, dass dann das Kleid unnatürlich verrutscht, da die Bewegung der Frau nicht auf das Kleid übertragen wird.

Ein weiteres Beispiel, ein rotierender Torus, wurde bereits in Abbildung 4.15 gezeigt. Ohne die korrekte Behandlung der Reibung, würde sich das Tuch überhaupt nicht bewegen, da das zugrunde liegende Distanzfeld invariant gegenüber dieser Rotation ist.

In der Implementierung des Verfahrens lassen sich die Reibungskoeffizienten konfigurieren, so dass unterschiedliche Materialien simuliert werden können. In den gezeigten Beispielen beträgt der Gleitreibungskoeffizient $\mu = 0,5$ und der Haftreibungskoeffizient $\mu_0 = 1,0$.

Durch die Kollisionsbehandlung anhand der Relativbewegung wird die Bewegung des Starrkörpers sofort auf den Stoff übertragen. Dies hat positiven Einfluss auf die Robustheit der gesamten Kollisionserkennung, da dadurch Durchdringungen im nächsten Zeitschritt vermieden werden. Ohne die korrekte Behandlung

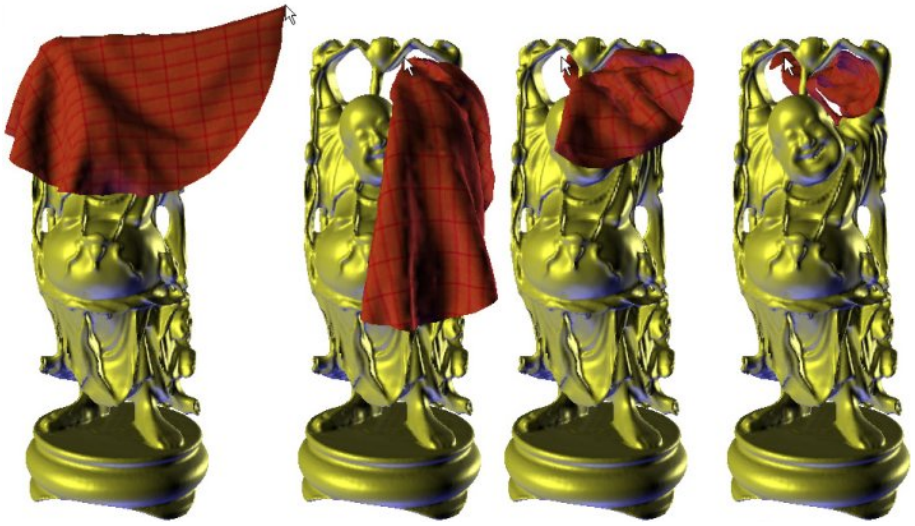


Abbildung 4.19: Mehrere Bilder aus einer Sequenz, in der ein Tischtuch über die Happy Buddha Figur gezogen wird.

ergeben sich schwerwiegende Simulationsfehler, da der Stoff der Bewegung des Starrkörpers nicht schnell genug folgen kann.

Der zusätzliche Aufwand für die Kollisionsbehandlung mit animierten Starrkörpern im Vergleich zu statischen Starrkörpern ist nicht messbar. Dies ist nicht weiter verwunderlich, da nur eine Transformation und einige Skalarprodukte zusätzlich ausgewertet werden müssen.

Die Kollisionsbehandlung mit dem virtuellen Menschen aus dem Beispiel von Abbildung 4.20 dauert $4,9\,ms$ pro Zeitschritt. Das Kleid besteht aus $4\,K$ Dreiecken. Insgesamt lässt sich mit dem Simulationssystem, das in dieser Arbeit vorgestellt wird, in diesem Beispiel eine Animation erzeugen, die mit 90Hz abläuft und somit hochgradig interaktiv ist. Die Zeitmessung wurde auf einem PC mit einer Intel Pentium-4 CPU mit $3,6\,GHz$ durchgeführt.

Beispiele mit animierten Körpern

Für das in Abschnitt 4.4.6 vorgestellte Verfahren zur Kollisionserkennung zwischen animierten Objekten und Bekleidung wurden einige Animationen eines Avatars mit Poser 6 [e f06] erstellt. Aus den Keyframes wurden dann die Distanz- und Geschwindigkeitsfelder berechnet. Die gezeigten Beispiele demonstrieren typische Bewegungen, die bei der virtuellen Anprobe von Kleidung helfen, die Passform zu beurteilen. Der Avatar befindet sich zunächst in einer Haltung mit gestreckten Armen, die eine optimale Vorpositionierung der Schnittteile erlaubt. Nach dem Vernähen können die Arme bewegt und neue Haltungen eingenommen

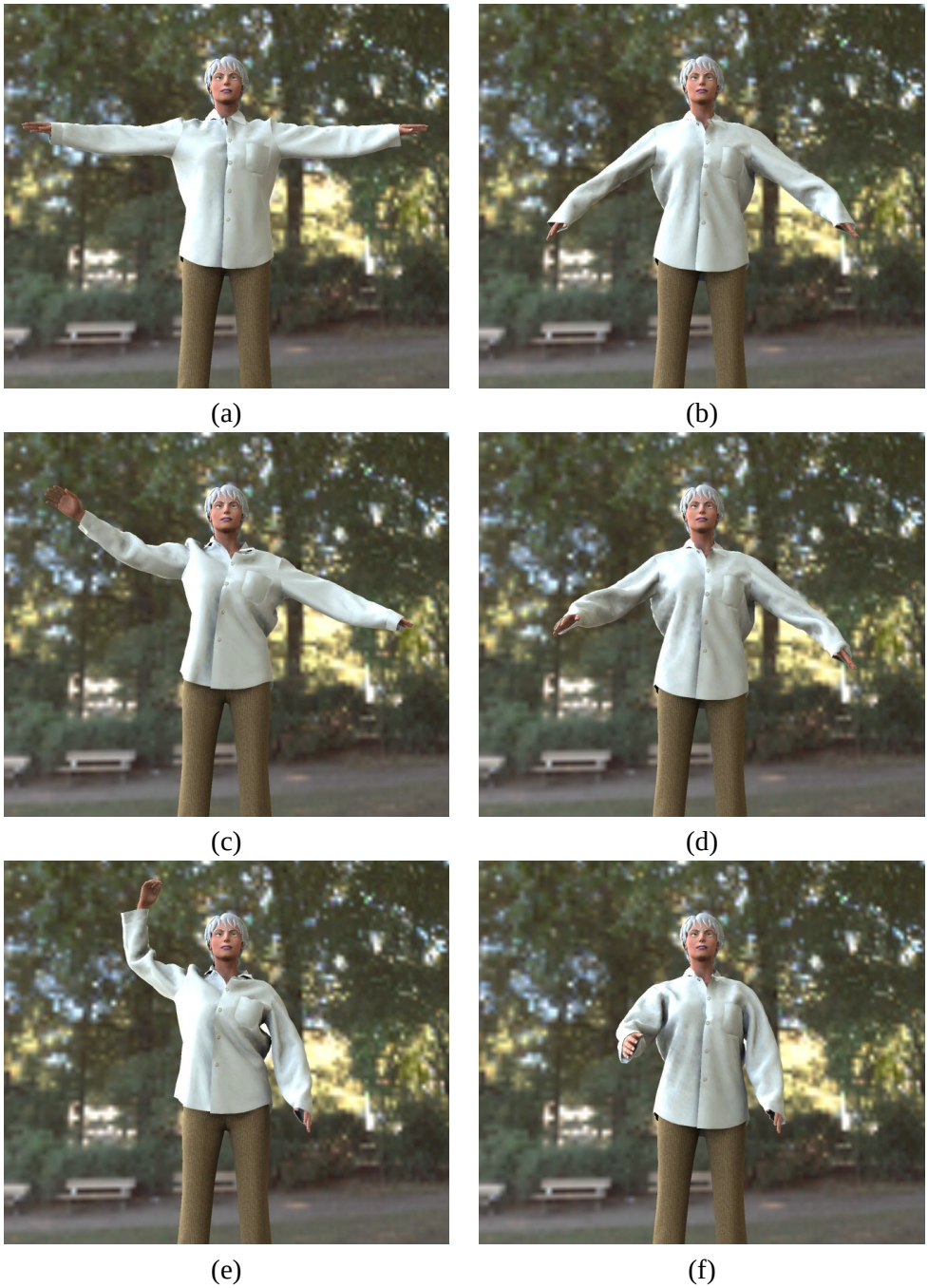
Abbildung 4.20: Mehrere Bilder aus einer Animation einer Frau, die sich um ihre eigene Achse dreht und ein Kleid mit $4K$ Dreiecken trägt. Anhand der Bewegung des Stoffes kann man die Drehrichtung erkennen. Aufgrund der Reibung zieht der Oberkörper das Kleid mit sich und es verrutscht nicht.

werden (siehe Abbildung 4.21). Bei den veränderten Haltungen der Arme kann bei der virtuellen Anprobe überprüft werden, ob keine Spannungen im Material auftreten und das Kleidungsstück auch optisch gut passt.

Mit Hilfe der Geschwindigkeitsfelder konnte die Anzahl der benötigten Keyframes in den Beispielen drastisch gesenkt werden. Ohne die $\mathbf{V}_n$ gab es beim Senken der Arme an einigen Stellen Artefakte, obwohl bereits 50 Keyframes verwendet wurden. Mit den $\mathbf{V}_n$ reichen 5 Keyframes aus. Allerdings sinkt die Performance des Verfahrens leicht, da neben der Distanz nun zusätzlich eine Geschwindigkeit aus einem Gitter mit Hilfe tri-linearer Interpolation rekonstruiert werden muss.

Für ein Geschwindigkeitsfeld ist wesentlich weniger Speicherplatz nötig als für ein Distanzfeld, da bei den hier gezeigten Fällen, bewegten virtuellen Menschen, die Geschwindigkeiten lokal sehr homogen sind. Daher wird nur eine geringe Auflösung des Gitters zur Speicherung des Geschwindigkeitsfeldes benötigt und der zusätzliche Speicheraufwand ist gering.

In der Haltung des Avatars am Anfang dauert die Kollisionserkennung $10,5\,ms$. Während der Animation benötigt diese in einem Zeitschritt dann $16\,ms$. Die Kleidung in dieser Szene besteht aus $13K$ Dreiecken.



(a) (b)

(c) (d)

(e) (f)

Abbildung 4.21: Mehrere Bilder, die zwei Animationen entnommen wurden ((a),(c),(e) und (a),(b),(d),(f)). Die Kleidung besteht aus $13K$ Dreiecken.

4.4.8 Bewertung

Der vorgestellte Algorithmus erlaubt die schnelle Kollisionserkennung zwischen starren und deformierbaren Objekten. In einer Erweiterung können auch Kollisionen zwischen animierten und deformierbaren Objekten effizient erkannt werden. Das Verfahren eignet sich insbesondere für die physikalisch basierte Simulation von Stoff, wie in den Beispielen gezeigt wurde. Es steht zu erwarten, dass der Algorithmus auch in anderen Gebieten wie der Haarsimulation angewandt werden kann. Da das Verfahren äußerst effizient ist, können damit immersive Anwendungen realisiert werden, in denen Stoff in Echtzeit animiert wird. Eine mögliche Erweiterung des Verfahrens bestünde in der Kollisionserkennung zwischen deformierbaren Avataren und deren Bekleidung, wobei der Stoff dann das Weichgewebe beeinflussen könnte.

Die Methode ist extrem robust, da durch die Vorzeichen des Distanzfeldes klar zwischen dem Inneren und dem Äußeren eines Objektes unterschieden werden kann. Sollte ein Partikel, aus welchem Grund auch immer, einmal innerhalb des Objektes sein, so wird er im nächsten Schritt der Kollisionsbehandlung zurück zur Oberfläche gebracht. Ein solcher Fall tritt häufig auf, wenn z.B. die initiale Position falsch ist oder sich das Objekt ruckartig bewegt. Diese Methode versagt nur dann, wenn der Partikel tiefer eindringt als die vorberechnete Distanzhülle oder der Partikel in einem Zeitschritt das Objekt durchstößt und auch wieder verlässt.

Das vorgestellte Modell zur Berechnung von Reibungskräften liefert gute Ergebnisse und ein realistisches Stoffverhalten. Eine Möglichkeit zur Erweiterung wäre die Behandlung von anisotroper Reibung. Beim bisherigen isotropen Reibungsmodell spielt es keine Rolle, in welcher Richtung zwei Objekte übereinander gleiten. Da aber die Webstruktur von Stoff vermuten lässt, dass sich textile Materialien häufig auch bei der Reibung anisotrop verhalten, wären eine weitere Untersuchung dieses Sachverhaltes und eine Erweiterung des Simulationsmodells sicherlich sehr interessant.

Als zukünftige Erweiterung könnte der Algorithmus als Basis für eine Kollisionserkennung-Hardware dienen. Da das Verfahren einfach zu implementieren ist und nur wenige bedingte Sprünge beinhaltet, eignet es sich gut für eine Hardware-Implementierung. Weiterhin werden keine Hierarchien benötigt, für die wiederum bedingte Sprünge nötig wären um sie zu traversieren. Zudem dauert jede Ausführung einer Abstandsberechnung ungefähr gleich lange. Dies ist sehr wichtig, wenn ein Pipelining-Schema realisiert werden soll.

4.5 Vermeidung von Selbstdurchdringungen

In diesem Kapitel wird ein neues Verfahren zur effizienten Selbstkollisionserkennung textiler Materialien beschrieben. Die Methode basiert auf einer hierarchischen Datenstruktur, die während der Simulation sowohl schnell upgedated als auch effizient abgefragt werden kann. Das Verfahren wurde mit dem Ziel entworfen, eine schnelle und stabile Methode zur Selbstkollisionserkennung zu erhalten.

Insbesondere die Stabilität ist für ein System, in das der Anwender interaktiv eingreifen können soll, sehr wichtig.

Ein Verfahren mit einer hohen Robustheit wurde von Bridson et al. [BFA02] vorgeschlagen. Die Robustheit impliziert auch eine hohe Stabilität, allerdings auf Kosten der Performanz. Das Verfahren modelliert Stoff mit unterschiedlichen Dicken und die gezeigten Animationen geben einen Eindruck von den Möglichkeiten, die ein robuster Algorithmus bietet, nämlich komplexe Animationen mit feinem Faltenwurf. Der Nachteil des Verfahrens ist seine hohe Rechenzeit, womit es für interaktive Anwendungen weniger in Frage kommt. Es existieren einige Techniken zur Geschwindigkeitssteigerung, die auf der Analyse der Krümmung der Stoffoberfläche basieren [VMT00b, Pro97]. Obwohl man mit diesen Verfahren flache Bereiche innerhalb eines Stück Stoffs schnell als nicht kollidierend erkennen kann, ist die Berechnung der Krümmung vergleichsweise aufwändig und in einem interaktiven System meist zu langsam.

Im Allgemeinen hat man zwei Möglichkeiten mit Selbstkollisionen umzugehen:

1. Auflösen, nachdem sie aufgetreten sind.

2. Niemals eine Selbstdurchdringung zulassen.

Das Verfahren von [BFA02] verfolgt die zweite Variante. Ein anderes Verfahren [BWK03] ist in der Lage bereits entstandene Durchdringungen aufzulösen.

Der hier vorgestellte Ansatz gehört in die zweite Kategorie. Aber anstatt zu garantieren, dass keinerlei Selbstdurchdringungen auftreten, wird nur verhindert, dass sie passieren. Hierzu genügt es Partikel gegeneinander zu testen, anstatt Dreiecke miteinander zu schneiden. Dies ist wesentlich performanter. Aber die Partikel müssen auf einem gewissen Abstand gehalten werden, um eine Selbstdurchdringung zu vermeiden und Artefakte von sich schneidenden Dreiecken zu unterbinden. Da für die Simulation hoch aufgelöste Meshes verwendet werden und die Distanz von der Größe der Dreiecke abhängt, ist dieser Ansatz gangbar.

Da diese Methode zunächst einmal sehr viel Rechenzeit spart, ist sie besonders gut für eine Echtzeitanwendung geeignet. Gleichzeitig toleriert sie vorhandene und entstehende Durchdringungen, was sie stabil macht. Weiterhin können sich Durchdringungen auch wieder auflösen und es entstehen keine Artefakte mit „eingefrorenen“ Dreiecken.

Das neue Verfahren zur effizienten Vermeidung von Selbstdurchdringungen wird im Folgenden in drei Schritten erklärt. Am Anfang stehen der Aufbau der Hierarchie und die Udpates der Hierarchie nach jedem Zeitschritt. Abschließend wird beschrieben, wie das Verfahren beschleunigt werden kann, indem die Eigenschaften der zu behandelnden textilen Materialien ausgenutzt werden.

4.5.1 Aufbau der Hierarchie

Am Anfang steht der Aufbau einer geeigneten hierarchischen Datenstruktur, um effizient herauszufinden welche Partikel sich nahe sind. Ohne eine Datenstruktur zur Beschleunigung wäre die Anzahl der Tests in $O(n^2)$, da jeder Partikel gegen jeden anderen getestet werden müsste. Eine gängige Methode ist es eine Bounding Volume Hierarchie (BVH) über den Dreiecken des zu simulierenden Meshes aufzubauen. Bei textilen Materialien stehen die einzelnen Schnittteile meist in der Ebene zur Verfügung. Da dieser Zustand dem Ausgangszustand ohne jegliche Deformation entspricht, liegt es nahe die Hierarchie auf diesen flachen Schnittteilen aufzubauen. Des Weiteren wird eine eigene Hierarchie für jedes Schnittteil aufgebaut. Diese Hierarchien können dann noch zusammengefasst werden. Diese Trennung wird später beim Testen auf Kollision benötigt.

Diese BVH kann dann genutzt werden um schnell überlappende oder sich nahe Dreiecke zu finden. In dem hier vorgestellten Verfahren genügt es, die Hierarchie über den Partikeln bzw. den Eckpunkten der Dreiecke aufzubauen. Dies hat den Vorteil, dass sich die Anzahl der Knoten in der Hierarchie halbiert, da nach der Euler-Formel immer ungefähr doppelt so viele Dreiecke wie Vertices in einem zusammenhängendem Mesh vorhanden sind.

Durch einen binären Baum kann die Selbstkollisionserkennung durch die Bestimmung sich naher Partikel erheblich beschleunigt werden. Für jeden Knoten der Hierarchie wird ein Bounding Volume geführt und die Kollisionserkennung reduziert sich auf Überlappungstests zwischen Bounding Volumes und die Traversierung der Hierarchie. Gewöhnlich überlappen sich die meisten Bounding Volumes nicht und die Anzahl der zu überprüfenden Partikel lässt sich stark verringern. Die Traversierung bei der Selbstkollisionserkennung erfolgt ähnlich zur Traversierung unterschiedlicher Objekte. Anstatt aber zwei verschiedene Hierarchien gegeneinander zu testen, wird die gleiche Hierarchie gegen sich selbst getestet.

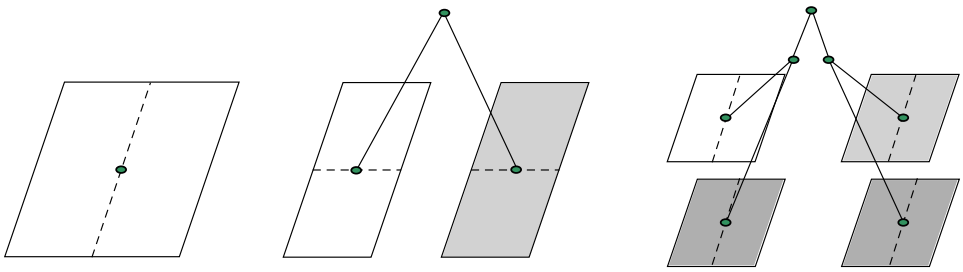


Abbildung 4.22: Schema der Konstruktion der Bounding-Box-Hierarchie zur Selbstkollisionserkennung.

In Abbildung 4.22 wird der Aufbau der Hierarchie durch rekursive Achsenteilungen in der Ebene dargestellt. Die Wurzel enthält alle Partikel. Diese werden

aufgeteilt und den Nachfolgeknoten zugewiesen. Die Partikel dieser Knoten werden wiederum aufgeteilt, allerdings wird in jedem Schritt an einer anderen Achse unterteilt. Die Achsen werden gewechselt, damit möglichst kompakte Nachfolgeknoten entstehen. Diese Vorgehensweise lässt sich noch optimieren, indem man untersucht bei welcher Wahl der Achse bessere Nachfolgeknoten entstehen. Genauere Erläuterungen hierzu findet man in [Zac02]. Der Punkt, an dem unterteilt wird, muss ebenfalls berechnet werden. Eine einfache Heuristik ist die Unterteilung in der Mitte des Bounding Volumes des aktuellen Knotens.

4.5.2 Hierarchie-Update

Textiles Material deformiert sich sehr stark während einer Simulation. Meist ändern sich die Positionen aller Partikel. Damit die Kollisionserkennung weiterhin funktioniert, muss die Hierarchie diesen Bewegungen Rechnung tragen. Man könnte die Hierarchie neu aufbauen, damit die Kollisionserkennung so rasch wie möglich arbeitet. Dieser Neuaufbau ist allerdings relativ aufwändig und wesentlicher komplexer als der Rest der eigentlichen Kollisionserkennung. Da die Hierarchie immer nur für einen Simulationsschritt gültig ist und die Kosten für den Neuaufbau in keinem Verhältnis zur gesparten Zeit bei der Kollisionserkennung stehen, ist diese Methode nicht effizient.

Aber da sich die Topologie der Schnittteile im Gegensatz zu flüssigen Materialien nicht ändert, kann die Struktur der Hierarchie beibehalten werden. Nach jedem Zeitschritt werden nur die Bounding Boxen der Knoten upgedatet. In [vdB97] wird gezeigt, dass ein Anpassen der Bounding Boxen zehn mal schneller als ein Neuaufbau ist.

Allerdings sind angepasste Bounding Volumes nicht mehr so eng anliegend wie solche, die direkt aufgebaut worden sind. An dieser Stelle wird auch ersichtlich, warum es sehr sinnvoll ist die Hierarchie auf den flachen Schnittteilen aufzubauen. Würde man die Hierarchie auf bereits deformierten Schnittteilen aufbauen, würde die Hierarchie für diesen Zustand besonders gut funktionieren. Eine Hierarchie, die auf den flachen Schnittteilen aufgebaut wird, ist wesentlich flexibler und bietet gute Performance für alle zukünftigen Deformationen.

Andererseits könnte im Fall von Bekleidung eine Hierarchie auf bereits deformierten Schnittteilen aber sogar von Vorteil sein, da sich das textile Material bei getragener Bekleidung meist nur wenig deformieren kann. Dieser Aspekt wurde in dieser Arbeit noch nicht untersucht und bietet Potential für Weiterentwicklung.

Das Update erfolgt Bottom-Up von den Blättern zur Wurzel. Zunächst werden die Bounding Volumes der Partikel neu berechnet. Danach werden jeweils zwei Bounding Volumes zusammengefasst. Besonders an dieser Stelle ist es vorteilhaft, dass nur halb so viele Knoten in der Hierarchie enthalten sind, wie bei einer Methode, die ganze Dreiecke im Baum speichert. Dadurch beschleunigt sich das Update der Hierarchie, das ja in jedem Zeitschritt durchgeführt werden muss.

Es sind eine ganze Reihe unterschiedlicher Bounding Volumes vorgeschlagen worden (siehe [TKH+05] für eine Übersicht). Für eine effiziente Selbstkollisionserkennung deformierbarer Objekte eignen sich nur Bounding Volumes, die schnell upgedated werden können und gleichzeitig das Volumen nicht zu sehr überschätzen. Diese Eigenschaft besitzen *k*-DOPs, da nur wenige Parameter angepasst werden müssen und die Kombination zweier *k*-DOPs ebenfalls äußerst einfach ist.

4.5.3 Kollisionstest

Problematisch bei der Selbstkollisionserkennung ist die Tatsache, dass benachbarte Dreiecke in einem Mesh von der Hierarchie ständig als kollidierend erkannt werden, da sich ihre Bounding Volumes meist überlappen. Daraus folgt ein enormer Overhead beim Erkennen der Kollisionen. Andere Verfahren nutzen die Tatsache aus, dass in Bereichen geringer Krümmung keine Selbstkollisionen entstehen können [VMT00b, Pro97, WB05]. Allerdings benötigt man hierfür Informationen über die Oberflächennormalen und der Rechenaufwand beim Update steigt. Deshalb wird hier ein Algorithmus vorgestellt, der schnell benachbarte Partikel erkennen kann, ohne Informationen über die Normalen zu verwenden. Die erkannten Partikel werden als nicht kollidierend angenommen. Diese Annahme kann aufgrund der Biegesteifigkeit von textilem Material getroffen werden.

Ob zwei Partikel benachbart sind, wird nur für einzelne Schnittteile geprüft. Bei unterschiedlichen Schnittteilen sind höchstens Partikel am Rand benachbart und werden an dieser Stelle vernachlässigt. Für zwei Partikel eines Schnittteils lässt sich schnell herausfinden, ob diese benachbart sind, indem man ihren geodätischen Abstand vergleicht. Für gekrümmte Flächen ist die Berechnung des geodätischen Abstandes für gewöhnlich sehr aufwändig. Hier kann dieser jedoch einfach bestimmt werden, da die Schnittteile in der Ebene vorliegen und der geodätische Abstand somit über den parametrischen Abstand gegeben ist.

Zusätzlich kann mit einer weiteren Heuristik sogar herausgefunden werden, ob zwei Bounding-Volumes benachbart sind. Man bestimmt beim Aufbau der Hierarchie den Mittelpunkt des Bounding-Volumes in der Ebene des Schnittteils. Später beim Traversieren der Hierarchie kann der Abstand zweier Mittelpunkte verwendet werden, um zwei Bounding-Volumes als höchstwahrscheinlich benachbart zu klassifizieren. In diesem Fall können sie von weiteren Tests ausgeschlossen werden.

Für den Kollisionstest wird geprüft, ob zwei Partikel sich näher sind als ein vorgegebener Schwellwert δ. Dies kann nur dann der Fall sein, wenn die zugehörigen *k*-DOPs ebenfalls einen geringen Abstand besitzen. Da die *k*-DOPs die Partikel eng umschließen, kann mit einem üblichen Überlappungstest für *k*-DOPs, wie z.B. in [AMH02] beschrieben, nicht der Abstand festgestellt werden. Um nicht die *k*-DOPs selbst vergrößern zu müssen, kann dieser Überlappungstest jedoch so angepasst werden, dass schnell ausgeschlossen werden kann, dass sich zwei *k*-DOPs näher sind als δ sind. In Algorithmus 3 ist dies für AABBs dargestellt. Eine Erweiterung für *k*-DOPs ist leicht möglich.

Algorithmus 3 : PROXIMITYTEST FÜR AABBS

Eingabe : BoundingBox a, BoundingBox b, Entfernungsschwellwert δ
Ausgabe : {benachbart|disjunkt}

foreach $i \in x,y,z$ **do**
  if $a_i^{max} < b_i^{min} - \delta$ *or* $a_i^{min} < b_i^{max} + \delta$ **then**
    return disjunkt
  end
end
return benachbart

Beim Traversieren der Hierarchie werden mit Algorithmus 3 k-DOPs, die weit genug entfernt voneinander sind, aus weiteren Test ausgeschlossen. Dies wird solange vollzogen, bis man auf zwei Blattknoten trifft. Für diese wird dann exakt geprüft, ob sie näher als δ sind und gegebenenfalls eine Kollisionsantwort generiert.

4.5.4 Integration in das Simulationssystem

Nachdem zwei Partikel als zu nahe aneinander erkannt wurden, muss noch gewährleistet werden, dass diese sich nicht weiter annähern bzw. dass der Abstand wieder vergrößert wird. Das Ziel ist es eine Selbstdurchdringung des Stoffes zu vermeiden. Hierzu kann eine steife gedämpfte Feder eingefügt werden [BW98] (siehe Abbildung 4.23). Alternativ kann auch eine abstoßende Kraft eingefügt werden [CK02]. Beide Methoden verhindern eine weitere Annäherung der Partikel. Durch Einstellen der Ruhelänge kann auch der Minimalabstand wiederhergestellt werden. Danach wird die Feder wieder eliminiert. Allerdings gefährden beide Varianten die Stabilität des Systems, da sich durch die neuen Kräfte die Steifigkeit des Partikelsystems erhöht. Zudem benötigt das Simulationssystem oft mehrere Zeitschritte bis der gewünschte Zustand erreicht ist.

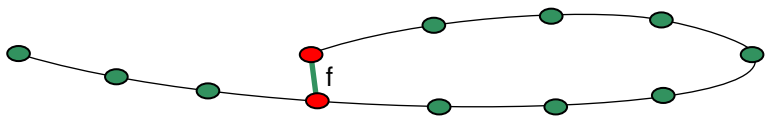


Abbildung 4.23: Vermeidung der Selbstkollision durch Einfügen einer steifen Feder.

Aus diesen Gründen verwendet dieses Verfahren eine Methode, die zum einen den Abstand in einem Schritt korrigiert und gleichzeitig durch eine Art Kraftstoß Impulse zwischen den Partikeln austauscht. Die Steifigkeit des Gesamtsystems wird dadurch kaum verändert. Es wird der gemeinsame Schwerpunkt der Partikel bestimmt und die Partikel entsprechend verschoben, so dass der gewünschte

Abstand erreicht wird. Dieses Verfahren ist ähnlich der Korrektur von zu langen Federn am Ende eines Zeitschritts (siehe auch Kapitel 5.3.1). Seien $\mathbf{x}_i, \mathbf{x}_j$ die Positionen der Partikel. Zunächst wird die Richtung und Länge des Korrekturvektors bestimmt

$$\mathbf{u} = \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|}(\|\mathbf{x}_j - \mathbf{x}_i\| - \delta) \tag{4.22}$$

wobei δ die minimale Entfernung bezeichnet. Die Verschiebung muss gewichtet werden, damit ein schwerer Partikel weniger bewegt wird als ein leichter. Daher wird der Korrekturvektor noch gewichtet

$$m_1 = \frac{m_i}{m_i + m_j} \tag{4.23}$$

$$m_2 = \frac{m_j}{m_i + m_j} \tag{4.24}$$

und folgendermaßen angewandt

$$\mathbf{x}_i \mathrel{+}= m_2 \cdot \mathbf{u} \tag{4.25}$$

$$\mathbf{x}_j \mathrel{-}= m_1 \cdot \mathbf{u} \tag{4.26}$$

Diese einfache Methode benötigt nur wenige arithmetische Operationen und überprüft gleichzeitig den häufigen Fall, dass zwei Partikel sich überhaupt nicht zu nahe sind. Da die Tests mit den Bounding Volumes konservativ sind, tritt dieser Fall sehr häufig ein.

4.5.5 Ergebnisse

Eine Implementierung der beschriebenen Methode zur Vermeidung von Selbstkollisionen hat gezeigt, dass das Verfahren effizient und stabil arbeitet. Das Ergebnis der Simulation eines Godetrockes ist in Abbildung 4.24 dargestellt. Besonders im unteren Bereich des Rocks wurden zahlreiche Selbstdurchdringungen erfolgreich verhindert. Die Simulation erfolgte nicht offline, sondern wurde interaktiv durchgeführt. Weitere Versuche mit verschiedenen Kleidungsstücken haben gezeigt, dass der Algorithmus für einlagige Bekleidung zuverlässig arbeitet.

Der oben beschriebene Algorithmus arbeitet sehr effizient und kann deshalb gut für interaktive VR-Applikationen eingesetzt werden. Insbesondere lassen sich einzelne Stoffstücke sogar in Echtzeit animieren. Die Qualität einer Simulation, die diesen Algorithmus verwendet, ist dabei wesentlich höher, als in bisherigen System, die aus Effizienzgründen gar keine Selbstkollisionserkennung durchführen. Obwohl nicht alle Durchdringungen verhindert werden, arbeitet der Algorithmus sehr stabil, da bei der Kollisionsantwort keine hohen Kräfte in das Partikelsystem eingeführt werden. Mehrfachkollisionen werden zwar nicht gesondert behandelt, führen aber auch nicht zu einer Instabilität. Die Anpassung der Abstände wird einfach mehrfach durchgeführt.

Abbildung 4.24: Ergebnis des Algorithmus zur Vermeidung von Selbstkollisionen bei einer Simulation mit komplexem Faltenwurf.

Dadurch, dass Impulse zwischen zwei Partikeln ausgetauscht werden, sind die entstehenden Animationen sehr realistisch (siehe Abbildung 4.25).

Der vorgestellte Algorithmus stößt an seine Grenzen, wenn mehrlagige Kleidung simuliert wird. Da er nur auf Basis der Partikel arbeitet, können Durchdringungen dicht aneinander liegender Lagen entstehen. Für solch schwierige Simulationen werden komplexere Methoden benötigt, die nicht nur die Partikel betrachten, sondern die einzelnen Flächen [Pro97, VMT00b, BFA02]. Allerdings sind diese Algorithmen für interaktive Anwendungen zu langsam. Zudem wurden von den Autoren keine Simulationen für mehrlagige Kleidung gezeigt.

Ein weiterer Nachteil ist die Tatsache, dass eine entstandene Durchdringung nicht mehr aufgelöst werden kann. Es kann sogar vorkommen, dass ein durchdrungener Partikel auf der falschen Seite festgehalten wird. Eine Lösung hierfür wurde

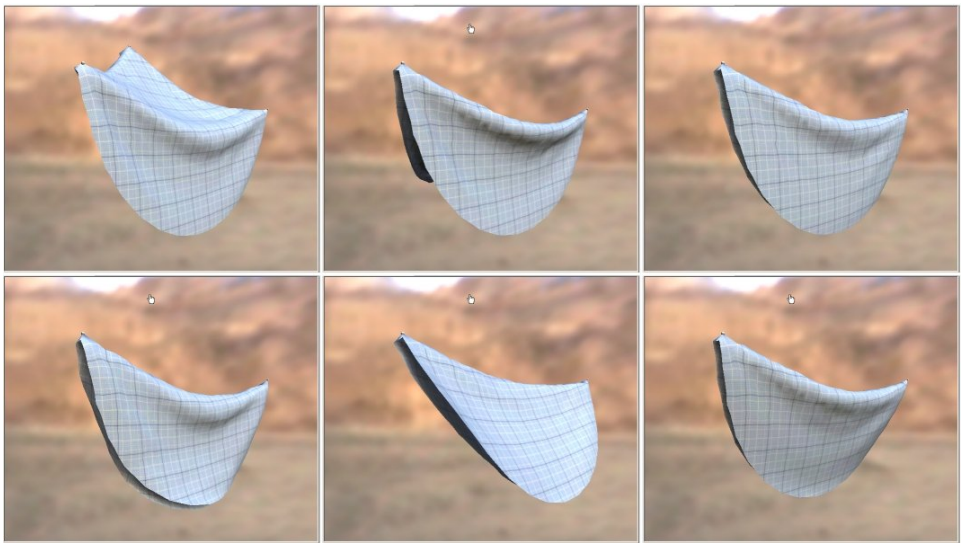


Abbildung 4.25: Ein Stofftuch wird zunächst an mehreren Punkten festgehalten und dann an einer Stelle losgelassen. Bei der Behandlung der Selbstkollisionen wird der Impuls auf den zuvor frei hängenden Teil des Tuches übertragen und er schwingt.

in [BWK03] vorgeschlagen. Das Verfahren ist allerdings zu aufwendig für eine interaktive Simulation.

Weiterhin kann sich der Schwellwert für den Abstand δ als zu groß erweisen. Da dieser von der Größe der simulierten Dreiecke abhängt, kann er nicht beliebig verkleinert werden. Manche Anwendungen erfordern jedoch sehr dicht anliegende Stoffschichten bei gleichzeitig grob aufgelösten Dreiecksnetzen.

4.6 Zusammenfassung

In diesem Kapitel werden verschiedene Aspekte der Kollisionserkennung und -behandlung für textile Materialien behandelt. Im ersten Teil wird auf die speziellen Probleme bei der Kollisionserkennung mit deformierbaren Objekten eingegangen. Danach werden einige bestehende Algorithmen zur Kollisionserkennung und verschiedene Ansätze zur Kollisionsbehandlung vorgestellt. Es zeigt sich, dass existierende Verfahren zwar das prinzipielle Problem der Kollisionserkennung zufriedenstellend lösen, aber die dafür benötigte Rechenzeit lässt bei komplexeren Geometrien, wie sie im Bereich der virtuellen Anprobe verwendet werden, keine interaktive Animation mehr zu. Daher wird in diesem Kapitel ein Verfahren zur Kollisionserkennung zwischen textilem Material und einem weitgehend starren Körper vorgestellt und ein weiteres Verfahren, mit dem Selbstkollisionen zwischen verschiedenen Teilen des textilen Materials vermieden werden können. Beide Al-

gorithmen sind äußerst effizient und erlauben zusammen genommen die interaktive Kollisionserkennung für textile Materialien.

Das Verfahren zur Kollisionserkennung zwischen textilem Material und anderen Körpern basiert auf *Distanzfeldern*, da diese direkt Informationen über Eindringtiefe und Normal liefern. Beides wird für eine realistische Kollisionsantwort im Rahmen einer Simulation benötigt. Daher wird zunächst auf Distanzfelder eingegangen. Es werden verschiedene Datenstrukturen zur Repräsentation von Distanzfeldern vorgestellt und miteinander verglichen. Dann wird eine neue Datenstruktur, das Sparse-Integer-Distance Grid, vorgestellt, die wenig Speicher benötigt und mit der Distanzen effizient abgefragt werden können. Die Berechnung eines Distanzfeldes aus einem gegebenen Dreiecksnetz ist äußerst rechenaufwendig, da ein Distanzfeld sehr komplexe Informationen über ein Objekt liefern kann. Deshalb wird auf deren effiziente Erzeugung in diesem Kapitel eingegangen.

Klassische Verfahren zur Kollisionserkennung zwischen starren Körpern verwenden Bounding Volume Hierarchien um kollidierende Primitive zweier Dreiecksnetze bestimmen zu können. Da in diesem Fall meist nur wenige Kontaktpunkte auftreten, lassen sich diese effizient bestimmen. Bei der Animation von textilem Material, welches sich leicht biegen lässt, passiert es aber häufig, dass der Stoff mit seiner ganzen Fläche im Kontakt mit anderen Objekten ist. In diesem Fall verschlechtert sich die Performanz der Kollisionserkennung mit einer Hierarchie enorm, da jetzt sehr viele Tests mit Primitiven durchgeführt werden müssen. Dieser Ansatz ist für interaktive Systeme nicht durchführbar. Insbesondere, wenn die Anwendung nicht nur Kollisionen behandeln muss, sondern auch noch Differentialgleichungen für die dynamische Bewegung der Objekte lösen muss.

Daher wird in diesem Kapitel ein neues Verfahren zur Kollisionserkennung vorgestellt, das die zuvor beschriebenen Distanzfelder verwendet, um die nötigen Test auf Nähe und Durchdringung durchzuführen. Jeder einzelne Test kann hierbei sehr effizient durchgeführt werden. Für die korrekte Berücksichtigung von Reibung und für animierte Körper liefern Distanzfelder jedoch nicht genügend Informationen. Daher werden zusätzlich noch *Geschwindigkeitsfelder* verwendet, um Informationen über die Bewegung der Objekte erhalten zu können. Eine Implementierung des vorgestellten Verfahrens zeigt, dass sich mit Distanz- und Geschwindigkeitsfeldern eine genaue und hoch effiziente Animation von Bekleidung realisieren lässt.

Im letzten Teil dieses Kapitels wird ein Verfahren zur effizienten *Vermeidung von Selbstdurchdringungen* beschrieben. Der Algorithmus nutzt die Tatsache aus, dass sich die Topologie von Stoff während der Simulation nicht ändert. Daher können geeignete hierarchische Strukturen aufgebaut werden, die während der Simulation sowohl schnell upgedated als auch effizient abgefragt werden können. Im Allgemeinen hat man zwei Möglichkeiten mit Selbstkollisionen umzugehen. Man kann sie entweder auflösen, nachdem sie aufgetreten sind, oder niemals eine Selbstdurchdringung zulassen. Der hier vorgestellte Ansatz gehört in die zweite Kategorie. Aber anstatt zu garantieren, dass keinerlei Selbstdurchdringungen auftreten, wird nur verhindert, dass sie passieren. Durch diese Approximation wird das Ver-

fahren sehr performant. Außerdem ist es äußerst stabil, da entstandene Durchdringungen toleriert werden und sich im weiteren Verlauf der Simulation auch wieder auflösen können. Die Stabilität ist insbesondere für eine interaktive Anwendung besonders wichtig.

Kapitel 5

Effiziente Materialsimulation

Im Bereich der Computergrafik haben sich Partikelsysteme zur Modellierung deformierbarer Objekte als sehr geeignet erwiesen, da sie einen guten Kompromiss zwischen Genauigkeit und Geschwindigkeit darstellen [VMT01]. Ein verformbarer Körper wird in einem Partikelsystem durch Massepunkte repräsentiert, die miteinander interagieren und Partikel genannt werden. Bewegung entsteht durch Wirkung von Kräften auf diese Partikel, den Gesetzen der Dynamik gehorchend. Daraus ergeben sich Differentialgleichungen, durch deren Lösen die Bahnen der einzelnen Partikel berechnet werden können.

Das mechanische Verhalten bei der Interaktion zwischen den Partikeln wird durch Kräfte beschrieben, die von den relativen Positionen der Partikel zueinander und der Stärke der Krümmung der Oberfläche des Körpers abhängen. Ein einfaches Modell für die Bestimmung der Kräfte ist ein Masse-Feder-System. Dabei ist die Kraft, die zwischen zwei Partikeln wirkt, gleich der Kraft einer Feder zwischen den Partikeln. Die Kraft der Feder kann proportional zum Abstand der Partikel sein oder aber auch mit komplizierten Formeln bestimmt werden. Vertiefende Informationen zu Partikelsystemen und eine gute Einführung in das Thema *physikalisch basierte Simulation* findet man in [WBK97].

5.1 Integration der Bewegungsgleichung

Die Wahl der Integrationsmethode ist entscheidend für die Performance und Stabilität einer numerischen Simulation. Deshalb werden an dieser Stelle einige Verfahren und deren Anwendung bei der Stoffsimulation vorgestellt und verglichen.

Bei der Simulation von textilen Materialien muss man beachten, dass bei wenig dehnbaren Stoffen so genannte steife Differentialgleichungssysteme gelöst werden müssen. Daraus folgt, dass bei der numerischen Integration in sehr kleinen Zeitschritten vorangegangen werden muss, um numerische Instabilitäten zu vermeiden. Erst in [BW98] wurde ein Modell vorgestellt, das es erlaubt, in großen Zeitschritten und damit effizient zu simulieren. Begnügt man sich mit einer approximierten Lösung, lässt sich die Simulation interaktiv in Echtzeit durchführen [MDDB01].

Bei der Simulation eines Partikelsystems entsteht Bewegung durch die Integration der Bewegungsgleichung

$$\ddot{\mathbf{x}} = \mathbf{M}^{-1}\mathbf{f}, \tag{5.1}$$

einer gewöhnlichen Differentialgleichung zweiter Ordnung, die durch Einführung der Geschwindigkeit auf ein System von Differentialgleichungen erster Ordnung reduziert werden kann:

$$\begin{pmatrix}\dot{\mathbf{x}}\\ \dot{\mathbf{v}}\end{pmatrix} = \begin{pmatrix}\mathbf{v}\\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{x},\mathbf{v})\end{pmatrix} \quad mit\ \mathbf{v} = \dot{\mathbf{x}} \tag{5.2}$$

Dieses System kann durch eine Vielzahl von numerischen Verfahren integriert werden, die mehr oder weniger für die Textilsimulation geeignet sind. In Frage kommen sowohl explizite (Euler, Mittelpunkt, Runge-Kutta 4. Ordnung) als auch implizite (Euler, Trapez, BDF-2) Integrationsverfahren.

Die neuen Positionen und Geschwindigkeiten der Partikel ergeben sich mit

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \Delta\mathbf{v} \tag{5.3}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{x}. \tag{5.4}$$

Je nach Integrationsverfahren werden die Änderungen von Position $\Delta\mathbf{x}$ und Geschwindigkeit $\Delta\mathbf{v}$ anders errechnet.

5.1.1 Explizite Verfahren

Explizite Verfahren sind einfach zu implementieren und schnell auszuwerten. Leider erlauben sie aber nur kleine Schrittweiten. Das explizite Eulerverfahren errechnet die Änderung der Geschwindigkeit $\Delta\mathbf{v}$ der Partikel und die Änderung der Positionen $\Delta\mathbf{x}$ nach folgenden Formeln:

Explizites Eulerverfahren:

$$\Delta\mathbf{v} = h\mathbf{M}^{-1}\mathbf{f}_n \tag{5.5}$$

$$\Delta\mathbf{x} = h\mathbf{v}_n \tag{5.6}$$

Dieses Integrationsverfahren liefert schnell die neuen Positionen der Partikel, da diese direkt, sprich *explizit*, berechnet werden können. Leider wird es aber auch sehr schnell instabil, wenn der Zeitschritt h zu groß gewählt wird. Besonders bei der Simulation von textilen Materialien, wo sehr hohe Kräfte entstehen, werden sehr kleine Zeitschritte benötigt, damit das System nicht explodiert.

Andere explizite Verfahren, wie z.B. das *Mittelpunktverfahren* oder *Runge-Kutta*, erlauben etwas größere Zeitschritte, wobei dies mit leicht erhöhtem Rechenaufwand erkauft wird. Das *Verlet Verfahren* [Ver67] ist wesentlich stabiler als das Eulerverfahren, kostet aber keine zusätzliche Performance. Auf dieses Verfahren wird in Abschnitt 5.3.2 näher eingegangen.

5.1.2 Implizite Verfahren

Implizite Verfahren liefern stabile Lösungen bei steifen Differentialgleichungen auch bei großen Zeitschritten. Das implizite Eulerverfahren errechnet Änderungen an den Zuständen der Partikel nach folgender Formel:

Implizites Eulerverfahren:

$$\Delta\mathbf{v} = h\mathbf{M}^{-1}\mathbf{f}_{n+1} \tag{5.7}$$
$$\Delta\mathbf{x} = h\mathbf{v}_{n+1} \tag{5.8}$$

Um die neuen Positionen bzw. Geschwindigkeiten der Partikel zu berechnen, muss jetzt ein Gleichungssystem gelöst werden, da Werte aus dem neuen Zeitschritt auf beiden Seiten der Gleichung auftauchen. Da sich die neuen Werte nicht direkt ergeben, nennt man diese Verfahren *implizit*. Im Allgemeinen erhält man aufgrund des Terms $\mathbf{f}_{n+1}$ ein nicht lineares Gleichungssystem, dessen Lösung mit dem Newton-Verfahren sehr aufwändig ist.

Daher verwendet man in der Stoffanimation meist semi-implizite Verfahren (siehe beispielsweise in [BW98, CK02]), bei denen die Kräfte im nächsten Zeitschritt mit Hilfe einer Taylorreihenentwicklung erster Ordnung approximiert werden:

$$\mathbf{f}_{n+1} \approx \mathbf{f}_n + \frac{\partial\mathbf{f}}{\partial\mathbf{x}}\Delta\mathbf{x} + \frac{\partial\mathbf{f}}{\partial\mathbf{v}}\Delta\mathbf{v} \tag{5.9}$$

Der Nachteil der semi-impliziten Verfahren ist die Einführung einer künstlichen Dämpfung in das System [DSB99].

Setzt man die approximierte Kraft aus Gleichung 5.9 in 5.7 ein, ergibt sich

$$\Delta\mathbf{v} = h\mathbf{M}^{-1}[\mathbf{f}_n + \frac{\partial\mathbf{f}}{\partial\mathbf{x}}\Delta\mathbf{x} + \frac{\partial\mathbf{f}}{\partial\mathbf{v}}\Delta\mathbf{v}]. \tag{5.10}$$

Stellt man nach $\Delta\mathbf{v}$ um ergibt sich mit $\Delta\mathbf{x} = h\Delta\mathbf{v}_{n+1} = h(\mathbf{v}_n + \Delta\mathbf{v})$ folgendes Gleichungssystem

$$\Delta\mathbf{v} = h\mathbf{M}^{-1}[\mathbf{f}_n + h\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{v}_n + h\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\Delta\mathbf{v} + \frac{\partial\mathbf{f}}{\partial\mathbf{v}}\Delta\mathbf{v}] \tag{5.11}$$
$$\left[\mathbf{M} - h^2\frac{\partial\mathbf{f}}{\partial\mathbf{x}} - h\frac{\partial\mathbf{f}}{\partial\mathbf{v}}\right]\Delta\mathbf{v} = h[\mathbf{f}_n + h\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\mathbf{v}_n] \tag{5.12}$$
$$\mathbf{A}\Delta\mathbf{v} = \mathbf{b} \tag{5.13}$$

In [CK02] wird das genauere *BDF-2 Verfahren* verwendet (Backward Difference Formula 2.ter Ordnung). Ein weiterer Ansatz ist die Verwendung von *IMEX Verfahren*, die die nicht steifen Anteile explizit integrieren und den Rest implizit [BA04, EEH00].

Lösen des linearen Gleichungssystems

Die semi-impliziten Verfahren erfordern eine möglichst effiziente Lösung der entstehenden linearen Gleichungssysteme $\mathbf{A}\Delta\mathbf{v} = \mathbf{b}$. Glücklicherweise sind diese Gleichungssysteme dünn besetzt und lassen sich mit der Methode der konjugierten Gradienten (CG-Verfahren) vergleichsweise schnell lösen. In [BW98] wird zum Lösen der Gleichungssysteme ein modifiziertes CG-Verfahren vorgestellt, das zusätzlich noch so genannte Constraints, d.h. Einschränkungen der Zustände eines Partikels, beim Lösen berücksichtigt. Mit Hilfe der Constraints lässt sich die Kollisionsbehandlung durchführen.

Bei größeren Partikelsystemen sinkt die Performanz des CG-Verfahrens jedoch stark ab. Daher stellt die Verwendung von *Mehrgitterverfahren* eine interessante Alternative dar und wird in Abschnitt 5.4 vorgestellt.

5.2 Modellierung von Textilien als Partikelsystem

5.2.1 Diskretisierung

Zur Animation eines Stück Stoffs mit Hilfe eines Partikelsystems wird eine geeignete Diskretisierung seiner Fläche benötigt. Man kann grob zwischen einer regulären Diskretisierung mit Rechtecksnetzen und irregulären Dreiecksnetzen unterscheiden. Die Wahl der Art der Diskretisierung beeinflusst entscheidend die weiteren Schritte der Modellbildung, d.h. auf welche Art und Weise die internen Kräfte im Material modelliert werden können.

Im Folgenden wird die Diskretisierung mit Rechtecksnetzen und mit Dreiecksnetzen näher erläutert. Für die Animation von Bekleidung sind Dreiecksnetze vorteilhaft, daher wird danach genauer auf diese Form der Diskretisierung eingegangen und ein Verfahren vorgestellt, mit dem sich Dreiecksnetze von hoher Qualität erzeugen lassen.

Rechtecksnetze vs. Dreiecksnetze

Einige Modelle zur Simulation nutzen reguläre Rechtecksnetze als Basis für die Koppelung der einzelnen Partikel [EWS96, BHW94]. Das Netz ist dabei an der Webstruktur des Stoffes ausgerichtet. Beim Weben unterscheidet man die Kett- und die Schussrichtung. Diese beiden Richtungen sind zueinander orthogonal und legen zu einem großen Teil das Verhalten des Stoffes fest. Außerdem lassen sich die Kräfte, die in diesen Richtungen wirken, mittels der Kawabata-Messung [Kaw80] für beliebige Textilien ermitteln. Man kann demnach die Simulation mit unterschiedlichen Materialien parametrisieren. Dies erlaubt eine physikalisch korrekte Simulation.

Will man nicht nur rechteckige Stoffe sondern komplette Kleidungsstücke, die aus mehreren Schnittteilen bestehen, simulieren, tritt ein im Folgenden beschriebenes Problem zu Tage. Betrachtet man die Randkurve eines Schnittteils, so stellt

man fest, dass diese nur selten an der Kett- oder Schussrichtung ausgerichtet ist. Die meisten Randkurven sind sogar gekrümmt. Diese Krümmung wird meist durch Einfügen vieler Eckpunkte mit kleinem Abstand approximiert. Legt man ein rechteckiges Gitter über ein solches Schnittteil erkennt man, dass Rechtecksnetze die Randkurve eines Schnittteils nur sehr schlecht approximieren (siehe Abbildung 5.1). Eine Lösung wäre die Auflösung des Gitters zu erhöhen. Dies würde aber gleichzeitig bedeuten, dass die Rechenzeit stark ansteigen würde, da in einem Partikelsystem die Rechenzeit im Wesentlichen von der Zahl der verwendeten Partikel abhängt [VMT01].

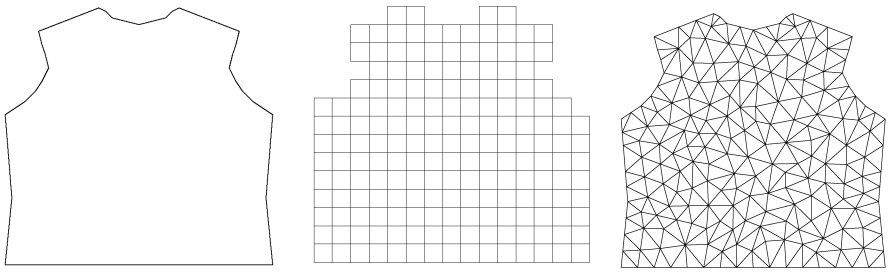


Abbildung 5.1: Randkurve, Dreiecksnetz und Rechtecksnetz. Trotz gleicher Anzahl Partikel approximiert das Rechtecksnetz die Randkurve nur sehr schlecht, das Dreiecksnetz hingegen sehr gut.

Alternativ kann man Dreiecksnetze zur Modellierung von Schnittteilen verwenden. Mit Dreiecksnetzen kann man beliebige Randkurven mit relativ wenigen Partikeln sehr gut approximieren. In dem Fall, dass die Randkurve als Polygonzug mit geradlinig verbundenen Segmenten vorgegeben ist, kann man so triangulieren, dass die Randkurve des Dreiecksnetzes mit der des Schnittteils übereinstimmt. Allerdings wird die Modellierung des anisotropen Verhaltens im Gegensatz zu Rechtecksnetzen wesentlich komplizierter. Viereckshetze werden nicht verwendet, da sie die Nachteile von Dreiecksnetzen und Rechtecksnetzen in sich vereinen.

Eine dritte Möglichkeit ist die Verwendung eines hybriden Netzes, das im Inneren aus Rechtecken und am Rand aus Dreiecken besteht. Dieses hybride Netz würde sowohl die Randkurve sehr gut approximieren, als auch im Inneren eine akkurate Simulation ermöglichen. Es ist aber unklar, ob Dreiecke und Rechtecke gemeinsam in einem Netz vernünftig simuliert werden können und wie das Materialverhalten am Rand abgebildet werden kann.

Anforderungen an die Dreiecksnetze

Die Randkurve eines Schnittteils liegt als geschlossener Polygonzug vor. Aus diesem Polygonzug soll ein Dreiecksnetz generiert werden, das alle Punkte und Kanten des Polygonzugs enthält. Das Netz dient als Grundlage für die Simulation. Im nächsten Abschnitt wird erläutert, wie ein Polygonzug trianguliert werden kann.

Dabei ist es für die spätere Verarbeitung wichtig, dass gewisse Eigenschaften des erzeugten Netzes garantiert werden können:

- Keine kleinen Winkel zwischen zwei Kanten (Angabe einer unteren Schranke).
- Maximaler Flächeninhalt der Dreiecke (Angabe einer oberen Schranke).

Aus diesen beiden Eigenschaften lässt sich ableiten, dass ein einzelnes Dreieck eine maximale räumliche Ausdehnung besitzt. Durch Beschränkung des Flächeninhalts und des minimalen Winkels ergibt sich eine maximale Kantenlänge. Die numerische Simulation profitiert von hochwertigen Netzen, da kleine Winkel und die daraus resultierenden spitzen Dreiecke zu numerischen Problemen führen können.

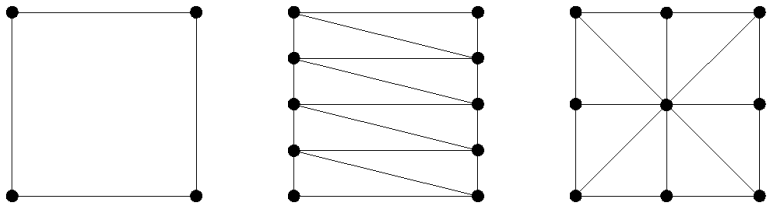


Abbildung 5.2: Links: Eine Fläche — Mitte: Die Triangulierung der Fläche mit spitzen Dreiecken. — Rechts: Die optimale Diskretisierung (maximale Winkel).

Erzeugung von Dreiecksnetzen hoher Qualität

Im Folgenden werden drei Strategien zur Triangulierung vorgestellt, die jeweils aufeinander aufbauen. Die dritte, die so genannte *Steiner Triangulierung*, entspricht den Anforderungen und wird in dieser Arbeit für die Triangulierung von Schnittteilen verwendet. Die verwendeten Begriffe tauchen im Kontext der algorithmischen Geometrie häufig auf. Weiterführende Informationen zur Triangulierung findet man in [BE92], und zur algorithmischen Geometrie im Allgemeinen in [FP85].

Folgende Definition wird für die Eingabe verwendet: Ein *planar straight line graph* (PSLG) besteht aus einer Menge von Punkten in der Ebene $\mathbf{P}$ und einer Menge von Liniensegmenten $\mathbf{S}$. Jedes $\mathbf{s} \in \mathbf{S}$ verbindet zwei Punkte $\mathbf{p}_1, \mathbf{p}_2 \in \mathbf{P}$. Die *Delaunay Triangulierung* einer Punktmenge ($\mathbf{S} = \emptyset$) hat die Eigenschaft, dass kein Punkt innerhalb des Kreises liegt, der durch die drei Punkte eines Dreiecks gebildet wird. Diese Eigenschaft wird auch Delaunay-Eigenschaft genannt. Die Delaunay Triangulierung maximiert den minimalen Winkel der Dreiecke.

Eine *Constrained Delaunay Triangulierung* (CDT) ist die Triangulierung eines PSLG, der zusätzliche Kanten enthalten darf. Diese Kanten bleiben in der Triangulierung erhalten. Ein Punkt $\mathbf{a} \in \mathbf{P}$ heißt *sichtbar* für $\mathbf{b} \in \mathbf{P}$, wenn die Verbindung

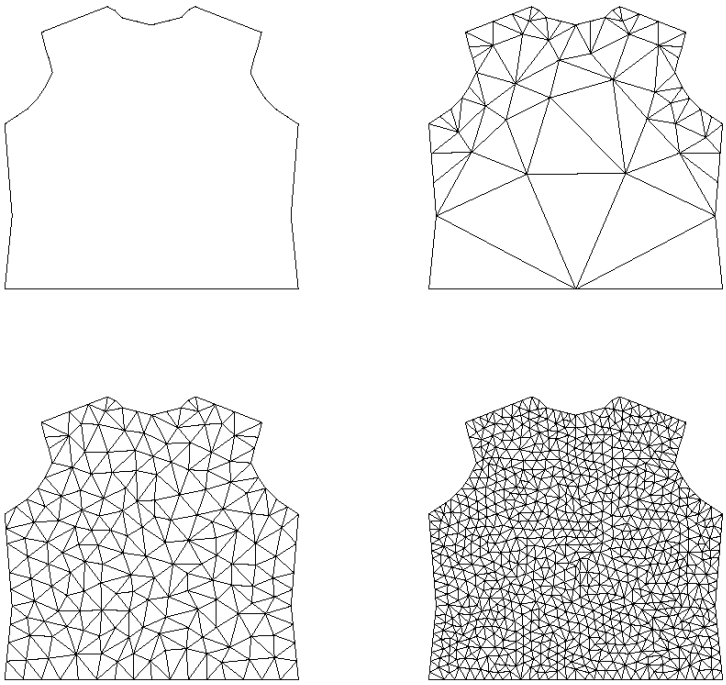


Abbildung 5.3: Vergleich zwischen verschieden fein triangulierten Dreiecksnetzen. Oben: Randkurve und deren Constrained Delaunay Triangulierung. — Unten: Steiner Triangulierung mit $25^2 mm$ und $5^2 mm$ maximaler Dreiecksfläche.

von **a** nach **b** kein Segment aus **S** schneidet. Eine CDT liegt dann vor, wenn für alle Segmente der Triangulierung $(\mathbf{p}_1, \mathbf{p}_2)$ gilt: $\mathbf{p}_1$ ist sichtbar zu $\mathbf{p}_2$ und jeder Kreis durch $\mathbf{p}_1$ und $\mathbf{p}_2$ enthält keinen Punkte $\mathbf{p}_3$, der sichtbar ist von einem Punkt der Kante $(\mathbf{p}_1, \mathbf{p}_2)$.

Bei der Delaunay Triangulierung und der CDT kann der minimale Winkel zwischen zwei Dreiecksseiten immer noch sehr klein sein. Deshalb kann man bei der Triangulierung das Einfügen weiterer Punkte zulassen. Dies wird *Steiner Triangulierung* genannt. Die eingefügten Punkte heißen Steiner Punkte. Sie werden benutzt, um bestimmte Optimalitätskriterien des Netzes zu erzwingen. Dabei muss man bedenken, dass beim Einfügen beliebig vieler Punkte fast jedes Kriterium erfüllbar wird. Deshalb darf ein Algorithmus zur Erzeugung eines Dreiecksnetzes nur eine begrenzte Zahl von Punkten einfügen.

Verbessern der Triangulierung

Eine mit dem Verfahren aus dem vorigen Abschnitt erzeugte Triangulierung kann noch verbessert werden, indem danach ein Glättungsverfahren angewandt wird. In [FJP95] wird ein iteratives Verfahren vorgestellt, dass in jedem Schritt die optimale Platzierung der Vertices berechnet. Es ist zwar pro Schritt wesentlich aufwändiger

als eine klassische Laplace-Filterung, konvergiert aber auch in weniger Iterationen. Die Laplace-Filterung ist jedoch wesentlich einfacher zu implementieren und erzeugt ebenfalls in sehr kurzer Zeit eine Triangulierung von hoher Qualität.

5.2.2 Interne Kräfte

Die meisten Textilien und insbesondere gewebter Stoff verändern ihre Länge kaum unter Zug. In einem Modell zur Simulation dieses Verhaltens bedeutet dies, dass die Abstände zwischen benachbarten Partikeln sich kaum ändern dürfen. Ansonsten würde das Kleidungsstück so aussehen, als ob es aus Gummi und nicht aus gewebtem Stoff wäre. Um dieses Verhalten zu simulieren, werden sehr steife Federn zwischen benachbarte Partikel gesetzt. Diese so genannten Strukturfedern sorgen dafür, dass sich die Abstände der Partikel zueinander nur wenig ändern. Des Weiteren muss noch das Biegeverhalten simuliert werden. Im Allgemeinen sind gewebte Stoffe recht leicht zu verbiegen, was dazu führt, dass diese Biegefedern recht weich sein können.

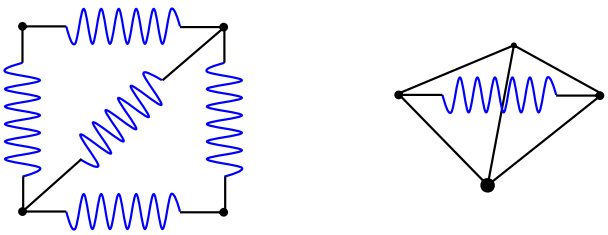


Abbildung 5.4: Links: Strukturfedern zwischen den Partikeln. — Rechts: Biegefeder zwischen zwei nicht benachbarten Partikeln.

In einem Dreiecksnetz kann man alle Kanten gleichzeitig auch als Strukturfedern ansehen. In einem Rechtecksnetz sollten zusätzlich noch Scherfedern eingefügt werden [EWS96]. Die Biegefedern können zwischen zwei Partikel gesetzt werden, die sich in zwei benachbarten Dreiecken gegenüberliegen (siehe Abbildung 5.4). Damit steuert man das Biegeverhalten der beiden Dreiecke an dieser gemeinsamen Kante.

Verbindungen zwischen zwei Partikeln lassen sich durch lineare Federn repräsentieren, deren Kraft nach dem Hooke'schen Gesetz bestimmt wird:

$$F = -dx \tag{5.14}$$

F	Resultierende Rückstellkraft der Feder
k[$\frac{N}{m}$]	Federkonstante
x[m]	Auslenkung der Feder

Für eine Verbindung zwischen den zwei Partikeln $\mathbf{p}_i$ und $\mathbf{p}_j$ ergibt sich für die Kraft, die auf $\mathbf{p}_i$ wirkt:

$$\mathbf{f}_i j = k \underbrace{(|\mathbf{x}_{ij}| - L)}_{Auslenkung} \underbrace{\frac{\mathbf{x}_{ij}}{|\mathbf{x}_{ij}|}}_{Federrichtung} \tag{5.15}$$

$\mathbf{f}_{ij}$ [N]	Kraft in $\mathbf{p}_i$ verursacht durch $\mathbf{p}_j$
$k_{ij}[\frac{N}{m}]$	Federkonstante
$\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ [m]	Positionsdifferenz in 3D
L_{ij} [m]	Ruhelänge der Feder.

Diese Kräfte werden für alle Partikel und alle Verbindungen mit verschiedenen Konstanten für strukturelle Federn, Scher- und Biegefedern berechnet. Für ein Partikel $\mathbf{p}_i$, welches mit einer Menge von Partikeln J verbunden ist, ergibt sich die Gesamtkraft zu

$$\mathbf{f}_i = \sum_{j \in J} k_{ij}(|\mathbf{x}_{ij}| - L_{ij}) \frac{\mathbf{x}_{ij}}{|\mathbf{x}_{ij}|}. \tag{5.16}$$

Die Federkonstanten können in einem uniformen Rechteckgitter alle gleich gewählt werden. Bei der Verwendung eines Dreiecksnetzes, oder wenn die Konstanten diskretisierungsunabhängig sein sollen, muss für jede Verbindung ein eigener Wert bestimmt werden. Für die Berechnung der k_{ij} in einem Dreiecksnetz sei auf die Arbeit von Van Gelder verwiesen [Gel98].

Da Stoff im Allgemeinen ein stark nichtlineares Last-Dehnungsverhalten hat (siehe Abbildung 5.5), wird dessen Verhalten wesentlich besser durch ein kubisches Federmodell als ein lineares abgebildet. Die Kraft, die zwischen zwei Parti-

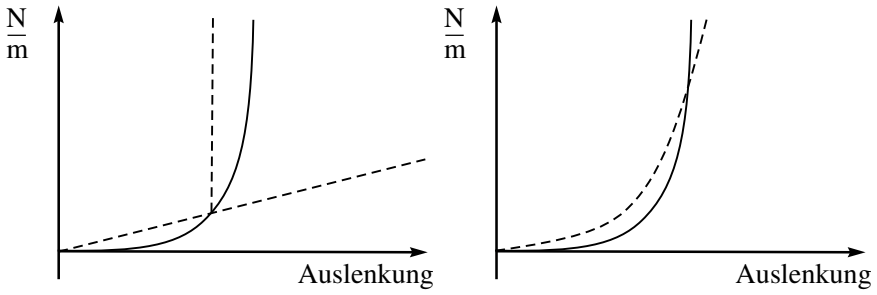


Abbildung 5.5: Durchgezogene Kurve: Nichtlineares Dehnungsverhalten realer Textilien. — Links: Unzureichende Näherung durch lineare Feder. — Rechts: Bessere Approximation durch kubisches Federmodell.

keln $\mathbf{p}_i$ und $\mathbf{p}_j$ auftritt, berechnet sich dann wie folgt:

$$\mathbf{f}_{ij} = [k_l(|\mathbf{x}_{ij}| - L) + k_c(|\mathbf{x}_{ij}| - L)^3] \frac{\mathbf{x}_{ij}}{|\mathbf{x}_{ij}|}, \tag{5.17}$$

Dabei beschreibt k_l den linearen und k_c den kubischen Koeffizienten.

5.2.3 Externe Kräfte

Die externen Kräfte wirken von außen auf das Material und tragen somit entscheidend zur Deformation von Stoff bei. In einem Modell muss entschieden werden, welche externen Kräfte verwendet werden und welche eher unwichtig sind. Die wichtigsten Kräfte sind die Gravitation, Eingaben des Anwenders und Kollisionen mit anderen Objekten. Weiterhin kann auch noch der Luftwiderstand oder sogar Wind modelliert werden.

Die *Gravitation* wirkt an jedem Partikel als Kraft

$$\mathbf{f}_i^g = 9,81 \cdot \mathbf{g} \cdot m_i \ , \tag{5.18}$$

wobei **g** der normalisierte Vektor in Richtung des Zentrums der Gravitation ist (sprich er zeigt auf den Boden einer Szene). Diese Kraft wird zu den anderen Kräften, die auf den Partikel wirken hinzu addiert.

Luftwiderstand wird gerne als idealer viskoser Zug resultierend in der Kraft

$$\mathbf{f}_i^d = -d\mathbf{v}_i,$$

modelliert. Damit der Parameter d unabhängig von der Diskretisierung des Partikelsystem ist kann folgende Formel verwendet werden:

$$\mathbf{f}_i^d = -d \cdot m_i \cdot \mathbf{v}_i \tag{5.19}$$

Übersicht über die Verwendeten Symbole und Einheiten:

$\mathbf{f}_i$ [N]	Resultierende Kraft (eines Partikels)
m_i [kg]	Masse (eines Partikels)
d $[\frac{1}{s}]$	Diskretisierungsunabhängiger Widerstandskoeffizient
$\mathbf{v}_i$ $[\frac{m}{s}]$	Geschwindigkeit (eines Partikels)
$\mathbf{a}_i$ $[\frac{m}{s^2}]$	Beschleunigung (eines Partikels)

Die *Eingaben eines Anwenders* können dazu verwendet werden einen Partikel zu greifen und seine Position zu verändern. Hierfür kann die Maus oder ein anderes geeignetes Eingabegerät verwendet werden. Bewegt der Anwender das Eingabegerät, so bewegt sich der Partikel ebenfalls. Dabei kann die Eingabe direkt umgesetzt werden, d.h. der Partikel folgt allen Bewegungen des Eingabegerätes. Alternativ kann zwischen der Position, die das Eingabegerät vorgibt, und den Partikel eine gedämpfte Feder gesetzt werden, die dann eine bestimmte Kraft überträgt. Der Vorteil dieser Variante besteht darin, dass Änderungen nicht so ruckartig erfolgen wie bei einer direkten Positionsänderung. Allerdings wird die Zielposition nicht immer erreicht.

Damit die Position eines Partikels direkt verändert werden kann, muss dieser Vorgang Priorität vor allen anderen Kräften besitzen. Dies kann dadurch modelliert werden, dass man die Masse des Partikels auf unendlich setzt und gleichzeitig die Geschwindigkeit auf 0. Danach verändern andere Kräfte seine Position nicht mehr.

Auch mehrere Partikel können auf diese Weise in ihrer Bewegung eingeschränkt werden.

Um die Beschleunigung zu errechnen wird das Konzept der inversen Masse verwendet [BW98]. Eine inverse Masse m_i^{inv} wird auf Null gesetzt, wenn m_i unendlich ist. Die Beschleunigung ergibt sich dann als

$$\mathbf{a}_i = \frac{\mathbf{f}_i}{m_i} = m_i^{inv} \cdot \mathbf{f}_i. \tag{5.20}$$

Dieses Vorgehen vermeidet Fallunterscheidungen, womit die Berechnung wesentlich vereinfacht wird.

5.3 Interaktive Simulation von Textilien

5.3.1 Robuste Modellierung interner Kräfte

In diesem Abschnitt wird ein Verfahren zur robusten Modellierung interner Kräfte vorgestellt. Das Verfahren ist äußerst stabil und erlaubt große Zeitschritte [FGL03].

Bei Verwendung von Partikelsystemen wird ein Stück Stoff in eine Menge von Partikeln diskretisiert. Da die Schnittteile von Kleidungsstücken oft kompliziert geformt sind, werden diese mittels eines Dreiecksnetzes diskretisiert. Die Kanten des Netzes repräsentieren die Strukturfedern und die Vertices sind die Partikel, in denen die gesamte Masse zusammengefasst wird. Zusätzlich werden Biegefedern eingefügt. Diese verbinden die nicht benachbarten Partikel zweier benachbarter Dreiecke (siehe auch Abbildung 5.6).

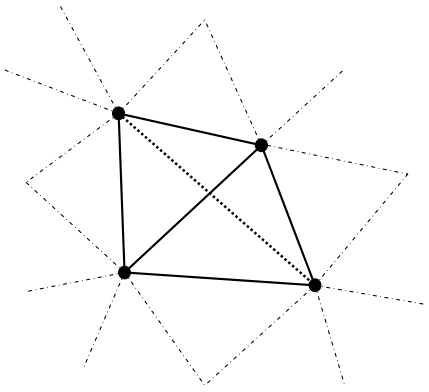


Abbildung 5.6: Verbindungen in einem Dreiecksnetz: Die Kanten (dicke Linien) entsprechen Strukturfedern und es werden zusätzlich noch Biegefedern hinzugefügt (gepunktete Linie).

Da die Triangulierung nicht regelmäßig ist, besitzen nicht alle Partikel die gleiche Masse. Um diese für einen Partikel zu berechnen wird folgende Formel ver-

wendet

$$\mathbf{m}_i = m_{pm} \cdot \sum_{\text{adj. triangles}} \frac{area_j}{3} \quad , \tag{5.21}$$

wobei $area_j$ die Fläche eines Dreiecks bezeichnet und m_{pm} ein Materialparameter des Stoffes ist, der die Masse pro Quadratmeter Stoff angibt. Die Formel ergibt sich aus der Idee, jedem Partikel ein Drittel der Fläche seiner adjazenten Dreiecke zuzuweisen.

Die Art und Weise auf die Partikel miteinander interagieren unterscheidet die verschiedenen Modelle zur Simulation von Stoff, die in der Literatur beschrieben sind. Viele Fortschritte wurden in der akkuraten Nachbildung des Stoffverhaltens erzielt [BHW94, EWS96, VCMT95, BW98, HE01, CK02]. In diesen Modellen wird Scherung, Anisotropie und das Biegeverhalten simuliert. Das hier vorgestellte Modell nimmt einige Vereinfachungen vor und approximiert das Dehn- und Biegeverhalten von Stoff. Die Anisotropie und die Scherung werden vernachlässigt, da sie nur für spezielle Stoffe von hoher Bedeutung sind. Allerdings kann das hier vorgestellte Modell auch diese Eigenschaften modellieren, indem Rechtecksnetze anstelle von Dreiecksnetzen verwendet werden.

Um das Modell zu veranschaulichen wird kurz ein einfaches Modell, ein Masse-Feder System, das in vielen interaktiven Systemen verwendet wird [MDDB01, KC02], vorgestellt. Die Interaktion zwischen verbundenen Partikeln wird durch lineare Federn abgebildet. Die resultierende Kraft ergibt sich als

$$\mathbf{f}_{ij} = k_{ij}(\|\mathbf{x}_j - \mathbf{x}_i\| - l_{ij}) \cdot \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|}, \tag{5.22}$$

wobei k_{ij} die Federkonstante bezeichnet und l_{ij} die Ruhelänge der Feder darstellt. Die Verwendung von linearen Federn ist jedoch nicht besonders hilfreich, da Stoff ein hochgradig nicht-lineares Verhalten zeigt. Er wird sehr steif, nachdem die Auslenkung einen bestimmten Schwellwert übersteigt. Eine Lösung hierfür ist es, sehr hohe Federkonstanten zu verwenden. Dann muss aber auch ein implizites Verfahren zur Lösung der Bewegungsgleichung verwendet werden[BW98, VMT01, CMT02]. Besser sind kubische Federn, wie in Abschnitt 5.2.2 beschrieben, da sie das Stoffverhalten besser annähern. Allerdings wird auch hier ein aufwendiges implizites Lösungsverfahren benötigt.

Ein ganz anderer, älterer Ansatz verwendet Federn mit kleinen k_{ij} und wendet eine Postkorrektur der Federn an, die zu stark ausgelenkt sind [Pro95]. Dieser Algorithmus iteriert über alle Federn und bewegt jeweils zwei Partikel entlang ihrer gemeinsamen Achse aufeinander zu. Die beiden Partikel werden entweder um die gleiche Distanz bewegt oder für den Fall von externen Einschränkungen wird nur einer der Partikel bewegt. Dieser Ansatz wurde auch im Kontext eines approximierenden impliziten Integrationsverfahrens verwendet [MDDB01].

In [VSC01] wird die Postkorrektur der Positionen der Partikel in eine Korrektur der Geschwindigkeiten konvertiert, so dass niemals eine zu starke Auslenkung vorkommt. Da der hier vorgestellte Ansatz ohne Geschwindigkeiten arbeitet, wie

später näher erläutert wird, wird dieser Ansatz nicht weiter verfolgt. Eine sehr inspirierende Idee wird in [Jak01] vorgestellt, wo jede Feder als Stab modelliert wird, der seine Länge überhaupt nicht ändert. Daraus ergibt sich ein sehr steifes Material, das schnell und vor allem robust animiert werden kann.

Die hier vorgestellt Methode geht in eine ähnlich Richtung, aber um einige Schritte weiter als die bisherigen Verfahren. Zunächst werden die k_{ij} auf Null gesetzt, d.h. es werden keine Federn im herkömmlichen Sinne mehr verwendet. Danach wird eine Postkorrektur, wie sie von Provot [Pro95] beschrieben wurde, angewandt. Allerdings wird jetzt die *Kompression* des Stoff nicht mehr von Federn verhindert und das Material kann kollabieren. Daher werden nicht nur zu lange Federn verkürzt, sondern auch zu kurze wieder verlängert.

Die neuen Positionen der Partikel werden bestimmt, indem zuerst die Richtung und die Länge des *Korrekturvektor* folgendermaßen errechnet werden

$$\mathbf{u} = \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|}(\|\mathbf{x}_j - \mathbf{x}_i\| - d_{max} \cdot l_{ij}). \tag{5.23}$$

Hierbei bezeichnet $d_{max} \cdot l_{ij}$ die maximal erlaubte Länge. Ersetzt man d_{max} mit d_{min}, so erhält man die Formel für komprimierte Federn.

Da zur Diskretisierung ein Dreiecksnetz verwendet wurde, haben alle Partikel unterschiedliche Masse. Daher können nicht einfach beide Partikel um $0.5 \cdot \mathbf{u}$, wie im Verfahren von Provot, verschoben werden. Dadurch würde sich der gemeinsame Schwerpunkt verändern. Gewichtet man den Korrekturvektor jedoch mit

$$m_1 = \frac{m_i}{m_i + m_j} \tag{5.24}$$

$$m_2 = \frac{m_j}{m_i + m_j} \tag{5.25}$$

und addiert dann

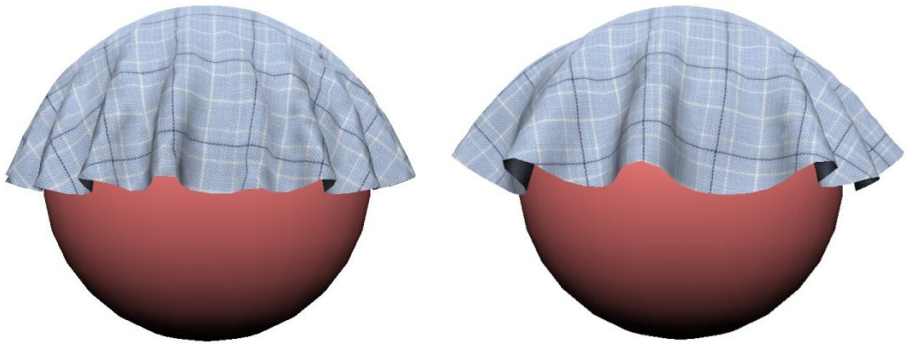
$$\mathbf{x}_i \mathrel{+}= m_2 \cdot \mathbf{u} \tag{5.26}$$

$$\mathbf{x}_j \mathrel{-}= m_1 \cdot \mathbf{u} \tag{5.27}$$

bleibt der Schwerpunkt der beiden Partikel konstant.

Weiterhin können *externe Einschränkungen* einfach behandelt werden, indem m_1 und m_2 auf die entsprechenden Werte gesetzt werden. Soll beispielsweise der Partikel $\mathbf{p}_j$ fixiert werden, werden nicht Gleichung 5.24 und 5.25 ausgewertet, sondern $m_1 = 0$ und $m_2 = 1$ gesetzt (in diesem Fall wäre ja m_j unendlich (siehe auch Abschnitt 5.2.3)). Als Ergebnis würde sich $\mathbf{p}_j$ gar nicht bewegen und der andere Partikel hingegen um die volle Länge von $\mathbf{u}$. Sollten beide Partikel unendliche Masse haben, wird $m_1 = m_2 = 0$ gesetzt.

Die *Biegefedern* werden ebenfalls durch die geometrischen Einschränkungen ersetzt. Sie werden genauso behandelt, wie Federn die aufgrund von Kompression zu kurz sind. Allerdings wird ein anderer Parameter verwendet, damit das



(a) (b)

Abbildung 5.7: Ein Tischtuch bestehend aus 3200 Dreiecken, das in Echtzeit auf einer Kugel fallen gelassen wird. Es wurden unterschiedliche Biegesteifigkeiten des Materials für (a) und (b) verwendet.

Verhalten unabhängig von der Kompression verändert werden kann. In konkreten Anwendungen ergeben sich für Werte zwischen $1,0$ und $0,8$ gute Ergebnisse. Die Wahl hängt von der Biegesteifigkeit des simulierten Materials ab. In Abbildung 5.7 sind zwei verschiedene Materialien dargestellt.

Eine einzelne Iteration über alle geometrischen Einschränkungen (Biege- und Strukturfedern) reicht natürlich nicht aus, da eine Änderung einer Länge einer Feder die Längen benachbarter Federn ändert. Diese müssen in der nächsten Iteration ebenfalls korrigiert werden. In Praxis genügen für gewöhnlich wenige Iterationen. Die Anzahl hängt dabei wesentlich von der Auflösung des Stoffs und dem Zeitschritt ab. Da die Änderungen nur lokal sind, muss die Information über Änderungen in den Positionen Schritt für Schritt durch das Mesh transportiert werden. Daraus ergibt sich, dass in Netzen hoher Auflösung die Information langsamer fließt als in niedrig aufgelösten.

Es sei noch angemerkt, dass das Partikelsystem *stabil* bleibt, auch wenn eine hohe Auflösung mit wenigen Iterationen verwendet wird. Allerdings sinkt dann die visuelle Qualität.

5.3.2 Stabile numerische Integration

Hier wird ein Verfahren zur stabilen numerischen Integration vorgestellt. Das Verfahren eignet sich besonders zur interaktiven Simulation höher aufgelöster Meshes, wie sie z.B. für die Erstellung der Beispiele in dieser Arbeit verwendet werden. Das Verfahren ist äußerst stabil und kann sich sogar von kurzfristigen Störungen erholen. Dies ist insbesondere in einem interaktiven System von Bedeutung, da der Anwender ja direkt mit dem Material interagiert und er es somit leicht sehr hohen Kräften aussetzen kann.

Das in Abschnitt 5.1.1 beschriebene explizite Eulerverfahren ist sehr schnell auszuwerten. In diesem Abschnitt werden die Formeln für das Eulerverfahren nicht nach Delta-Werten aufgelöst. Dann ergibt sich

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + dt \cdot \frac{\mathbf{f}_i^n}{m_i} \tag{5.28}$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + dt \cdot \mathbf{v}_i^{n+1}. \tag{5.29}$$

Leider ist das explizite Eulerverfahren instabil für große Zeitschritte, wenn große Kräfte zwischen den Partikeln wirken. Da das hier beschriebene Verfahren die hohen internen Kräfte durch geometrische Einschränkungen ersetzt und die externen Kräfte (z.B. Gravitation) nicht hoch sind, kann jedoch ein einfaches explizites Verfahren verwendet werden.

Ein weiteres Problem sind schnelle Änderungen in den Positionen der Partikel, die durch die Kollisionsantwort oder Benutzereingaben entstehen. Diese können zu Instabilitäten führen. Um dieses Problem anzugehen, schlagen Meyer et al. [MDDB01] vor die Geschwindigkeiten nach jedem Zeitschritt zu korrigieren. Dies ist äquivalent zu einer Korrektur vor einem Zeitschritt durch

$$\mathbf{v}_i^n = \frac{\mathbf{x}_i^n - \mathbf{x}_i^{n-1}}{dt}. \tag{5.30}$$

Das Resultat dieser Gleichung ist, dass die Geschwindigkeiten sich direkt aus den Positionen der Partikel ergeben und daher Änderungen in den Positionen gleichzeitig die Geschwindigkeiten im nächsten Zeitschritt ändern. Dadurch wird die Stabilität der numerischen Integration erhöht. Dies ist nicht verwunderlich, da dieses Integrationsverfahren äquivalent zum Verletverfahren ist [Ver67]. In diesem Verfahren ergeben sich die Positionen, ohne dass Geschwindigkeiten berechnet werden, durch

$$\mathbf{x}_i^{n+1} = -\mathbf{x}_i^{n-1} + 2\mathbf{x}_i^n + dt^2 \frac{\mathbf{f}_i^n}{m_i}. \tag{5.31}$$

Die Äquivalenz wird erkennbar, wenn Gleichung 5.28 in Gleichung 5.29 eingesetzt wird. Dies ergibt

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + dt(\mathbf{v}_i^n + dt \cdot \frac{\mathbf{f}_i^n}{m_i}). \tag{5.32}$$

Setzt man weiter Gleichung 5.30 in Gleichung 5.32 ein und schreibt sie um, ergibt sich

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \mathbf{x}_i^n - \mathbf{x}_i^{n-1} + dt^2 \cdot \frac{\mathbf{f}_i^n}{m_i} \quad , \tag{5.33}$$

was äquivalent zu Gleichung 5.31 ist.

Wie in [HEE+02] beschrieben ist das Verletverfahren eine gute Wahl für Probleme mit geringer oder keiner Dämpfung. Trotzdem kann Dämpfung eingeführt werden, indem Gleichung 5.30 mit einem geeigneten Wert multipliziert wird. Steht

der Wert auf $1,0$ so ergibt sich keine Dämpfung. Für Werte kleiner Eins kann die entstehende Dämpfung verwendet werden, um den Luftwiderstand zu simulieren. In den gezeigten Beispielen ergibt ein Wert von $0,99$ ein realistisches Verhalten von Stoff.

5.3.3 Ergebnisse

Um die Eigenschaften des vorgestellten Verfahrens zur interaktiven Simulation von textilen Materialien zu demonstrieren, sind einige Beispiele abgebildet. In Abbildung 5.7 wird ein Vergleich zwischen unterschiedlichen Biegesteifigkeiten gemacht. Durch Veränderung der zugehörigen Materialparameter können unterschiedlichste Materialien animiert werden. Die Möglichkeiten zur Veränderung der Parameter für die Strukturfedern sind begrenzt, da Werte jenseits von $1,2$ keine befriedigenden Ergebnisse mehr liefern. In diesen Fällen können sich die Dreiecke zu stark und zu unregelmäßig deformieren.

In Abbildung 5.8 ist eine Sequenz aus einer Echtzeitanimation eines Tischtuchs dargestellt, die aus einem Video entnommen wurde. Die Sequenz validiert das vorgestellte Simulationsmodell und zeigt, dass sich natürlich aussehende Bewegungen damit erzeugen lassen. Dabei kann der Benutzer in Echtzeit mit dem Tuch interagieren. Das Dreiecksnetz des Tuches besteht aus $2K$ Dreiecken und der Zeitschritt dt ist $0,0083s$, d.h. nach jedem vierten Simulationsschritt wird ein Frame angezeigt. Die geometrischen Einschränkungen, die die internen Kräfte repräsentieren, wurden viermal pro Zeitschritt angewandt. Das Beispiel mit der Kugel hatte den gleichen Zeitschritt, es wurden aber mehr Iterationen für die geometrischen Einschränkungen benötigt, da mehr Partikel beteiligt waren.

Das Verfahren wurde in Java implementiert und die gezeigten Beispiele wurden mit Java3D gerendert. Das System lief auf einem Dual Intel Xeon^{TM} Prozessor mit $2,0GHz$, wobei aber keinerlei Parallelisierungen des Algorithmus zur Animation durchgeführt wurden. Nur das Rendering in Java3D [Sun03] verwendete den zweiten Prozessor.

5.3.4 Bewertung

Es wurde ein Algorithmus zur interaktiven Animation von textilen Materialien vorgestellt, der für Dreiecksnetze mit mehreren tausend Dreiecken sogar in Echtzeit arbeitet. Das Verfahren ist auch bei großen Zeitschritten stabil, da die hohen internen Kräfte durch geometrische Einschränkungen modelliert werden. Nur die externen Kräfte, wie Gravitation und die Änderungen in der Geschwindigkeit werden auf herkömmliche Weise über die Zeit integriert.

Weiterhin können mit dem Verfahren unterschiedliche Materialien animiert werden. Obwohl das Ziel keine absolute physikalische Korrektheit war, sehen die erzeugten Animationen realistisch aus. Ein offenes Problem ist die Abbildung von gemessen Materialparametern auf das vorgestellte Modell. Ein weiteres Problem ist Stoff der sich stark dehnen kann, da sich mit den geometrischen Ein-

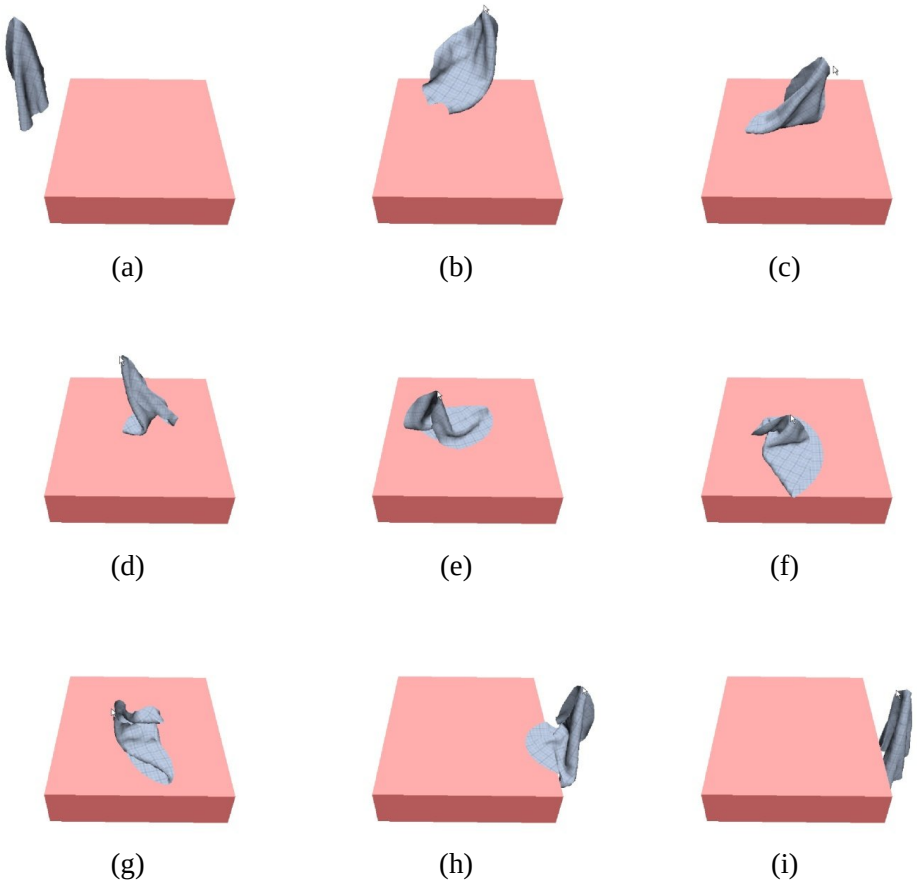


Abbildung 5.8: Ein Tischtuch wird in Echtzeit vom Anwender bewegt. Die Bilder sind aus einem Video entnommen, dass direkt vom Bildschirm aufgenommen wurde.

schränkungen Stoffe mit geringer Dehnung besser darstellen lassen. Im Falle einer starken Auslenkung der Federn stehen keine Kräfte für die Zwischenzustände zur Verfügung und es entstehen schnell Artefakte in der Visualisierung. Diese ließen sich beheben, indem die Federkräfte wieder eingeführt werden. Aber dann kann die Stabilität nicht mehr garantiert werden und die Performanz sinkt.

5.4 Mehrgitterverfahren in der Textilsimulation

Eine große Herausforderung im Bereich der Stoffsimulation ist die effiziente numerische Integration der Bewegungsgleichung, da die Integration den wesentlichen Teil des Rechenaufwands benötigt. Die Differentialgleichungen sind im Allgemeinen steif, da gewebter Stoff sich zwar kaum dehnen lässt, aber leicht gebogen werden kann. Daher kann ein explizites Integrationsverfahren nur mit klei-

nen Zeitschritten arbeiten, da ansonsten das System divergiert. Verwendet man das im vorigen Kapitel beschriebene Verfahren zur interaktiven Simulation können zwar größere Zeitschritte gewählt werden, aber Stoffe, die sich stark dehnen lassen können nicht akkurat simuliert werden.

Große Zeitschritte können genommen werden, wenn man das in [BW98] vorgestellte semi-implizite Verfahren verwendet. Dabei muss in jedem Zeitschritt ein dünn-besetztes lineares Gleichungssystem (LGS) gelöst werden. Hierfür wird ein modifiziertes vorkonditioniertes CG-Verfahren verwendet (siehe auch 5.1.2), welches wesentlich effizienter als andere iterative Lösungsverfahren arbeitet.

Nichtsdestotrotz ist das Lösen des LGS bei feiner aufgelösten Netzen äußerst zeitaufwändig. Daher wurde vorgeschlagen, die Anzahl der Iterationen beim CG-Verfahren zu begrenzen um die Berechnungen schneller durchführen zu können [VMT01]. Allerdings führt dies zu verringerter Genauigkeit und, wesentlich gravierender, dazu dass in Folgezeitschritten aufgrund akkumulierter Fehler keine Lösung mehr gefunden werden kann.

Um sowohl effizient als auch genau rechnen zu können, bieten sich Mehrgitterverfahren an. Diese werden häufig im Bereich der Fluiddynamik eingesetzt und sind in der Lage das Konvergenzverhalten von iterativen Lösern zu verbessern. Daher wird in dieser Arbeit der Einsatz von Mehrgitterverfahren in der Stoffanimation untersucht.

Im Folgenden werden zunächst Mehrgitterverfahren eingeführt und danach die Übertragung auf die Lösung der bei der Stoffsimulation auftretenden linearen Gleichungssysteme vorgestellt. Die Ergebnisse zeigen, dass insbesondere leicht gedämpfte Systeme (in diesem Fall Dämpfung durch das Material), von der Verwendung eines Mehrgitterverfahrens profitieren.

5.4.1 Mehrgitterverfahren

Mit Hilfe von Mehrgitterverfahren lässt sich die Lösung linearer Gleichungssystem der Form

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{5.34}$$

oft schneller bestimmen als mit klassischen iterativen Lösungsverfahren (Jacobi, Gauss-Seidel, Successive Over-Relaxation, etc.). Dabei greifen die Mehrgitterverfahren während der einzelnen Schritte auf ebendiese iterativen Verfahren zurück.

Wesentlich für die Mehrgitterverfahren ist die Verwendung von Rechengittern unterschiedlicher Auflösung während der Berechnung. Das feinste dieser Gitter ergibt sich dabei aus dem gegebenen Problem, beispielsweise einer partiellen Differentialgleichung. Während des Lösens wird gegebenenfalls mehrfach zwischen den einzelnen Gittern gewechselt (siehe Abbildung 5.9 für ein Beispiel mehrerer Gitter).

Wechselt man auf ein gröberes Gitter, so spricht man von *Restriktion*, beim Wechsel auf ein feineres von *Prolongation*. Das Ziel dieser Wechsel ist es die Anzahl der benötigten Schritte eines iterativen Lösungsverfahrens auf dem feinsten

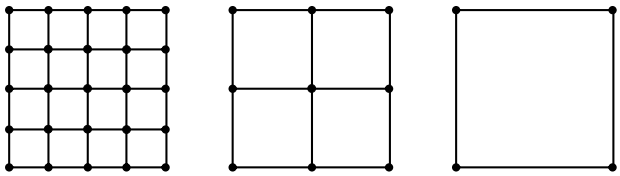


Abbildung 5.9: Ein rechteckiges Simulationsgitter in verschiedenen Diskretisierungsstufen, wobei auf jeder Ebene damit auch ein Rechengitter assoziiert ist. Die gröberen Gitter ergeben sich durch Streichen jeder zweiten Spalte und Zeile.

Gitter zu reduzieren. Da die Rechenzeit auf den gröberen Gittern weniger ins Gewicht fällt ergibt sich dadurch eine höhere Ausführungsgeschwindigkeit. Dabei unterscheidet man zwei Methoden:

1. **Kaskadische Verfahren:** Man startet auf dem gröbsten Gitter und bestimmt dort eine Näherungslösung, die dann auf die nächste Stufe prolongiert wird. Rekursiv gelangt man zum feinsten Gitter, auf dem dann bis zur gewünschten Genauigkeit gelöst wird.

2. **Verfahren mit Grobgitterkorrektur:** Man startet auf dem feinsten Gitter und restringiert das Residuum der Näherung auf ein gröberes Gitter. Die dort berechnete Lösung wird prolongiert und ergibt einen Korrekturvektor mit dem die Lösung verbessert wird.

Bei den Verfahren mit Grobgitterkorrektur haben sich einige gängige Schedules für die Gitterwechsel etabliert (siehe Abbildung 5.10). Die Zyklen werden von links nach rechts durchlaufen, wobei sich die feineren Gitter oben und die groben Gitter unten befinden.

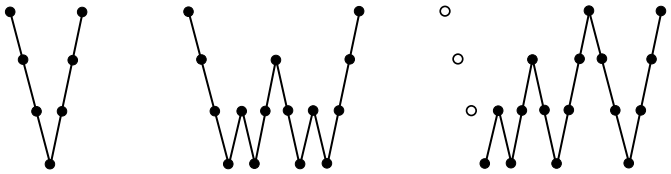


Abbildung 5.10: V-, W- und FMV-Zyklus zum Wechsel zwischen den Gittern. Das feine Gitter befindet sich hierbei jeweils oben.

Das Prinzip der Mehrgitterverfahren

Die Kernidee der Mehrgitterverfahren ist der schnellere Informationsaustausch zwischen im feinen Gitter weit voneinander entfernten Gitterpunkten. Auf einem gröberen Gitter sind diese näher beieinander und somit in der Lage sich in wenigen Iterationen zu beeinflussen.

Die iterativen Lösungsverfahren (Jacobi, etc.) sind Relaxationsverfahren, die für Gleichung 5.34 nur lokal schnelle Lösungen liefern. Für weit voneinander entfernt liegende Punkte werden viele Iterationen benötigt. Nicht so bei den Mehrgitterverfahren, die quasi die groben Gitter als Abkürzung verwenden.

Nach jedem Schritt eines der Relaxationsverfahren bleibt ein Fehler bestehen. Spektralanalysen haben ergeben, dass oszillierende Anteile dieses Fehlers sehr gut durch Relaxationsverfahren beseitigt werden können, glatte (sprich niederfrequente) Anteile jedoch nicht [Wes92]. Daher approximieren die Mehrgitterverfahren den glatten Fehleranteil auf den groben Gittern und glätten dann den nicht-glatten Anteil mittels einiger Schritte eines der iterativen Verfahren auf dem feinen Gitter. Weiterhin kann ein glatter Fehler auf einem feinen Gitter, wenn er auf ein grobes Gitter projiziert wird, nicht glatt sein [Ale02a]. Man erhält somit auf dem groben Gitter einen stärker oszillierenden Fehler.

Die Verfahren mit Grobgitterkorrektur zielen darauf ab, einen auf dem groben Gitter approximierten Fehler und einen glatten Fehler auf dem feinen Gitter miteinander zu verknüpfen und daraus einen schwächeren aber stark oszillierenden Fehler zu gewinnen, der dann von einem Relaxationsverfahren schnell reduziert werden kann.

Die Grobgitterkorrektur

Zu einer *Näherungslösung* $\mathbf{u}$ für das lineare Gleichungssystem $\mathbf{Ax} = \mathbf{b}$ lässt sich folgendes *Residuum* $\mathbf{r} = \mathbf{b} - \mathbf{Au}$ angeben. Es ergibt sich einerseits:

$$\mathbf{Au} = \mathbf{b} + \mathbf{r}.$$

Andererseits kann durch Korrektur der Näherungslösung $\mathbf{u}$ um den unbekannten *Fehler* $\mathbf{e} = \mathbf{x} - \mathbf{u}$ die Gleichung

$$\mathbf{A}(\mathbf{u} - \mathbf{e}) = \mathbf{b}$$

exakt erfüllt werden. Somit folgt insgesamt:

$$\mathbf{A}(-\mathbf{e}) = \mathbf{r}.$$

Da $\mathbf{Ax} = \mathbf{b}$ ebenfalls der Bedingung

$$\overline{\mathbf{A}}\overline{\mathbf{x}} = \overline{\mathbf{b}} \tag{5.35}$$

$\overline{\mathbf{A}}$	Grobgitter(matrix)
$\overline{\mathbf{x}}$	Tatsächliche Lösung übertragen auf das Grobgitter
$\overline{\mathbf{b}}$	Restringierte rechte Seite von $\mathbf{Ax} = \mathbf{b}$

genügt, kann über das grobe Gitter ein Korrekturvektor durch Lösen der Gleichung

$$\overline{\mathbf{A}}\overline{\mathbf{e}} = \overline{\mathbf{r}} \tag{5.36}$$

$\bar{\mathbf{e}}$ Gesuchte Korrektur auf Grobgitter
$\bar{\mathbf{r}}$ Residuum übertragen auf das grobe Gitter

berechnet werden. Die Feingitterlösung wird dann wie folgt aktualisiert:

$$\mathbf{u} = \mathbf{u} + \tilde{\mathbf{e}}, \qquad \tilde{\mathbf{e}} = \mathbf{P}\bar{\mathbf{e}}. \qquad \mathbf{P}\ \textit{Prolongationsoperator}$$

Zusammenfassend besteht die Grobgitterkorrektur also aus folgenden Schritten:

* Vorglättung	$\mathbf{u} = solve(\mathbf{u}_0, \mathbf{A}, \mathbf{b}, \#steps)$
Berechnung des Residuums	$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{u}$
* Restriktion	$\bar{\mathbf{r}} = restrict(\mathbf{r})$
* Lösung	$\bar{\mathbf{e}} = solve(\bar{\mathbf{e}}_0, \overline{\mathbf{A}}, \bar{\mathbf{r}})$ aus $\overline{\mathbf{A}}\bar{\mathbf{e}} = \bar{\mathbf{r}}$
* Prolongation	$\tilde{\mathbf{e}} = prolongate(\bar{\mathbf{e}})$
Korrektur	$\mathbf{u} = \mathbf{u} + \tilde{\mathbf{e}}$
* Nachglättung	$\mathbf{u} = solve(\mathbf{u}, \mathbf{A}, \mathbf{b}, \#steps)$

Übergang zwischen Gittern

Die *Prolongation* beschreibt den Übergang von einem groben Gitter auf ein feineres:

$$\mathbf{P} : \overline{\mathbf{U}} \rightarrow \mathbf{U}$$

Häufig verwendete Standardoperatoren sind lineare, bilineare oder sogar trilineare Interpolation. Oft ist es jedoch notwendig, auf das Problem abgestimmte Operatoren zu verwenden. Gleiches gilt auch für die Restriktion.

In der Stencil-Notation wird beschrieben welcher Anteil eines Grobgitterpunktes auf die benachbarten Punkte des feineren Gitters prolongiert wird (siehe Abbildung 5.11).

$$\mathbf{P}_{stencil} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix} \quad \mathbf{R}_{stencil} = \begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

Abbildung 5.11: Links: Prolongationsoperator. — Rechts: Full-Weighting Restriktionsoperator.

Die *Restriktion* beschreibt den Übergang von einem feinen Gitter auf ein gröberes Gitter

$$\mathbf{R} : \mathbf{U} \rightarrow \overline{\mathbf{U}}$$

und bildet einen beliebigen Vektor $\mathbf{v}$ in den Raum des gröberen Gitters ab.

In der Stencil-Notation wird für einen Grobgitterpunkt beschrieben, welche Anteile der gespeicherten Werte von umliegenden Feingitterpunkten im Grobgitterpunkt akkumuliert werden sollen. Der 2D-Full-Weighting-Operator aus Abbildung 5.11 akkumuliert die Werte umliegender Punkte (und sich selbst) so, dass wie auf dem feineren Gitter in einem Punkt ein Gesamtgewicht von 1 gespeichert wird.

Konstruktion von Grobgittern

Es existieren zwei grundlegend verschiedene Verfahren zur Konstruktion von Grobgittern:

1. Grobgitterapproximation durch Diskretisierung (Discretization coarse grid approximation, DCA) - Das gegebene Problem wird auf dem gröberen Gitter erneut diskretisiert. Analog zur Bestimmung der Matrix $\mathbf{A}$ für das Problem auf dem feinen Gitter wird $\overline{\mathbf{A}}$ auf dem groben Gitter bestimmt.

2. Galerkin Grobgitterapproximation (Galerkin coarse grid approximation, GCA) - Die Grobgittermatrix $\overline{\mathbf{A}}$ wird direkt aus den Restriktions- und Prolongationsoperatoren $\mathbf{R}$ und $\mathbf{P}$ konstruiert:

$$\overline{\mathbf{A}} = \mathbf{RAP} \tag{5.37}$$

Die DCA hat den Nachteil, dass das gegebene Problem für jedes Gitter erneut diskretisiert werden muss. Allerdings kann danach sehr effizient zwischen den Gittern gewechselt werden.

Die GCA hat den großen Vorteil, dass bei gegebenen Prolongations- und Restriktionsoperatoren die neuen Gitter automatisch bestimmt werden können. Daher bietet sich die GCA als Teil eines generischen Lösungsverfahrens an.

5.4.2 Übertragung auf die Stoffanimation

Operatoren

Die beschriebenen Operatoren für Restriktion und Prolongation (Interpolation bzw. Full-Weighting) können bei der Verwendung von Rechtecksnetzen verwendet werden, da diese der Konnektivität im Stoffmodell von Struktur- und Scherverbindungen entsprechen. Die Biegefedern und somit die Verbindungen zu den übernächsten Nachbarn werden allerdings nicht berücksichtigt, da diese nur einen geringen Einfluss auf die Lösung haben. Sie werden bis zum letzten Wechsel auf das feine Gitter nicht berücksichtigt und dann für die Bestimmung der endgültigen Lösung mit herangezogen.

Allgemein sind Restriktion und Prolongation Matrix-Vektor-Multiplikationen, wobei sich die jeweiligen Matrizen aus den Operatoren ergeben. Bei den verwendeten Stencil-Operatoren kann allerdings auf die explizite Aufstellung dieser Matrizen verzichtet werden. Durch Iteration über alle Grobgitterpunkte können sowohl

die Restriktion als auch die Prolongation effizient durchgeführt werden, indem akkumuliert bzw. verteilt wird.

Somit kann effizient zwischen den Gittern gewechselt werden. Ein Wechsel ist dabei in etwa so aufwändig, wie eine CG-Iteration auf dem jeweils feineren Gitter.

Konstruktion der Grobgitter

Die beiden grundsätzlichen Methoden zur Konstruktion der Grobgitter (DCA und GCA) werden im Folgenden kurz vorgestellt. Bei der DCA werden mechanisch äquivalente Grobgitter benötigt. Diese lassen sich in einem Vorverarbeitungsschritt berechnen. Auf folgende Punkte muss dabei geachtet werden:

- Erneute Diskretisierung mit geringerer Auflösung durch Elimination jedes zweiten Partikels
- Neuberechnung der Materialparameter (Federkonstanten, Masse, Dämpfungskoeffizienten, etc.)
- Geeignete Abbildung der Constraints für die Kollisionsbehandlung auf das gröbere Modell

Als problematisch erweisen sich bei der Stoffanimation die Kompressionskräfte der Federn, da bei gekrümmten Stoff und der Elimination von Partikeln stark komprimierte Federn in den Grobgittern entstehen können. Daher sollten Kompressionskräfte nur im feinsten Gitter berücksichtigt werden.

Mit Hilfe der GCA-Methode können die Matrizen für die Grobgitter automatisch allein aus den Prolongations- und Restriktionsoperatoren durch $\overline{\mathbf{A}} = \mathbf{RAP}$ gebildet werden. Würde man die Matrizen $\mathbf{R}$ und $\mathbf{P}$ direkt aufstellen wäre der Rechenaufwand allerdings beträchtlich. In [Gan03] wird ein effizientes Verfahren zur Bestimmung von $\overline{\mathbf{A}}$ vorgestellt.

Integration mit Mehrgitterverfahren

Ein konkretes Verfahren zur effizienten Integration mit einem Mehrgitterverfahren erhält man durch Kombination eines impliziten Integrationsverfahrens (z.B. Euler, siehe Abschnitt 5.1.2), einem der Schedules (z.B. V-Zyklus) und Operatoren zur Restriktion und Prolongation. Weiterhin kann entweder DCA oder GCA für die Konstruktion der Grobgitter verwendet werden. Aufgrund dieser vielfältigen Kombinationsmöglichkeiten ergibt sich eine hohe Flexibilität bei der Erstellung eines Mehrgitterverfahrens für die Stoffsimulation.

Zur Evaluierung der Performance wurden Zeitmessungen für die Simulation eines rechteckigen Stück Stoffs in unterschiedlichen Auflösungen (25^2, 49^2 und 99^2) durchgeführt. Das kaskadische Verfahren war dabei in allen Auflösungen langsamer als der V-Zyklus. Ebenso war die DCA immer schlechter als die GCA. Dies ist auf der einen Seite erstaunlich, da der Vorverarbeitungsschritt der GCA fehlt. Allerdings scheint die Bestimmung der Werte für die gröberen Gitter noch nicht

optimal zu sein. Das automatische GCA Verfahren liefert offenbar eine bessere Grobgitterapproximation und das Mehrgitterverfahren konvergiert damit schneller.

Verglichen mit dem CG-Verfahren ist der V-Zyklus mit GCA bei der Auflösung von 25^2 Partikeln noch langsamer, bei 49^2 Partikeln ist der V-Zyklus etwas schneller und bei 99^2 sogar fast doppelt so schnell. Weiterhin hat sich gezeigt, dass die Mehrgitterverfahren insbesondere von einer höheren Dämpfung profitieren. Ohne Dämpfung entstehen hochfrequente Schwingungen auf dem feinen Gitter, die auf den gröberen Gittern nicht berücksichtigt werden können. Daher werden im letzten Schritt auf dem feinsten Gitter zu viele Iterationen benötigt, so dass kein Geschwindigkeitsvorteil gegenüber dem CG-Verfahren erreicht wird.

5.4.3 Bewertung

Die Entwicklung eines Mehrgitterverfahrens zur Stoffanimation hat mehrere Fragen aufgeworfen, deren Beantwortung entscheiden wird, ob diese Klasse von Verfahren für die Animation von Bekleidung geeignet ist. Zunächst bereitet die Behandlung von Kollisionen große Probleme beim Übergang zwischen den Gittern, da dies und die dadurch entstehenden Constraints bei der GCA nicht erfasst werden. Zur Lösung in weiteren Untersuchungen könnten die Constraints entweder bei der GCA berücksichtigt oder aber bei der DCA integriert werden. Letzteres scheint der gangbarere Weg, da bei der erneuten Diskretisierung mehr Kontrolle über die entstehenden Gleichungssysteme gegeben ist.

Ein weiteres Problem hat sich in abschließenden Experimenten mit speziellen Vorkonditionierungsverfahren für das CG-Verfahren gezeigt, da dadurch die Geschwindigkeit des CG-Verfahrens deutlich gesteigert werden kann. Dabei geht der Vorteil der Mehrgitterverfahren stark zurück. An dieser Stelle müssen weitere Untersuchungen mit anderen Schedules und Operatoren zur Prolongation bzw. Restriktion zeigen, ob die Performance der Mehrgitterverfahren nicht ebenfalls verbessert werden kann.

5.5 Interaktive Optimierung der Passform

Dieser Abschnitt beschäftigt sich mit der Entwicklung eines interaktiven Verfahrens zur Optimierung der Passform von Kleidungsstücken an die Maße bzw. Wünsche des Kunden. Besonders wichtig sind vor allem Längenänderungen wie sie beispielsweise an Arm- und Beinschnittteilen oft durchgeführt werden. Aber auch das Erzeugen von Zwischengrößen wird häufig benötigt. Da ein individuelles Anpassen der Schnittteile mit CAD Software nur durch geschultes Fachpersonal möglich ist und außerdem sehr zeitaufwändig ist, wird in dieser Arbeit ein Algorithmus zur automatischen Anpassung der Schnittteile anhand weniger intuitiver Parameter vorgestellt.

Als Ausgangsbasis dienen Schnittteile eines Kleidungsstücks in einer Grundgröße. Des Weiteren liegen die Randkurven von Schnittteilen in veränderter Größe

bzw. mit angepassten Längen vor. Aus diesen Eingabedaten sollen neue Schnittteile mit individuellen Größen erzeugt werden.

Die bereits beschriebenen Methoden zur Simulation von Textilien erlauben das Durchführen einer virtuellen Anprobe. Dabei werden Kleidungsstücke in einer vom Kunden ausgewählten Größe vorpositioniert und simuliert. Passt dieses Kleidungsstück jedoch nicht, muss ein neuer Simulationsdurchlauf mit einer anderen Größe gestartet werden. Obwohl die in dieser Arbeit vorgestellten Algorithmen schon eine schnelle Simulation erlauben, dauert ein solcher Durchlauf dennoch mehrere Sekunden, da auch die gesamte Vorpositionierung und virtuelle Vernähung erneut durchgeführt werden. Diese Wartezeit ist unakzeptabel, vor allem, wenn der Kunde mehrere Änderungen hintereinander durchführen möchte. Ist die passende Größe nicht in der Datenbank, wäre sogar ein geschulter Schnittmacher oder Schneider nötig, um passende Schnittteile mit den individuellen Maßen konstruieren zu können. Damit ist diese Vorgehensweise nicht nur sehr zeitaufwändig, sondern auch entsprechend kostenintensiv.

Daher wird in diesem Kapitel ein Algorithmus zur Optimierung der Passform von bereits simulierten Kleidungsstücken vorgestellt. Das Verfahren arbeitet automatisch und setzt einfache User-Eingaben in komplexe Geometrieänderungen um. Die Kernidee ist es, zunächst Kleidungsstücke in verschiedenen Größen in einer Datenbank bereitzustellen. Hierzu werden die entsprechenden Schnittteile benötigt. Dann werden während der Laufzeit mit Hilfe einer linearen Gradierung die fehlenden Größen erzeugt. Bei einer solchen Gradierung werden korrespondierende Punkte auf zwei Schnittteilen linear interpoliert. Dazu wird ein automatisches Verfahren, das diese Korrespondenzen für die Vertices der Schnittteile berechnet, vorgestellt.

Dieser Ansatz lässt sich in allen gängigen Systemen zur Textilsimulation umsetzten. Den größten Vorteil erzielt man aber in einem interaktiven System, da sich die Passform in Echtzeit optimieren lässt. Weiterhin bietet der Ansatz ein intuitives User-Interface, so dass auch ungeschulte Anwender einfach individuelle Anpassungen an Kleidungsstücken vornehmen können.

5.5.1 Andere Verfahren

Ein spezieller Aspekt bei einer virtuellen Anprobe ist auch die Optimierung des Sitzes eines Kleidungsstücks. Hierzu wurden zwei interaktive Verfahren vorgestellt, die im Folgenden beschrieben werden. In [KSFS03] wird eine VR-Anwendung vorgestellt, die es erlaubt Schnittteile interaktiv auf einem Avatar zu platzieren. Dem Anwender wird hierzu ein Eingabegerät (Tracker) mit sechs Freiheitsgraden (6DOF) zur Verfügung gestellt, mit dem sich die Schnittteile bewegen und rotieren lassen. Nachdem alle Schnittteile platziert sind, werden sie mittels einer Simulation vernäht. Danach kann das Kleidungsstück von allen Seiten betrachtet werden und einzelne Teile können während der Simulation bewegt werden. Damit lässt sich zwar der Sitz des Kleidungsstückes optimieren, es ist allerdings nicht möglich, die Passform zu ändern.

Einen etwas anderen Ansatz verfolgen die Autoren von [IH02]. Auch hier ist es das Ziel einen Avatar einzukleiden und nachher den Sitz zu optimieren. Es wird aber kein 6DOF-Eingabegerät genutzt, sondern der Anwender kann Markierungen auf dem Avatar und dazu korrespondierende Markierungen auf den Schnittteilen einzeichnen. Der Algorithmus legt dann die Schnittteile entsprechend um den Körper, so dass die Markierungen übereinstimmen. Dies dauert nur wenige Sekunden. Danach kann der Stoff auf der Oberfläche noch verschoben werden. Es wird allerdings keine Stoffsimulation durchgeführt. Das Verfahren eignet sich nach Aussage der Autoren aber als Eingabe für eine solche Simulation.

Ein Verfahren zum interaktiven Optimieren der Form eines digitalen Kleidungsstückes wird in [EGB+02] vorgeschlagen. Die Basis bildet die Geometrie eines eingescannten Kleidungsstückes. Dieses wird an die Körperform mittels intelligentem Morphing angepasst. Das beschriebene Verfahren erlaubt sehr flexible Anpassungen und liefert eine gute Visualisierung des getragenen Kleidungsstückes. Allerdings wird keinerlei Simulation durchgeführt. Daher kann auch keine Aussage über die Passform getroffen werden.

Volino et al. [VCMT05] schlagen, unter anderem, ein Verfahren zum interaktiven Verändern der Form von Schnittteilen vor. Dabei wird die veränderte Form der Randkurve direkt auf das triangulierte Simulationsmesh übertragen. Verändert der Anwender die Position von Vertices auf der Randkurve, werden benachbarte Vertices im Mesh ebenfalls verschoben. Hierzu werden in jedem Vertex Gewichte gespeichert, die definieren wie stark ein Randpunkt den Vertex beeinflusst. Dadurch lassen sich die Koordinaten des Vertex relativ zum Rand berechnen. Das Verfahren erlaubt sehr schnelle Änderungen der Form mit sofortigem Update der Simulation. Allerdings wird immer ein Fachmann benötigt, der weiß wie die Randkurven zu editieren sind. Ein Verfahren zur automatischen Zuordnung neuer Randkurven wird nicht beschrieben. Zudem muss eine starke Integration zwischen 2D-CAD und 3D-Simulation bestehen.

Ein weiterer Update-Algorithmus, der Änderungen an 2D Schnittteilen auf eine 3D Simulation abbildet, wird in [LY05] vorgestellt. Auch hier berechnet der Simulator die neue Passform. Der Algorithmus erstellt ausgehend von einem bestehenden Simulationsmesh ein topologisch äquivalentes Mesh mit veränderter Form. Die Autoren schlagen hierfür ein Verfahren vor, das mit Hilfe einer 2D Simulation die neuen Vertex Koordinaten berechnet. Vertices am Rand des alten Meshes $\mathbf{M}_o$ werden mit dem Rand des neuen Meshes $\mathbf{M}_n$ mit einer Feder der Ruhelänge 0 verbunden. Die Vertices am Rand von $\mathbf{M}_n$ sind während der Simulation fixiert. Durch die Federn wird das alte Mesh wie gewünscht deformiert. Allerdings bereitet die Wahl der Federkonstanten gewisse Probleme, da bei falscher Wahl das Verfahren nicht konvergiert. Insgesamt ist der Prozess äußerst aufwendig, da ein einzelnes Update mehrere Sekunden benötigt. Die erzielten deformierten Meshes können zudem noch stark optimiert werden.

5.5.2 Überblick über das Verfahren

Als Eingabedaten dienen triangulierte Schnittteile eines Kleidungsstücks in einer Grundgröße als Ausgangsbasis. Des Weiteren liegen die Randkurven von Schnittteilen in veränderter Größe bzw. mit angepassten Längen vor. Beispielsweise werden ausgehend von einem Kleid in Größe 38 noch die Schnittteile für die Größen 34 und 42 verwendet. Die Grundgröße kann dann sowohl verkleinert als auch vergrößert werden.

Die Randkurven können in gängigen CAD-Systemen erstellt und exportiert werden. Meist wird zunächst mit Splines oder anderen parametrisierten Kurven modelliert. Grundlagen zu solchen Kurven finden sich in [ESK96]. Im Gegensatz zu Polygonzügen erleichtern solche Kurven das Modellieren geschwungener Linien. Beim Exportieren aus dem CAD-System werden diese dann in einen Polygonzug umgewandelt. Die Diskretisierung erfolgt dabei anhand frei wählbarer Fehlerschranken. In Abbildung 5.12 ist links ein solcher Polygonzug dargestellt. Man erkennt leicht die höhere Auflösung in Bereichen starker Krümmung, die für eine gute Approximation der ursprünglichen parametrisierten Kurve sorgt.

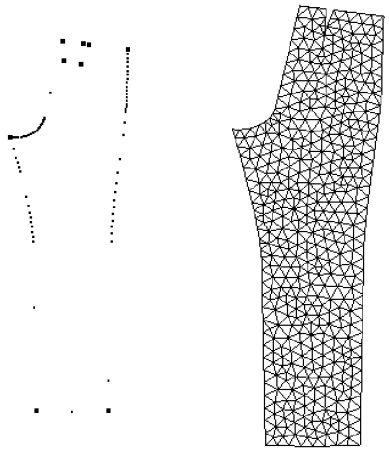


Abbildung 5.12: Links: Die Eckpunkte einer Randkurve eines Schnittteils. — Rechts: Das Mesh dessen Topologie übertragen werden soll.

Das Ziel des Verfahrens ist es die Randkurven genau so zu triangulieren wie das bereits gegebene Quellmesh. Dabei entstehen ein oder mehrere neue Meshes, die im folgenden Zielmeshes genannt werden. Durch Morphing dieser Meshes können dann beliebige Zwischengrößen erzeugt werden. Der Vorteil bei diesem Ansatz liegt darin, dass sich die Topologie der Meshes beim Anpassen der Größe nicht ändert. Aus diesem Grund kann der Simulator effizient den Übergang vom alten auf das neue Mesh berechnen, indem einfach die neuen Materialparameter vom Zielmesh übernommen werden.

Im folgenden Abschnitt 5.5.3 wird erklärt wie für eine gegebene Randkurve und Triangulierung ein kompatibler Rand erzeugt werden kann. Dieser wird benötigt, um danach die Triangulierung vom Quell- auf das Zielmesh zu übertragen 5.5.4. Danach wird erläutert wie die Materialparameter upgedated werden 5.5.5. Abschließend werden in Abschnitt 5.5.6 Ergebnisse zu diesem Verfahren präsentiert.

5.5.3 Kompatible Randkurven

Für das allgemeine Problem zu zwei gegeben Polygonen **S** und **T** eine bijektive Abbildung zwischen den Vertices zu finden, wurde in [SG92] eine Lösung vorgeschlagen. Dazu werden in **S** und **T** solange Vertices eingefügt bis beide Polygone die gleiche Anzahl von Vertices besitzen. Das Verfahren baut auf einem physikalischen Modell auf und stellt anhand dessen fest, an welchen Stellen die Vertices eingefügt werden müssen. Gleichzeitig wird damit auch das Korrespondenzproblem zwischen den Polygonen gelöst und man erhält kompatible Polygone.

Das Einfügen neuer Vertices würde aber den Rand des Quellmeshes verändern und somit Änderungen im Simulator erzwingen. Deshalb wird ein Verfahren benötigt, welches das Korrespondenzproblem löst, ohne neue Vertices einzufügen. Im Allgemeinen kann dies sehr schwierig sein. Im vorliegenden Fall sind sich die Randkurven aber sehr ähnlich. Deshalb genügt es die markanten Punkte der beiden Kurven zu finden und abzugleichen.

Im ersten Schritt wird aus dem Quellmesh der Rand extrahiert. Mit einer geeigneten Repräsentation des Meshes, die die Topologie zur Verfügung stellt, kann zunächst ein Vertex am Rand bestimmt werden. Von diesem ausgehend wird dann das Mesh einmal umlaufen, wobei man eine Liste mit Punkten am Rand erhält. Die zweite Randkurve liegt bereits als Polygon vor, muss allerdings eventuell per Hand richtig herum orientiert werden. In den meisten Fällen sind Schnittteile in verschiedenen Größen jedoch immer gleich orientiert, da dies vom Konstruktionsprozess vorgegeben wird.

Die Randkurven von Schnittteilen sind bis auf Ecken geometrisch glatt, d.h. unter anderem zweimal stetig differenzierbar. Diese Ecken lassen sich auch in der diskretisierten Form anhand der Winkel zwischen angrenzenden Kanten leicht erkennen und dienen als markante Punkte. Markante Punkte auf dem Rand des bereits triangulierten Schnittteils zu finden ist etwas schwieriger, da aufgrund der gröberen Diskretisierung einige Punkte fälschlicherweise als markant erkannt werden können. Aber auch hier werden Vertices, deren angrenzende Kanten einen Winkel unterhalb eines bestimmten Schwellwertes haben, als markant gekennzeichnet. Die Anzahl der markanten Punkte ist daher aber für **S** höher als für **T**.

Die Korrespondenz wird zunächst zwischen den markanten Punkten hergestellt. Dazu werden die Bounding Boxen von **S** und **T** berechnet. Dann wird **T** so verschoben, dass die Mittelpunkte der Boxen übereinander liegen. Zwei markante Punkte korrespondieren, wenn sie nahe beieinander liegen und einen ähnlichen Öffnungswinkel haben. Letztere Bedingung kann aufgestellt werden, da die Rand-

kurven gleich orientiert sind. Mit ihr werden die fälschlicherweise erkannten Punkte herausgefiltert. Sollte diese beschriebene Heuristik fehlschlagen, können die markanten Punkte jedoch auch manuell bestimmt und abgeglichen werden.

Nachdem die markanten Punkte korrespondieren, können $\mathbf{S}$ und $\mathbf{T}$ in $2n$ offene Polygone $\mathbf{S}_i$, $\mathbf{T}_i$ mit $1 < i < n$ unterteilt werden, wobei n die Anzahl der markanten Punkte darstellt. Jetzt wird ein neues offenes Polygon $\hat{\mathbf{T}}_i$ mit der gleichen Anzahl Vertices wie $\mathbf{S}_i$ erzeugt. Bezeichne $\mathbf{t}_k$ den k-ten Vertex von $\hat{\mathbf{T}}_i$. Dieser wird an den Punkt auf $\mathbf{T}_i$ gesetzt, der die gleiche normalisierte Bogenlänge wie der k-te Vertex von $\mathbf{S}_i$ besitzt. Zur Bestimmung der Bogenlänge wird einer der markanten Punkte als Startpunkt definiert. Die normalisierte Bogenlänge eines Vertex $\mathbf{p}_k$ ergibt sich dann als

$$b(\mathbf{p}_k) = \frac{\sum_1^{k-1} |\mathbf{p}_i - \mathbf{p}_{i+1}|}{\sum_1^{n-1} |\mathbf{p}_i - \mathbf{p}_{i+1}|} \tag{5.38}$$

mit n als Anzahl von Vertices und $\mathbf{p}_1$ als Startpunkt.

Zum Auffinden der Position von $\mathbf{t}_k$ werden zunächst die beiden aufeinander folgenden Vertices $\mathbf{q}, \mathbf{r} \in \mathbf{T}_i$ bestimmt, für die gilt $b(\mathbf{q}) < b(\mathbf{p}_k)$ und $b(\mathbf{r}) > b(\mathbf{p}_k)$. Jetzt muss nur noch entsprechend interpoliert werden, um $\mathbf{t}_k$ zu erhalten:

$$\alpha = \frac{b(\mathbf{p}_k) - b(\mathbf{q})}{b(\mathbf{r}) - b(\mathbf{q})} \tag{5.39}$$

$$\mathbf{t}_k = (1-\alpha)\mathbf{q} + \alpha\mathbf{r} \tag{5.40}$$

Fasst man die $\hat{\mathbf{T}}_i$ zu einem Polygon zusammen, ist dieses zu $\mathbf{S}$ kompatibel. Ein Beispiel zu diesem Algorithmus ist in Abbildung 5.13 dargestellt.

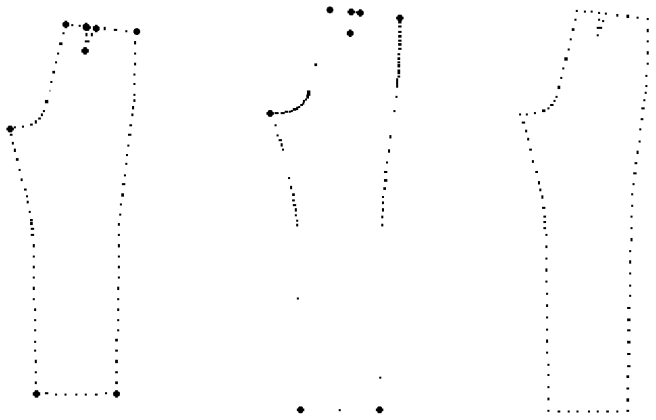


Abbildung 5.13: Links: Die Randkurve eines Quellmeshes mit erkannten markanten Punkte. — Mitte: Der Rand eines größeren Schnittteils mit markanten Punkten. — Rechts: Die mit diesem Verfahren erzeugte kompatible Randkurve des größeren Schnittteils .

5.5.4 Triangulierung der Zielgeometrie

Kompatible Triangulierung

Im Folgenden werden die Begriffe Triangulierung und kompatible Triangulierung definiert, die Notation ist [SG01] entlehnt. Diese Definitionen beschränken sich auf zwei Dimensionen. In dieser Arbeit ist mit einer Triangulierung auch immer eine zwei dimensionale Triangulierung gemeint (im Gegensatz zu einer Unterteilung eines Objekts des $\mathbb{R}^3$ in Tetraeder).

Definition 5.1. Triangulierung *Eine Triangulierung* $\mathbf{T} = \mathbf{T}(\mathbf{V}, \mathbf{E})$ *ist eine Menge von Vertices* $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ *mit* $\mathbf{v}_i \in \mathbb{R}^2$ *und Menge von Kanten* $\mathbf{E}$*, die eine Untermenge der ungeordneten Paaren* $\{\mathbf{v}_i, \mathbf{v}_j\}$ *von Vertices mit* $i \neq j$ *ist, für die gilt*

- *Die einzigen Schnittpunkte von Kanten treten an den gemeinsamen Vertices auf.*
- *Alle begrenzten Gebiete (Polygone) haben genau drei Kanten.*

Intuitiv kann man sagen, dass zwei Triangulierungen *kompatibel* (oder auch *isomorph*) sind, wenn sie topologisch äquivalent sind. Etwas formaler kann man folgende Definition angeben:

Definition 5.2. Kompatible Triangulierung *Zwei Triangulierungen* $\mathbf{T}_1$ *und* $\mathbf{T}_2$ *sind kompatibel (oder isomorph), wenn folgende Bedingungen gelten*

1. *Es existiert eine Eins-zu-Eins Korrespondenz zwischen den Vertices und den Kanten, so dass korrespondierende Kanten mit korrespondierenden Vertices verbunden sind.*
2. *Alle korrespondierenden Dreiecke haben die gleiche Orientierung.*

Erstellen einer kompatiblen Triangulierung

Seien zwei Polygone (bzw. Randkurven) $\mathbf{P}_1$ und $\mathbf{P}_2$ mit gleicher Anzahl n an Vertices gegeben. Ein einzelnes Polygon kann immer nur unter der Verwendung seiner Vertices trianguliert werden [Cha91]. Möchte man aber $\mathbf{P}_1$ und $\mathbf{P}_2$ ebenfalls nur unter Verwendung der gegebenen Vertices *kompatibel triangulieren*, ist dies nicht immer möglich. Ein Beispiel hierfür ist in Abbildung 5.14 links dargestellt. In [ASS93] wird gezeigt, dass $\mathbf{P}_1$ und $\mathbf{P}_2$ unter Verwendung von höchstens $O(n^2)$ zusätzlichen Vertices, so genannten *Steinerpunkten*, kompatibel trianguliert werden können.

Hierzu werden die Polygone zunächst einzeln trianguliert. Danach werden beide auf ein n-eckiges Polygon abgebildet, wodurch eine Überlagerung der beiden Triangulierungen entsteht. An den resultierenden Schnittpunkten, von denen es höchstens $O(n^2)$ geben kann, werden Steinerpunkte eingefügt. Dabei entstehen einige Facetten mit mehr als drei Vertices. Diese müssen noch ausgehend von einem ihrer Vertices trianguliert werden, wobei keine zusätzlichen Vertices eingefügt

werden. Bildet man die entstandene Triangulierung zurück auf $\mathbf{P}_1$ und $\mathbf{P}_2$ ab, erhält man eine kompatible Triangulierung mit maximal $O(n^2)$ zusätzlichen Steinerpunkten. Der Nachteil dieser Methode ist, dass sehr spitze Dreiecke entstehen können. Dies führt zu numerischen Problemen, wie auch in Abschnitt 5.2.1 beschrieben.

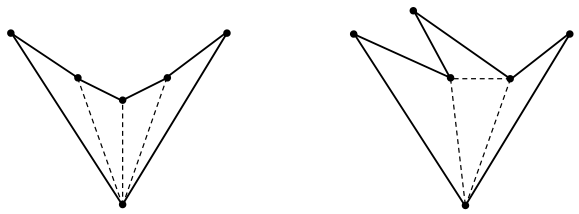


Abbildung 5.14: Zwei Polygone, die nicht kompatibel trianguliert werden können.

Die spitzen Dreiecke kann man umgehen, indem man die Methode, die in [ACOL00] beschrieben ist, verwendet. Zunächst wird für $\mathbf{P}_1$ und $\mathbf{P}_2$ jeweils die Delaunay-Triangulierung berechnet. Diese optimiert die Qualität der entstehenden Triangulierung, da der minimale Winkel in den Dreiecken maximiert wird. Dadurch entstehen weniger spitze Dreiecke. Überlagert man diese Triangulierungen können natürlich wieder spitze Dreiecke entstehen. Deshalb werden die beiden nach der Überlagerung entstandenen Triangulierungen nochmals simultan optimiert. Dabei werden innere Vertices bewegt und Kanten umgedreht [1], so dass sich die minimalen Winkel in beiden Triangulierungen verbessern. Das Verfahren wird *Compatible Mesh Smoothing* genannt. Zusätzlich werden lange Kanten geteilt, um eine weitere Verbesserung der Qualität der Triangulierung zu erhalten.

Das Verfahren liefert Dreiecksnetze von hoher Qualität. Allerdings müssen beide Polygone trianguliert werden und bei der Überlagerung der beiden Netze entstehen zusätzliche Vertices. Überlagert man mehr als zwei Netze verschärft sich dieses Problem und der Algorithmus muss diese alle simultan bearbeiten. Für die vorliegende Anwendung ist es besser, wenn die bestehende Triangulierung des Quellmeshes beibehalten und diese auf das Zielmesh übertragen wird. Ein Algorithmus, der dies leistet wird im folgenden Kapitel vorgestellt.

Transfer einer Triangulierung

Aus dem gegebenen Quellmesh $\mathbf{S}$ und einer oder mehreren kompatiblen Randkurven $\mathbf{t}_i$ werden mehrere kompatible Triangulierungen $\mathbf{T}_i$ erstellt. Der folgende Algorithmus arbeitet für jedes $\mathbf{t}_i$ gleich und jeder Transfer ist unabhängig von den anderen, daher beschränkt sich die Beschreibung auf ein $\mathbf{t}$. Da die Vertices am Rand bereits korrespondieren und die Topologie durch $\mathbf{S}$ festgelegt ist, müssen nur noch die Koordinaten der inneren Vertices von $\mathbf{T}$ bestimmt werden.

[1] Dazu betrachte man eine Kante und die beiden angrenzenden Dreiecke. Beim Umdrehen wird die Kante zunächst entfernt und dann mit den beiden anderen Vertices verbunden.

Der Algorithmus überträgt hierzu die Koordinaten der Dreiecke vom Rand ins Innere. Man startet mit einem Dreieck $\mathbf{d} \in \mathbf{S}$, das eine Kante auf dem Rand von $\mathbf{S}$ besitzt. Jetzt läuft man an den Kanten des Randes entlang und fügt nacheinander die Dreiecke hinzu. Danach werden ausgehend von den Vertices des Randes solange Dreiecke hinzugefügt bis ein Band von Dreiecken entsteht, so dass alle Dreiecke, die adjazent zum Rand sind, übertragen wurden. Dieses Band definiert einen neuen Rand, von dem aus das nächste Band gebildet wird. Die Rekursion stoppt, sobald alle Dreiecke übertragen wurden.

Die Reihenfolge bei der Abarbeitung wurde so gewählt, damit immer die Koordinaten zweier Vertices eines Dreiecks schon übertragen worden sind. Auf diese Weise kann der fehlende dritte Vertex gut berechnet werden. Des Weiteren werden zuerst Dreiecke mit einer Kante am Rand bearbeitet, damit man gute neue Koordinaten für den Vertex erhält. Die Triangulierung wird demnach vom Rand nach Innen hin übertragen. Folgender Algorithmus führt diesen Schritt durch:

1. Füge in $\mathbf{V}_{neu}$ alle Vertices $\mathbf{v}_i$ von $\mathbf{S}$ ein, die am Rand liegen und markieren die $\mathbf{v}_i$ als besucht.

2. Nehme jeweils alle aufeinander folgenden $\mathbf{v}_i, \mathbf{v}_{i+1} \in \mathbf{V}_{neu}$ und suche alle Dreiecke $\mathbf{t}$, die von $\mathbf{v}_i, \mathbf{v}_{i+1}$ gebildet werden. Füge das $\mathbf{t}_i$, dass noch nicht markiert ist, in $\mathbf{T}_{neu}$ ein und markiere es. Der dritte Vertex $\mathbf{v}_t$ von $\mathbf{t}_i$ wird in die Liste $\mathbf{V}_{neu}$ eingefügt, sofern er noch nicht markiert ist.

3. Ermittle alle nicht markierten Dreiecken $\mathbf{t}_i$, die adjazent zu $\mathbf{v}_i \in \mathbf{V}_{neu}$ sind. Jetzt werden drei Fälle unterschieden.

   (a) Man findet ein Dreieck. Dieses wird markiert und ein $\mathbf{T}_{neu}$ eingefügt.

   (b) Man findet zwei Dreiecke. Diese haben einen gemeinsamen Vertex $\mathbf{v}_i$, der markiert wird und zu $\mathbf{V}_{neu}$ hinzugefügt wird. Die beiden Dreiecke werden zu $\mathbf{T}_{neu}$ hinzugefügt.

   (c) Man findet mindestens drei Dreiecke. Man fügt die Dreiecke so in $\mathbf{T}_{neu}$ ein, dass beim Einfügen immer schon zwei Vertices in $\mathbf{V}_{neu}$ sind. Der fehlende Vertex wird in $\mathbf{V}_{neu}$ eingefügt.

4. Wiederhole die Schritte 2 und 3 bis alle Dreiecke markiert sind. Bei diesen Wiederholungen startet man bei den Vertices aus $\mathbf{V}_{neu}$, die in dieser Runde gerade eingefügt worden sind.

Wie lassen sich die neuen Koordinaten der Vertices $\mathbf{v}_{neu}$ berechnen? Gegeben ist ja nur eine Kante des neuen Dreiecks. Diese hat eine andere Länge als die korrespondierende alte Kante. Daher wird von $\mathbf{v}_{alt}$ die relative Position auf der Kante und der Abstand zur Kante bestimmt. Dann werden diese Werte für $\mathbf{v}_{neu}$ bestimmt und somit auch seine neuen Koordinaten. Der Abstand zur Kante kann noch skaliert werden, um die Dreiecke im Zielmesh kleiner bzw. größer als im Quellmesh

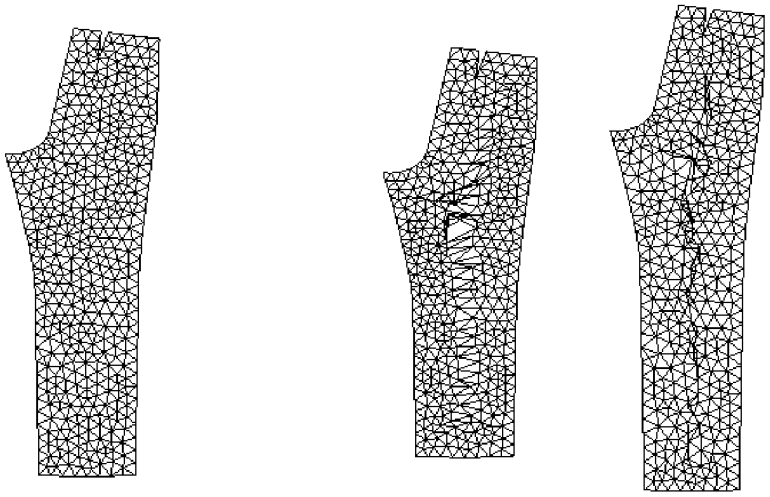


Abbildung 5.15: Transfer einer Triangulierung vom linken Mesh auf die beiden rechten Meshes. Dabei entstehen große gestreckte Dreiecke (Mitte) oder Überschneidungen (Rechts).

zu machen. Dies ist wichtig, wenn sich die Meshes in ihrer Größe sehr stark unterscheiden.

Abbildung 5.15 zeigt links ein Quellmesh und rechts zwei Zielmeshes, die mit dem oben beschrieben Algorithmus erzeugt wurden. Das eine Mesh ist kleiner und das andere größer als das Quellmesh. Man erkennt deutlich, dass beim kleineren in der Mitte viele sehr große, spitze Dreiecke liegen. Beim Übertragen war am Ende einfach noch Platz übrig. Andererseits sieht man beim großen Mesh, dass es zu Überschneidungen kommt, da die Dreiecke zu schnell den gesamten Platz aufgebraucht haben. Die beiden Probleme können auch gleichzeitig auftreten, je nach Form der Randkurve des Zielmeshes. Die bisher berechneten Koordinaten dienen daher als Eingabe für einen weiteren Algorithmus, der eine Glättung des Meshes vornimmt.

Es wurden bereits zwei Methoden zur Mesh-Optimierung vorgestellt 5.2.1. Der in [FJP95] vorgestellte Ansatz zur lokalen Mesh-Optimierung kann allerdings an dieser Stelle nicht eingesetzt werden, da die aktuelle Triangulierung noch nicht gültig ist (einige Dreiecke sind falsch orientiert). Deshalb wird die einfachere, aber auch langsamer konvergierende Laplace-Filterung durchgeführt. Der Vorteil bei dieser Filtermethode liegt darin, dass die initialen Überschneidungen aufgelöst werden. In der Praxis genügen für gewöhnlich aber wenige Iterationen, um das Zielmesh zu glätten und insgesamt die Qualität aller Dreiecke zu erhöhen.

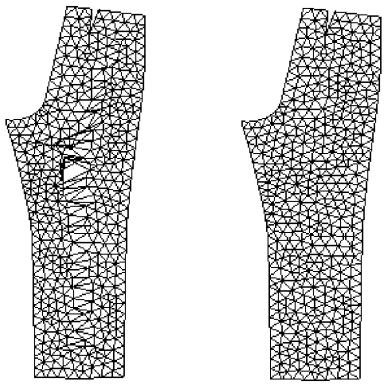


Abbildung 5.16: Glättung und Bereinigung eines Meshes. Links: Das ursprüngliche Mesh. — Rechts: Das Ergebnis einer Laplace-Filterung.

5.5.5 Updaten der Materialparameter

Nach dem Transfer der Triangulierung liegen Quell- und Zielmesh kompatibel trianguliert vor. Beim interaktiven Optimieren der Passform werden aus diesen unterschiedlichen Schnittteilen neue Zwischengrößen erzeugt. Hierfür muss geeignet zwischen den korrespondierenden Vertices gemorpht werden. Im Allgemeinen erfordert dieses *Vertex Path* Problem genauere Betrachtung [Ale02b]. Da sich im vorliegenden Fall aber Quell- und Zielmesh sehr ähnlich sind, genügt ein einfaches lineares Interpolieren der Koordinaten der Vertices. Aus korrespondierenden Vertices $\mathbf{v}_s$ und $\mathbf{v}_t$ wird mit

$$\mathbf{v}' = \alpha\mathbf{v}_t + (1-\alpha)\mathbf{v}_s \quad \text{mit} \quad \alpha \in [0,1] \tag{5.41}$$

die neue Koordinate berechnet. Führt man dies für alle Vertices durch erhält man ein Schnittteil in einer neuen Größe. Liegen mehrere Zielmeshes vor, kann davon ausgegangen werden, dass sie eine geordnete Reihe von Größen bilden. Daher wird in diesem Fall zwischen zwei aufeinander folgenden Größen interpoliert.

Bisher wurden alle Berechnung im $\mathbb{R}^2$ durchgeführt. Daher muss die veränderte Schnittteilgeometrie noch an den Simulator übertragen werden, damit sich die dreidimensionale Form des Kleidungsstückes ändern kann. Der hier gewählte Ansatz sieht vor, dass die Koordinaten der Partikel indirekt geändert werden. Hierzu werden nur die Materialparameter der Partikel verändert, der Rest wird vom Simulator übernommen.

Beispielsweise kann man die Ruhelänge von Federn im System anhand des gemorphten Schnittteils neu berechnen. Mit diesen neuen Materialparametern kann der Simulator die neuen Koordinaten der Partikel errechnen. Damit keine sprunghaften Änderungen auftreten, werden die Materialparameter schrittweise über mehrere Zeitschritte von alt nach neu geblendet.

Problematisch beim Updaten der Materialparameter ist, dass das simulierte Kleidungsstück aus einem einzigen Mesh besteht, dass durch Vernähen der Schnittteile entstanden ist. Bei diesem Prozess sind Partikel am Rand miteinander verschmolzen worden. Dies verkompliziert beispielsweise die Berechnung der Biegefedern, da sich diese über zwei unterschiedliche Schnittteile erstrecken. Auch die Information, welche Masse die beiden einzelnen Partikel hatten, geht verloren. Aus diesen Gründen werden in den verschmolzenen Partikeln einige Zusatzinformationen gespeichert, aus denen sich alle nötigen Materialparameter zur Laufzeit effizient berechnen lassen.

Die Behandlung der Texturkoordinaten verlangt besondere Erwähnung, da diese für die visuelle Qualität entscheidend sind. Werden die Texturkoordinaten beim Optimieren der Passform nicht korrekt angepasst, stimmt der Rapport nicht mehr und es entsteht der Eindruck die Textur würde über das Schnittteil wandern. Ein in der Praxis äußerst störender Effekt. Deshalb muss der Rapport von Ziel- und Quellmesh ebenfalls mit Formel 5.41 interpoliert werden.

5.5.6 Ergebnisse

Das beschriebene Verfahren erlaubt das interaktive Optimieren der Passform verschiedenster Kleidungsstücke. Die Größenänderung, die dabei durchgeführt wird, entspricht einer *linearen Gradierung*, wie sie bei Schnittkonstruktion häufig verwendet wird, um ausgehend von eine Basisgröße andere Größen mit wenig Aufwand zu Erzeugen. Bei der linearen Gradierung werden mehrere ausgewählte Punkte der Randkurve entlang einer Geraden entsprechend der gewünschten Größenänderung nach außen verschoben. Vergleiche von Schnittteilen, die mit dem oben beschriebenen Verfahren erzeugt wurden und welchen, die mit spezieller CAD Software aus dem Textilbereich konstruiert wurden, haben ergeben, dass sich die Randkurven bis auf kleinere Ungenauigkeiten, die auf die Diskretisierung zurückzuführen sind, decken. Die Vergleiche konnten durchgeführt werden, da der Export der Randkurve von neu erzeugten Schnittteilen einfach möglich ist.

Die Interaktivität des Verfahrens wurde an mehreren Beispielen überprüft. In allen Beispielen lag die Rechenzeit für die Änderung der Größe weit unter den Rechenzeiten für die eigentliche Simulation, die wiederum in Echtzeit abläuft. Die Anforderung an ein interaktives System zur Optimierung der Passform ist damit erfüllt. Exemplarisch sind in Abbildung 5.5.6 die Längenänderungen von Hemdsärmeln und in Abbildung und 5.18 die Größenänderung eines Kleides dargestellt. Die Längenänderung der Ärmel wurde mit Hilfe von zwei unterschiedlich langen Schnittteilen für die Ärmel durchgeführt. Die Länge im mittleren Bild entspricht der Ausgangsgröße. Weiterhin wurde noch ein kurzer Ärmel verwendet. Auf der linken Seite ist eine Zwischengröße dargestellt und rechts die Extrapolation auf einen sehr langen Ärmel. Dazu beschränkt man α aus Formel 5.41 nicht mehr auf Werte zwischen Null und Eins.

Eine solche Extrapolation kann allerdings nur bis zu einem gewissen Grad durchgeführt werden, da die Dreiecke immer spitzer und größer werden, je länger

Abbildung 5.17: Interaktives Optimieren der Passform: Die Ärmel wurden in der Länge verändert.

der Ärmel ist. Da das Simulationsverfahren sehr stabil arbeitet, treten aber weniger numerische Probleme auf, sondern die Qualität der Visualisierung ist nicht mehr ausreichend, um eine vernünftige Aussage über die Passform zur erhalten. Die einzelnen Dreiecke sind so sehr gestreckt, dass sich der Faltenwurf nicht mehr korrekt ausprägen kann.

Beim zweiten Beispiel werden beim Optimieren der Passform alle Schnittteile des Kleides verändert 5.18. Man startet mit der Größe 38 (unten im Bild) und kann das Kleid sowohl enger als auch weiter machen.

Zur Eingabe kann ein einfacher Slider verwendet werden, der es entweder ermöglicht die Größen kontinuierlich zu verändern oder aber ein Raster anbietet um mehrere Konfektionsgrößen nacheinander zu testen. Die erste Variante ist für individuelle Maßbekleidung wichtig, die zweite für das Anprobieren von Konfektionsgrößen.

Betrachtet man die verschiedenen Größen in 3D genauer, so stellt man fest, dass das verwendete Kleid in keiner der möglichen Größen richtig passt. Es ist entweder an der Hüfte zu weit oder im Bereich des Brustkorbs zu eng. Es sind in diesem Fall demnach sehr individuelle Änderungen an der Geometrie der Schnittteile nötig. Je nach Anwendungsszenario kann jetzt ein ausgebildeter Schnittmacher diese Änderungen vornehmen und danach die neue Passform visualisieren

Abbildung 5.18: Optimieren der Passform: Ausgehend vom Kleid in Größe 38 (Mitte) wurden interaktiv ein engeres Kleid in Größe 36 (links) und ein weites Kleid in Größe 40 (rechts) erzeugt.

lassen. Steht kein Fachmann zur Verfügung, weiß die Kundin zumindest, dass sie ein anderes Modell wählen sollte.

Bewertung

Besonders am letzten Beispiel wird eine interessante Möglichkeit zur Erweiterung des Verfahrens offenbar. Anstatt nur zwei verschiedene Geometrien zu verwenden, könnte man zwischen mehreren Schnittteilen morphen. Zusätzlich zu einem langen und normalen Ärmel könnte noch ein weiter Ärmel hinzugefügt werden. Dann wird die neue Geometrie aus drei Schnittteilen erzeugt und man erhält z.B. einen weiten langen Ärmel. Extrapoliert man wieder wäre auch ein dünner langer Ärmel möglich.

Ein weiterer Schritt wäre die Anwendung im Virtual Prototyping. Bei direkter Anbindung eines CAD-Systems zur Schnittkonstruktion, könnten zuerst Schnitte konstruiert und simuliert werden. Danach würde der Schnittmacher sein Werk überprüfen und könnte gegebenenfalls Änderungen an den Randkurven vornehmen. Mit dem oben beschriebenen Verfahren kann das virtuelle Kleidungsstück dann sehr schnell an die neue Form angepasst werden. Es entsteht ein Workflow ohne Wartezeiten. Somit kann der Schnittmacher in kürzester Zeit verschiedene Varianten durchprobieren und den besten Schnitt schließlich verwenden.

Ein Nachteil des Verfahrens ist, dass sich bei großen lokalen Änderungen an einem Schnittteil die Qualität des Meshes sehr verschlechtern kann. In einem solchen Fall wäre eine Mesh-Optimierung nötig, die zur Laufzeit das Simulations-

gitter anpasst und Dreiecke schlechter Qualität entsprechend unterteilt oder die Koordinaten der Vertices anpasst. Ein weiteres Problem stellt die Diskretisierung der Randkurve dar. Wird bei einer Änderung der Randkurve ein zuvor wenig gekrümmtes Teilstück so modifiziert, dass es eine hohe Krümmung ausweist, wird an dieser Stelle ein feiner aufgelöstes Mesh benötigt. Da aber die Diskretisierung durch den Rand des Quellmesh vorgegeben ist, kann darauf nicht reagiert werden. Auch hier wird ein Remeshing benötigt. Alternativ könnte ein Multi-Resolution Ansatz weiterhelfen, bei dem das Quellmesh in mehreren Auflösungen vorliegt und adaptiv verfeinert werden kann. Wird dann bei einer Änderung der Randkurve an einer anderen Stelle des Schnittteils eine hohe Auflösung benötigt, könnte das Mesh an dieser Stelle adaptiv verfeinert werden und erst danach auf die Randkurve übertragen werden.

5.6 Zusammenfassung

In diesem Kapitel wird die effiziente numerische Simulation von textilen Materialien beleuchtet. Hierzu wird zunächst in das Thema eingeführt und es werden Partikelsysteme vorgestellt, die sich im Bereich der Computergrafik zur Modellierung deformierbarer Objekte als sehr geeignet herausgestellt haben, da sie einen guten Kompromiss zwischen Genauigkeit und Geschwindigkeit darstellen. Das mechanische Verhalten bei der Interaktion zwischen den Partikeln wird durch Kräfte beschrieben, die unter anderem von den relativen Positionen der Partikel zueinander abhängen. Die Wahl der Integrationsmethode ist entscheidend für die Performance und Stabilität einer numerischen Simulation. Deshalb werden explizite und implizite Verfahren und deren Anwendung bei der Stoffsimulation vorgestellt.

Da man zur Animation eines Stück Stoffs mit Hilfe eines Partikelsystems eine geeignete Diskretisierung seiner Fläche benötigt, werden verschiedene Ansätze hierfür vorgestellt. Für die Animation von Bekleidung sind Dreiecksnetze vorteilhaft, da sich damit die Vernähung der einzelnen Schnittteile sehr gut realisieren lässt. Im Gegensatz dazu bereiten Rechtecksnetze dabei große Probleme.

Im Weiteren wird ein Verfahren zur robusten Modellierung interner Kräfte vorgestellt. Das Verfahren ist äußerst stabil und erlaubt große Zeitschritte. Das hier vorgestellte Modell nimmt einige Vereinfachungen vor und approximiert das Dehn- und Biegeverhalten von Stoff. Die Anisotropie und die Scherung werden vernachlässigt. Allerdings kann das hier vorgestellte Modell auch diese Eigenschaften nachbilden, indem Rechtecksnetze anstelle von Dreiecksnetzen verwendet werden.

Es wird eine explizite Methode zur *stabilen numerischen Integration* vorgestellt, die sich besonders gut zur interaktiven Simulation höher aufgelöster Meshes eignet, wie sie z.B. für die Erstellung der Beispiele in dieser Arbeit verwendet werden. Das Verfahren ist äußerst stabil und kann sich sogar von kurzfristigen Störungen erholen. Dies ist insbesondere in einem interaktiven System von Bedeutung, da der Anwender direkt mit dem Material interagieren kann, wodurch sehr hohe Kräfte entstehen können.

Verwendet man implizite Verfahren für die numerische Integration der Bewegungsgleichung, wird ein wesentlicher Teil des Rechenaufwands bei der Animation von Bekleidung hierfür benötigt. Mehrgitterverfahren erlauben in vielen Bereichen eine Beschleunigung beim Lösen der entstehenden Gleichungssysteme. Daher wird ein Mehrgitterverfahren zur Animation von Stoff vorgestellt. Allerdings bereitet die Behandlung von Kollisionen Probleme beim Übergang zwischen den Gittern und führt dazu, dass die Performance im Vergleich zum CG-Verfahren nicht wesentlich gesteigert werden kann.

Der letzte Teil dieses Kapitels beschäftigt sich mit der Entwicklung eines neuen interaktiven Verfahrens zur *Optimierung der Passform* von Kleidungsstücken an die Maße bzw. Wünsche des Kunden. Da ein individuelles Anpassen der Schnittteile mit CAD Software nur durch geschultes Fachpersonal möglich ist und außerdem sehr zeitaufwändig ist, wird ein Algorithmus zur automatischen Anpassung der Schnittteile anhand weniger intuitiver Parameter vorgestellt. Als Ausgangsbasis dienen Schnittteile eines Kleidungsstücks in einer Grundgröße. Des Weiteren liegen die Randkurven von Schnittteilen in veränderter Größe bzw. mit angepassten Längen vor. Aus diesen Daten werden neue Schnittteile mit individuellen Größen erzeugt.

Die Kernidee des vorgestellten Verfahrens ist es, zur Laufzeit mit Hilfe einer linearen Gradierung die fehlenden Größen zu erzeugen. Bei einer solchen Gradierung werden korrespondierende Punkte auf zwei Schnittteilen linear interpoliert. Dazu wird eine Methode, die diese Korrespondenzen für die Vertices der Schnittteile berechnet, vorgestellt.

Dieser Ansatz lässt sich in allen gängigen Systemen zur Textilsimulation umsetzen. Den größten Vorteil erzielt man aber in einem interaktiven System, da sich die Passform in Echtzeit optimieren lässt. Weiterhin bietet der Ansatz ein intuitives User-Interface, so dass auch ungeschulte Anwender individuelle Anpassungen an Kleidungsstücken vornehmen können.

Kapitel 6

Echtzeit Visualisierung

Bei der Echtzeit Visualisierung von textilen Materialien bzw. ganzer Kleidungsstücke ergeben sich eine Reihe von speziellen Herausforderungen aufgrund der Eigenschaften und der Verarbeitung dieses Materials.

Da Kleidungsstücke aus miteinander vernähten Schnittteilen bestehen, tragen diese Nähte oft in hohem Maße zum Gesamteindruck bei der Betrachtung eines Kleidungsstücks bei. Daher ist es wichtig diese bei der Visualisierung darzustellen, andernfalls kann bei einer virtuellen Anprobe das Aussehen nicht vernünftig beurteilt werden.

Textiles Material kann leicht gebogen werden. Daher ergeben sich bei der Simulation Falten und Schichten, die sehr dicht übereinander liegen. Damit dies bei der Visualisierung vom Anwender erkannt werden kann, ist es wichtig die Selbstabschattung zu berechnen, weil ansonsten keine Differenzierung womöglich gleich gefärbter Stoffe mehr möglich ist. Durch den Schattenwurf wird zudem der Tiefeneindruck erheblich verbessert, was das Bild natürlicher erscheinen lässt.

Weiterhin erfordert ein echtzeitfähiges Simulations- und Visualisierungssystem eine spezielle Architektur des Szenegraphen um dynamische Szenen mit den hier beschriebenen Anforderungen an die Darstellungsqualität effizient rendern zu können.

6.1 Visualisierung des Simulationszustands

Während der Simulation ist es für eine virtuelle Anprobe wichtig, dass die Passform beurteilt werden kann. Als Hilfestellung für diese Beurteilung wäre es vorteilhaft, wenn sich neben der photorealistischen Darstellung der Zustand des Stoffs, wie etwa der Abstand des Kleidungsstücks zum Körper oder im Stoff auftretende Spannungen visualisieren lassen. Mit diesen Informationen wird es leichter die passende Größe zu finden oder auch Fehler bei der Konstruktion von Kleidung zu entdecken. Wichtige Informationen die visualisiert werden sollen sind:

- Spannung

- Ausdehnung (in %)
- Distanz zum Körper

Da der Simulator auf der Basis eines Partikelsystems arbeitet, sind alle benötigten internen Zustände in jedem Partikel abrufbar. So wird beispielsweise während der Kollisionserkennung die Distanz zum Körper für jeden Partikel berechnet und für die Bestimmung der Kräfte, die auf einen Partikel wirken, die Dehnung des Stoffes ausgewertet. Auch die Biegekräfte, die wirken, lassen sich speichern.

Abbildung 6.1: Visualisierung der Spannung zur Passformkontrolle. Links: Der Rock in normaler Visualisierung. — Mitte: Der Rock in Größe 36. — Rechts: Der Rock in Größe 40. Man erkennt, dass weniger Spannungen im Material vorhanden sind.

Um diese Informationen darzustellen kann ein Color-Mapping verwendet werden. Hierzu wird ein Farbverlauf definiert und den auftretenden Werten in den Partikeln ein entsprechender Farbwert zugewiesen. Beim Rendern wird dann zwischen den Farbwerten interpoliert und man erhält optisch ansprechende Farbverläufe, die zudem in etwa dem Verlauf der Werte in den Partikeln entsprechenden. Einfache Farbverläufe lassen sich mit Vertexcolors darstellen, die linear in den Dreiecken interpoliert werden. Für komplexere Farbverläufe wird eine Textur, die diesen Farbverlauf beschreibt benötigt. Der Vorteil bei der Verwendung einer Textur ist, dass nur die Texturkoordinaten und nicht die Vertexattribute neu berechnet werden müssen. Außerdem sind Erstellung und Änderung der Farbverläufe mit Standardprogrammen zur Bildverarbeitung komfortabel möglich.

In Abbildung 6.1 kann man die verwendete Textur für den Farbverlauf zur Darstellung von Spannungen im Material erkennen. Hierbei bedeutet Blau, dass keine Spannung vorhanden ist, Grün bedeutet geringe Spannung, und Rot stellt hohe Spannungen dar. Der Farbverlauf ähnelt dem Farbspektrum und orientiert sich da-

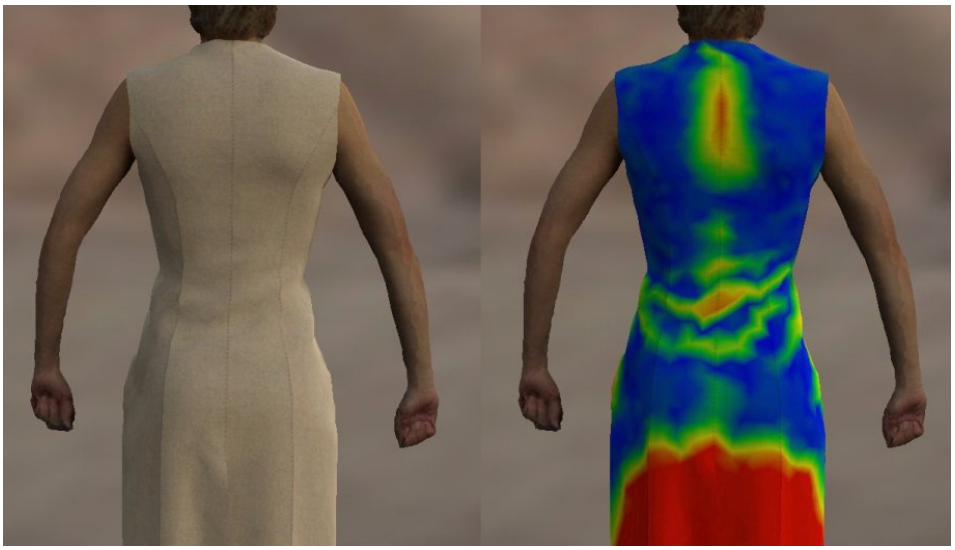


Abbildung 6.2: Links: Zur Passformkontrolle eignet sich die normale Visualisierung kaum, da die Falten am Rücken kaum sichtbar sind. — Rechts: Visualisiert man aber die Distanz zum Körper, werden die Falten sehr gut hervorgehoben.

bei an gängigen Farbverläufen zur Darstellung von Skalarwerten (z.B. Temperaturen).

Zur Verfeinerung der Anzeige für die darzustellenden Informationen kann der Wert, der auf Blau und der Wert der auf Rot abgebildet wird, interaktiv verschoben werden. Wenn beispielsweise ein Rock mit einem Stoff der 10% Dehnung zulässt entworfen wird, ist es oft sinnvoll diesen Wert auf Blau abzubilden. Ansonsten wird eine hohe Dehnung, die eigentlich nicht vorhanden ist, in der Darstellung suggeriert.

6.2 Accessoires und Nähte

Zur realistischen Visualisierung von Kleidung ist es nicht nur wichtig, das eigentliche textile Material korrekt zu rendern, sondern auch Details wie Nähte, Knöpfe und Nieten spielen eine große Rolle. Diese Elemente könnten zwar schon im Simulator behandelt werden und müssten dann nur noch dargestellt werden. Dies würde aber die Rechenzeit enorm ansteigen lassen. Insbesondere die exakte Simulation des Nahtfadens wäre um Größenordnungen zu aufwändig. Andere Objekte wie Gürtellaschen, Taschen und Embleme können hingegen als Schnittteile behandelt werden und somit wie die restlichen Schnittteile visualisiert werden.

Im Folgenden wird beschrieben, wie Accessoires und Nähte als zusätzliche Texturen aufgebracht werden können. Die besondere Herausforderung besteht in Berechnung der entsprechenden Texturkoordinaten. Eine alternative Herangehensweise wäre die Berechnung einer neuen Textur für jedes Schnittteil, die eine Kom-

bination aus der Basistextur und allen Nähten und Knöpfen darstellt. Problematisch ist dabei, dass für jedes Schnittteil eine eigene hoch aufgelöste Textur benötigt werden würde. Der damit verbundene Speicheraufwand und der Aufwand zur Neuberechnung dieser Textur (z.B. beim Wechsel der Naht) machen diese Methode äußerst unpraktikabel.

Zur Darstellung von Accessoires und Nähten über zusätzliche Texturen wird hier ein Algorithmus vorgestellt, der es erlaubt mehrere Texturen (Stoffmuster, Nahttextur und dann Aufdrucke) übereinander darzustellen. Er hat die nützliche Eigenschaft, dass die verschiedenen Texturen jeweils unabhängig sind und daher in hoher Auflösung dargestellt werden können. Hierfür werden mehrere Sätze an Texturkoordinaten verwendet. Die Kombination der Farbwerte erfolgt dann in einem speziellen Fragmentprogramm der Graphikkarte. Prinzipiell wird für jede einzelne Naht ein eigener Satz an Texturkoordinaten benötigt. Dies würde aber den Datenaufwand unnötig erhöhen. Daher werden zunächst unabhängige Nähte identifiziert, die sich einen Satz an Texturkoordinaten teilen können.

6.2.1 Erzeugung der Texturkoordinaten

Texturkoordinaten der Nähte

Ein Stück Stoff wird im Simulator als indiziertes Dreiecksnetz repräsentiert. Um die Nähte aufzubringen, muss für jeden Eckpunkt eine passende Texturkoordinate erzeugt werden. Die übliche Festlegung der Texturkoordinaten pro Eckpunkt statt pro Pixel erzeugt durch die sehr spezielle Positionierung, die Nähte benötigen, mannigfaltige Interpolationsprobleme. Kommen sich zwei Nähte zu nahe, muss deswegen eine der Nähte in einem weiteren Textur-Pass gezeichnet werden.

Eine gute Verteilung der Nähte auf Textur-Passes wird durch folgenden Algorithmus erreicht:

- Alle Texturen werden Textur-Pass 1 zugeteilt.
- Für jede Naht wird überprüft, ob sie einer anderen zu nahe kommt. Ist dies der Fall, wird die zweite Naht dem nächst höheren Textur-Pass zugewiesen und zunächst nicht weiter berücksichtigt.
- Sind alle Nähte überprüft, wird dieselbe Prüfung mit dem nächsten Textur-Pass durchgeführt, bis kein zusätzlicher Textur-Pass mehr erstellt wurde.

Da eine Naht nach einer bestimmten Strecke abbricht, sind zudem die OpenGL Texture-Clamping-Modi nicht ausreichend. Durch eine Erweiterung mit einem speziellen Fragmentprogramm kann eine dritte Texturkoordinate eingeführt und zur Ausmaskierung genutzt werden. Zur Erstellung wird zunächst die gesamte Fläche ausmaskiert - letztlich sollen die Texturkoordinaten nur innerhalb der Nähte von Bedeutung sein, außerhalb dieser entstehen durch Interpolation allenfalls Graphikfehler.

Abbildung 6.3: Darstellung von Nähten, die als zusätzliche Textur über die Textur der Schnittteile gelegt werden. Damit erhält man eine sehr hohe Qualität der Darstellung der Nähte, was in der Ausschnittsvergrößerung zu erkennen ist.

Der Nahtverlauf am Rand eines Stoffstücks ist bereits vorgegeben. Für die Kanten des Dreiecksnetzes, die entlang der Naht liegen werden die Texturkoordinaten ermittelt. Im Folgenden wird die etwas kompliziertere Berechnung der Texturkoordinaten im Inneren des Schnittteils beschrieben. Durch jeden Partikel wird eine Gerade gelegt, die den Winkel zwischen den angrenzenden Kanten der Naht halbiert. Dadurch ergibt sich eine Unterteilung des Inneren des Schnittteils. In jedem dieser Bereiche entspricht die zweite Texturkoordinate dem Abstand zur zugehörigen Kante.

Die erste Texturkoordinate ist die Länge aller Kanten, die vor dem aktuellen Bereich liegen plus dem Abstand des ersten Punktes der Kante vom parallel auf die Kante projizierten Punkt, für den die Texturkoordinate ermittelt werden soll. Die Texturkoordinaten werden nur auf der Naht und einem variablen, umgebenden Bereich festgelegt, der mittels der dritten Texturkoordinate demaskiert wird. In Richtung der ersten Texturkoordinate wird die dritte Texturkoordinate zudem so festgelegt, dass unter Berücksichtigung der Interpolation der Texturkoordinate die Textur exakt am gewünschten Ende der Naht abgebrochen wird.

An spitzen Ecken im Schnittteil sollte die Naht unterbrochen werden, da in diesem Fall keine korrekten durchgängigen Texturkoordinaten berechnet werden können. Dies rührt daher, dass der obige Algorithmus in einigen Partikeln mehr als eine Texturkoordinate zuweisen würde, da sich einige der Unterteilungen dann überlappen.

Texturkoordinaten der Aufdrucke

Die Erzeugung der Texturkoordinaten für Aufdrucke und Knöpfe gestaltet sich wesentlich einfacher, da hierzu nur der Bezugspunkt zum Ursprung des Objekts, die Ausrichtung und die Größe auf dem Schnittteil bekannt sein müssen. Aus diesen Informationen lassen sich die Texturkoordinaten berechnen. Erstreckt sich ein Aufdruck über mehrere Schnittteile, so müssen die Texturkoordinaten auf allen weiteren Schnittteilen entsprechend fortgesetzt werden.

(a) (b)

Abbildung 6.4: Darstellung von Accessoires: (a) Knopfleiste und Knöpfe mittels Texturen. — (b) Darstellung eines Aufdrucks mit einer teiltransparenten Textur.

Als Ergebnis der oben beschriebenen Algorithmen erhält man eine sehr realistische Visualisierung von Nähten und Accessoires (siehe Abbildung 6.3 und 6.4). Die Visualisierung bleibt auch bei Nahaufnahmen von hoher Qualität und eine Änderung der Naht oder der Basistextur zur Laufzeit ist effizient möglich.

Als Erweiterung könnte zusätzlich die Oberflächenstruktur der Nähte betrachtet werden. Realer Stoff hat an den Nähten unterschiedliche Höhen. Hierfür ist könnten Bumpmaps verwendet werden, die die Normalen einer glatten Oberfläche variieren um eine nicht glatte Fläche darzustellen.

6.3 Stoffdicke und Säume

Noch ist der Stoff eine unendlich dünne Fläche und wird genauso gerendert. Dies führt zu unrealistischer Visualisierung am Rand eines Kleidungsstücks. Ausgehend von einem simulierten Kleidungsstück kann eine detailreichere Geometrie, die die

Schnittteile dicker aussehen lässt, erzeugt und dargestellt werden. Der Simulator stellt die einzelnen Schnittteile als einfache Dreiecksnetze zur Verfügung. Die Kanten am Rand können nun je nach gewünschter Dicke des Stoffs extrudiert und gegebenenfalls noch abgerundet werden. Dabei sollte diese Extrusion lokal erfolgen, damit die Anzahl der darzustellenden Dreiecke nicht übermäßig ansteigt.

Säume entstehen, wenn Stoff am Rand eines Schnittteils umgeklappt und vernäht wird. Beim Ausschneiden der Schnittteile aus den Stoffbahnen werden für die Säume sogenannte Nahtzugaben zur Randkurve hinzugefügt. Die Größe der Nahtzugabe kann dann beim Rendering berücksichtigt werden. Säume lassen sich ganz ähnlich darstellen wie die Stoffdicke, nur dass das Schnittteil am Saum jetzt doppelt so dick ist wie der eigentliche Stoff und dass Säume nur so tief sind wie die Nahtzugabe war.

Wenn man davon ausgeht, dass bei den meisten Kleidungsstücken alle offen Kanten gesäumt sind, kann auf die Erzeugung zusätzlicher Geometrie zur Darstellung der Stoffdicke verzichtet werden, da die Säume bereits den Eindruck von dickerem Stoff erwecken (siehe auch Abbildung 6.5). Somit lässt sich mit Säumen eine lokale Extrusion der Schnittteile realisieren.

Im Prinzip erfolgt die Erzeugung neuer Geometrie ausgehend von den Randkanten eines Schnittteils. An eine Randkante wird entlang der Normalen des zugehörigen Dreiecks ein Viereck gesetzt. Die Höhe des Dreiecks entspricht der Höhe des Saums. An dieses Viereck wird ein weiteres gesetzt, das dann unterhalb des Schnittteils verläuft. Die Koordinaten der Vertices benachbarter Vierecke müssen noch auf ihren gemeinsamen Schwerpunkt gesetzt werden, damit man ein durchgehendes Mesh erhält. Dann werden die Vierecke tesselliert und gegebenenfalls noch unterteilt um sie abzurunden.

Zusammen mit der entsprechenden Fortsetzung der Texturkoordinaten der Vertices am Rand des Schnittteils erhält man einen fließenden Übergang. Die Normalen der Dreiecke werden vom Renderer anschließend noch geglättet, so dass der Eindruck eines runden Saums entsteht. Die Koordinaten der Vertices der so erzeugten Geometrie werden in jedem Zeitschritt neu berechnet, damit der Saum den Bewegungen der Schnittteile folgen kann.

6.4 Visualisierung von Bekleidung mit Schatten

Die Berechnung von Schatten ist essentiell für eine realistische Visualisierung von Stoff, da dieses Material sehr häufig Falten wirft, die zu Abschattungen führen. Aber auch wenn mehrere Lagen Stoff übereinander liegen, ermöglichen Schatten die Differenzierung der einzelnen Lagen, was ohne Schatten nicht möglich wäre.

6.4.1 Berechnung des Schattenwurfs von Punktlichtquellen

Basierend auf Shadow Volumes [Cro77] wird hier ein Algorithmus zur schnellen Visualisierung bekleideter Menschen skizziert, der vom Autor dieser Arbeit vorge-

Abbildung 6.5: Verbesserung der Darstellung durch Säume: Oben: Ohne Säume erscheint der Stoff sehr dünn. — Unten: Das Hemd mit Säumen sieht realistischer aus.

stellt wurde [FLG03]. Der Algorithmus verwendet intensiv moderne programmierbare Grafikkarten um die komplexen Geometrien in Echtzeit unter Einbeziehung der Selbstabschattung zu rendern.

Um den Schatten von Punktlichtquellen zu berechnen gibt es zwei wohl bekannte Techniken, die mit interaktiven Frameraten arbeiten: Shadow Maps und Shadow Volumes [Wil78, Cro77]. Shadow Maps arbeiten im Bildraum. Daher verursachen sie oft Artefakte durch Aliasing. Kürzlich wurde ein verbesserter Algorithmus vorgestellt [SCH03], der allerdings den Nachteil besitzt, dass er nicht komplett auf der Grafikkarte ablaufen kann. Shadow Volumes werden hingegen im Objektraum berechnet und erzeugen präzise Schatten. Sie lassen sich auch auf der GPU implementieren [EK02]. Ein wichtiger Teil des Verfahrens ist die Berechnung von potentiellen Silhouettenkanten eines Dreiecksnetzes. Für gewöhnlich werden diese in Software bestimmt. In [BS03] wird demonstriert, dass Vertexprogramme verwendet werden können, um diesen Berechnungsschritt zu beschleunigen.

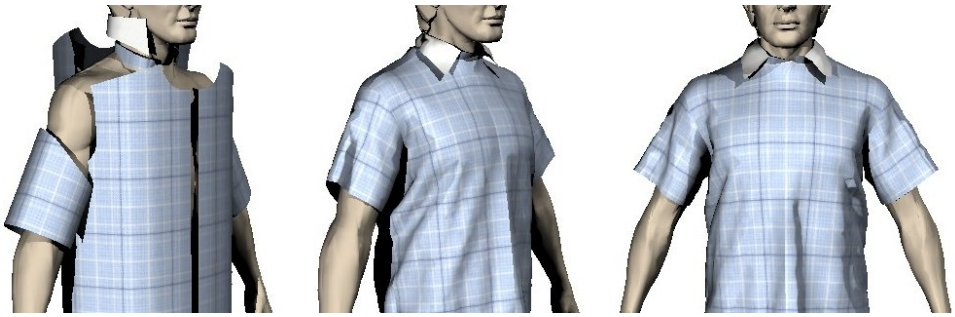


Abbildung 6.6: Interaktive Visualisierung mit zwei Punktlichtquellen, die harte Schatten werfen. Sehr gut zu erkennen ist der präzise Schattenwurf am Kragen, der den dreidimensionalen Eindruck der Szene verstärkt.

Um einen Frame zu rendern werden mehrere Durchgänge benötigt. In jedem Durchgang wird ein anderes Vertexprogramm verwendet. Der erste Durchgang rendert die Geometrie in den Z-Buffer. Der zweite Durchgang erzeugt die Vierecke, die die Schattenvolumen entlang der Silhouettenkanten erzeugen. Im dritten Durchgang werden diese Vierecke, die jeweils dem Betrachter abgewandt oder zugewandt sein können, gerendert. Schließlich wird im vierten und letzten Durchgang die eigentliche Geometrie gerendert und bestimmt, ob ein Pixel im Schatten liegt oder nicht. Die Durchgänge zwei bis vier werden für jede zusätzliche Lichtquelle wiederholt. Ein Beispiel zu diesem Verfahren ist in Abbildung 6.6 dargestellt.

Obwohl das Verfahren sehr präzise Schattenwürfe berechnet, ist dies gleichzeitig ein Nachteil, da man dadurch harte Schattenkanten erhält. Diese wirken sofort unnatürlich, da in der Realität nur selten Punktlichtquellen vorkommen. Daher wird im folgenden Abschnitt ein Algorithmus beschrieben, der effizient die Schattenwürfe von Flächenlichtquellen approximieren kann.

6.4.2 Berechnung des Schattenwurfs von Flächenlichtquellen

Realistische Beleuchtungssituation

Zur Aufnahme einer realen Beleuchtungssituation verwendet man eine Stahlkugel auf einem Stativ und eine Kamera. Die Umgebung spiegelt sich in der Stahlkugel und wird von der Kamera erfasst. Diese Information kann verwendet werden, um später beim Rendern die Szene zu beleuchten. Man erhält mit diesem Verfahren eine Flächenlichtquelle, die aus einem Bild der Umgebung besteht.

Um den Dynamikumfang von realem Licht erfassen zu können, wird nicht nur eine Aufnahme gemacht, sondern mehrere Aufnahmen mit unterschiedlichen Belichtungszeiten. Die Information aus diesen Aufnahmen kann in ein High-Dynamic-Range Bild umgerechnet werden. Man hat anstatt 8-Bit dann 24-Bit pro Farbkanal. Dieses Bild wiederum kann in eine Environment-Map umgerechnet werden, die dann später zur Beleuchtung der Geometrie verwendet wird.

Schnelle Berechnung weicher Schatten

Dieser Abschnitt beschreibt kurz ein Verfahren zur interaktiven Berechnung von weichen Schatten. Dieses Verfahren wurde in den Renderer des vorgestellten Systems integriert und erlaubt eine qualitativ hochwertige Visualisierung von Bekleidung.

Abbildung 6.7: Links: Normale Visualisierung mit OpenGL. — Mitte: Beleuchtung mit einer Environment Map. – Rechts: Berücksichtigung des Schattenwurfs. Bild entnommen aus [KF05].

Ein Sampling der HDR-Environment-Map liefert eine Reihe von Lichtquellen. Verwendet man diese zusammen mit dem Shadow Map Algorithmus, erhält man

ein Bild mit weichen Schatten. Allerdings benötigt man eine Vielzahl von Lichtquellen, damit das Verfahren realistische Bilder ohne Artefakte liefert.

Zur Erzeugung einer Shadow Maps berechnet man ein Tiefenbild der Geometrie aus Sicht der Lichtquelle. Mit dieser Tiefeninformation kann beim Rendern für jeden Pixel entschieden werden, ob er von der Lichtquelle aus sichtbar ist. Das Verfahren ist deshalb sehr interessant, da es komplett im Bildraum arbeitet und somit beliebige Geometrien verarbeiten kann. Zudem ist es höchst effizient und eignet sich sehr gut für die Erzeugung weicher Schatten.

Mit Structured Importance Sampling der Environment-Map werden wesentlich weniger Lichtquellen benötigt, oft reichen ein paar hundert. Die Zeit zum Rendern dieser Lichtquellen, ist für Echtzeitanwendungen jedoch immer noch zu lange. Eine weitere Reduktion der Anzahl der Lichtquellen lässt sich durch eine Glättung der Schattenwürfe erreichen. Hierzu wird für jede Lichtquelle der Übergang vom Schatten in den voll sichtbaren Bereich leicht geblurrt. Zusammen mit einigen weiteren Optimierungen und der Verwendung spezieller Vertex- und Fragmentprogramme erreicht man interaktive Frameraten. Weitere Details und Ergebnisse finden sich in [KF05].

6.5 Der CSG SceneGraph

Ein echtzeitfähiges Simulations- und Visualisierungssystem erfordert eine spezielle Architektur des Szenegraphen um dynamische Szenen mit hoher Darstellungsqualität effizient rendern zu können. Bei vielen Architekturen beeinflussen sich Darstellungsqualität und Simulationsgeschwindigkeit sehr stark, d.h. je besser die Darstellung desto langsamer läuft die Simulation. Im schlimmsten Fall wird die Simulation so langsam, dass kein interaktives Arbeiten mit dem System mehr möglich ist.

Da die Simulation und das Rendering jeweils sehr viel Rechenzeit beanspruchen aber auch zwei getrennte Einheiten sind, liegt es nahe beides zu parallelisieren. Dies ist insbesondere vorteilhaft, da aktuelle Grafikkarten in der Lage sind, fast alle Berechnungen zur Erzeugung eines Bildes eigenständig durchzuführen. Somit lässt sich bei Parallelisierung von Simulation und Rendering die verfügbare Rechenleistung moderner PCs hervorragend nutzen. Der Simulator nutzt die Ressourcen des Hauptprozessors und der Renderer die der Grafikkarte.

Dabei entsteht das Problem, dass jetzt mehrere Threads synchronisiert werden müssen. Insbesondere die Zugriffe auf den Szenegraph müssen konsistent erfolgen, da ansonsten Fehler beim Rendering auftreten können. Als Beispiel sei an dieser Stelle Java3D [Sun03] aufgeführt, da dieses Szenegraph-API in einer frühen Version des Systems verwendet wurde. Bei Java3D in der Version 1.3.1 sind keine konsistenten Änderungen am Szenegraphen möglich, da die Anwendung beispielsweise keine Kontrolle über die Abarbeitung von Transformationen in den entsprechenden Knoten hat. So kann es bei einer Animation einer kinematischen Kette passieren, dass ein Teil der Transformationen schon durchgeführt wurde, während

ein anderer Teil noch an der alten Position steht. Als Folge kommt es zum Flackern verschiedener Teile der Szene. Auch die Verwendung von so genannten Behaviours hilft hier nicht weiter, da nur Änderungen an einzelnen Knoten des Szenegraphen korrekt synchronisiert werden.

Als weiteres Problem von Java3D stellt sich die fehlende Kontrolle darüber heraus, wann ein neuer Frame gerendert werden soll. Damit wird oft ein neuer Frame gerendert, obwohl keine Änderungen an der Szene erfolgt sind. Besonders bei aufwändigen Simulationen, in denen die Berechnung eines Simulationsschrittes wesentlich länger dauert als das Rendern eines Frames, kostet dieses Verhalten von Java3D unnötig Rechenzeit.

OpenGL Performer [RH94] war das erste System, welches bereits sehr früh threadsichere Datenhaltung unterstützte. Allerdings sind in diesem System das Rendering und der eigentliche Szenegraph sehr eng miteinander verknüpft.

In [RVR04] wird ein Konzept für einen threadsicheren Szenegraphen vorgeschlagen, das in OpenSG umgesetzt wurde. Daten im Szenegraph werden selektiv repliziert, wenn von verschiedenen Threads auf sie zugegriffen wird. Der Vorteil dabei ist, dass Daten nur gepuffert werden, wenn sie auch wirklich von mehreren Threads benötigt werden. Die Synchronisation der verschiedenen Threads erfolgt mit Hilfe von ChangeLists. Für die Anwendungsentwicklung nachteilig ist die Tatsache, dass bei diesem Konzept der Anwender bei jedem Zugriff auf den Szenegraph die ChangeLists selbst aktualisieren muss.

Neben der Parallelisierung von Simulation und Rendering, ergeben sich durch einen threadsicheren Szenegraph weitere Vorteile in der Anwendungsentwicklung, da von nahezu beliebigen Stellen in der Anwendung konsistent auf den Szenegraph zugegriffen werden kann.

Des Weiteren werden Anwendungen von zukünftigen Prozessorgenerationen profitieren, wenn sie einen threadsicheren Szenegraph verwenden, da der Trend zu immer mehr parallelen Ausführungseinheiten innerhalb eines PCs weiter anhält. Aktuell sind CPUs mit zwei Kernen, die jeweils HyperThreading unterstützen. Dadurch ergeben sich vier Ausführungseinheiten, die nahezu parallel ablaufen.

6.5.1 Anforderungen

Aus der bereits skizzierten Anforderung an eine Parallelisierung von Anwendung und Renderer ergibt sich, dass der Szenegraph threadsicher sein muss. Ansonsten können Artefakte beim Rendern auftreten oder schlimmstenfalls sogar das System zum Absturz kommen. Die Threadsicherheit sollte dabei für die Anwendung möglichst transparent erfolgen, da ansonsten der Entwicklungsaufwand unnötig steigt. Trotzdem muss darauf geachtet werden, dass die hierfür nötige Synchronisation nur geringen Zeitverlust verursacht. Ein Beispiel für eine ineffektive Synchronisation ist es den Zugriff auf den Szenegraphen während der Simulation zu blockieren und dann am Ende eines Zeitschritts kurz freizugeben. Daraufhin würde der Renderer den Zugriff blockieren, bis das Bild fertig gestellt ist. Die beiden

Komponenten laufen dann zwar in unterschiedlichen Threads, die Laufzeit profitiert aber nicht davon.

Da der Szenegraph in einem Simulationssystem eingesetzt wird, muss die Änderung der Positionen der Partikel höchst effizient erfolgen, da sich diese in jedem Zeitschritt und damit in jedem Frame ändern. Änderungen an anderen Elementen, wie beispielsweise dem Material, der Lichtquellen und auch der Topologie der Dreiecksnetze, spielen nur eine untergeordnete Rolle und müssen nicht gesondert optimiert werden.

Weiterhin ist eine klare Trennung zwischen dem Szenegraph und dem Renderer, der die eigentliche Bilderzeugung übernimmt, sehr sinnvoll. In vielen Szenegraph APIs wie z.B. OpenInventor [Sil06] erfolgt das Rendern in Methoden der einzelnen Knoten. Dies hat gewiss einige Vorteile, wenn spezielle Renderer für eigene Knoten entwickelt werden sollen. Aber das Rendern in mehreren Durchgängen (Multipass-Rendering), wie es beispielsweise für die Berechnung von Schatten oder den Nähten benötigt wird, lässt sich dann nur schwierig realisieren, da der Renderer keine Kontrolle über fremd entwickelte Knoten hat. Daher sollte der Szenegraph nur die Verwaltung der Szene übernehmen und der Renderer die Bilderzeugung. Ein weiterer Vorteil einer solchen Architektur ist die Unabhängigkeit vom Renderer, der dann flexibel ausgetauscht werden kann.

Zusammenfassend ergeben sich folgende Kernanforderungen an den Szenegraphen:

- Parallelisierung von Szenegraph und Renderer
- Effizientes Ändern der Vertexpositionen
- Threadsicherheit
- Trennung zwischen Szenegraph und Renderer

Deren Umsetzung wird im folgenden Kapitel beschrieben, welches das Konzept des CSG SceneGraph vorstellt. Dabei stellt CSG ein rekursives Akronym[1] dar und bedeutet *CSG SceneGraph*.

6.5.2 Architektur

Die *Parallelisierung von Szenegraph und Renderer* kann erfolgen, da die Anwendung sich nur um die Neuberechnung der 3D Szene kümmern muss. Hierzu wird die Simulation durchgeführt und je nach Nutzereingaben auch die Struktur der Szene modifiziert, wenn beispielsweise neue 3D Objekte geladen werden. Der rechenaufwändige Teil ist dabei die Simulation. Die Anwendung greift für diese Änderungen ausschließlich auf den Szenegraphen zu.

Der Renderer hingegen muss im Prinzip nur Dreiecke verarbeiten. Dabei durchlaufen diese die Rendering-Pipeline, d.h. sie werden transformiert, gerastert

[1] Andere bekannte Bespiele rekursiver Akronyme sind: CAVE Automated Virtual Environment und PHP Hypertext Preprocessor.

und dann werden für die entsprechenden Pixel die Farbwerte berechnet. Für diesen Vorgang benötigt der Renderer keine Information über die Struktur der Szene oder den Simulator. Bei der Berechnung von hochwertigen Bildern mit speziellen Reflektionsmodellen für textile Materialien und der Berücksichtigung der Selbstabschattung ist das Durchlaufen der Rendering-Pipeline sehr aufwändig und muss insbesondere in mehreren Durchläufen (engl. Passes) erfolgen.

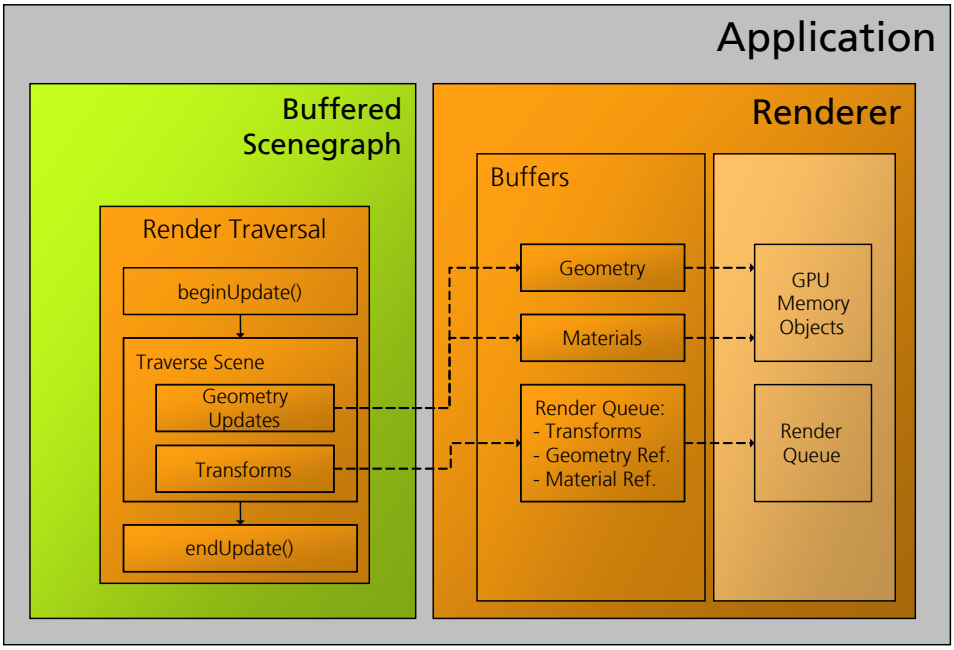


Abbildung 6.8: Parallelisierung von Szenegraph und Renderer. Der Renderer und der Buffered SceneGraph (beide rot umrandet) sind jeweils ein eigener kritischer Abschnitt. Somit kann immer nur ein Thread auf den jeweiligen Block zugreifen.

Aus Sicht des Szenegraphen genügt es das Rendering in drei Teilschritte auf zu teilen. Zunächst werden die neuen Geometriedaten und Materialdaten von der Anwendung an den Renderer übertragen. Danach werden diese Daten in den Speicher der Grafikkarte kopiert. Im letzten Schritt wird schließlich die klassische Rendering-Pipeline durchlaufen (siehe hierzu Abbildung 6.8).

Der Datenaustausch zwischen Renderer und Szenegraph erfolgt zu einem dedizierten Zeitpunkt und überträgt dabei die Änderungen am Szenegraph in den Speicherbereich des Renderers. Während dieser Übertragung darf der Renderer keine Kopie der Daten in die Grafikkarte vornehmen, da ansonsten ein Bild mit teilweise ungültigen Daten gerendert wird. Der aufwändige Schritt des Abarbeitens der Rendering-Pipeline hingegen arbeitet auf der Kopie und kann daher parallel erfolgen. Die Synchronisation gestaltet sich einfach, wenn man die einzelnen Schritte des Renderings in einen gemeinsamen kritischen Abschnitt setzt. Falls ver-

schiedene Threads den Renderer gleichzeitig ansprechen wollen, wird dies damit verhindert. Sinnvollerweise können wartende Threads gleich weiter arbeiten und müssen nicht in eine Warteschlange aufgenommen werden, da ja bereits ein anderer Thread mit der Bilderzeugung beschäftigt ist.

Weiterhin dürfen während des Datenaustauschs keine Änderungen am Szenegraph vorgenommen werden, da ansonsten die Gefahr besteht, dass die Szene inkonsistent wird. Als Folge ergeben sich Artefakte beim Rendern. Die hierfür benötigte Synchronisation des Szenegraphen wird im Abschnitt 6.5.2 beschrieben.

Effizientes Ändern der Vertexpositionen lässt sich mittels durchgängiger Verwendung indizierter Dreiecksnetze erreichen. Im Gegensatz zu nicht indizierten Datenstrukturen wird jeder Vertex immer nur einmal gespeichert bzw. kopiert. Nach der Eulerformel (siehe [ESK97]) ergibt sich für geschlossene Dreiecksnetze, dass es ungefähr doppelt so viele Dreiecke f wie Vertices v gibt, d.h. $v \approx \frac{f}{2}$. Speichert man ein Dreiecksnetz jedoch nicht indiziert, benötigt man $v' = 3 \cdot f = 3 \cdot 2v$ Vertices. Gegenüber der indizierten Variante bedeutet dies bei der Übertragung den sechsfachen Aufwand. Beim Speichern ergibt sich bei Verwendung von Floats (4 Byte) für die Vertices und Integern (4 Byte) für jeden Index der doppelt Aufwand, wenn man keine Indexstruktur verwendet.

Moderne Grafikkarten erlauben es ebenfalls ein Dreiecksnetz als Indexstruktur in den Speicher der Karte (GPU-MEM) zu kopieren. Für das eigentliche Verarbeiten der Dreiecke auf der GPU reicht ein kurzer Befehl um das ganze Rendern zu starten. Es liegen dann alle Daten in der GPU-MEM und es erfolgt keinerlei Kommunikation mit der CPU mehr.

Des Weiteren werden nur die Vertexpositionen an den Renderer übertragen. Alle abgeleiteten Daten, wie Normalen oder der Tangentspace für das Bumpmapping werden vom Renderer selbst berechnet. Die Attribute (Texturkoordinaten, Farben, etc.) werden, sofern sie sich nicht geändert haben, beibehalten.

Synchronisation

Die Synchronisation des Szenegraphen gegenüber Änderungen bzw. Zugriffen von mehreren Threads der Anwendung erfolgt darüber, dass alle Funktionen des Szenegraphen in einem kritischen Abschnitt liegen. Hierdurch wird beispielsweise verhindert, dass ein Thread beim Setzen der Koeffizienten einer Transformationsmatrix unterbrochen wird und dann ein anderer Thread die zur Hälfte überschriebene Matrix auslesen kann. In der Implementierung lässt sich diese Synchronisation transparent für die Anwendung realisieren, indem innerhalb der Funktionen ein Semaphore verwendet wird.

Allerdings wird mit diesem Mechanismus noch nicht sichergestellt, dass der Renderer immer eine konsistente Szene rendert, da sich Änderungen am Szenegraphen oft über mehrere Aufrufe erstrecken, die zwischendurch immer den kritischen Abschnitt verlassen. Daher wird ein AtomicUpdate eingeführt, dass den kritischen Abschnitt ausdehnt. Nachdem ein Thread mit einem AtomicUpdate be-

gonnen hat, kann kein anderer Thread mehr auf den Szenegraph zugreifen, auch der Renderer nicht mehr. Sind alle Änderungen durchgeführt, wird das AtomicUpdate beendet und der Szenegraph wieder freigegeben. Somit erhält die Anwendung die Möglichkeit beliebig komplexe atomare Änderungen vorzunehmen.

Im vorgestellten Konzept zur Synchronisation des Zugriffs auf den Szenegraphen blockiert ein solcher weitere Threads. Für die meisten PC-Systeme bedeutet dies keinen Nachteil, da meist nur ein Prozessor zur Verfügung steht. Auf einem Mehrprozessorsystem kann das Blockieren jedoch zeitraubend sein. Das Konzept kann aber erweitert werden, indem die Zugriffe nicht global blockieren, sondern nur für einzelne Knoten und deren Nachfolger. Diese Erweiterung muss aber noch auf ihrer Praxistauglichkeit hin überprüft werden, da unklar ist, wie hoch der Rechenaufwand für die Verwaltung einer solchen Synchronisationsmethode ist.

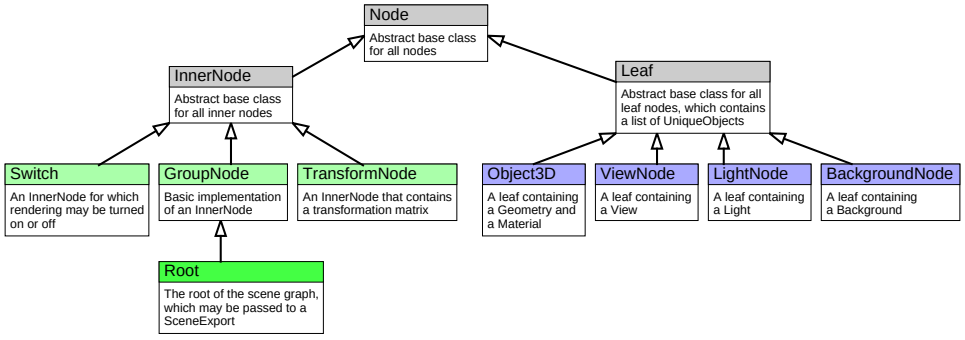


Abbildung 6.9: Klassenhierarchie der Knoten des CSG SceneGraph.

6.5.3 Anbindung an den Renderer

Der Renderer wird über eine schlanke Schnittstelle an den Renderer angebunden. Hierzu werden zunächst die Geometriedaten übertragen, wobei bereits in den Szenegraph hinzugefügte Geometrien komplett übertragen werden und andere nur noch teilweise, je nachdem welche sich geändert haben. Zur Verwaltung und Referenzierung auf der Seite des Renderers wird jeder Geometrie und jedem Material eine eindeutige ID zugewiesen.

Nach der Übertragung der Geometriedaten, erfolgt eine weitere Traversierung des Szenegraphen. Dabei wird für jedes Objekt3D (siehe Abbildung 6.9) ein Eintrag in eine Renderqueue vorgenommen. Jeder Eintrag besteht aus einer Transformation, einer Material ID und einer Geometrie ID (siehe 6.10). Die Renderqueue kann als nicht-hierarchische Repräsentation der Szene angesehen werden. Ist die Renderqueue komplett, kann der Renderer diese Liste arbeiten, die IDs auflösen und jeden einzelnen Eintrag rendern. Dabei ist es der konkreten Implementierung des Renderers überlassen, ob eine Sortierung nach Material erfolgt oder anderweitige Optimierungen vorgenommen werden.

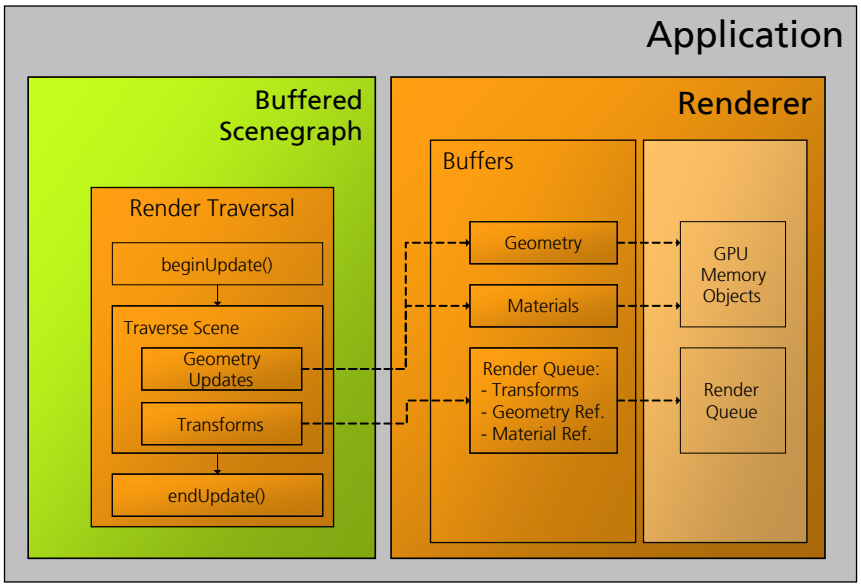


Abbildung 6.10: Anbindung des Renderers an den CSG SceneGraph: Die Kernidee besteht in der Übertragung der Renderqueue, die beschreibt, wo welche Objekte, mit welchem Material, dargestellt werden sollen. Die Renderqueue lässt sich als eine nicht-hierarchische Repräsentation des aktuellen Zustands des Szenegraphen interpretieren.

Zur Entkopplung von Szenegraph und Renderer wird der Szenegraph zweifach vorgehalten, ähnlich dem Double Buffer beim Rendern. Auf den ersten Szenegraph greift die Anwendung lesend und schreibend zu, der zweite wird vom Renderer ausgelesen. Zu einem dedizierten Zeitpunkt erfolgt eine Kopie vom ersten zum zweiten Szenegraph. Während diesem Vorgang dürfen natürlich weder die Anwendung, noch der Renderer auf den Szenegraph zugreifen. Das Kopieren ist nur nötig, wenn die Anwendung entweder einen neuen Simulationsschritt fertig berechnet hat oder aus anderen Gründen ein Update durchführen muss.

Würde der Renderer, wenn er die Daten für einen neuen Frame benötigt, nur ein AtomicUpdate machen, könnte ihn zwar die Anwendung nicht unterbrechen, aber dieses AtomicUpdate könnte zu einem Zeitpunkt geschehen, an dem der Simulator den Zeitschritt noch nicht fertig berechnet hat. Würde man nun alle Schritte des Simulators in einem AtomicUpdate Block ausführen, entstünde das Problem, dass Renderer und Simulator sehr eng miteinander verzahnt sind. Die dabei entstehenden Wartezeiten würden die Performance stark senken.

Durch die schlanke Schnittstelle zwischen Renderer und Szenegraph und der Entkopplung mittels des Double Buffering können unterschiedliche Renderer einfach an den Szenegraph angeschlossen werden. Diese können sogar gleichzeitig aktiv sein. So ist es beispielsweise möglich die Simulationsergebnisse auf der HEy-

Szene		**CSG**	**CSG ohne HT**	**Sequentiell**
		Hz	*Hz*	*Hz*
Kleid	Zeitschritte	110	80	65
	Frames	30	28	30
Hemd, Hose und Pulli	Zeitschritte	20	20	16
	Frames	20	15	16
Stofftuch mit Schattenwurf	Zeitschritte	72	60	5,5
	Frames	6	5,5	5,5

Tabelle 6.1: Vergleichsmessung zur Bestimmung der Performanz des CSG SceneGraph während einer interaktiven Simulation. Die Messung zur Spalte CSG wurde auf einem Hyper-Threading Prozessor durchgeführt. Bei der sequentiellen Messung wurden die Simulation und das Rendern nacheinander ausgeführt.

eWall darzustellen und gleichzeitig auf einem Einzelbildschirm die gewohnte Anwendung. Eine weitere Anwendung ist das Speichern der Szene in einer Datei.

6.5.4 Ergebnisse

Um die Performance des vorgestellten threadsicheren Szenegraphen zu bestimmen, wurde für unterschiedliche Szenen die Anzahl der Zeitschritte pro Sekunde und die Framerate bestimmt. In der Praxis erlaubt eine Framerate von $10\,Hz$ interaktives Arbeiten. Die Anzahl der Zeitschritte sollte immer möglichst hoch sein, um nahe an eine Simulation in Echtzeit heranzukommen. Je nach Komplexität der Szene wird dieses Ziel auch erreicht.

Die Ergebnisse der Messung sind in Tabelle 6.1 aufgeführt. In der Spalte CSG wurde das System auf einem Intel Pentium 4 mit Hyper-Threading und $3,6\,GHz$ getestet. In der nächsten Spalte wurde das Hyper-Threading deaktiviert und in der letzten zuerst simuliert und dann gerendert.

In der Szene *Kleid* war die maximale Framerate per Programm auf $30\,Hz$ begrenzt. Bei Verwendung des CSG SceneGraph ergibt sich für ein Hyper-Threading-fähigen Prozessor ein Speedup von $1,7$ und auch ohne Hyper-Threading immerhin ein Speedup von $1,23$. Das vorgestellte Konzept eines parallelen Szenegraphen bietet demnach auch auf herkömmlichen Systemen Geschwindigkeitsvorteile.

Die Szene *Hemd, Hose und Pulli* ist wesentlich komplexer. Daher wird eine insgesamt geringere Performance erreicht. Sowohl Framerate als auch die Anzahl der Zeitschritte liegt deutlich unter der Begrenzung von $30Hz$. Daher wird auf dem HT-System das Ergebnis eines jeden Zeitschritts auch gerendert. Ohne Hyper-Threading wird dieses Ziel aus Zeitmangel nicht erreicht, d.h. ein neuer Zeitschritt wurde berechnet, aber der Renderer ist noch nicht fertig und der Zeitschritt wird beim Rendern übersprungen. Gegenüber einem sequentiellen Ansatz ergibt sich hier ein Speedup von $1,25$.

Im letzten Test wurde das Verhalten bei äußerst niedriger Framerate untersucht, indem beim Rendern ein sehr *hochwertiger Schattenwurf* berechnet wurde. Da-

her erreicht der Renderer auch nur ca. sechs Frames pro Sekunde. Man erkennt gut, dass beim CSG SceneGraph der Simulator unabhängig vom Renderer arbeiten kann. Gegenüber einem sequentiellen Verfahren ergibt sich ein sehr hoher Speedup von 13, der jedoch relativiert werden muss, da es sicherlich möglich gewesen wäre, jeweils drei Zeitschritte zu berechnen und dann erst zu rendern. Allerdings müsste ein solches Verhalten speziell implementiert werden. Die Architektur des in dieser Arbeit vorgestellten Szenegraphen vereinfacht durch die Unabhängigkeit von Simulator und Renderer die Steuerung der Performance und die Verteilung der Rechenleistung erheblich.

Aus Sicht der Anwendungsentwicklung ergeben sich bei der Verwendung des vorgestellten parallelen und threadsicheren CSG SceneGraph zwei wesentliche Vorteile. Beliebige Threads können das Rendern eines Frames veranlassen. Dies ist beispielsweise vorteilhaft, wenn sowohl aus dem GUI als auch von einer Simulationskomponente das Rendern gestartet werden soll. Zudem können beliebige Threads Änderungen an der Szene vornehmen. Somit kann eine Anwendung parallel Daten berechnen und diese dann unabhängig voneinander in den Szenegraphen einfügen.

6.6 Zusammenfassung

In diesem Kapitel wird beschrieben, wie textile Materialien bzw. ganze Kleidungsstücken in Echtzeit visualisiert werden können. Zunächst wird eine Methode zum Visualisieren von Zusatzinformationen bei der virtuellen Anprobe vorgestellt. Mit entsprechenden Color-Mappings kann die Passform besser beurteilt werden, indem der Abstand des Kleidungsstücks zum Körper oder im Stoff auftretende Spannungen dargestellt werden.

Es werden eine Reihe von Algorithmen vorgestellt, die das Ziel haben die Kleidung möglichst realistisch aussehen zu lassen. Zunächst wird ein Verfahren zu Darstellung von Nähten vorgestellt. Für die Nähte werden zusätzliche Texturen und eigene Texturkoordinaten erzeugt. Damit lassen sich sehr detailreiche Nähte darstellen. Zur realistischen Visualisierung des Randes eines Kleidungsstücks wird ausgehend von einem simulierten Kleidungsstück eine detailreichere Geometrie erzeugt, die die Schnittteile dicker aussehen lässt. Dieses Verfahren kann auch zur Visualisierung von Säumen verwendet werden.

Da die Berechnung von Schatten essentiell für eine realistische Visualisierung von Stoff ist, werden bei der Visualisierung weiche Schatten mittels Shadow Mapping erzeugt. Die Schatten verbessern den Tiefeneindruck und ermöglichen die Differenzierung übereinander liegender Stofflagen.

Für echtzeitfähige Simulations- und Visualisierungssysteme wird der CSG SceneGraph vorgestellt. Bei diesem arbeiten Anwendung und Renderer voneinander getrennt und auf entsprechender Hardware (HyperThreading oder SMP) auch parallel. Dadurch kann gleichzeitig eine hochwertige Visualisierung erfolgen und eine hohe Framerate bei der Simulation erreicht werden.

Kapitel 7

Anwendungen

7.1 Virtual Prototyping von Bekleidung

Die Textil- und Bekleidungsindustrie in Deutschland steht unter immer höherem internationalem Konkurrenzdruck. Daher sind die Unternehmen gezwungen, ihre Produkte noch schneller und günstiger herzustellen. Der Arbeitsablauf bis zur Produktion neuer Kleidungsstücke ist noch nicht vollständig digitalisiert und umfasst eine lange Prozesskette. Insbesondere bei der Überprüfung der Passform müssen in aufwändiger Handarbeit eine hohe Anzahl von Prototypen entworfen werden, anhand derer entschieden wird, ob das entworfene Kleidungsstück noch abgeändert werden muss. Gelingt es hier, die Anzahl der benötigten Prototypen zu reduzieren, so bringt dies enorme Zeit- und Kosteneinsparungen mit sich. Dies kann durch ”Virtual Prototyping”der Bekleidung erreicht werden.

Das Bundesministerium für Wirtschaft und Arbeit (BMWA) hat hierfür im Rahmen des PRO INNO II Programms ein Projekt ins Leben gerufen, welches ein virtuelles Produktentwicklungsverfahren zur lückenlosen CAD-gestützten 2D-Schnittkonstruktion und 3D-Passformsimulation von Bekleidung realisieren soll. Die beteiligten Partner sind die Assyst Bullmer GmbH, deutscher Marktführer im Bereich von CAD-CAM Lösungen für die Bekleidungs- und Textilindustrie, und das Fraunhofer IGD. Die in dieser Arbeit entwickelten Methoden zur Simulation und Visualisierung von Bekleidung fließen direkt in dieses Projekt mit ein.

Die Beurteilung der Passform und des optischen Erscheinungsbildes eines Kleidungsstückes wird in diesem Produktentwicklungsverfahren schon während der Konstruktionsphase in realitätsähnlicher Qualität an anthropometrisch korrekten Avataren ermöglicht, ohne dass die Fertigung von physischen Prototypen aus den verwendeten Stoffen erforderlich ist. Sowohl die 3D-Passformsimulation als auch die realitätsnahe Visualisierung erfolgen hierbei in Echtzeit, um ein unterbrechungsfreies Arbeiten zu ermöglichen.

Arbeitsablauf zur Erstellung neuer Kleidungsstücke

Der Arbeitsablauf zur Erstellung neuer Kleidungsstücke ist heutzutage noch nicht vollständig rechnergestützt und umfasst eine lange Prozesskette mit mehreren Stationen. Das Bekleidungsdesign erfolgt zunächst auf Papier als Skizze (quasi als grobe räumliche Ansicht) oder durch Drapieren des Stoffs an einer Schneiderbüste. Daraus erstellt ein Schnittkonstrukteur mit Hilfe von 2D-CAD Programmen die entsprechenden Schnittteile. Mit diesen als Vorlage schneiden Cutter die eigentlichen Stoffstücke aus Stoffbahnen. Schließlich werden die Einzelteile zu einem ersten Prototypen vernäht. Dieser kann einer Schneiderbüste, d.h. einer Puppe mit idealisierter Körperform, oder einer Person, die als Firmenmodel zur Verfügung steht, angezogen werden. Jetzt entscheidet der Designer, ob das Kleidungsstück seinen Vorstellungen entspricht. Ist dies nicht der Fall, beginnt der aufwändige Konstruktions- und Musterprozess von vorn.

In diesem Entwicklungszyklus arbeitet der Designer meist ohne computergestützte Hilfsmittel. Dem Schnittkonstrukteur erlauben zur Zeit verfügbare CAD-Lösungen nur eine 2D-Konstruktion. Die Ergebnisse dieser Konstruktion können vom Designer nur schlecht beurteilt werden und müssen deshalb zunächst in einen Prototyp umgesetzt werden. Folglich wäre ein CAD-System wünschenswert, das die gesamte Prozesskette, vom Design bis zum Erstellen des realen Kleidungsstücks, unterstützt. Integriert man in ein solches System auch noch eine physikalisch korrekte virtuelle 3D-Darstellung von Bekleidung an generischen Avataren oder realen Menschen, so entfallen die aufwändigen Schritte zur Erzeugung der physischen Prototypen.

Arbeitsablauf beim Virtual Prototyping

Zunächst wird das zweidimensionale Schnittmuster mithilfe der bewährten 2D-Bekleidungs CAD Lösung konstruiert [ASS06]. Anschließend werden die Schnittteile mit Informationen zur Positionierung und dem Verlauf der Nähte versehen. Die Zusatzinformationen ermöglichen die automatische Berechnung einer dreidimensionalen Position der noch unvernähten Schnittteile um den Avatar. Die physikalisch basierte Simulation näht die vorpositionierten Schnittteile zusammen und berechnet die Passform der Kleidungsstücke, wobei die triangulierten Schnittteile als Grundlage für das Partikelsystem dienen. Gleichzeitig werden die simulierten Kleidungsstücke und der Avatar in Echtzeit unter Berücksichtigung von Selbstabschattungen visualisiert. Als Lichtquelle dient eine Aufnahme einer realen Szene.

Während der Simulation können die Passform beurteilt und Änderungen am Sitz des Kleidungsstücks vorgenommen werden. Als Hilfestellung für eine gute Beurteilung lassen sich weitere Eigenschaften, wie etwa der Abstand des Kleidungsstücks zum Körper oder im Stoff auftretende Spannungen (siehe hierzu Abbildung 7.2), visualisieren. Der Bekleidungskonstrukteur hat jetzt die Möglichkeit, Änderungen am zweidimensionalen Schnittbild im CAD System vorzunehmen und kann dann direkt die Auswirkung auf die Passform dreidimensional begutachten.

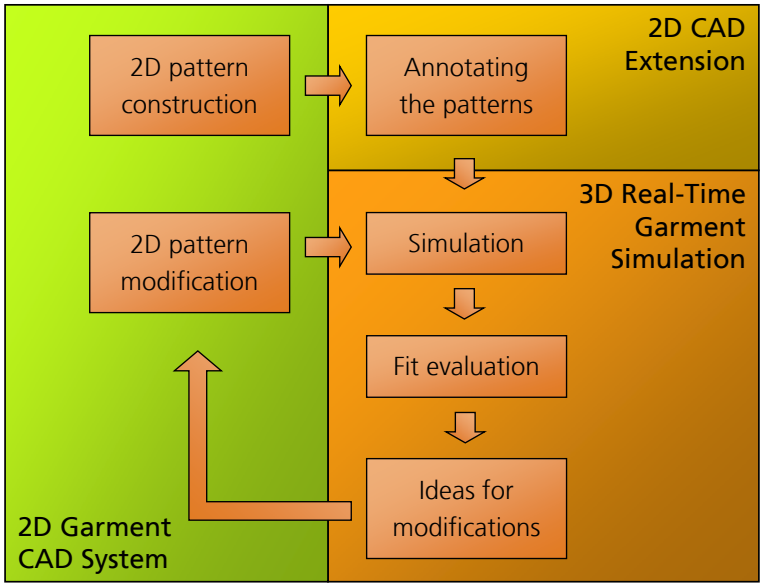


Abbildung 7.1: Virtual Prototyping Workflow.

Mithilfe dieser virtuellen Passformkontrolle kann Bekleidung viel schneller konstruiert werden. Der gesamte Arbeitsablauf ist in Abbildung 7.1 zusammengefasst.

7.1.1 3D Schnittkonstruktion

Die Schnittkonstruktion des im vorigen Abschnitt beschriebenen Verfahrens zum Virtual Protopying von Bekleidung erfolgt noch klassisch rein zwei dimensional. Aber eine solche 2D-Schnittkonstruktion stellt hohe Anforderungen an die Erfahrung des Konstrukteurs. Er muss Schnittteile in der Ebene so konstruieren, dass sie sich im zusammen genähten Zustand der 3D-Körperform der Schneiderbüste bzw. dem Firmenmodel in einer bestimmten Konfektionsgröße anpassen und zusätzlich den Vorstellungen des Designers entsprechen. Um dem Schnittkonstrukteur die Arbeit zu erleichtern sind neue Herangehensweisen erforderlich. Ein viel versprechender Ansatz ist der Übergang auf drei Dimensionen. Die Schnittteile können dann direkt an einer virtuellen Schneiderbüste oder auch an einem virtuellen Model konstruiert werden. Die Auswirkungen einer Änderung der Konstruktion werden dann sofort sichtbar und können leichter beurteilt werden. Zudem wird die Konstruktion eines Kleidungsstücks einfacher, da die Positionierung und die Lage der Schnittteile zueinander direkt erkennbar werden. Es steht daher zu erwarten, dass sogar wenig erfahrene Konstrukteure in der Lage sein werden, eigene Kreationen zu entwerfen.

Zur Umsetzung eines solchen Szenarios können die Ergebnisse dieser Arbeit als Grundlage verwendet werden [FGKK05]. Zusätzlich müssten entspre-

Abbildung 7.2: Links: Interaktives drapieren mit Stecknadeln. — Rechts: Visualisierung von Spannungen im Material

chende Werkzeuge geschaffen werden, die dem Konstrukteur und Designer die Möglichkeit eröffnen wie in der Realität textile und andere Materialien virtuell an 3D Formkörpern anzubringen, zu schneiden und an den Schnittkanten zu verbinden, d.h. virtuell zu nähen.

7.2 Virtual Try-On

Die Bekleidungsindustrie und der Bekleidungshandel sind mit über 50 Mrd. Euro Umsatz jährlich allein in der Bundesrepublik Deutschland ein wichtiger Konsumgütermarkt. Dabei spielt das Design neuer Kleidungsstücke eine entscheidende Rolle, da der Markt ständig nach neuen Kreationen verlangt. Noch vor wenigen Jahren war die Erstellung von vier Kollektionen (Frühjahr, Sommer, Herbst und Winter) üblich. Heute geht der Trend ganz deutlich zu 12 Kollektionen im Jahr und damit zu einem monatlichen Wechsel der Produkte. In diesem Kontext ist eine virtuelle Anprobe auf Basis einer akkuraten Textilsimulation äußerst hilfreich. Ergebnisse dieser Arbeit bilden die Basis für eine interaktive Java-basierte Textilsimulation. Diese wurde im Kontext des Virtual Try-On Projektes entwickelt [WKK+04, DTE+04, WKK+05].

Ein wichtiger Faktor bei der Bewertung eines Simulationssystems ist der Vergleich mit der Realität. Hierzu wurden mit dem System erzeugte Bilder mit realen Kleidungsstücken verglichen. Die reale Kleidung wurde nach den Schnittmustern geschneidert, die auch in der Simulation verwendet werden. Aus den 3D-Laserscans wurde eine Puppe gefertigt, der anschließend die Kleidung angezogen wurde.

Abbildung 7.3: Oben: Vergleich zwischen realem und virtuellem Anzug. — Unten: Vergleich zwischen realem und virtuellem Hemd.

Abbildung 7.4: Virtuelle Anprobe eines Hemds.

7.3 Virtuelle Maßkonfektion

Die Anfertigung von Maßbekleidung ist noch immer ein langwieriger komplexer Prozess, zu dessen zentralen Bestandteilen die Ermittlung der Körpermaße des Kunden, die Auswahl der Schnittmuster, Stoffe und Accessoires zählen. All diese Informationen werden anschließend vom Maßschneider zur Herstellung des Prototypen verwendet, dessen Passform unter Umständen in mehreren Iterationen überprüft und optimiert werden muss. Dazu muss der Kunde jedes Mal den Verkaufsraum aufsuchen und das Kleidungsstück anprobieren - ein zeitraubender Vorgang. Dem Kunden fehlt außerdem die Möglichkeit bereits während der Auswahl im Ladengeschäft eine exakte Vorstellung über das endgültige Aussehen des Kleidungsstücks zu gewinnen, da nur eine beschränkte Musterkollektion und kleinformatige Stoffproben zur Verfügung stehen.

Bisherige Lösungen, die den Maßschneider bei seiner Arbeit unterstützen, erlauben das Eingeben der Kundenmaße, woraus automatisch die 2D-Schnittmuster erstellt werden. Eine Überprüfung, ob die Bekleidung dem Kunden passen wird,

Abbildung 7.5: Vergleich zwischen realer und virtueller Bluse.

kann nicht durchgeführt werden. Ebenso ist eine Visualisierung der Kleidung im Voraus nicht möglich.

Mit den Ergebnissen dieser Arbeit wird die Entwicklung eines Systems zur Erstellung von Maßkonfektion mit der Methoden der virtuellen Realität möglich. Auf Basis einer interaktiven Textilsimulation und -präsentation kann individuelle Bekleidung entworfen und am virtuellen Doppelgänger des Kunden begutachtet werden. Im Gegensatz zu bestehenden Systemen wird damit eine Überprüfung der Passform eines Kleidungsstücks möglich, ohne dass ein reales Muster erstellt werden muss. Dadurch sinken der Zeitaufwand und als Folge die Kosten.

Im Gegensatz zur herkömmlichen Produktentwicklung können mit Hilfe von virtueller Maßkonfektion bereits während der Konstruktionsphase Aussagen über die Passform und das optische Erscheinungsbild eines Kleidungsstückes getroffen werden, ohne dass die Fertigung eines Prototypen erforderlich ist. Die Optimierung der Passform vereinfacht sich aufgrund der besseren Kontrolle während der Modellentwicklung wesentlich, wodurch der gesamte Erstellungsprozess beträchtlich verkürzt wird.

IntExMa - Interaktives Expertensystem für Maßkonfektion

Derzeit wird das in dieser Arbeit vorgestellte System zur Bekleidungssimulation und Visualisierung im InExMa (www.intexma.info) Projekt weiterentwickelt. Ziel dieses Projektes ist es Maßkonfektion kundenindividuell und interaktiv zu präsentieren. Die Maße des Kunden werden verwendet, um einen virtuellen Menschen mit den gleichen Maßen zu erzeugen. Dieser wird dann mit den in dieser Arbeit vorgestellten Methoden bekleidet. Besonderes Augenmerk liegt in diesem Projekt auf der Visualisierung von Herrenhemden mit Knöpfen, Nähten und den vielen anderen wichtigen Accessoires, die für den optischen Eindruck so entscheidend sind.

Das Projekt verbindet die Vorteile traditioneller Einzelhandelsgeschäfte mit den Stärken moderner Informationstechnologien. Daraus entsteht ein völlig neues Einkaufserlebnis für den Kunden. Das System unterstützt dabei sowohl den Käufer bei der Präsentation seiner Produkte, als auch den Verkäufer bei der Auswahl der Maßkonfektion.

Kapitel 8

Zusammenfassung

In dieser Arbeit werden mehrere Verfahren und Konzepte vorgestellt, die zusammengenommen ein komplettes System zur Simulation und Visualisierung von textilen Materialien bzw. Bekleidung ergeben, das sich für die virtuelle Anprobe und das Virtual Prototyping von Bekleidung eignet. Besonderes Augenmerk wird dabei auf das Zusammenspiel der einzelnen Algorithmen, die Stabilität der Simulation und die Echtzeitfähigkeit des Komplettsystems gelegt.

Ontologien für Bekleidung. Für die Berechnung der Anfangswerte für die Simulation von Bekleidung werden in dieser Arbeit Ontologien für Kleidungsstücke und ein Verfahren zum interaktionsfreien Einkleiden virtueller Menschen vorgestellt. Es wird zunächst eine Ontologie für die Schnittteile spezifiziert und danach auf Kleidungsstücke erweitert. Diese Ontologien stellen semantische Informationen bereit, die es zusammen mit einer Methode zur geometrischen Vorpositionierung erlauben, Schnittteile ohne Nutzereingaben an virtuellen Menschen zu platzieren.

Die Ontologien können eingesetzt werden um auf einer hohen bzw. abstrakten Ebene intuitiv die Eigenschaften von mehreren gleichzeitig getragenen Kleidungsstücken zu ändern. Beispielsweise kann die Reihenfolge, in der die Kleidungsstücke angezogen werden, damit einfach verändert werden. Dabei ergibt sich die dreidimensionale Geometrie der Kleidung an einem speziellen Avatar als Resultat der Anwendung der Semantik auf die tiefer gelegenen Schichten der Modellierung. Hierbei sind keinerlei Eingriffe des Anwenders in diese Schichten nötig. Er spezifiziert nur anfangs einige semantische Eigenschaften und dies ergibt die gewünschten Änderungen an der Geometrie.

Auch die Kollisionserkennung zwischen unterschiedlichen Kleidungsstücken bzw. Lagen von Stoff kann mit Hilfe der vorgestellten Ontologien verbessert werden. Obwohl es bekannt ist, dass die Kollisionserkennung davon profitiert, wenn auf der geometrischen Ebene auch Informationen der physikalisch basierten Ebene (Geschwindigkeit, maximale Krümmung des Materials, etc.) in Betracht gezogen werden, wurden bisher keine Information aus höheren Ebenen der Modellierung verwendet.

Schnelle Kollisionserkennung. In dieser Arbeit wird ein Verfahren zur Kollisionserkennung auf der Basis von Distanzfeldern vorgestellt. Mit diesem lassen sich effizient die Kollisionen zwischen stark deformierbaren und weitgehend starren Körpern erkennen und behandeln. Dabei werden die weitgehend starren Körper durch ihre Distanzfelder repräsentiert, die für die nötigen Tests auf Abstand und Durchdringung verwendet werden. Das Verfahren eignet sich insbesondere für die physikalisch basierte Simulation von Stoff, wie in mehreren Beispielen demonstriert wird. Da das Verfahren äußerst effizient ist, können damit immersive Anwendungen realisiert werden, in denen Stoff oder sogar ganze Bekleidung in Echtzeit animiert wird. Die Methode ist extrem robust, da durch die Vorzeichen des Distanzfeldes klar zwischen dem Inneren und dem Äußeren eines Objektes unterschieden werden kann.

Weiterhin wird in dieser Arbeit ein neues Verfahren zur effizienten Vermeidung von Selbstkollisionen textiler Materialien beschrieben. Die Methode basiert auf einer hierarchischen Datenstruktur, die während der Simulation sowohl schnell upgedated als auch effizient abgefragt werden kann. Anstatt zu garantieren, dass keinerlei Selbstdurchdringungen auftreten, wird nur verhindert, dass sie passieren. Auch komplexe Kleidungsstücke wie ein Godetrock, der viele Falten wirft, können mit dieser Methode interaktiv simuliert werden. Obwohl nicht alle Durchdringungen verhindert werden, arbeitet der Algorithmus sehr stabil, da bei der Kollisionsantwort keine hohen Kräfte in das Partikelsystem eingeführt werden. Mehrfachkollisionen werden zwar nicht gesondert behandelt, führen aber auch nicht zu einer Instabilität.

Effiziente Materialsimulation. Es wird ein Algorithmus zur interaktiven Animation von textilen Materialien vorgestellt, der sogar für Dreiecksnetze mit mehreren tausend Dreiecken in Echtzeit arbeitet. Das Verfahren ist auch bei großen Zeitschritten stabil, da die hohen internen Kräfte durch geometrische Einschränkungen modelliert werden. Nur die externen Kräfte, wie die Gravitation, werden auf herkömmliche Weise über die Zeit integriert. Weiterhin können mit dem Verfahren unterschiedliche Materialien animiert werden. Obwohl das Verfahren keine absolute physikalische Korrektheit besitzt, erhält man sehr realistische Animationen.

Weiterhin wird ein Algorithmus zur Optimierung der Passform von bereits simulierten Kleidungsstücken vorgestellt. Das Verfahren arbeitet automatisch und setzt einfache User-Eingaben in komplexe Geometrieänderungen um. Die Kernidee ist es, zunächst Kleidungsstücke in verschiedenen Größen in einer Datenbank bereitzustellen. Hierzu werden die entsprechenden Schnittteile benötigt. Dann werden während der Laufzeit mit Hilfe einer linearen Gradierung die fehlenden Größen erzeugt. Bei einer solchen Gradierung werden korrespondierende Punkte auf zwei Schnittteilen linear interpoliert. Dazu wird ein automatisches Verfahren, das diese Korrespondenzen für die Vertices der Schnittteile berechnet, vorgestellt. Zur Eingabe der gewünschten Größe der Bekleidung kann ein einfacher Slider verwendet werden, der es entweder ermöglicht die Größen kontinuierlich zu verändern oder aber ein Raster anbietet um mehrere Konfektionsgrößen nacheinander zu testen.

Echtzeit Visualisierung. Zur realistischen Visualisierung von Kleidung wird ein Verfahren vorgestellt, das auch Details wie Nähte, Knöpfe und Säume rendern kann. Diese Elemente werden nicht vom Simulator berücksichtigt, sondern nur dargestellt, was Rechenzeit spart. Durch die Verwendung von zusätzlichen Texturen lassen sich diese Elemente in hoher Auflösung darstellen, die auch bei Nahaufnahmen nicht an Qualität verlieren.

Bei herkömmlichen Architekturen beeinflussen sich Darstellungsqualität und Simulationsgeschwindigkeit sehr stark, d.h. je besser die Darstellung desto langsamer läuft die Simulation. Daher wird in dieser Arbeit der CSG SceneGraph vorgestellt, dessen spezielle Architektur es erlaubt dynamische Szenen mit hoher Darstellungsqualität effizient zu rendern. Die Kernidee dieser Architektur ist die Parallelisierung von Szenegraph und Renderer, wodurch beide Einheiten entkoppelt werden. Weiterhin können unterschiedliche Renderer einfach an den CSG SceneGraph angeschlossen werden, die sogar gleichzeitig aktiv sein können.

Kapitel 9

Ausblick

Die in dieser Arbeit vorgestellten Verfahren liefern vielfältige Ideen für weitere Arbeiten. Die Ontologien für Bekleidung erlauben das automatische Einkleiden von virtuellen Menschen. Die Grundidee des Verfahrens ist das zweistufige Einkleiden: Zuerst eine geometrische Vorpositionierung und danach die physikalisch basierte Endpositionierung. Hier wäre auch ein völlig anderer Ansatz denkbar. Angenommen, ein Avatar mit durchschnittlichen Maßen wäre bereits bekleidet. Dann könnte sowohl der Avatar, als auch die Kleidung an die individuellen Maße des Kunden angepasst werden. Voraussetzung für die Anpassung des Avatars wäre ein auf Körpermaße parametrisierbares Menschmodell. Die Kleidung könnte mit den Techniken aus Kapitel 5.5 verändert werden, d.h. man würde kontinuierlich mit Hilfe der Simulation von einer Kleidergröße auf die neue übergehen. Wichtig ist hierbei, dass das Menschmodell ebenfalls kontinuierlich verändert werden kann.

Das vorgestellte Modell zur Berechnung von Reibungskräften während der Kollisionsbehandlung liefert gute Ergebnisse und ein realistisches Stoffverhalten. Eine Möglichkeit zur Erweiterung wäre die Behandlung von anisotroper Reibung. Beim bisherigen isotropen Reibungsmodell spielt es keine Rolle, in welcher Richtung zwei Objekte übereinander gleiten. Da aber die Webstruktur von Stoff vermuten lässt, dass sich textile Materialien häufig auch bei der Reibung anisotrop verhalten, wären eine weitere Untersuchung dieses Sachverhaltes und eine Erweiterung des Simulationsmodells sicherlich sehr interessant.

Die Kollisionserkennung mit Distanzfeldern könnte als Basis für eine Hardware zur Kollisionserkennung oder ein GPU-basiertes Verfahren dienen. Da der Algorithmus einfach zu implementieren ist und nur wenige bedingte Sprünge beinhaltet, eignet er sich gut hierfür. Weiterhin werden keine Hierarchien benötigt, für die wiederum bedingte Sprünge nötig wären um sie zu traversieren. Zudem dauert jede Ausführung einer Abstandsberechnung ungefähr gleich lange. Dies ist sehr wichtig, wenn ein Pipelining-Schema realisiert werden soll.

Bei der virtuellen Anprobe sind die virtuellen Menschen nicht deformierbar und nur eingeschränkt beweglich, obwohl viele Kleidungsstücke, wie z.B. enge Hosen oder BHs, den Körper deformieren. Bei vielen Kleidungsstücken ist diese

Deformation des Körpers erheblich und eine Aussage über die genaue Passform ist nur dann möglich, wenn dies berücksichtigt wird. Damit dieses Verhalten bei der Simulation berücksichtigt werden kann, muss das Weichgewebe des Menschen entsprechend modelliert werden. Dabei entsteht eine gekoppelte Simulation, die das Zusammenspiel zwischen dem textilen Material und dem Körper des Menschen berücksichtigen muss. Eine Konzeption eines Simulationsmodells, das diese Kopplung unterstützt, ist eine ebenso große Herausforderung, wie die zweckmäßige Simulation des Weichgewebes. Zusätzlich könnte das Verfahren zur Kollisionserkennung auf der Basis von Distanzfeldern erweitert werden. Die Deformation des Körpers könnte in einem zusätzlichen Vektorfeld repräsentiert werden, das zusammen mit dem Distanz- und Geschwindigkeitsfeld die Rekonstruktion der Oberfläche des Avatars ermöglichen würde.

Auch der vorgeschlagene Algorithmus zur Vermeidung von Selbstkollisionen bietet Potential für weitere Entwicklungen. Derzeit werden nur die Positionen der Partikel im neuen Zeitschritt betrachtet. Bei Verwendung eines kontinuierlichen Tests auf Basis der Bewegung der beteiligten Partikel könnte die Genauigkeit des Verfahrens gesteigert werden. Damit wäre es vermutlich möglich auch mehrlagige Kleidung mit nur geringem Effizienzverlust zu simulieren.

Für die in Kapitel 7.1.1 vorgestellte 3D Schnittkonstruktion können die Ergebnisse dieser Arbeit als Grundlage verwendet werden. Zusätzlich müssten entsprechende Werkzeuge geschaffen werden, die dem Konstrukteur und Designer die Möglichkeit eröffnen wie in der Realität textile und andere Materialien virtuell an 3D Formkörpern anzubringen, zu schneiden und an den Schnittkanten zu verbinden, d.h. virtuell zu nähen.

Literaturverzeichnis

[AB03] Uri Ascher and Eddy Boxerman. On the modified conjugate gradient method in cloth simulation. *The Visual Computer*, 19:526–531, 2003.

[ACOL00] Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. In *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series, pages 157–164, New Orleans, LA, USA, July 2000. ACM SIGGRAPH.

[ACP03] Brett Allen, Brian Curless, and Zoran Popović. The space of human body shapes: reconstruction and parameterization from range scans. *Transactions on Graphics (ACM SIGGRAPH 2003)*, 22(3):587–594, 2003.

[Ale02a] Obermaier Alefeld, Lenhardt. *Parallele numerische Verfahren*. Springer-Verlag, 2002.

[Ale02b] Marc Alexa. Recent advances in mesh morphing. *Computer Graphics Forum*, 21(2):173–196, 2002.

[AMH02] Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering*. A K Peters, LTD, 2002.

[Arv90] James Arvo. A simple method for box-sphere intersection testing. In *Graphics gems*, pages 335–339. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[ASS93] Boris Aronov, Raimund Seidel, and Diane Souvaine. On compatible triangulations of simple polygons. *Computational Geometry: Theory and Applications*, 3(1):27–35, 1993.

[ASS06] ASSYST - Gesellschaft für Automatisierung, Software und Systeme mbH. cad.assyst, http://www.assyst-intl.com/, 2006.

[BA04] Eddy Boxerman and Uri Ascher. Decomposing cloth. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 153–161, New York, NY, USA, 2004. ACM Press.

[BA05] J. Andreas Baerentzen and Henrik Aanaes. Signed distance computation using the angle weighted pseudonormal. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):243–253, 2005.

[BE92] Marshall Wayne Bern and David Eppstein. Mesh generation and optimal triangulation. In Ding-Zhu Du and Frank Kwang-Ming Hwang, editors, *Computing in Euclidean Geometry*, number 1 in Lecture Notes Series on Computing, pages 23–90. World Scientific, 1992.

[BFA02] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3), 2002.

[BHW94] David E. Breen, Donald H. House, and Michael J. Wozny. Predicting the drape of woven cloth using interacting particles. In *SIGGRAPH 94 Conference Proceedings*, Annual Conference Series, pages 365–372, Orlando, FL, USA, July 1994. ACM SIGGRAPH.

[BMF03] Robert Bridson, S. Marino, and Ronald Fedkiw. Simulation of clothing with folds and wrinkles. In *Proc. ACM/Eurographics Symposium on Computer Animation*, pages 28–36, 2003.

[BMWM01] David E. Breen, Sean Mauch, Ross T. Whitaker, and Jia Mao. 3d metamorphosis between different types of geometric models. *Eurographics 2001 Proceedings*, 20(3):36–48, 2001.

[BPK+02] Peer-Timo Bremer, Serban Porumbescu, Falko Kuester, Bernd Hamann, Kenneth I. Joy, and Kwan-Liu Ma. Virtual clay modeling using adaptive distance fields. In *Proceedings of the 2002 International Conference on Imaging Science, Systems, and Technology (CISST 2002)*, 2002.

[Bri03] Robert Bridson. *Computational aspects of dynamic surfaces*. PhD thesis, Stanford University, 2003.

[BS03] Stefan Brabec and Hans-Peter Seidel. Shadow volumes on programmable graphics hardware. In *Eurographics 2003 (Computer Graphics Forum)*, 2003.

[BVIG91] Chakib Bennis, Jean-Marc Vézien, Gérard Iglésias, and André Gagalowicz. Piecewise surface flattening for non-distorted texture mapping. In *SIGGRAPH 91 Conference Proceedings*, Annual Conference Series, Las Vegas, NV, USA, July 1991. ACM SIGGRAPH.

[BW98] David Baraff and Andrew Witkin. Large steps in cloth simulation. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 43–54, Orlando, FL, USA, July 1998. ACM SIGGRAPH.

[BWAK03] David Baraff, Andrew Witkin, John Anderson, and Michael Kass. Physically based modeling. In *SIGGRAPH Course Notes, ACM SIGGRAPH*, 2003.

[BWK03] David Baraff, Andrew Witkin, and Michael Kass. Untangling cloth. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, 22(3):862–870, 2003.

[Cha91] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(5):485–524, 1991.

[CK02] Kwang-Jin Choi and Hyeong-Seok Ko. Stable but responsive cloth. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3), 2002.

[CMT02] Frédéric Cordier and Nadia Magnenat-Thalmann. Real-time animation of dressed virtual humans. *Computer Graphics Forum*, 21(3):327–335, 2002.

[CMT04] Frédéric Cordier and Nadia Magnenat-Thalmann. A data-driven approach for real-time clothes simulation. *Pacific Graphics 2004*, October 2004.

[COSL98] Daniel Cohen-Or, Amira Solomovic, and David Levin. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17(2):116–141, 1998.

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. In *Computer Graphics (SIGGRAPH '77 Proceedings)*, pages 242–248, Jul 1977.

[DDBT99] L. Dekker, I. Douros, B. Buxton, and P. Treleaven. Building symbolic information for 3d human body modeling from range data. In *Proceedings of the Second International Conference on 3-D Digital Imaging and Modeling*. IEEE Computer Society, 1999.

[DMB00] Mathieu Desbrun, Mark Meyer, and Alan H. Barr. Interactive animation of cloth-like objects for virtual reality. In House and Breen [HB00], chapter 9, pages 219–239.

[DSB99] Mathieu Desbrun, Peter Schröder, and Alan Barr. Interactive animation of structured deformable objects. In *Graphics Interface '99*, Kingston, Canada, June 1999.

[DTE+04] A. Divivier, R. Trieb, A. Ebert, H. Hagen, C. Groß, A. Fuhrmann, V. Luckas, J.L. Encarnação, E. Kirchdörfer, M. Rupp, S. Vieth, S. Kimmerle, M. Keckeisen, M. Wacker, W. Straßer, M. Sattler, R. Sarlette, and R. Klein. Virtual try-on: Topics in realistic, individualized dressing in virtual reality. In *Proceedings of the Virtual and Augmented Reality Status Conference*, February 2004.

[e f06] e frontier. Poser 6, http://www.e-frontier.com/, 2006.

[EEH00] Bernhard Eberhardt, Olaf Etzmuß, and Michael Hauth. Implicit-explicit schemes for fast animation with particle systems. In *Eurographics Computer Animation and Simulation Workshop 2000*, 2000.

[EEHS00] Olaf Etzmuß, Bernhard Eberhardt, Michael Hauth, and Wolfgang Straßer. Collision adaptive particle systems. *Proceedings Pacific Graphics 2000*, 2000.

[EGB+02] Achim Evert, Ingo Ginkel, Henning Barthel, Andreas Divivier, and Michael Bender. Efficient assistance of virtual dress fitting using intelligent morphing. In *Proceedings of the Second IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP 2002)*, pages 306–311, 2002.

[EHK+00] Bernhard Eberhardt, Jens-Uwe Hahn, Reinhard Klein, Wolfgang Straßer, and Andreas Weber. Dynamic implicit surfaces for fast proximity queries in physically based modeling. Technischer Bericht WSI–2000–11, Wilhelm-Schickard-Institut für Informatik, Graphisch-Interaktive Systeme (WSI/GRIS), Universität Tübingen, 2000.

[EK02] Cass Everitt and Mark J. Kilgard. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. In *NVIDIA White paper*, 2002.

[EKK+01] Olaf Etzmuß, Michael Keckeisen, Stefan Kimmerle, Johannes Mezger, Michael Hauth, and Markus Wacker. A cloth modelling system for animated characters. In *Proceedings Graphiktag*, 2001.

[EKS03] Olaf Etzmuß, Michael Keckeisen, and Wolfgang Straßer. A fast finite element solution for cloth modelling. *Proceedings of Pacific Graphics*, pages 244–251, 2003.

[ESK96] José L. Encarnação, Wolfgang Straßer, and Reinhard Klein. *Graphische Datenverarbeitung 1*. Oldenburg, 1996.

[ESK97] José L. Encarnação, Wolfgang Straßer, and Reinhard Klein. *Graphische Datenverarbeitung 2*. Oldenburg, 1997.

[Etz02] Olaf Etzmuß. *Animation of Surfaces with Applications to Cloth Modelling*. PhD thesis, Universität Tübingen, Fakultät für Informatik, 2002.

[EW99] B. Eberhardt and A. Weber. A particle system approach to knitted textiles. *Computers & Graphics*, 23(4):599–606, September 1999.

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

[FGKK05] Arnulph Fuhrmann, Clemens Groß, Martin Knuth, and Jörn Kohlhammer. Virtual prototyping of garments. In *ProSTEP iViP Science Days 2005*, 2005.

[FGL03] Arnulph Fuhrmann, Clemens Groß, and Volker Luckas. Interactive animation of cloth including self collision detection. *Journal of WSCG*, 11(1):141–148, February 2003.

[FGLW03] Arnulph Fuhrmann, Clemens Groß, Volker Luckas, and Andreas Weber. Interaction-free dressing of virtual humans. *Computers & Graphics*, 27(1):71–82, January 2003.

[FGW05] Arnulph Fuhrmann, Clemens Groß, and Andreas Weber. Ontologies for virtual garments. In *Workshop towards Semantic Virtual Environments (SVE 2005)*, pages 101–109, Villar, Switzerland, mar 2005.

[FJP95] Lori Freitag, Mark Jones, and Paul Plassmann. An efficient parallel algorithm for mesh smoothing. In *Proceedings of the Fourth International Meshing Roundtable*, pages 47–58, 1995.

[FLG03] Arnulph Fuhrmann, Volker Luckas, and Clemens Groß. Improving the clothing design process: Fast 3d visualization of garments. In *Proceedings of Innovation and Modelling of Clothing Engineering Processes (IMCEP)*, October 2003.

[FP85] M. Shamos F. Preparata. *Computational Geometry*. Springer Verlag, 1985.

[FPRJ00] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 249–254, 2000.

[FSG03] Arnulph Fuhrmann, Gerrit Sobotka, and Clemens Groß. Distance fields for rapid collision detection in physically based modeling. In *Proceedings of GraphiCon 2003*, pages 58–65, September 2003.

[FTT99] John Funge, Xiaoyuan Tu, and Demetri Terzopoulos. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 29–38. ACM Press/Addison-Wesley Publishing Co., 1999.

[FUF06] Christoph Fünfzig, Torsten Ullrich, and Dieter W. Fellner. Hierarchical spherical distance fields for collision detection. *IEEE Computer Graphics and Applications*, 26(1):64–74, 2006.

[Fuh01] Arnulph Fuhrmann. Automatische Vorpositionierung von Schnittteilen an individuellen 3D-Figurinen zur Faltenwurfsimulation. Diplomarbeit, Technische Universität Darmstadt, Fachbereich Informatik, 2001.

[Fun99] John Funge. *AI for Games and Animation: A Cognitive Modeling Approach*. A K Peters, 1999.

[Gan03] Sebastian Gantzert. Effiziente Animation virtueller Bekleidung. Diplomarbeit, Technische Universität Darmstadt, Fachbereich Informatik, 2003.

[Gel98] Allen Van Gelder. Approximate simulation of elastic membranes by triangulated spring meshes. *Journal of Graphics Tools*, 3(2):21–42, 1998.

[GFL03] Clemens Groß, Arnulph Fuhrmann, and Volker Luckas. Automatic pre-positioning of virtual clothing. In *Proceedings of the Spring Conference on Computer Graphics*, pages 113–122, April 2003.

[GHZ99] Leonidas J. Guibas, David Hsu, and Li Zhang. H-walk: hierarchical distance computation for moving convex bodies. In *Proceedings of 15th ACM Symposium on Computational Geometry*, pages 344–351, 1999.

[GKJ+05] Naga K. Govindaraju, David Knott, Nitin Jain, Ilknur Kabul, Rasmus Tamstorf, Russell Gayle, Ming C. Lin, and Dinesh Manocha. Interactive collision detection between deformable models using chromatic decomposition. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3):991–999, 2005.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 171–180, New Orleans, LA, USA, August 1996. ACM SIGGRAPH.

[HB00] Donald H. House and David E. Breen, editors. *Cloth Modeling and Animation*. A. K. Peters, Natick, MA, USA, 2000.

[HE01] Michael Hauth and Olaf Etzmuß. A high performance solver for the animation of deformable objects using advanced numerical methods. In *Proc. Eurographics 2001*, 2001.

[HEE+02] Michael Hauth, Olaf Etzmuß, Bernd Eberhardt, Reinhard Klein, Ralf Sarlette, Mirko Sattler, Katja Daubert, and Jan Kautz. Cloth animation and rendering. In *Eurographics 2002 Tutorials*, 2002.

[Her02] Horst Herr. *Technische Mechanik*. Europa-Lehrmittel, 2002.

[HKL+99] Kenneth E. Hoff III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of ACM SIGGRAPH 1999*, pages 277–286, 1999.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. In *Eurographics*. Eurographics, Eurographics, 2003. State-of-the-Art Report.

[Hub96] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[Hum06] Human Solutions GmbH. Body Scanning, http://www.human-solutions.de, 2006.

[IH02] Takeo Igarashi and John F. Hughes. Clothing manipulation. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 91–100, New York, NY, USA, 2002. ACM Press.

[IZLM01] Kenneth E. Hoff III, Andrew Zaferakis, Ming C. Lin, and Dinesh Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. In *Symposium on Interactive 3D Graphics*, pages 145–148, 2001.

[Jak01] Thomas Jakobson. Advanced character physics. In *Proceedings of Game Developers Conference*, USA, March 2001.

[JGW04] Jein Jeong, Tolga Goktekin, and Xunlei Wu. An interactive parallel multigrid fem simulator. In *Medical Simulation: International Symposium, ISMS*, 2004.

[JS01] Mark W. Jones and Richard Satherley. Using distance fields for object representation and rendering. In *Proc. of Eurographics UK Conference EGUK 2001*, pages 37–44, London, 2001.

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

[KC02] Young-Min Kang and Hwan-Gue Cho. Bilayered approximate integration for rapid and plausible animation of virtual cloth with realistic wrinkles. In *Computer Animation 2002*, page 203, Geneva, Switzerland, 2002.

[KCCL01] Young-Min Kang, Jeong-Hyeon Choi, Hwan-Gue Cho, and Do-Hoon Lee. An efficient animation of wrinkled cloth with approximate implicit integration. *The Visual Computer*, 17:147–157, 2001.

[KCL+00] Young-Min Kang, Jeong-Hyeon Choi, Do-Hoon Lee, Chan-Jong Park, and Hwan-Gue Cho. Real-time animation technique for flexible and thin objects. In *WSCG 2000*, pages 322–329, Plzen, Czech., 2000.

[KF05] Martin Knuth and Arnulph Fuhrmann. Self-shadowing of dynamic scenes with environment maps using the gpu. In *WSCG FULL papers proceedings*, pages 31–38, Feb 2005.

[KHM+98] J. Klosowski, H. Held, J. Mitchell, K. Zikan, and H. Sowizral. Efficient collision detection using bounding volume hierarchies of *k*-DOPs. *IEEE Trans. Visual. Comput. Graph.*, 4(1), 1998.

[Kim05] Stefan Kimmerle. *Collision Detection and Post-Processing for Physical Cloth Simulation*. PhD thesis, Universität Tübingen, Fakultät für Informatik, 2005.

[KNF04] Stefan Kimmerle, Matthieu Nesme, and François Faure. Hierarchy accelerated stochastic collision detection. In *Vision, Modeling, and Visualization 2004*, 2004.

[KPLM98] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shells: A higher-order bounding volume for fast proximity queries. In *Proceedings of the 1998 Workshop on the Algorithmic Foundations of Robotics*. A K Peters, LTD, 1998.

[KSFS03] Michael Keckeisen, Stanislav L. Stoev, Matthias Feurer, and Wolfgang Straßer. Interactive cloth simulation in virtual environments. In *Proceedings of IEEE Virtual Reality 2003*, page 71, Washington, DC, USA, 2003. IEEE Computer Society.

[Lip80] M. Lipschutz. *Differentialgeometrie - Theorie und Anwendung*. McGraw-Hill, New York, 1980.

[LMTT91] B. Lafleur, N. Magnenat-Thalmann, and D. Thalmann. Cloth animation with self-collision detection. In *Proc. of the IFIP conference on Modelling in Computer Graphics, SIGGRAPH '91*, pages 179–187, 1991.

[LY05] Ze Gang Luo and Matthew Ming-Fai Yuen. Reactive 2D/3D garment pattern design modification. *Computer-Aided Design*, 37(6):623–630, 2005.

[MC94] Brian Mirtich and John Canny. Impulse-based dynamic simulation. In *Proceedings of Workshop on Algorithmic Foundations of Robotics*, 1994.

[MDDB01] Mark Meyer, Gilles Debunne, Mathieu Desbrun, and Alan H. Barr. Interactive animation of cloth-like objects for virtual reality. *The Journal of Visualization and Computer Animation*, 12:1–12, May 2001.

[Mir98] Brian Mirtich. V-Clip: fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, 1998.

[MIR05] MIRALab. Forschungsgruppe der Universität Genf, http://miralabwww.unige.ch/, 2005.

[MKE03] Johannes Metzger, Stefan Kimmerle, and Olaf Etzmuß. Hierarchical techniques in collision detection for cloth animation. *Journal of WSCG*, 11(2):322–329, February 2003.

[MPT99] William A. McNeely, Kevin D. Puterbaugh, and James J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. In *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series, pages 401–408, Los Angeles, CA, USA, August 1999. ACM SIGGRAPH.

[NG96] H. N. Ng and R. L. Grimsdale. Computer graphics techniques for modelling cloth. *IEEE Computer Graphics and Applications*, 16(5):52–60, September 1996.

[OWL04] OWL. Web Ontology Language, http://www.w3.org/2004/owl/, 2004.

[PF94] Larry Palazzi and David R. Forsey. A multilevel approach to surface response in dynamically deformable models. In *Proceedings of Computer Animation '94*, 1994.

[PG95] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14(2):105–116, June 1995.

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

[Pro97] Xavier Provot. Collision and self-collision handling in cloth model dedicated to design garments. In *Graphics Interface '97*, pages 177–189, 1997.

[PT92] Bradley A. Payne and Arthur W. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications*, 12(1):65–71, January 1992.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[RC02] Isaac Rudomin and Jose Luis Castillo. Realtime clothing: geometry and physics. In *WSCG 2002 Posters*, pages 45–48, Plzen, Czech., 2002.

[RH94] John Rohlf and James Helman. Iris performer: a high performance multiprocessing toolkit for real-time 3d graphics. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 381–394, New York, NY, USA, 1994. ACM Press.

[RVR04] Marcus Roth, Gerrit Voss, and Dirk Reiners. Multi-threading and clustering for scene graph systems. *Computers & Graphics*, 28(1):63–66, 2004.

[Sam90] Hanan Samet. *The design and analysis of spatial data structures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[SCH03] Pradeep Sen, Mike Cammarano, and Pat Hanrahan. Shadow silhouette maps. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, 22(3):521–526, July 2003.

[SE03] Philip J. Schneider and David H. Eberly. *Geometric Tools for Computer Graphics*. Morgan Kaufmann Publishers, 2003.

[Set96] J.A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Science*, 93(4):1591–1595, 1996.

[Set99] J.A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, Cambridge, UK, 1999.

[SF96] Meng Sun and Eugiene Fiume. A technique for constructing developable surfaces. In Wayne A. Davis and Richard Bartels, editors, *Graphics Interface '96*, pages 176–185. Canadian Human-Computer Communications Society, 1996.

[SG92] Thomas W. Sederberg and Eugene Greenwood. A physically based approach to 2-D shape blending. In *SIGGRAPH 92 Conference Proceedings*, Annual Conference Series, pages 25–34, Chicago, IL, USA, July 1992. ACM SIGGRAPH.

[SG01] Vitaly Surazhsky and Craig Gotsman. Controllable morphing of compatible planar triangulations. *ACM Transactions on Graphics*, 20(4):203–231, 2001.

[Sil06] Silicon Graphics, Inc. Open Inventor, http://oss.sgi.com/projects/inventor/, 2006.

[SMT03] Hyewon Seo and Nadia Magnenat-Thalmann. An automatic modeling of human bodies from sizing parameters. In *Proceedings of the 2003 Symposium on Interactive 3D graphics*, pages 19–26. ACM Press, 2003.

[SOM04] Avneesh Sud, Miguel A. Otaduy, and Dinesh Manocha. Difi: Fast 3d distance field computation using graphics hardware. *Computer Graphics Forum (Proc. of Eurographics)*, 23(3):557–566, 2004.

[SPG03] Christian Sigg, Ronald Peikert, and Markus Gross. Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization '03*. IEEE Computer Society Press, October 2003.

[Sun03] Sun Microsystems. Java3D 1.3.1, http://java.sun.com/products/java-media/3d/, 2003.

[TF88] Demetri Terzopoulos and Kurt Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, December 1988.

[TKH+04] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Straßer, , and P. Volino. Collision detection for deformable objects. In *Eurographics 2004: State of the Art Report*, August 2004.

[TKH+05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Straßer, and P. Volino. Collision detection for deformable objects. *Computer Graphics forum*, 24(1):61–81, March 2005.

[TW88] Demetri Terzopoulos and Andrew Witkin. Physically based models with rigid and deformable components. *IEEE Computer Graphics*, 8:41–51, 1988.

[Vas00] T. Vassilev. Dressing virtual people. In *Proceedings of Systemics, Cybernetics and Informatics 2000*, 2000.

[VCMT95] P. Volino, M. Courchese, and N. Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 137–144, Los Angeles, CA, USA, August 1995. ACM SIGGRAPH.

[VCMT05] Pascal Volino, Frédéric Cordier, and Nadia Magnenat-Thalmann. From early virtual garment simulation to interactive fashion design. *Computer-Aided Design*, 37(6):593–608, 2005.

[vdB97] G. van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[Ver67] Loup Verlet. Computer ëxperimentsön classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical Review*, 159(1):98–103, 1967.

[Vis01] Visual Effects Society. From ivory tower to silver screen. *SIGGRAPH 2001 Course Notes*, 36, 2001.

[VMT97a] P. Volino and N. Magnenat-Thalmann. Developing simulation techniques for an interactive clothing system. In *Proceedings of Virtual Systems and MultiMedia '97*, pages 109 – 118, 1997.

[VMT97b] P. Volino and N. Magnenat-Thalmann. Interactive cloth simulation: Problems and solutions. In *JWS97-B, Geneva, Switzerland*, 1997.

[VMT00a] P. Volino and N. Magnenat-Thalmann. Accurate collision response on polygonal meshes. In *Proc. of Computer Animation Conference*, pages 179–188, 2000.

[VMT00b] Pascal Volino and Nadia Magnenat-Thalmann. *Virtual Clothing, Theory and Practice*. Springer, 2000.

[VMT01] P. Volino and N. Magnenat-Thalmann. Comparing efficiency of integration methods for cloth animation. In *Proceedings of Computer Graphics International 2001 (CGI'01)*, 2001.

[VSC01] T. Vassilev, B. Spanlang, and Y. Chrysanthou. Fast cloth animation on walking avatars. In *Proceedings of Eurographics 2001*, 2001.

[WB05] Wingo Sai-Keung Wong and George Baciu. Dynamic interaction between deformable surfaces and nonsmooth objects. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):329–340, 2005.

[WBK97] Andrew Witkin, David Baraff, and Michael Kass. Physically based modeling: Principles and practice. In *SIGGRAPH Course Notes, ACM SIGGRAPH*, 1997.

[Wei86] J. Weil. The synthesis of cloth objects. *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4):49–54, August 1986.

[Wes92] P. Wesseling. *An Introduction to Multigrid Methods*. John Wiley & Sons, 1992.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, pages 270–274, Aug 1978.

[WK03] Jianhua Wu and Leif Kobbelt. Piecewise linear approximation of signed distance fields. In *Vision, Modeling and Visualization 2003 Proceedings*, pages 513–520, 2003.

[WKE99] Rudiger Westermann, Leif Kobbelt, and Thomas Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15(2):100–111, 1999.

[WKK+04] M. Wacker, M. Keckeisen, S. Kimmerle, W. Straßer, V. Luckas, C. Groß, A. Fuhrmann, R. Sarlette, M. Sattler, and R. Klein. Virtual Try-On: Virtuelle Textilien in der Graphischen Datenverarbeitung. *Informatik Spektrum*, 27(6), December 2004.

[WKK+05] M. Wacker, M. Keckeisen, S. Kimmerle, W. Straßer, V. Luckas, C. Groß, A. Fuhrmann, M. Sattler, R. Sarlette, and R. Klein. Simulation and Visualisation of Virtual Textiles for Virtual Try-On. *Special Issue of Research Journal of Textile and Apparel: Virtual Clothing Technology and Applications*, 9(1), 2005.

[Zac98] Gabriel Zachmann. Rapid collision detection by dynamically aligned dop-trees. In *Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98*, Atlanta, Georgia, March 1998.

[Zac02] Gabriel Zachmann. Minimal hierarchical collision detection. In *Proc. ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 121–128, Hong Kong, China, November11–13 2002.

[ZWF+03] Ye Zhao, Xiaoming Wei, Zhe Fan, Arie Kaufman, and Hong Qin. Voxels on fire. In *Proceedings of IEEE Visualization*, 2003.

[ZY00] Dongliang Zhang and Matthew M.F. Yuen. Collision detection for clothed human animation. In *Proceedings of the 8th Pacific Graphics Conference on Computer Graphics and Application*, pages 328–337, 2000.

[ZY01] Dongliang Zhang and Matthew M. F. Yuen. Cloth simulation using multilevel meshes. *Computers & Graphics*, 25(3):383–389, June 2001.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Veröffentlichungen

In diesem Anhang sind die wissenschaftlichen Veröffentlichungen des Autors dieser Arbeit aufgelistet.

A.1 Artikel in Fachzeitschriften

- M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Straßer, P. Volino
  Collision Detection for Deformable Objects
  Computer Graphics forum, 24(1):61–81, March 2005.

- M. Wacker, M. Keckeisen, S. Kimmerle, W. Straßer, V. Luckas, C. Groß, A. Fuhrmann, R. Sarlette, M. Sattler, and R. Klein
  Simulation and Visualisation of Virtual Textiles for Virtual Try-On
  Special Issue of Research Journal of Textile and Apparel: Virtual Clothing Technology and Applications 9(1), 2005.

- M. Wacker, M. Keckeisen, S. Kimmerle, W. Straßer, V. Luckas, C. Groß, A. Fuhrmann, R. Sarlette, M. Sattler, and R. Klein
  Virtual Try-On: Virtuelle Textilien in der Graphischen Datenverarbeitung.
  Informatik Spektrum, 27(6):504–511, December 2004.

- Arnulph Fuhrmann, Clemens Gross, Volker Luckas, and Andreas Weber.
  Interaction-free Dressing of Virtual Humans.
  Computers & Graphics, 27(1):71–82, January 2003.

A.2 Internationale Veröffentlichungen

- Arnulph Fuhrmann and Clemens Groß and Martin Knuth and Jörn Kohlhammer

Virtual Prototyping of Garments.
ProSTEP iViP Science Days 2005, September 2005.

- M. Teschner, B. Heidelberger, D. Manocha, N. Govindaraju, G. Zachmann, S. Kimmerle, J. Mezger, A. Fuhrmann
  Collision Handling in Dynamic Simulation Environments
  Eurographics 2005, Full-Day Tutorial, Dublin, Ireland, August 2005.

- Arnulph Fuhrmann and Clemens Groß and Andreas Weber
  Ontologies for Virtual Garments.
  Workshop towards Semantic Virtual Environments (SVE 2005), 101–109, March 2005.

- G. Zachmann, M. Teschner, S. Kimmerle, B. Heidelberger, L. Raghupathi, A. Fuhrmann
  Real-Time Collision Detection for Dynamic Virtual Environments.
  IEEE Virtual Reality 2005, Tutorials, March 2005.

- Martin Knuth and Arnulph Fuhrmann
  Self-Shadowing of dynamic scenes with environment maps using the GPU.
  WSCG FULL papers proceedings, pages 31–38, February 2005.

- M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, , and P. Volino:
  Collision Detection for Deformable Objects.
  Eurographics 2004: State of the Art Report, August 2004.

- A. Divivier, R. Trieb, A. Ebert, H. Hagen, C. Gross, A. Fuhrmann, V. Luckas, J.L. Encarnação, E. Kirchdörfer, M. Rupp, S. Vieth, S. Kimmerle, M. Keckeisen, M. Wacker, W. Strasser, M. Sattler, R. Sarlette, and R. Klein;
  Virtual try-on: Topics in realistic, individualized dressing in virtual reality
  Proceedings of the Virtual and Augmented Reality Status Conference, February 2004.

- Arnulph Fuhrmann, Volker Luckas, and Clemens Gross:
  Improving the Clothing Design Process: Fast 3D Visualization of Garments.
  Proceedings of Innovation and Modelling of Clothing Engineering Processes (IMCEP), October 2003.

- Arnulph Fuhrmann, Gerrit Sobotka, and Clemens Gross:
  Distance Fields for Rapid Collision Detection in Physically Based Modeling.
  Proceedings of GraphiCon 2003, pages 58–65, September 2003.

- Clemens Gross, Arnulph Fuhrmann, and Volker Luckas:
  Automatic Pre-Positioning of Virtual Clothing.
  Proceedings of the Spring Conference on Computer Graphics, pages 113–122, April 2003.

- Arnulph Fuhrmann, Clemens Gross, and Volker Luckas:
  Interactive Animation Of Cloth Including Self Collision Detection.
  Journal of WSCG, 11(1):141–148, February 2003.

A.3 Nationale Veröffentlichungen

- Arnulph Fuhrmann:
  Automatische Vorpositionierung von Schnittteilen an individuellen 3D-Figurinen zur Faltenwurfsimulation
  Diplomarbeit am Fachbereich Informatik, TU-Darmstadt, 2001

A.4 FhG - Veröffentlichungen

- Clemens Groß, Arnulph Fuhrmann:
  Automatic Pe-positioning and Physically Based Simulation of Cloth
  Computer Graphik Topics, 6/2002 Vol. 14, 2002

- Arnulph Fuhrmann, Clemens Groß, Martin Knuth:
  Virtual Prototyping of Garments
  Computer Graphik Topics, 6/2005 Vol. 17, 2005

Anhang B

Betreute studentische Arbeiten

In diesem Anhang sind die vom Autor dieser Arbeit betreuten studentischen Arbeiten aufgeführt. Ergebnisse dieser Arbeiten sind in diese Arbeit eingeflossen.

B.1 Diplom- und Masterarbeiten

- Roland Sarrazin: Kollisionserkennung und -behandlung zwischen stark deformierbaren und starren Körpern, Diplomarbeit TU-Darmstadt, FG Graphisch Interaktive Systeme (GRIS), FB Informatik , Oktober 2002, Betreuer: A. Fuhrmann, Dr.-Ing. V. Luckas, Prüfer: Professor Dr.-Ing. José L. Encarnação

- Sebastian Gantzert: Effiziente Animation virtueller Bekleidung, Diplomarbeit TU-Darmstadt, FG Graphisch Interaktive Systeme (GRIS), FB Informatik, 2003, Betreuer: A. Fuhrmann, Dr.-Ing. V. Luckas, Prüfer: Professor Dr.-Ing. José L. Encarnação

- Martin Knuth: Interaktives Rendering von Bekleidung, Diplomarbeit TU-Darmstadt, FG Graphisch Interaktive Systeme (GRIS), FB Informatik, 2004, Betreuer: A. Fuhrmann, Dr.-Ing. V. Luckas, Prüfer: Professor Dr.-Ing. José L. Encarnação

- Michael Benz: Cluster basiertes Rendern mit verteilten Frame Buffern, Diplomarbeit TU-Darmstadt, FG Graphisch Interaktive Systeme (GRIS), FB Informatik, 2006, Betreuer: M. Knuth, A. Fuhrmann, Prüfer: Professor Dr.-Ing. José L. Encarnação

- Marco Hutter: Kollisionserkennung für mehrlagige Bekleidung, Diplomarbeit TU-Darmstadt, FG Graphisch Interaktive Systeme (GRIS), FB Informatik, 2006 Betreuer: A. Fuhrmann, Prüfer: Professor Dr.-Ing. José L. Encarnação

- Martin Feyrer: Adaptive Netze zur physikalisch basierten Animation von Bekleidung, Diplomarbeit FH Darmstadt, FB Informatik, Juni 2003, Betreuer: A. Fuhrmann, Prüfer: Prof. Dr.-Ing. W. Groch

- Stefan Böhm: Erzeugung von Avataren anhand von Körpermaßen für das Bekleidungsdesign, Masterarbeit FH Darmstadt, FB Informatik, April 2005, Betreuer: A. Fuhrmann, Prüfer: Prof. Dr. W. Kestner

- Frank Freund: Robuste Kollisionserkennung zur Bekleidungssimulation an interaktiv animierbaren Avataren, Masterarbeit FH Darmstadt, FB Informatik, April 2005, Betreuer: A. Fuhrmann, Prüfer: Prof. Dr.-Ing. W. Groch

B.2 Bachelorarbeiten

- Frank Freund: Animation von Avataren bei physikalisch basierten Kleidungssimulationen, Bachelor-Arbeit, FH Darmstadt, Betreuer: A. Fuhrmann, Dr.-Ing. V. Luckas, 2003, Prüfer: Prof. Dr.-Ing. W. Groch

- Andreas Altmannsberger: Interaktive Bearbeitung von Schnittteilen an virtuellen Menschen, Bachelor-Arbeit, FH Darmstadt, Betreuer: A. Fuhrmann, 2003, Prüfer: Prof. Dr. Frank

- Christina Gerstmann: Rendering virtueller Bekleidung mit Detailtexturen, Bachelor-Arbeit, FH Darmstadt, Betreuer: A. Fuhrmann, 2003, Prüfer: Prof. Dr.-Ing. W. Groch

- Tobias Franke: Kollisionserkennung von skelettanimierten Avataren mit Soffsimulationen, Bachelor-Arbeit, FH Darmstadt, Betreuer: A. Fuhrmann, 2004, Prüfer: Prof. Dr. H.P. Weber

- Reda El-Jazouli: Interaktive Größenänderung von Kleidungsstücken, Bachelor-Arbeit, FH Darmstadt, Betreuer: A. Fuhrmann, 2004, Prüfer: Prof. Dr. W.-D. Groch

B.3 Praktika

- Ivica Aracic: Efficient Handling of Dynamic Geometry in Java3D, Praktikum im Rahmen der Vorlesung GDV III an der TU Darmstadt, Betreuer: A.Fuhrmann, Dr.-Ing. V. Luckas, 2002

- Sebastian Eifert, Gert Kremser, Kai Stroh: Stencilbuffer- und schattenvolumenbasierte Schattendarstellung Praktikum im Rahmen der Vorlesung GDV III an der TU Darmstadt, Betreuer: A. Fuhrmann, T. May, Dr.-Ing. V. Luckas, 2002

- Ivica Aracic: Efficient Handling of Dynamic Geometry in Java3D, Praktikum im Rahmen der Vorlesung GDV III an der TU Darmstadt, Betreuer: A.Fuhrmann, Dr.-Ing. V. Luckas, 2002

- Marco Hutter: 3D-Simulation von Faltdächern Praktikum im Rahmen der Vorlesung GDV III an der TU Darmstadt, Betreuer: A. Fuhrmann, Dr.-Ing. V. Luckas, 2003

- Annette Göttmann, Sarah Kraatz, Bastian Wormuth: Implementierung des CG-Verfahrens für dünnbesetzte Matrizen, Praktikum im Rahmen der Vorlesung GDV III an der TU Darmstadt, Betreuer: A. Fuhrmann, Dr.-Ing. V. Luckas, 2003

- Victoria Engau: Optimized Spatial Hashing for Collision Detection, Praktikum im Rahmen der Vorlesung GDV III an der TU Darmstadt, Betreuer: A. Fuhrmann, Dr.-Ing. V. Luckas, 2004

- Robert Konrad: Integration von Accessoires (Nähte, Embleme, Knöpfe) in ein System zur Bekleidungssimulation, Praktikum der TU Darmstadt - Programmierung eines graphischen Systems, Betreuer: A. Fuhrmann, M. Knuth, 2005

- Bastian Linneweber: Interaktiver Abgleich von Materialparametern in der Stoffsimulation, Praktikum der TU Darmstadt - Programmierung eines graphischen Systems, Betreuer: A. Fuhrmann, M. Knuth, 2005

- Robert Konrad: Verbessertes Rendering von Kleidung durch geometrische Details, Praktikum der TU Darmstadt - Programmierung eines graphischen Systems, Betreuer: A. Fuhrmann, M. Knuth, 2006

Anhang C

Lebenslauf

Persönliche Daten

Arnulph Fuhrmann
Dresdener Str. 38, 64372 Ober-Ramstadt

geb. am 22. 08. 1973 in Wiesbaden
ledig, deutsch

Schulbildung

08/1980–07/1984	Grundschule in Fulda
08/1984–06/1993	Rabanus-Maurus-Schule in Fulda (Gymnasium) Abschluss: Abitur

Studium

10/1993–10/2001	Informatik an der TU-Darmstadt mit Nebenfach Mathematik, Abschluss als Diplom Informatiker
10/1997–10/1998	Studienunterbrechung zur Ableistung des Zivildienstes beim CBF Darmstadt e.V.

Berufliche Tätigkeiten

03/1996–04/2001	Studienbegeleitende Tätigkeit als wissenschaftliche Hilfskraft am ZGDV e.V., an der TU-Darmstadt und am Fraunhofer IGD
12/2001-01/2005	Wissenschaftlicher Mitarbeiter in der Abteilung Echtzeitlösungen für Simulation und Visual Analytics des Fraunhofer IGD, Darmstadt
seit 02/2005	Stellvertretender Abteilungsleiter in der Abteilung Echtzeitlösungen für Simulation und Visual Analytics des Fraunhofer IGD, Darmstadt