Echtzeitdeformation und Kollisionserkennung zur virtuellen Operationssimulation

Diplomarbeit

vorgelegt von

Christian Wienss

Institut für Computervisualistik
Arbeitsgruppe Computergraphik

Competence Center
Virtual Environments

Betreuer: Dr. Gernot Goebbels, Fraunhofer IMK.VE
Dr. Igor Nikitin, Fraunhofer IMK.VE
Prüfer: Prof. Dr. Stefan Müller, Universität Koblenz-Landau

Dezember 2004

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass die vorliegende Arbeit selbständig verfasst wurde und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet wurden. Die Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

....., den ....

(Ort,Datum) (Unterschrift)

Danksagung

An dieser Stelle möchte ich mich für die vielseitige mir zugewandte Hilfe bedanken, ohne die eine Arbeit wie die vorliegende nicht realisierbar gewesen wäre.

Zum Ersten möchte ich mich bei meinem Arbeitskollegen Klaus Bernhard Troche bedanken, der mir mit so vielen kleinen und großen Hilfen zur Seite stand, dass ich ohne Ihn oft verzweifelt wäre. Der selbe Dank gilt meinen Betreuern Dr. Gernot Goebbels und Dr. Igor Nikitin, die mich auf die spannende Aufgabenstellung aufmerksam gemacht haben und immer für Fragen offen waren.

Mein großer Dank gilt auch Herrn Prof. Dr. Stefan Müller, der sich trotz vieler Termine immer Zeit für mich genommen hat und mir stets mit gutem Rat und Hilfe beistand.

Vielen Dank an Albert Schaeffer von der PolyDimensions GmbH für das zur Verfügung stellen von medizinischen Datensätzen.

Bei allen Mitarbeitern der Abteilung Virtual Environments des Fraunhofer Instituts für Medienkommunikation in St. Augustin möchte ich mich für die angenehme und kollegiale Arbeitsatmosphäre bedanken.

Zu guter Letzt möchte ich in Liebe und Dankbarkeit meine Eltern für die Ermöglichung des Studiums aufführen.

Christian Wienss

Inhaltsverzeichnis

1 Einleitung

In virtuellen Simulationsumgebungen ist es wichtig, dass sich Objekte so realitätsnah wie möglich verhalten. Dies ist insbesondere bei der Simulation von chirurgischen Operationen unabdingbar. Hierzu muss nicht nur eine sehr genaue Kollisionserkennung stattfinden, sondern auch ein realitätsnahes Objektverhalten simuliert werden. Zudem kann ein solches System nur zur Anwendung kommen, wenn es echtzeitfähig ist. Die Immersion und die Benutzerführung darf nicht durch Latenzen gestört werden.

Dieses sich sehr schnell entwicklende Teilgebiet der VR stellt die Simulation chirurgischer Eingriffe dar. Diese dienen sowohl dem Planen und dem Training chirurgischer Eingriffe als auch der Resultatsanalyse solcher Operationen. Um einen möglichst genauen Realitätseindruck zu vermitteln, ist es wichtig, den Anwender speziell in seiner visuellen und kinästhetischen Wahrnehmung des Sachverhaltes anzusprechen. Deshalb besteht ein chirurgischer Simulator im Wesentlichen aus den folgenden drei Hauptkomponenten:

1. Die grafische Schnittstelle
   Sie soll einen möglichst realistischen visuellen Eindruck des simulierten Eingriffs vermitteln.

2. Die haptische Schnittstelle
   Diese stellt kraftrückkoppelnde Devices dar und dient der reinen Vermittlung kinästhetischer Eindrücke (Härte von Knochen, Widerstand von weichem Bindegewebe und beispielweise härterem Muskelgewebe etc.) an den Benutzer.

3. Physikalische Modellierung
   Diese Modellierung bildet das Herzstück eines chirurgischen Simulators. Ihre Aufgabe ist es, eine auf physikalischen Grundlagen basierende Interaktion mit dem zu examinierenden Gegenstand zu erzeugen. Dazu gehören in diesem Zusammenhang elastische Deformationen von menschlichem Gewebe, Darstellung von Blutungen, Schnitte in Gewebe etc..

Diese Diplomarbeit soll sich mit der physikalischen Simulation von Gewebeverhalten unter Krafteinwirkungen auseinandersetzen. Zur haptischen Vermittlung von Kräften und zur korrekten Deformation von flexiblen Materialien an starren Objekten wird eine genaue Kollisionserkennung und -behandlung benötigt. Zur genauen Errechnung und Darstellung der Deformation ist die präzise Bestimmung der Eindringtiefe des flexiblen Objekts in das starre Objekt entscheidend. Hier gibt es bislang zwei Varianten:

- Näherungsberechnung von Eindringtiefen und Kollisionsflächen auf Kosten der Genauigkeit (echtzeitfähig)
- Exakte Berechnungen der Eindringtiefen und Kollisionsflächen zu Lasten der Geschwindigkeit (nicht echtzeitfähig)

Das Ziel dieser Arbeit ist, einen echtzeitfähigen Algorithmus zur Berechnung der Eindringtiefe und der Kollisionspunkte zu entwickeln, welcher diese an einen Deformationssimulator übergibt. Dieser kann in Folge dessen eine korrekte Deformation durchführen. Die hohe zeitliche Komplexität dieser Berechnungen soll durch geeignete Ansätze gesenkt werden.

Die entwickelten Verfahren und Programme sollen soweit möglich an medizinischen Datensätzen eingesetzt und validiert werden.

Kapitel 2 (*Grundlagen*) erläutert den mathematischen und physikalischen Hintergrund dieser Arbeit.

Im Kapitel 3 (*Methodik*) werden bereits realisierte Ansätze erläutert, auf die die spätere Arbeit aufbaut. Hierbei werden sowohl die Bereiche der Kollisionserkennung als auch der Deformation dargestellt.

Unter Kapitel 4 (*Vorgehensweise*) wird dargestellt, wie und warum spezielle Algorithmen und Ansätze weitergeführt und neu entwickelt werden. Bestimmende Faktoren sind hierbei vor allem die Echtzeitaspekte und die Genauigkeit des Verfahrens.

Gegenstand des Kapitels 5 (*Implementation*) ist die genaue Beschreibung der Umsetztung der Lösungsidee. Hierbei wird eine verständliche Erklärung der Auflistung von Codefragmenten vorgezogen.

Die Resultate dieser Arbeit werden in Kapitel 6 (*Ergebnisse*) präsentiert. Diese werden hauptsächlich in Bildern mit Informationen zur Echtzeitfähigkeit dargestellt.

In den Kapiteln 7 (*Zusammenfassung und Ausblick*) wird die Arbeit mit ihren Ergebnissen knapp zusammengefasst und später einige Möglichkeiten der Arbeitsweiterführung dargestellt.

Der Anhang beinhaltet die Erklärung wichtiger Begriffe und der Vorstellung der Simulationsumgebung AVANGO.

2 Grundlagen

In diesem Kapitel wird der mathematische und physikalische Hintergrund dieser Arbeit erläutert, auf den die Simulationsumgebung später aufbaut.

2.1 Finite Elemente

Die Finite-Elemente-Methode (FEM)[1] ist ein numerisches Verfahren zur näherungsweisen Lösung von partiellen Differentialgleichungen mit Randbedingungen. Das untersuchte Lösungsgebiet wird zunächst in Gitterzellen, die finiten Elemente eingeteilt. Innerhalb des Finiten Elements werden für die gesuchte Lösung je n Ansatzfunktionen definiert, die nur auf endlich vielen der Gitterzellen ungleich Null sind. Durch eine Linearkombination der n Ansatzfunktionen innerhalb des Elementes werden die möglichen Lösungen der numerischen Näherung festgelegt. Die Differentialgleichungen und die Randbedingungen werden mit Gewichtungsfunktionen multipliziert und über das Lösungsgebiet integriert. Das Integral wird durch eine Summe über einzelne Integrale der Finiten Elemente ersetzt. Da die Ansatzfunktionen nur auf wenigen der Elemente ungleich Null sind, ergibt sich ein dünnbesetztes, häufig sehr großes Gleichungssystem, bei dem die Faktoren der Linearkombination unbekannt sind. Dieses wird dann mittels spezieller Methoden der numerischen linearen Algebra gelöst [2]. In dieser Arbeit wird die Verwendung von FEM aus vier Knoten bestehenden Tetraedern realisiert. Die Definition basiert auf Bro-Nielsen et al.[3].

Die Knoten jedes Elemtes Ω^e werden als P_i^e bezeichnet, wobei i die Nummer des

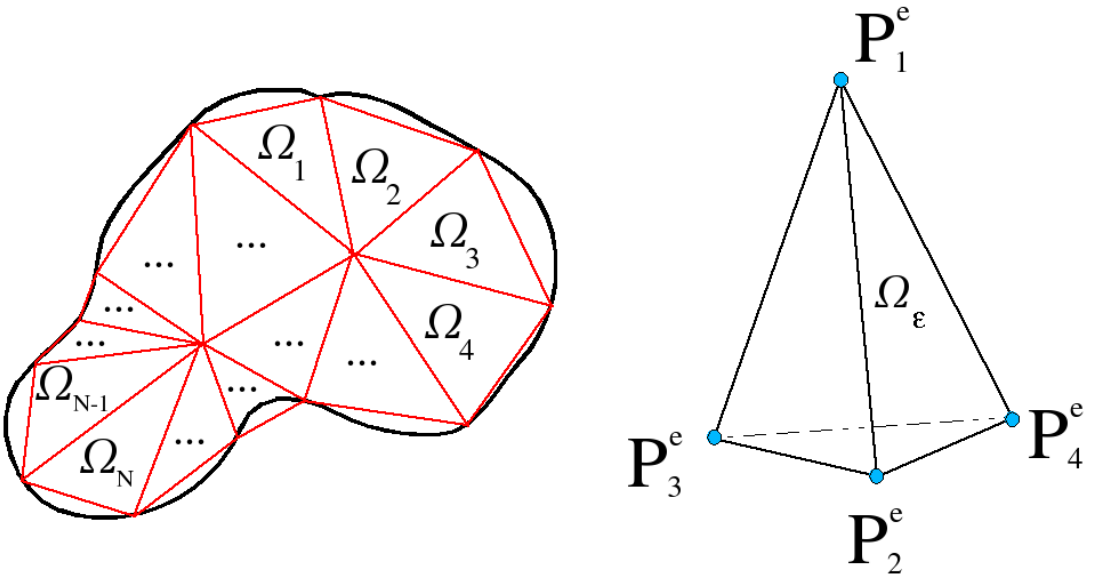

Abbildung 2.1: Links die Diskretisierung des Objekts in Finite Elemente, zur besseren Anschauung zweidimensional. Rechts die in dieser Arbeit zur Unterteilung verwendete Tetraederstruktur.

Knotens innerhalb des Tetraeders darstellt. Die globale Nummerierung ist hiervon unabhängig. Als Finite Elemente fungieren in dieser Arbeit Tetraeder, zwischen deren Stützpunkten mit linearer Interpolaion das Veschiebungsfeld (vgl. Abb. 4.4 und 4.5) errechnet wird:

$$u^e(x) = \sum_{i=1}^{4} N_i^e(x) u_i^e.$$

Die Basisfunktionen $N_i^e(x)$ sind definiert als die sogenannten natürlichen oder baryzentrischen Koordianten L_i der Tetraeder:

$$N_i^e(x) = L_i = \frac{1}{6V^e}(a_i^e + b_i^e x + c_i^e y + d_i^e z) \text{ mit } i = 1, 2, 3, 4$$

oder

$$\begin{bmatrix} L_1 \\ L_2 \\ L_3 \\ L_4 \end{bmatrix} = \frac{1}{6V^e} \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_4 & b_4 & c_4 & d_4 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix}$$

mit

$$6V = \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{vmatrix} \qquad a_1 = \begin{vmatrix} x_2 & x_3 & x_4 \\ y_2 & y_3 & y_4 \\ z_2 & z_3 & z_4 \end{vmatrix}$$

$$b_1 = -\begin{vmatrix} 1 & 1 & 1 \\ y_2 & y_3 & y_4 \\ z_2 & z_3 & z_4 \end{vmatrix} \qquad c_1 = \begin{vmatrix} 1 & 1 & 1 \\ x_2 & x_3 & x_4 \\ z_2 & z_3 & z_4 \end{vmatrix}$$

$$d_1 = -\begin{vmatrix} 1 & 1 & 1 \\ x_2 & x_3 & x_4 \\ y_2 & y_3 & y_4 \end{vmatrix}$$

Bei der Verwendung von Relativkoordinaten müssen im Falle einer Deformation nur die Koordinaten der Eckpunkte berechnet werden, die Koordinaten des Verschiebungsfeldes bleiben gleich. Die anderen Koordinaten erhält man durch zyklische Vertauschung der Indizes mit Vorzeichenwechsel. Das Verschiebungsfeld (engl.: *displacement-field*) ist der Überlappungsraum zwischen Objekten. Die Oberflächenpunkte des flexiblen Objekts müssen um dieses Feld verschoben werden.

2.2 Tensoren

Für manche Anwendungen, zum Beispiel in der Elastizitätstheorie, ist es vollkommen ausreichend, sich Tensoren als eine Verallgemeinerung von Skalar, Vektor, Matrix vorzustellen [4]. Dabei unterscheidet man Tensoren verschiedener Stufe (auch Rang genannt):

- Ein Tensor nullter Stufe ist ein Skalar.
- Ein Tensor erster Stufe wird durch einen Vektor dargestellt; im n-dimensionalen Raum hat ein solcher Tensor genau n Koeffizienten.

- Ein Tensor zweiter Stufe wird durch eine quadratische Matrix dargestellt, also ein Zahlenschema, in dem jeder der n^2 Koffizienten des Tensors durch zwei Indizes bezeichnet ist.
- Ein Tensor dritter Stufe ließe sich durch eine würfelförmige Anordnung seiner n^3 Koeffizienten darstellen, die durch je drei Indizes *adressiert* werden.
- Ein Tensor m-ter Stufe hat dementsprechend n^m Koeffizienten, die mit Hilfe von m Indizes auseinander gehalten werden.

2.3 Materialeigenschaften

Modelldeformationen auf Basis der Finiten Elemente können elementspezifisch definiert werden: jedes Element kann eine andere Materialeigenschaft haben. So können komplexe Modellstrukturen simuliert werden. Hierzu wäre die Entwicklung eines Verfahrens sinnvoll, welches beispielsweise aus MR oder MRI Daten direkt eine Volumendarstellung mit den passenden Materialeigenschaften extrahieren könnte. Da wir uns auch mit der Simulation von erkrankten Organen befassen, müssen Verhärtungen oder Aufschwemmungen erkennbar und darstellbar sein. Manche Materialien verhalten sich bei geringen Deformationen allerdings wie eine homogene Struktur, auch wenn die Innenstruktur heterogen ist. Dies ist z.B. bei der Simulation des Deformationsverhaltens von Niere und Leber der Fall. Bei der Definition von Material genügt eine Angabe von drei Eigenschaften:

- Dichte des Materials (kg/m^3)
- Elastizitätsmodul (Pa)
- Reziproke Querkontraktion (Poisson Verhältnis)

2.3.1 Elastizitätsmodul

Der Elastizitätsmodul, auch Young's Modulus genannt, ist ein Materialkennwert aus der Werkstofftechnik, der den Zusammenhang zwischen Spannung und Verfor-

mung (meist Dehnung) bei der mechanischen Beanspruchung eines festen Körpers beschreibt [4]. Der Zahlenwert des Elastizitätsmoduls ist um so größer, je mehr Widerstand ein Material seiner Verformung entgegensetzt. Ein Material mit hohem Elastizitätsmodul ist also steif, ein Material mit niedrigem Elastizitätsmodul ist weich. Der Elastizitätsmodul ist als Steigung des Graphen im Spannungs-Dehnungs-Diagramm innerhalb des Elastizitätsbereichs definiert.

$$E = \frac{d\sigma}{d\varepsilon}$$

Dabei bezeichnet σ die mechanische Spannung (Zugspannung, nicht Schubspannung) und ε die Dehnung. Die Dehnung ist das Verhältnis von Längenänderung zur ursprünglichen Länge. Die Einheit ist die einer Spannung:

$$[E] = \frac{N}{mm^2}$$

Bei linearem Verlauf des Spannungs-Dehnungs-Graphen (Proportionalitätsbereich) gilt:

$$E = \frac{\sigma}{\varepsilon}$$

Im Prinzip ist das nur eine andere Schreibweise für das Hookesche Gesetz, wobei der Elastizitätsmodul der Federkonstante entspricht. Der Elastizitätsmodul hängt von verschiedenen Umgebungsbedingungen wie z. B. dem Druck oder der Temperatur ab, die auf die Materialeigenschaften Einfluss haben. Im Rahmen dieser Arbeit wird er jedoch wegen der dadurch entstehenden geringen Änderung als konstant betrachtet.

Beispiele: Elastizitätsmodul von

- Stahl ca.: 190 000 bis 210 000 N/mm2 (bei der Raumtemperatur)
- Messing: 78 000 bis 123 000 N/mm2
- Beton: 40 000 bis 45 000 N/mm2
- Holz, parallel zur Faser: 9 000 bis 12 000 N/mm2
- Holz, quer zur Faser: 300 bis 1 000 N/mm2
- Silikonkautschuk: 10 bis 100 N/mm2

2.3.2 Querkontraktion

Die Querkontraktion ist ein Spezialfall der Deformation. Sie beschreibt das Verhalten eines Körpers unter dem Einfluss einer Zugkraft [4]. In Richtung der Kraft reagiert der Körper mit einer Längenänderung Δl, senkrecht dazu mit einer Verringerung seines Durchmessers d um Δd. Die Längenänderung kann durch das vereinfachte Hookesche Gesetz bestimmt werden. Über die Dickenänderung macht das Hookesche Gesetz in seiner vereinfachten Form jedoch keine Aussagen.

Dennoch kann häufig auf die kompliziertere Anwendung des allgemeinen Hookeschen Gesetzes verzichtet werden, da in vielen Fällen die relative Durchmesseränderung $\frac{\Delta d}{d}$ proportional zu der über das vereinfachte Hookesche Gesetz bestimmbaren relativen Längenänderung $\frac{\Delta l}{l}$ ist:

$$\frac{\Delta d}{d} = \mu \frac{\Delta l}{l}$$

Der Proportionalitätsfaktor μ ist eine dimensionslose Größe und heißt Poissonzahl oder auch Querkontraktionszahl.

Im Gültigkeitsbereich der Proportionalität zwischen Längen- und Dickenänderung erlaubt die Poissonzahl auch eine Berechnung der relativen Volumenänderung $\frac{\Delta V}{V}$, mit der ein Körper auf Dehnung reagiert:

$$\frac{\Delta V}{V} = (1 - 2\mu) \frac{\Delta l}{l}$$

Physikalisch sinnvolle Werte für μ liegen zwischen -1 und 0,5 (-1 $< \mu <$ 0,5). Negative Werte ergeben eine Querdehnung; entgegen der weitverbreiteten Meinung gibt es solches Verhalten tatsächlich z. B. bei gewissen Polymerschäumen. Werte über 0,5 würden eine Verkleinerung des Volumens bei Längsdehnung beschreiben. Das kann aus energetischen Gründen nicht auftreten.

Die Poissonzahl ist eine Materialkonstante, d.h. sie ist abhängig vom Material des verwendeten Werkstücks. Sie kann aus den elastischen Konstanten des jeweiligen Materials berechnet werden. Typische Werte der Poissonzahl liegen zwischen 0,3 und 0,4.

Allgemein ist die Dickenänderung, mit der ein Körper auf eine angelegte mechanische Spannung reagiert nicht in alle Richtungen gleich, auf die erzwungene Län-

genänderung kann ein Körper z.B. mit unterschiedlicher Änderung von Höhe und Breite reagieren. Dies ist insbesondere bei kristallinen Festkörpern zu beachten. Wenn es auf diese Unterschiede ankommt, so muss das Hookesche Gesetz in seiner allgemeinen Form angewandt werden.

Darüber hinaus gelten bei der Behandlung der Querkontraktion die selben Einschränkungen wie beim Hookeschen Gesetz selbst: sie gilt nur für lineare elastische Deformationen [4].

2.4 Das Hookesche Gesetz

Das Hookesche Gesetz [4] besagt, dass eine elastische Deformation ε eines Körpers proportional zur anliegenden Spannung σ ist. Im allgemeinen Fall wird das Hookesche Gesetz ausgedrückt durch die lineare Tensorgleichung

$$\sigma = \tilde{\tilde{C}}\varepsilon,$$

mit dem Elastizitätstensor $\tilde{\tilde{C}}$, der die elastischen Eigenschaften der deformierten Materie kennzeichnet.

Das Hookesche Gesetz gilt nur für lineare elastische Deformationen. Diese Bedingung ist in der Regel für kleine Deformationen erfüllt. Bei Deformationen oberhalb der so genannten Proportionalitätsgrenze werden die Verformungen nicht-linear, d.h. die Verzerrung ε ist nicht mehr proportional zur Verspannung σ, die Verformung kann aber dennoch reversibel sein. Erst für noch größere Deformationen wird die Verformung irreversibel (plastische Deformation), und es findet keine vollständige Rückformung beim Nachlassen der Spannung statt.

2.4.1 Verallgemeinertes Hookesches Gesetz

Die allgemeine Form des Hookeschen Gesetzes als lineare Tensorgleichung mit dem Elastizitätstensor (4. Stufe) $\tilde{\tilde{C}}$ mit 81 Komponenten $C_{ijkl},\ i,j,k,l = 1\ldots3$ ist

schwierig zu handhaben. Aufgrund der Symmetrie von Verzerrungs- und Spannungstensor reduziert sich die Zahl der unabhängigen Komponenten jedoch auf 36. Damit lässt sich das Hookesche Gesetz in eine einfacher zu handhabende Matrixgleichung überführen, wobei die elastischen Konstanten in einer 6×6-Matrix, sowie die Verzerrung und die Verspannung als sechskomponentige Vektoren dargestellt werden (Voigtsche Notation):

$$\begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46} \\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56} \\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66} \end{pmatrix} \cdot \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \end{pmatrix} .$$

Aus energetischen Überlegungen ergibt sich, dass auch diese 6×6-Matrix symmetrisch ist. Die Anzahl der unabhängigen $C_{ij},\ i, j = 1 \ldots 3$ (elastische Konstanten) reduziert sich damit weiter auf maximal 21.

Die maximal sechs unabhängigen der beiden symmetrischen Tensoren für Dehnung und Spannung werden häufig in der folgenden Weise auf zwei sechskomponentige Vektoren verteilt:

$$\begin{gathered} \epsilon_{xx} := \varepsilon_x\,,\ \epsilon_{yy} := \varepsilon_y\,,\ \epsilon_{zz} := \varepsilon_z\,,\ \epsilon_{xy} = \epsilon_{yx} := \sqrt{2}\gamma_{xy}\,, \\ \epsilon_{yz} = \epsilon_{zy} := \sqrt{2}\gamma_{xz}\,,\ \epsilon_{zx} = \epsilon_{xz} := \sqrt{2}\gamma_{yz}\,, \end{gathered}$$

mit $\gamma \in \varepsilon$, also

$$\varepsilon^T = \left(\varepsilon_x\,,\ \varepsilon_y\,,\ \varepsilon_z\,,\ \sqrt{2}\gamma_{xy}\,,\ \sqrt{2}\gamma_{xz}\,,\ \sqrt{2}\gamma_{yz}\right)\,,$$

und analog

$$\sigma^T = \left(\sigma_x\,,\ \sigma_y\,,\ \sigma_z\,,\ \sqrt{2}\tau_{xy}\,,\ \sqrt{2}\tau_{xz}\,,\ \sqrt{2}\tau_{yz}\right) \text{ mit } \tau \in \varepsilon.$$

Der Faktor $\sqrt{2}$ ist notwendig, um Übereinstimmung zwischen der hier eingeführten Matrix/Vektor-Darstellung Tensorgleichung und der Tensorgleichung $\sigma = \tilde{\tilde{C}}\varepsilon$ herzustellen. Statt $\sqrt{2}$ bei den Vektordarstellungn von sowohl Verzerrung als auch Verspannung kann auch der Faktor 2 bei nur einem der beiden Vektoren verwendet werden.

2.4.2 Isotrope Medien

Im Spezialfall isotroper Medien reduziert sich die Anzahl der unabhängigen elastischen Konstanten von 21 auf 2. Wesentliche Eigenschaften der Deformation lassen sich dann durch die Querkontraktionszahl charakterisieren. Das Hookesche Gesetz lässt sich dann darstellen in der Form

$$\bar{\varepsilon} = L^{-1}\bar{\sigma}, mit L^{-1} = \frac{1}{E}\begin{bmatrix} 1 & -\nu & -\nu & 0 & 0 & 0 \\ \cdot & 1 & -\nu & 0 & 0 & 0 \\ \cdot & \cdot & 1 & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & 1+\nu & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & 1+\nu & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1+\nu \end{bmatrix}, bzw.$$

$$L = \frac{E}{1+\nu}\begin{bmatrix} \frac{1-\nu}{1-2\nu} & \frac{\nu}{1-2\nu} & \frac{\nu}{1-2\nu} & 0 & 0 & 0 \\ \cdot & \frac{1-\nu}{1-2\nu} & \frac{\nu}{1-2\nu} & 0 & 0 & 0 \\ \cdot & \cdot & \frac{1-\nu}{1-2\nu} & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & 1 & 0 & 0 \\ \cdot & \cdot & \cdot & 0 & 1 & 0 \\ \cdot & \cdot & \cdot & 0 & 0 & 1 \end{bmatrix},$$

wobei E der Elastizitätsmodul und ν die Querkontraktionszahl sind. Beide sind durch den Werkstoff bestimmt. Für eindimensionale Deformationen vereinfacht sich die Beziehung zu

$$\varepsilon = \frac{1}{E}\sigma.$$

2.4.3 Vereinfachtes Hookesches Gesetz

Das Hookesche Gesetz gilt für einen großen Dehnungsbereich bei Zug- und Druckfedern. In diesem Spezialfall einer eindimensionalen linearen elastischen Deformation vereinfacht sich der Elastizitätsmodul E zur Federkonstanten D, die Verzerrung ε des Körpers zu seiner relativen Längenänderung $\frac{\Delta l}{l}$ und statt der mechanischen Spannung σ (Kraft pro Angriffsfläche) kann direkt die angelegte Kraft F angegeben werden. Das Hookesche Gesetz kann dann in der einfachen Form

$$\Delta l = \frac{1}{D} F$$

als eine lineare Relation zwischen der angelegten Kraft F und der daraus resultierenden Längenänderung Δl dargestellt werden (Hookesche Gerade). Bei Berechnung der rücktreibenden Kraft kehrt sich das Vorzeichen um ($F_r = -D\Delta l$).

2.5 Linearität in der Deformation

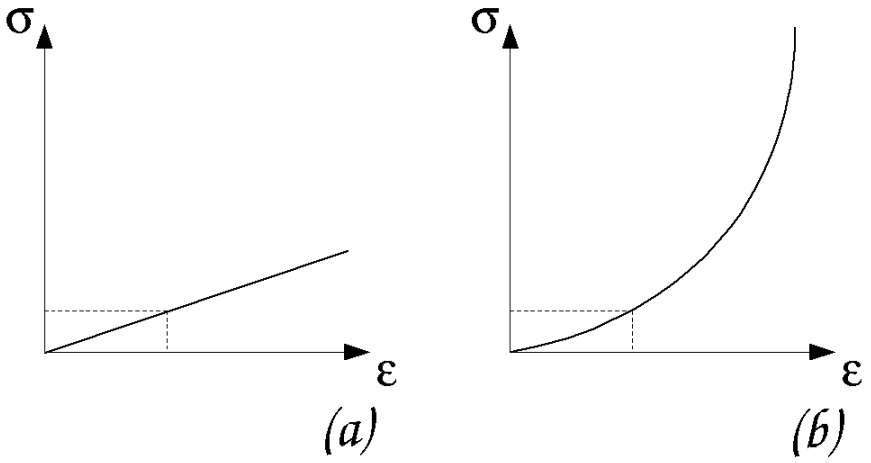

Abbildung 2.2: Lineares Dehnungs-Spannungsverhältnis (a), nichtlinear Fall (b). Bei kleinen Verschiebungen lässt sich die nichtlineare Deformationseigenschaft noch gut nähern (umrandete Bereiche). Definition σ und ε aus Abschnitt 2.4.

Es gibt zwei Arten der Nichtlinearität [5], die materialbedingte und die geometrisch bedingte [6]. Die materialbedingte Nichtlinearität, hervorgerufen durch Verletzung der Dehnungs-Spannungsrelation (Hookesches Gesetz (vgl. Abschnitt 2.4)), tritt meist unter Extrembelastungen auf, wenn das Material seine elastischen Eigenschaften verliert. Die geometrisch bedingte Nichtlinearität tritt bei starken Verschiebungen der Knoten untereinander auf. *Beispiel:* Gegeben sind zwei nahe beieinander liegende Punkte. Der Eine wird verschoben mit $x \rightarrow x+\gamma(x)$, der Andere mit $(x+\delta x) \rightarrow (x+\delta x)+\gamma(x+\delta x)$. Zieht man die beiden Positionen voneinader ab, erhält man $\delta x \rightarrow \delta x+\gamma(x+\delta x)-\gamma(x) = \delta x+\delta\gamma$. Das Quadtrat der Distanz hat sich geändert zu $\delta x^2 \rightarrow \delta x^2 + 2\delta x * \delta\gamma + \delta\gamma^2$. Für kleine Deformationen kann man den Term $\delta\gamma^2$ in der Regel weglassen, so bleibt die Gleichung linear. Bei großen Deformationen, z. B. wenn die Differenz der Deformationen $\delta\gamma$ in die Größenordnung des Abstands zwischen den beiden Punkten kommt, kann man diesen Term nicht mehr vernachlässigen, die Deformation wird nichtlinear. Diese Eigenschaft ist von der Materialeigenschaft unabhängig und ist die rein geometrische Nichtlinearität. Aus diesem Beispiel wird deutlich, warum Objekte, die ähnliche Ausdehnungen in

allen Dimensionen haben, linear deformierbar sind. Deformationen dieser Objekte, die in ihrem Deformationsausmaß vergleichbar mit ihren Objektgrößen sind, sind in der Realität sehr schwer durchführbar. Hingehen kann man eine plattenähnliche Form z.B. in einen Zylinder rollen oder aus einem Stab eine Spirale machen (vlg. Abb. 2.3).

In Folge dessen kann hier die Nichtlinearität vernachlässigt und die *Theorie der kleinen Deformationen* kann hierauf angewendet werden [6]. Objekte mit ungleichmäßiger Ausdehnung können so auch dargestellt werden, allerdings nur unter Einschränkung auf sehr kleine Deformationen. Über diese Einschränkungen hinausgehende Anforderungen müssen entweder durch das Hinzufügen von allgemeinen nichtlinearen Methoden [7] [8] [9] oder objektspezifische Näherungen realisiert werden.

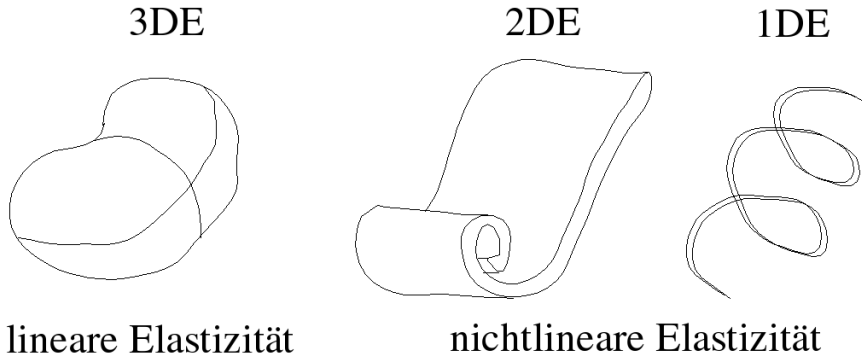

Abbildung 2.3: Verschiedene Elastizitätstypen, dimensionsabhängig.

2.6 Greensche Funktionen

Eine Greensche Funktion ist ein Integralkern, welcher zur Lösung von inhomogenen Differenzialgleichungen unter Randbedingungen verwendet werden kann.

In dieser Arbeit wird das lineare System der Form $\mathcal{K}u = f$ anhand der Greenschen Funktion gelöst. f wird durch δ_{ik} [1] substituiert, welcher an k-ter Position eine

[1] Kronecker Delta $\delta_{ik} = \begin{cases} 1, & \text{falls } i = k \\ 0, & \text{falls } i \neq k \end{cases}$

1 beinhaltet, ansonsten mit Nullen gefüllt ist. Die entsprechende Lösungsvektoren werden als $u^{(k)}$ bezeichnet. Diese Vektoren heißen Greensche Funktionen des Systems $\mathcal{K}u = f$.

In dieser Arbeit ist der physikalische Hintergrund dieser Funktionen die Deformation des Körpers unter Anwendung von Kraft, welche auf den k-ten Knoten ausgeübt wird. Diese Kraft kann aus drei Dimensionen ausgeübt werden und die Deformation hat drei Komponenten, so können die Greenschen Funktionen in 3×3 - Blöcken dargestellt werden.

Das Konzept der Greenschen Funktionen ist sehr allgemein. In dieser Arbeit sind $u^{(k)}$ die Spalten der inversen Matrix $\mathcal{K}^{-1}$. Wenn alle Greenschen Funktionen des linearen Systems bekannt sind, kann dieses System für jeden Vektor f schnell gelöst werden, da f immer in die Linearkombination δ_{ik} zerlegt werden kann. Die Zugehörige Lösung u ist nur noch eine Linearkombination der Greenschen Funktionen.

Zusammengefasst kann man die Greenschen Funktionen als die Spalten der invertierten $\mathcal{K}$-Matrix in dem System $u = \mathcal{K}^{-1} * f$ betrachten. Diese Definitionen gelten für diskrete und endliche Systeme, wie sie in dieser Arbeit zur Verwendung kommen. Die Hauptanwendung der Greenschen Funktion liegt in den kontinuierlichen Systemen:

Die Vektoren u_i und f_i können als die Werte einer kontinuierlichen Funktion gesehen werden, welche in einem Gitter diskretisiert wurden mit $u(x_i)$, $f(x_i)$. Wenn der Gitterabstand gegen 0 geht, werden die kontinuierlichen Grenzen erreicht, in der $u(x)$ und $f(x)$ Funktionen sind und $\mathcal{K}$ der lineare Operator. Analog zu der Matrixmultiplikation $\sum_j \mathcal{K}_{ij} * u_j = f_i$, kann der lineare Operator als $\int \mathcal{K}(x,y) * u(y) * dy = f(x)$ dargestellt werden, wobei $\mathcal{K}(x,y)$ den Integralkern des Operators darstellt.

Die oben definierte diskrete Greensche Funktion tendiert in kontinuierlichen Grenzen zur kontinuierlichen Greenschen Funktion, welches die Lösung folgender Gleichung ergibt:

$$\int \mathcal{K}(x,y) * G(y,z) * dy = \delta(x-z).$$

$\delta(x)$ ist die Dirac-Funktion, welche als hohe Wertekonzentration nahe $x = 0$ be-

trachtet werden kann mit $\int \delta(x) * dx = 1$. Dies ist die Analogie zu Kroneckers Symbol δ_{ij}. Wenn man die oben genannte Gleichung mit einer beliebigen Funktion $u(z)$ multipliziert, sieht man, dass $G(y, z)$ der Integralkern des inversen Operators $\mathcal{K}^{-1}$ ist. Der Zweck der Greenschen Funktionen in kontinuierlichen Systemen ist der selbe: alle Lösungen des linearen Gleichungssystems $\mathcal{K}u = f$ zu bestimmen. In dem Fall, in dem $\mathcal{K}$ ein Differenzialoperator (z.B. $\mathcal{K} = \frac{d}{dx}, \frac{d^2}{dx^2}, \frac{d^2}{dx*dy}$ usw.) ist, kann man die Greenschen Funktionen analytisch finden und die Differenzialgleichungen $\mathcal{K}u = f$ für beliebiges f lösen.

3 Motivation

In diesem Kapitel werden Vorgehensweisen bisheriger Ansätze erläutert, die im Rahmen dieser Arbeit modifiziert wurden.

3.1 Kollisionserkennung

Unter Berücksichtigung einiger standardisierter Kollisionserkennungstechniken [10] kamen aus Performanzgründen nur einige wenige kollisionserkennde Algorithmen in Frage (vgl. Abschnitt 4.1). In dieser Arbeit werden Teile der im Folgenden beschriebenen Algorithmen abgeändert verwendet.

3.1.1 Separierende Ebene

Definitionen:

- Gegeben:
$$P, Q \subseteq \mathbb{R}^3$$
- Erkennungsproblem (detection problem)
P und Q kollidieren:
$$P \cap Q \neq \emptyset \Leftrightarrow \exists x \in \mathbb{R}^3 : x \in P \wedge x \in Q$$

- Konstruktionsproblem (construction problem)

$$R := P \cap Q$$

- Definition Kollision für polygonale Objekte:
  P und Q kollidieren:

$$\exists f \in F^P, \exists f' \in F^Q : f \cap f' \neq \emptyset$$

Der Ansatz der Kollisionserkennung mit einer separierenden Ebene basiert auf der Idee, dass eine Ebene im Raum zwischen zwei Objekten definiert wird. Sollte eine solche Ebene existieren, so dass jedes Objekt vollständig einer Seite des Ebene zugeordnet werden kann, kann keine Kollision stattfinden. Diese Methode stellt einen sehr schnellen, aber groben und auf konvexe Objekte beschränkten Algorithmus dar. Ziel ist es nicht, eine separierende Ebene zu bestimmen, sondern zu beweisen, dass eine solche vorhanden sein muss: P liegt ganz auf der einen Seite von H_i, Q liegt ganz auf der anderen Seite:

$$P \cap Q = \bigcap_{i=1...n_1+n_2} H_i = \emptyset \Leftrightarrow \exists i : P \subseteq H_i \wedge Q \subseteq H_i^c$$

In vielen Fällen wird dieser Algorithmus auch so implementiert, dass die separierende Ebene bestimmt wird. In diesen Fällen versucht man zur Steigerung der Geschwindigkeit Kohärenzen zum vorherigen Betrachtungszeitpunkt herzustellen. Oft ist es so, dass die selbe gefundene trennende Ebene wiederverwendet werden kann. Die Suche nach einer solchen Ebene ist nicht deterministisch, man muss nach einer Anzahl von Schritten die Suche einstellen.

3.1.2 Voxelbasiertes Sampling

Zum haptischen Rendering wurde von der Firma Boeing [11] [12] der Voxmap-Pointshell™-Algorithmus (VPS) veröffentlicht, der einen einfachen, schnellen, nähernden und voxelbasierenden Ansatz mit sechs Freiheitsgraden vorstellt. Die hohen Ansprüche der Echtzeitkollisionserkennung mussten wegen des Einsatzes im haptischen Umfeld erweitert werden:

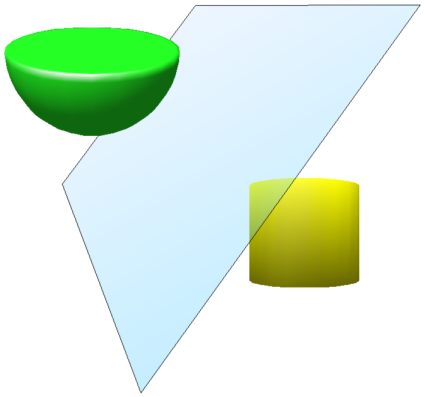

Abbildung 3.1: Kann man den Raum in zwei Teilräume unterteilen und jedes Objekt einem Teilraum vollständig zuordnen, kann keine Kollision stattfinden.

- Alle Oberflächenkontakte müssen erkannt werden, nicht nur einer.
- Eine Reaktionskraft muss daraufhin zusätzlich berechnet werden.
- Eine zuverlässige Wiederholrate von 1000 Hz muss gesichert sein.

Voxelbasierte Methoden wurden schon umgesetzt, aber nicht auf Basis von haptischen Schnittstellen [13] [14] [15].

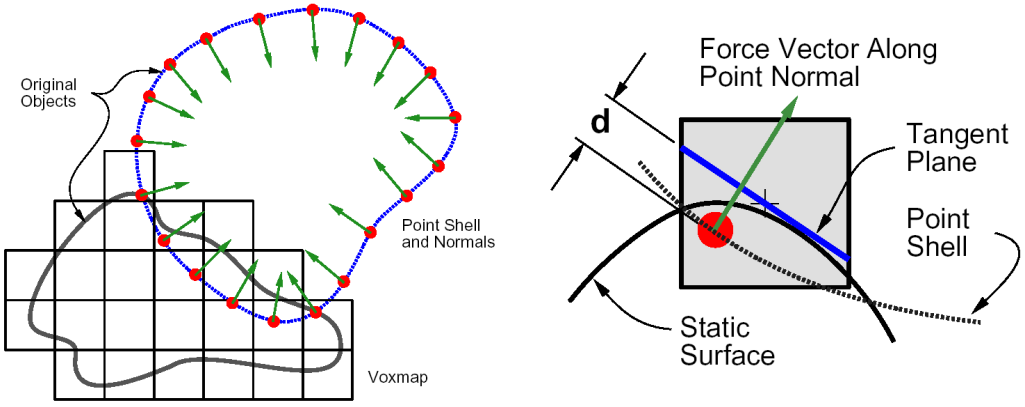

Abbildung 3.2: Links: Voxmap kollidiert mit der Punktschale, rechts tangentenbasiertes Kraftmodell [11].

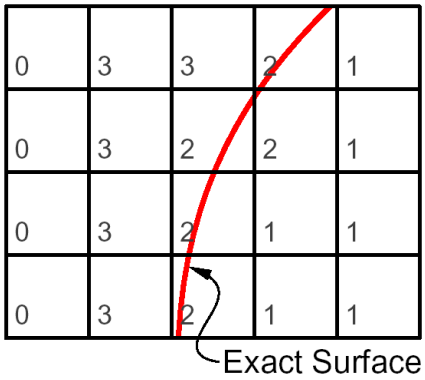

Abbildung 3.3: Zuweisung von 2-bit-Voxelwerten. 0 stellt den leeren Raum dar, 1 das Innere, 2 die Oberfläche und 3 oberflächennahe Voxel [11].

In diesem Ansatz werden dynamische Objekte durch ihre Oberflächenpunkte beschrieben, zu jedem Oberflächenpunkt wird die inverse Normale angegeben. Diese Darstellung wird als Punktschale (*point shell*) beschrieben. Die statische Umgebung ist durch die räumliche Belegung der statischen Objekte innerhalb einer Voxelumgebung beschrieben, die sogenannte *voxmap* (vgl. Abb. 3.3 und 3.2, links). Für jedes Voxel im Raum wird ein Wert definiert, ob er sich in der Nähe eines statischen Objekts befindet, die Oberfläche beinhaltet oder ob er sich innerhalb dessen befindet. Bei jedem Frame wird jeder Oberflächenpunkt gegen diese voxmap abgetastet. Wenn sich nun ein Punkt innerhalb eines Oberflächenvoxels befindet (vgl. Abb. 3.2, rechts) wird die Eindringtiefe als die Distanz d vom Punkt zur Tangentenebene errechnet. Die aus der Summe der kollidierenden Punkte genäherte Kraft wird an das haptische Interaktionsgerät zurückgeliefert. Der oben beschriebene und in [11] und [16] vorgestellte Ansatz wurde von Renz et al. [12] durch das Hinzufügen von folgenden Eigenschaften erweitert:

1. Ursprünglich wurde das Objekt in Voxel zerlegt und die Sammlung der Mittelpunkte aller Oberflächenvoxel bildete die Pointshell (vgl. Abb. 3.4, 1-3). Nun werden zur Steigerung der Genauigkeit und der Stabilität diese Punkte auf die Oberflächendreiecke abgebildet (vgl. Abb. 3.4, 4-6).

2. Es entstanden Kraftunterbrechungen, sobald ein Oberflächenpunkt die Voxelgrenzen überschreitet. Dieser Effekt trat vor allem dann auf, wenn eine

gleitende Bewegung auf der Oberfläche ohne Änderung der Eindringtiefe durchgeführt wurde und war durch Vibrationen des haptischen Ausgabegerätes zu erkennen. Hier wurde in diesen Grenzfällen durch Mittelung des Gesamtwiderstandes das Aussetzen der Kraft verhindert.

3. In McNeely et al. [11] und Adams and Hannaford [16] wurde das Feder-Dämpfer-Modell verwendet, welches eine gute mechanische Interpolation liefert, aber in den meisten Fällen zu einer heuristischen Optimierung der Parameter führt, da die Kraftsprünge reduziert werden, die durch Unterbrechungen und Diskretisierung dieser Werte entstehen. In Renz et al. [12] wurde eine mechanische Lösung des Feder-Dämpfer-Modells umgesetzt, die diese Probleme durch das Ersetzten der Translationsvektoren durch die Kollisionsvektoren realisiert.

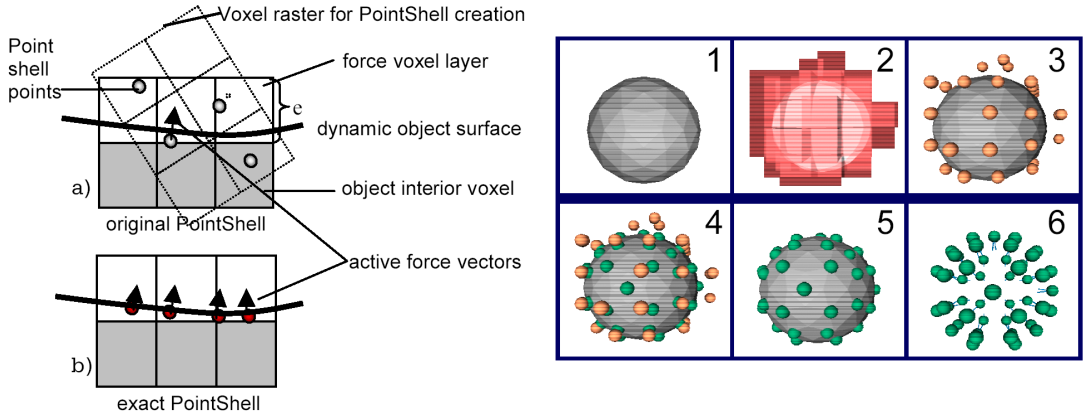

Abbildung 3.4: Links: Verschiedene Kollisionseffekte im Vergleich zwischen dem originalen Pointshell Algorithmus und dem durch Renz et al. erweiterten. Rechts: Schritte der exakten Pointshellerstellung [12].

3.1.3 Occupancy Map Method

Wegen der hohen Elementanzahl in volumetrischen Modellen wurde in der Arbeit von Dr. Gibson [17] ein Algorithmus vorgestellt, welcher voxelbasiert Überschneidungen erkennen kann [18]. Hierbei wird der gesamte Raum in Voxel aufgeteilt.

Befindet sich kein Objekt in dem Voxel, gibt es einen *NULL*-Pointer zurück. Sollten sich zwei Objekte in dem selben Voxel befinden, d.h. wenn ein Pointer eines Objekts auf ein anderes Objekt zeigt, findet eine Kollision statt. Selbstüberschneidung im Falle von Kollision kann so auch erkannt werden. Die Genauigkeit der Kollisionserkennung ist extrem von der Auflösung der Voxelstruktur abhängig.

3.2 Deformation

3.2.1 Volumetrische Modelle

Vor 1996 wurde im Bereich Deformation hauptsächlich mit Oberflächensimulationen gearbeitet. In einer Oberflächensimulation besteht das Modell nur aus Punkten, die sich auf der Oberfläche befinden. Diese sind durch Dreiecke miteinander verbunden. Im Inneren des Objekts befinden sich keine Punkte. Es kann also keinerlei Strukturinformation aus dem Inneren des Objekts bekannt sein. Dies führt zu zwei primären Problemen: Zum einen können sich Punkte bei der Deformation nur auf Nachbarn in einer Ebene stützen, was größere Deformationssimulationen physikalisch nicht mehr korrekt erscheinen lässt. Zum anderen können Topologieänderungen wie Schnitte und Risse nicht dargestellt werden, da bei einem Einschnitt in die Oberfläche nicht die darunter liegende Schicht, sondern nur die Rückseite der gegenüberliegenden Oberfläche zu sehen ist. Aus diesen Gründen sind Oberflächendatensätze zur Simulation chirurgischer Operationen nur bedingt einsetzbar.

Die Veröffentlichung von Bro-Nielsen und Cotin [3] war Pionierarbeit zur Volumensimulation. Unter Volumendatensätzen versteht man Modelle, die sowohl auf der Oberfläche als auch im Inneren des Objekts Punke besitzen. Diese sind durch Tetraeder miteinander verbunden. So kann Innenstruktur simuliert werden, welche bei Schnittsimulation eine realitätsnahe Situation darstellt. Ein weiterer großer Vorteil liegt in der Zuweisung von Materialeigenschaften. Jedes Element kann eine andere Beschaffenheit besitzen, so dass Strukturen im Inneren von Modellen simuliert werden können (zum Beispiel Nierensteine).

Auf Basis der Finiten Element Methode (vgl. Abschnitt 2.1) wurden drei Ansätze realisiert, die die Komplexität reduzieren und Rechenzeit durch Vorberechnungen verringern.

- **Verdichtung**
  Die Matrix, die aus dem volumetrischen Modell entstanden ist kann auf ein Verhalten der Oberflächenpunkte reduziert werden, welche dann ein Volumenverhalten simulieren (Beweis in [19]).

- **Invertierung**
  Invertierung der Matrix im Rahmen einer Vorberechnung und die Verwendung von Matrix-Vektormultiplikation zur Reduzierung der Berechnungszeit.
- **Selektive Matrix-Vektormultiplikation**
  Ausnutzung des dünn besetzten Kraftvektors.

Bro-Nielsen und Cotin haben außerdem eine Definition von Kriterien zur Echtzeitdeformation aufgestellt, welche für die von ihnen vorgelegte Arbeit als Leitfaden gilt:

1. **Geschwindigkeit**
   Das wichtigste Kriterium ist Geschwindigkeit, Deformation sollte in der kürztmöglichsten Zeit durchgeführt werden.
2. **Vorberechnungszeit**
   Wenn ein Algorithmus um 0.01 Sekunden verschnellert werden kann, ist ein höherer Vorberechnungsaufwand von 24 Stunden vertretbar. Es gilt, so viele Arbeitsschritte vorzuberechnen wie möglich.
3. **Optisch überzeugend**
   Laut Bro-Nielsen und Cotin soll eine Deformation vor allem optisch Überzeugen, physikalische Korrektheit steht im Hintergrund.
4. **Topologieänderungen**
   Ein Simulationssystem soll in der Lage sein, plastische Änderungen wie Schnitte am Modell darzustellen. Hierzu ist eine volumetrische Darstellung nötig.

Um die Punkte 1. und 2. zu realisieren, wurde das linear elastische Deformationsmodell (Hooksches Gesetz, vgl. Abschnitt 2.4) verwendet. Hiermit wurde sich auf geringe Deformation von Daten beschränkt, welche in allen drei Dimensionen eine ähnliche Ausdehnung haben, da starke Deformationen dünner Materialien linear nicht näherbar sind (vgl. Abschnitt 2.5).

Als Beispiel wurde in der Arbeit von Bro-Nielsen et al. ein auf Voxel basierender Datensatz manuell in finite Elemente unterteilt. Es konnte eine Deformationssimulation realisiert werden, die bei einer Anwendung von Kraft auf drei Knoten des Modells 20 Bilder pro Sekunde darstellen konnte.

Im Rahmen dieser Arbeit können zwei Kernpunkte aus den Thesen von Bro-Nielson et al. übernommen und ergänzt werden:

1. Geschwindigkeit gilt weiterhin als höchste Forderung und wird durch den Anspruch auf Echtzeitfähigkeit ergänzt.
2. Die Vorberechnungszeit ist auch in dieser Arbeit unkritisch. Alle Arbeitsschritte, die im Vorraus zu errechnen sind, werden vorberechnet und gespeichert.

Von Lemma **3.** muss sich im Rahmen dieser Arbeit distanziert werden, da gerade physikalische Korrektheit für chirurgische Operationssimulationen unabdingbar ist. Topologieänderungen (plastische Deformation) wird in der hier vorgelegten Arbeit nur am Rande berücksichtigt.

3.2.2 Deformation mit Interaktionselementen

In Nikitin et al.(2002)[20] wird ein Ansatz beschrieben, der sowohl unter Verwendung von FEM als auch BEM mit Hilfe von Interaktionselementen die Deformationen vorberechnet. Es findet eine Unterteilung in vorberechnete Daten und im Simulator berechnete Daten statt. Das Verfahren basiert auf der linearen Gleichung $\mathcal{K} * u = f$, in der f die bekannte Kraft in den Knoten des Objekts, u die unbekannte Verformung derselben ist. $\mathcal{K}$ ist die Steifigkeitsmatrix, welche durch die Materialeigenschaften und die Form des Objekts gegeben ist. Offline wird die sehr dünn besetzte Matrix $\mathcal{K}$ zur dichtbesetzten Matrix $\mathcal{K}^{-1}$ invertiert. Zur Darstellung der Deformation reicht es aus, die Steifigkeitsmatrix auf diejenigen Punkte zu beschränken, auf welche Kraft ausgeübt werden kann: die Oberflächenpunkte. Diese auf die Oberflächenpunkte beschränkte Submatrix von $\mathcal{K}$ wird im folgenden

als $\mathcal{K}'$ bezeichnet. Vor der Reduzierung auf die Submatrix muss $\mathcal{K}$ invertiert werden, um die Kräfte der innen wirkenden Knoten mit einzubeziehen. Hieraus kann online mit $u = (\mathcal{K}^{-1})' * f$ die Verformung errechnet werden. Es muss nicht die ganze, kaum zu komprimierende, invertierte Matrix gespeichert werden, sondern eine Beschränkung auf diejenigen Oberflächenelemente, auf welche externe Kraft einwirken kann, reicht aus (Beweis in [19]). In diesem Ansatz wird die Kraft durch Interaktionselemente ausgeübt. Es gibt endlich viele physikalisch durchführbare Zustände, in die die Objekte überführt werden können. Diese werden vorberechnet und in einem dafür entworfenen Datenformat gespeichert, so dass diese im Simulator ausgelesen, dargestellt und gegebenenfalls interpoliert werden können.

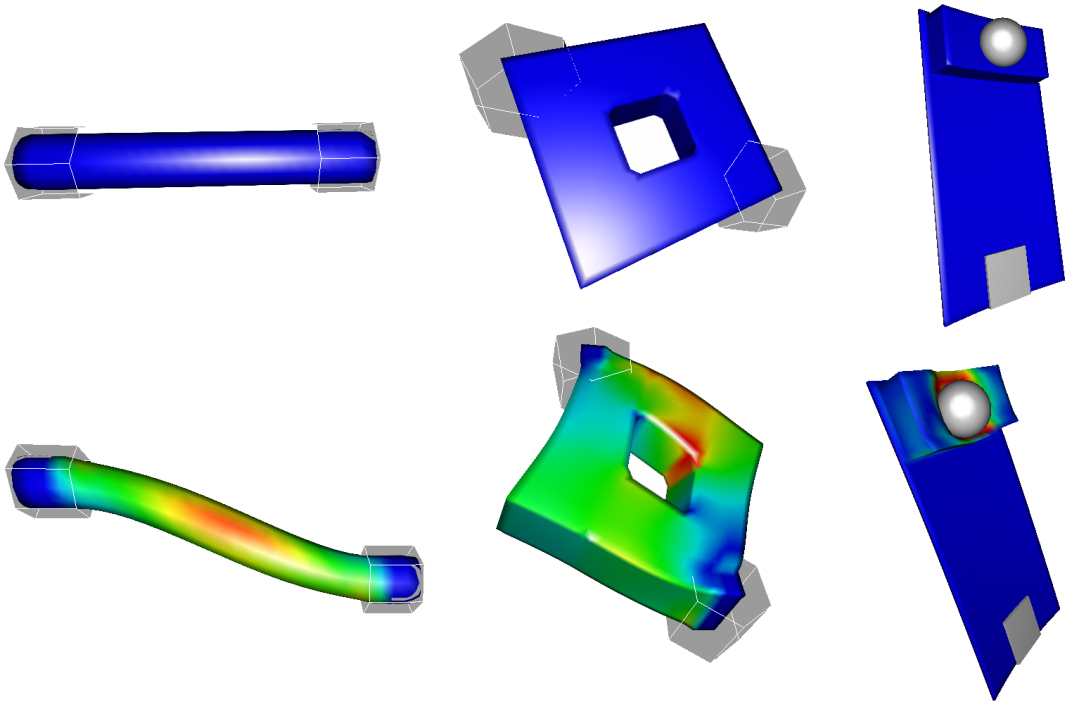

Abbildung 3.5: Ergebnisse eines Deformationssimulators [20] [24]. Kraft wird durch den Benutzer über die Interaktionslemente auf die Objekte übertragen.

3.2.3 Deformation mit Kollision

In Klimenko et al. [5] wurde neben den Ergebnissen aus [20] auch ein Ergebnis präsentiert, dass Deformation anhand von Kollision beschreibt. Es basiert auf einem flexiblen BEM-Modell, als starres Objekt dient eine Kugel festgelegter Größe. Die Eindringtiefe lässt sich sehr schnell durch den Abstand der Oberflächenpunkte zum Kugelmittelpunkt bestimmen. Dieser Ansatz lässt sich nur mit einer Kugel umsetzen. Hieraus wird dann die Verschiebung u berechnet und das Objekt deformiert. Da es sich um eine BEM-Simulation handelt, kann keine Innenstruktur mit anderen Materialeigenschaften simuliert werden.

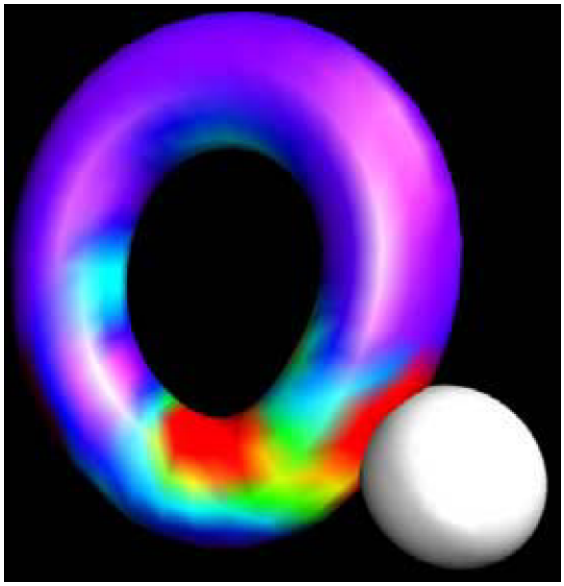

Abbildung 3.6: Ergebnisse eines Deformationssimulators [5]. Kraft wird durch den Benutzer über die Kugel auf den Torus übertragen.

3.2.4 Linked Volumes

In einer Arbeit von Dr. Gibson [17] wird ein Ansatz zur Deformation beschrieben, der nicht auf finiten Elementen beruht, sondern auf der so genannten *linked volume* Repräsentation. Hierbei wird jedes Element mit seinen nächsten sechs Nachbarn verbunden. Diese Verbindungen können gedehnt, gestaucht, getrennt und wiederhergestellt werden. In Bezug auf die Modellierung der Interaktionen unterscheidet sich dieser Ansatz von Masse-Feder-Systemen und finiten Element Methoden. Das *linked volume* besteht aus einer Aufstellung von Elementen, welche Informationen über die Struktur und die Materialeigenschaft tragen. Alle Operationen werden lokal betrachtet, d.h. zum Beispiel wenn eine Verbindung getrennt wird betrifft das nur genau zwei Knoten. In diesem Modell wird die Krafteinwirkung anhand eines Wellenprinzips weitergegeben. Wird beispielsweise ein Knoten verschoben, wird dies über die Verbindungen den anderen Knoten mitgeteilt. Je nach Materialeigenschaft, welche in den Verbindungen gespeichert werden kann, wird die Kraft übertragen und verteilt sich iterativ im gesamten Objekt. Der Vorteil dieses

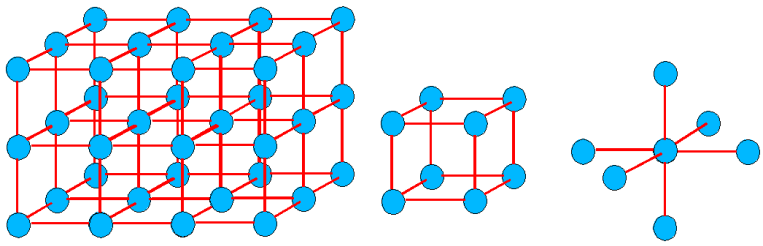

Abbildung 3.7: Gitterstruktur der Linked Volumes (links), mit Knoten (blau) und Links (rot). Mittig eine Zelle, rechts die Link-Struktur.

Ansatzes besteht in der Möglichkeit, sehr große Systeme in Echtzeit zu deformieren, ohne zeitaufwändige Vorberechnungen durchzuführen. Außerdem gewährt er die Möglichkeit von Topologieveränderungen, welche in der FEM nur nichtlinear und damit nicht vorberechenbar realisiert werden können. Nachteile dieses Systems liegen in der enormen Speicherbelegung und in der Dauer der Darstellung.

4 Vorgehensweise

Aufgrund der vorgestellten Vorgehensweisen wird in diesem Kapitel dargestellt, welche Ansätze modifiziert und später in der Simulationsumgebung verwendet werden.

Kern der hier vorgestellten Lösungsidee ist die Trennung von offline vorberechneten und online zu erechnenden Daten. Ziel ist es, so viel Information vorzuberechnen wie möglich, so dass in der online Simulationsumgebung so wenige Berechnungen zu tätigen sind, dass das System echtzeitfähig ist. Wie in Abb. 4.6 verdeutlicht wird, werden alle Berechnungen, die vorher getätigt werden können, offline durchgeführt und in schnell auslesbaren Datenstrukturen gespeichert (vlg. Abschnitt 5.1).

4.1 Wahl der Technik

Die im Folgenden dieser Arbeit vorgestellten Mechanismen wurden unter dem Hintergrund ausgewählt, echtzeitfähig und realitätsnah zu arbeiten. Ein voxelbasierter Ansatz eröffnet die Möglichkeit, neben der Kollisionserkennung (PointShell-Algorithmus™) auch einen Weg zur Bestimmung der Überlappung der Objekte zu finden (Occupancy Map Method). Hieraus lässt sich ein Ansatz entwickeln, der eine Eindringtiefe berechnen kann (vgl. Abschnitt 4.3). Aufgrund der Struktur der Objekte ist vor dem Inkrafttreten des voxelbasierten Kollisionserkennungsverfahrens zunächst eine sehr schnelle und einfache Verfahrensweise zur Kollisionserkennung erforderlich. Hierzu wurde sich zu der einfachen Abfrage entschieden, ob eine Überschneidung in einer achsenparallelen Ebene der Bounding Boxes stattfindet. Sollte

dies der Fall sein, werden wiederum alle Punkte der flexiblen Objekte gegen die Bounding Boxes der starren Objekte getestet. Danach erfolgt das für das Ermitteln der Eindringtiefe wichtige voxelbasierte Verfahren. Andere Verfahren konnten mit der Einfachheit dieses Vorgehens insofern nicht konkurrieren, da sie darauf ausgelegt sind, exakte Kollisionsdaten zu liefern, was im Rahmen dieser Arbeit im ersten Kolliosionserkennungsschritt nicht erforderlich ist. Zur Feinkollisionserkennung wurde ein Ansatz ausgewählt, der dem Voxmap Pointshell™-Algorithmus von Boeing ähnelt (vgl. Abschnitt 3.1.2). Dieser Ansatz bietet die Erweiterungsmöglichkeit, gleichzeitig eine Kollisionserkennung und eine Eindringtiefe zu errechnen. Zudem lassen sich Ansätze entwickeln, um Eindringtiefen und nächste Oberflächenpunkte offline vorzuberechnen (vgl. Abschnitt 5.1.1). Aus folgenden Gründen wurden andere kollisionserkennende Algorithmen wie die in Lin et al. [10] beschrieben nicht verwendet:

1. Eine genauere Berechnung als die Information der überschneidenden Ebene ist im ersten Schritt nicht erforderlich.
2. Die Rechenzeit, die trotz Optimierung durch Baumstruktur und Ähnlichem immer auf eine genaue Kollisionsinformation zielt, muss nicht aufgewendet werden.
3. Ein voxelbasierender Ansatz kann bei geeigneter Datenvorbereitung durch Interpolation sehr schnell sehr genaue Kollisionsentscheidungen teffen.
4. Das Berechnen von Eindringtiefen auf Basis einer Voxelstruktur ist Echtzeitfähig.

Zur Deformationssimulation wurde eine Erweiterung der Boundary Element Methode zur Finiten Element Methode durchgeführt. So können Innenstrukturen der Modelle simuliert werden, ohne den Vorteil der Vorberechnungsfähigkeit zu verlieren. Der Ansatz der Linked Volumes ist durchaus interessant, wurde aber in dieser Arbeit wegen der in Abschnitt 3.2.4 dargestellten Nachteile nicht verwendet. Zudem konnte die Verwendung von Finiten Elementen auf bereits vorhandene Strukturen aufsetzen.

4.2 Kollisionserkennung

Sind in der Szene nur starre Objecte vorhanden, besteht eine Kollision aus einem Austausch von Energie. In dem Falle, in dem die Objekte flexibel sind, wird Energie nicht nur ausgetauscht, sondern in Form von Deformation auch abgeleitet und gespeichert. Unter Umständen kann von dem Objekt ein Verhalten wie brechen oder reißen erwartet werden. Es wird vorerst von dem Fall ausgegangen, dass es ein beliebig geformtes statisches Objekt und ein beliebig geformtes dynamisches Objekt gibt, welches mit dem statischen Objekt kollidieren kann. Ein Objekt kollidiert mit einem anderen Objekt, wenn sich mindestens ein Punkt innerhalb oder auf der Oberfläche eines anderen Objekts befindet (vgl. Abschnitt 3.1). Das dynamische Objekt wird zunächst über eine Bounding Box repräsentiert, gegen die auf Kollision getestet wird. Bei Überlappung werden alle Oberflächenpunkte getestet.

Die Kollisionserkennung in dieser Arbeit wird durch eine Unterteilung in drei Stufen realisiert.

1. Kollisionserkennung zwischen den Bounding Boxes der Objekte.
2. Kollisionserkennung zwischen den Punkten des flexiblen Objekts und der Bounding Box des starren Objekts.
3. Kollisionserkennung durch Interpolation zwischen Gitterpunkten.

Wenn von einer der Stufen zurückgegeben wird, dass keine Kollision stattfindet, wird der Vorgang abgebrochen. Die Punkte **1.** und **2.** werden durch eine Abwandlung des *Separating Plane*-Algorithmus auf der Basis von achsenparallelen Ebenen durchgeführt (vgl. Abschnitt 3.1.1), da eine achsenparallele, rechteckige Bounding Box, wie sie in dieser Arbeit verwendet wird, konvex ist. Im ersten Fall werden die Grenzen der Bounding Box getestet, im zweiten Fall jeder Punkt des flexiblen Objekts. Solange sich die Bounding Boxes nicht überschneiden, kann keine Kollision stattfinden. In zwei räumlich getrennten Regionen ist die einfache Abfrage ausreichend, ob sich die Bounding Boxes in einer Ebene nicht überschneiden. Hierzu genügt die Abfrage, ob eine Dimension existiert, in der alle Werte der einen Bounding Box größer oder kleiner sind als in der anderen, zum Beispiel wenn die größte

X-Koordinate der einen Bounding Box bei *-100* liegt und die kleinste X-Koordinate der anderen bei *340*. Es ist somit mindestens eine separierende Ebene [25] (vgl. Abschnitt 3.1.1) gefunden worden, nämlich beispielsweise die $X{=}0$-Ebene. Die Ebene wird nicht durch ihre Gleichung definiert, ihre Existenz wird nur angenommen. Durch diese nicht iterative Vorgehensweise ist der Algorithmus deterministisch.

Sobald eine Überschneidung der Bounding Boxes stattfindet, wird über die Informationen der Gitterpunkte interpoliert (vgl. Abschnitt 5.2.2) und für die betreffenden Punkte die Eindringtiefe berechnet.

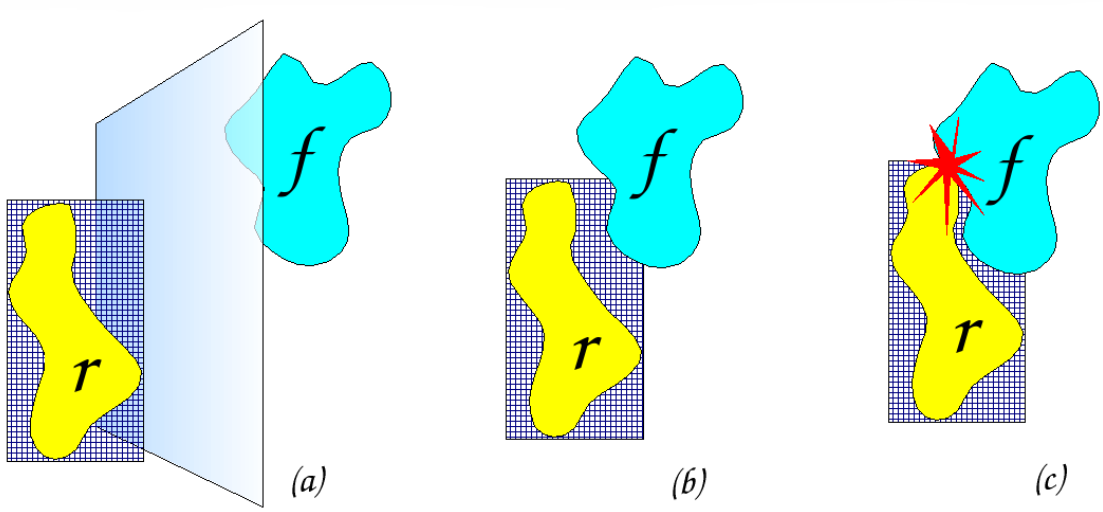

Abbildung 4.1: Eine separierende Ebene kann gefunden werden (a). Sobald ein Punkt des flexiblen Objekts f innerhalb der Bounding Box des rigiden Objekts r liegt, wird der Abstand zur Oberfläche über die Gitterpunktinformationen interpoliert. Als Ergebnis wird der Wert *außerhalb* zuückgeliefert (b) oder die berechnete Eindringtiefe bei Kontakt (c).

4.3 Deformation

4.3.1 online vs. offline

Zur Lösung von linearen Deformationen existiert die Möglichkeit, die Elastizitätsgleichung online zu lösen und gleichzeitig die graphische Darstellung durchzuführen [20]. Lineare Problemstellungen in der Elatizitätstheorie erfordern die Lösung von großen linearen Systemen der Form $\mathcal{K}u = f$ mit konstanter Matrix $\mathcal{K}$ und variabler Kraft f. So können moderat große Modelle berechnet werden [7] (1.400 Knoten bei 45 fps). Bei der Verwendung von vorberechneten Daten und invertierter Submatrix $(\mathcal{K}^{-1})'f$ und der Gleichungsdarstllung $u = (\mathcal{K}^{-1})'f$, kann eine größere Performanz erreicht werden [20] (10.000 Knoten bei 85 fps). Der Vorteil der online-Lösung liegt in der Möglichkeit der Verwendung von nicht-linearem Objektverhalten [7], welches annähernd in Echtzeit durchführbar ist (1.400 Knoten bei 8 fps). Dafür kann interaktiv die Matrix $\mathcal{K}$ verändert werden und somit die Randbedingungen und die Topologie des Objekts. So sind z.B. Schnitte im Modell realisierbar. Dieses Vorgehen wurde in aktuellen Arbeiten [21][22] schon für offline-Vorberechnungen implementiert. Hierbei wurde ein Algorithmus verwendet, der die schnelle Evaluation von $(\mathcal{K}^{-1})'$ auf Basis der Boundary Element Methode umsetzt.

4.3.2 FEM vs. BEM

Das in dieser Arbeit modifizierte System basiert auf Boundary Element Methods (BEM)[21]. Dieses wurde in eine Finite Element Methode (FEM) überführt, da sich mit BEM folgende wichtige Eigenschaften nicht realisieren lassen:

- Die Deformation berücksichtigt zwar innere Objektbeschaffenheit, diese kann jedoch nur aus dem selben Material beschaffen sein wie die Außenhaut. Da gerade bei medizinischen Datensätzen verschiedene Gewebetypen miteinander wechselwirken, war eine Umwandlung in FEM naheliegend. So können jetzt zum Beispiel das Verhalten von Haut über Muskel über Knochen dargestellt werden.

- Das Auftreten von scharfen Kanten führt bei BEM zu Artefakten.

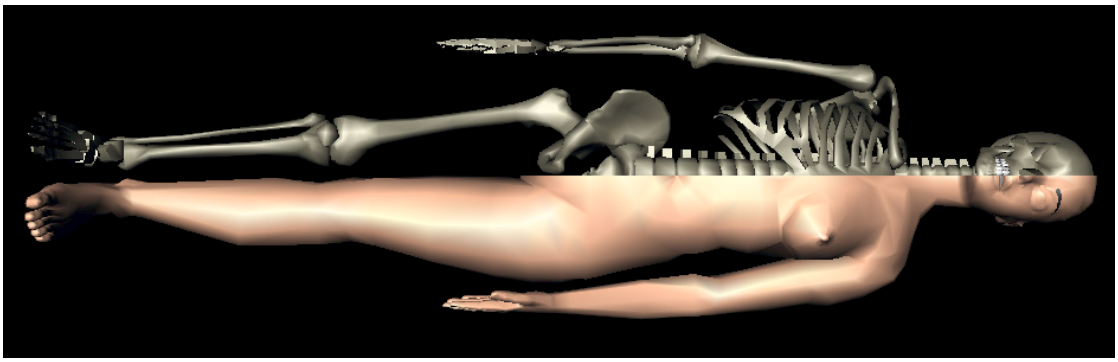

Abbildung 4.2: In einem Datensatz können bei Verwendung der Finiten Element Methode verschiedene Materialien mit den jeweiligen Deformationseigenschaften definiert werden, um ein realitätsnahes Objektverhalten zu simulieren.

Das modifizierte System stellt eine Vorberechnung zur Verfügung, die Datensätze anhand von Interaktionselementen deformiert. Hierzu werden alle endlich vielen Zustände vorberechnet, so dass die Deformationsinformation im Folgenden direkt abrufbar ist. Dieses Vorgehen ist bei der Verwendung von beliebigen Objekten und beliebiger Konstellation nicht mehr möglich. Die Zustände, die erreicht werden können, sind nicht mehr endlich und können so nicht vorherbestimmt werden. Infolgedessen musste ein Weg gefunden werden, die Deformationsinformation f (einwirkende Kraft) aus der Steifigkeitsmatrix $\mathcal{K}$ und der durch den Benutzer einwirkenden Verschiebung u der Elemente in Echtzeit rückzuberechnen.

$$\mathcal{K} * u = f$$

Hierzu muss die Steifigkeitsmatrix $\mathcal{K}$ in der Vorberechnung invertiert werden, um aus der Verschiebung der Elemente die Deformation und die Gesamteinwirkung auf das flexible Objekt zu berechnen. Hierfür reicht es aus, eine Submatrix von $\mathcal{K}$ zu bilden und zu invertieren, welche nur diejenigen Punkte innehat, auf welche Kraft ausgeübt werden kann: die Oberflächenpunkte.

$$(\mathcal{K}^{-1})' * f = u$$

Kern dieser Vorgehensweise ist, die durch Interaktion mit dem Benutzer entstehende Verschiebung der Elemente u in Form eines Verschiebungsfeldes in Echtzeit zu errechnen, um dieses in die Gleichung einbinden zu können.

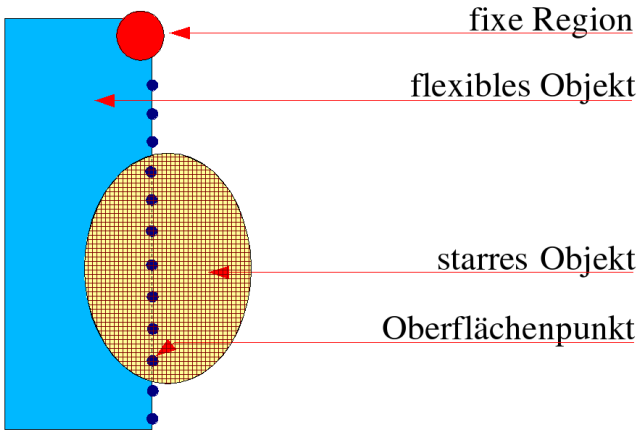

Abbildung 4.3: Ein starres Objekt dringt in ein fexibles Objekt ein. Eine fixe, von Deformationen nicht betroffene Region bezweckt, dass sich das flexible Objekt nicht verschiebt und ist als Randbedingung zur Lösung des Systems nötig.

Eine solche Vorgehensweise funktioniert nur aufgrund der Annahme, dass ein bestimmter Teil des flexiblen Objekts fixiert ist. Ist dies nicht der Fall, könnte man die oben genannte Gleichung nicht in dieser Form ausdrücken [23], die Steifigkeitsmatrix $\mathcal{K}$ wäre degeneriert und könnte nicht invertiert werden.

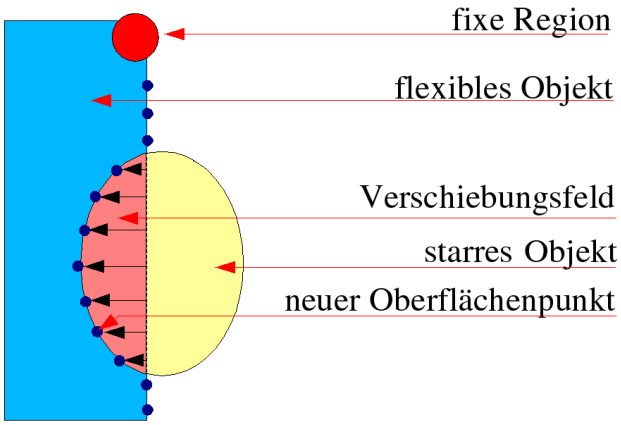

Abbildung 4.4: Vor der Berechnung der auf das Objekt wirkenden Kraft wird die Größe des Verschiebungsfeldes berechnet. Hierzu werden die Oberflächenpunkte des flexiblen Objekts auf die Oberfläche des starren Objekts projiziert und die Eindringtiefe errechnet.

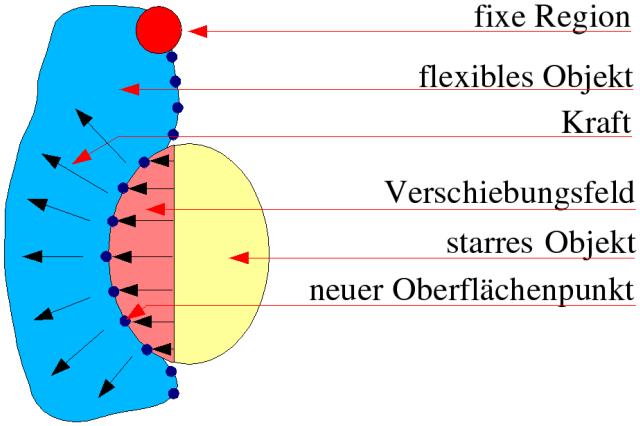

Abbildung 4.5: Eine mögliche Deformation eines sehr weichen flexiblen Objekts aufgrund des Verschiebungsfeldes.

4.3.3 FEM Oberflächen

In dieser Arbeit werden zur Darstellung der Deformation nur die Oberflächenpunkte verrechnet. Hierfür muss die Bedingung gelten, dass sich die inneren Knoten gegenüber den Oberflächenknoten unabhängig verhalten. In diesem Falle verhält sich die Oberfläche, bei der nur Oberflächenknoten verrechnet werden, genauso wie die Oberfläche, bei der alle Knoten (innere Knoten und Oberflächenknoten) in der Berechnung verwendet werden. Dadurch können Multiplikatoren eingespart werden, um die Deformation des Körpers zu berechnen. Beweis dieser These in Bro-Nielsen [19].

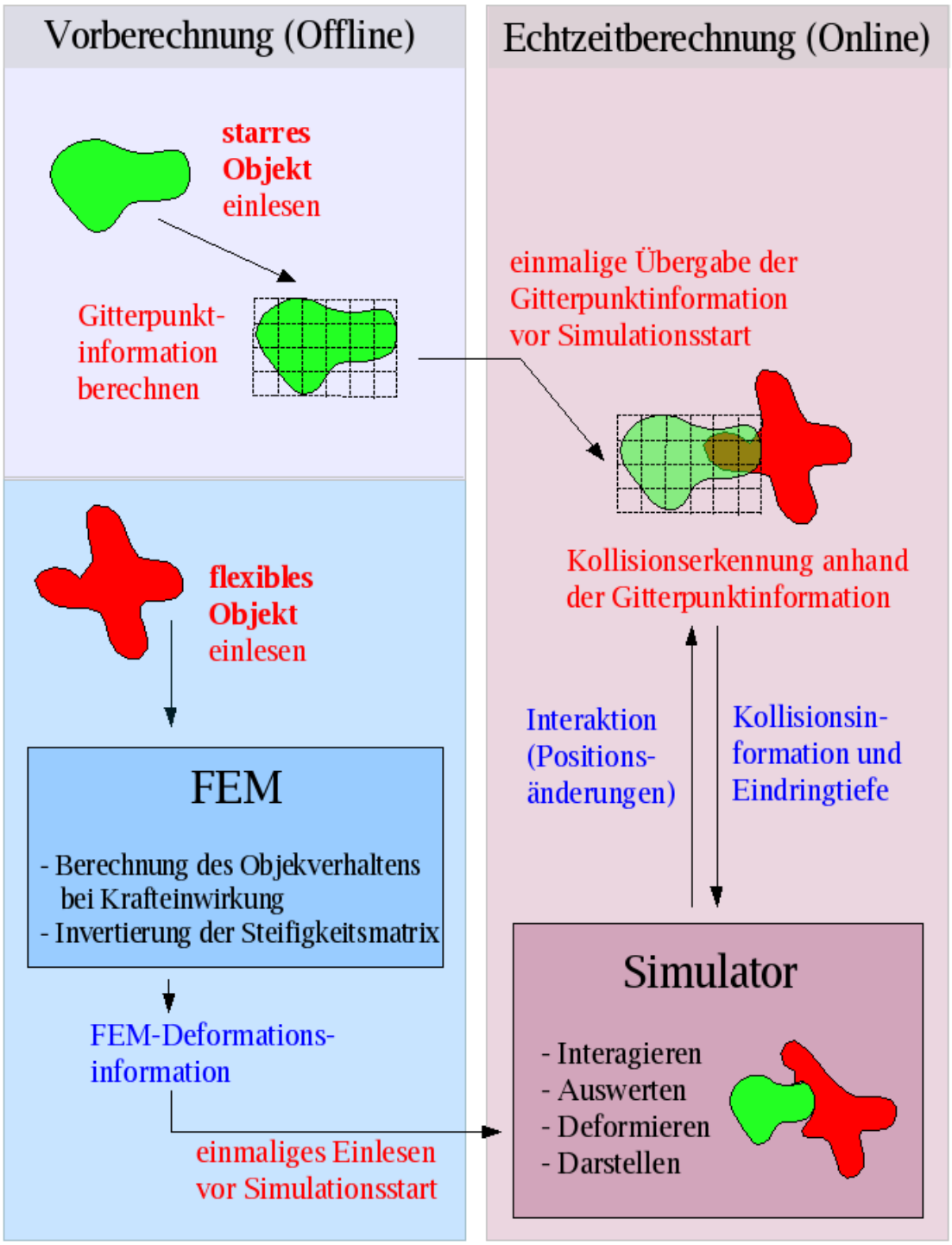

Abbildung 4.6: Schema der Online- und Offlineoperationen

5 Implementation

Diese Lösungsidee wurde im Rahmen dieser Arbeit in der Programmiersprache C++ und mit Hilfe der Skriptsprache Scheme umgesetzt. Hierbei musste die Integration in die Simulationsumgebung AVANGO (vgl. Anhang A.2) realisiert werden. Eine optimierte Programmierung stand wegen des Anspruchs auf Echtzeitfähigkeit stets im Vordergrund.

5.1 Offline-Berechnungen

Berechnungen, deren Ergebnis vorab berechnet werden kann, die also nicht auf Interaktionsdaten aus dem Simulator begründen, werden offline durchgeführt. Hierzu gehören das Berechnen und Anpassen der Steifigkeitsmatrix $\mathcal{K}$ und das Erstellen des Gitternetzes.

5.1.1 Gitternetz im Inventor-Objekt erstellen

Das im Rahmen dieser Arbeit erstellte Programm *CompLattice* liest eine im Inventor-Format geschriebene Datei anhand eines eigens erstellten Einleseprogramms aus, die ein Objekt beschreibt. Dieses Objekt wird durch eine achsenparallele, rechteckige Bounding Box umhüllt. Kein Teil des Objekts befindet sich außerhalb dieser Box.

Nach dem Einlesen des Inventor-Objekts, welches später das starre Objekt in der

Simulation darstellt, übergibt der Benutzer dem Programm die gewünschte Auflösung des Gitternetzes. Er gibt hierzu einen Wert an, der die Unterteilungsanzahl des Objekts betrifft. Durch diesen Wert wird die kleinste Seite der Bounding Box dividiert und unterteilt. Das Objekt wird also bildlich gesprochen längs in Scheiben unterteilt. Da das Objekt in eine homogene Gitterstruktur überführt werden soll, wird jetzt im selben Abstand quer und horizontal unterteilt. Da die Addition der Gitterabstände selten genau mit der Objektausdehnung der anderen Dimensionen übereinstimmt, wird die Bounding Box in diesen Dimensionen etwas größer. Dies hat keine Auswirkungen auf die Genauigkeit der Berechnungen und vernachlässigbare Auswirkungen auf die Gesamtperformanz.

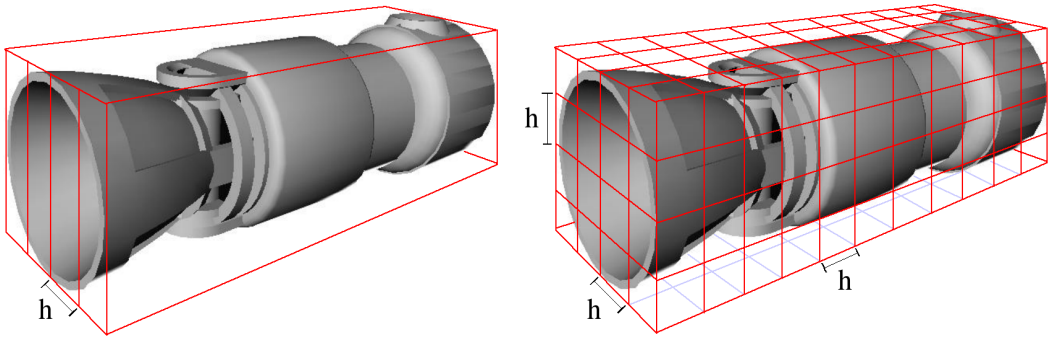

Abbildung 5.1: Die Bounding Box wird durch eine homogene Gitterstruktur aufgeteilt. Der Abstand h ist in allen Dimensionen gleich.

Es wird eine Datei erstellt, in der nicht mehr das Objekt, sondern nur noch die Gitterpunkte zeilenweise mit folgenden Informationen gespeichert werden:

- Das erste Zahlentripel definiert die Position des Gitterpunktes. Die Zahlentripel werden als Fließkommazahlen gespeichert. Jedes Tripel enthält die Koordinaten in folgender Reihenfolge: X-, Y- und Z-Koordinate. Als Trennzeichen wird ein Leerzeichen verwendet. Die Koordinaten können in exponentieller Schreibweise (0.123E3) oder ohne Nachkommaanteile (123) vorliegen.
- Das zweite Zahlentripel definiert die Position des nächsten Oberflächenpunktes. Die Speicherung erfolgt wie bei den Gitterpunktkoordinaten.

- Die Distanz zwischen dem Gitterpunkt und dem Oberflächenpunkt. Dieser Wert wird negativ angegeben, wenn sich der Gitterpunkt außerhalb des Objekts befindet. Ansonsten ist der Wert positiv. Wegen dieser Zusatzinformation und weil sie häufig verwendet wird, wird die Distanz zusätzlich gespeichert.

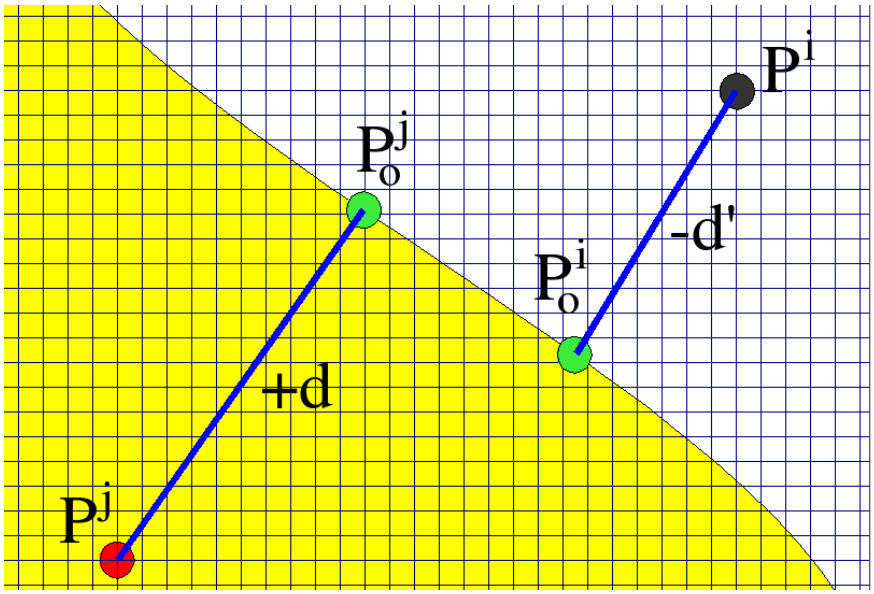

Abbildung 5.2: Für jeden Gitterpunkt P^x wird die Entfernung d zum nächsten Oberflächenpunkt P_o^x gespeichert. Die Entfernung wird negativ angegeben, wenn der Gitterpunkt außerhalb des Objekts liegt.

Die Struktur dieser Datei sieht demnach wie folgt aus:

```
.
.
.
-0.02 0 0.04 -0.0180249 -0.000926081 0.0357645 0.00476427
-0.02 0 0.06 -0.024125 0.00315815 0.0753948 0.0162477
-0.02 0 0.08 -0.019287 0 0.077339 -0.00275488
-0.02 0.02 -0.08 -0.0177766 0.0141421 -0.0717022 -0.0103977
-0.02 0.02 -0.06 -0.0199667 0.0193548 -0.0598756 -0.000657944
.
.
.
```

Die ersten beiden Gitterpunkte liegen innerhalb des Objekts, die drei unteren Gitterpunkte außerhalb. Eine Indizierung der Punkte erfolgt beim Auslesen über die Zeilennummer.

Die Ermittlung des nächsten Oberflächenpunktes ist zeitintensiv, da das Objekt in Triangel-Strips angegeben ist und die Koordinaten innerhalb eines Dreiecks erforderlich sind. Bei der effizienten Ermittlung des nächsten Oberflächenpunktes können Sonderfälle eintreten, deren Verhinderung recht rechenintensiv ist.

Zuerst werden die Triangle-Strips aufgelöst und als Dreiecke gespeichert. Aus der Anordnung der Dreiecke im Strip kann erkannt werden, welche Seite des Dreieckes nach außen zeigt. Bei der Speicherung wird dieser Wert als Normale behandelt. Die Entscheidung, ob ein Punkt innerhalb des Objekts liegt, wird zunächst über den Winkel zur Normalen des nächsten Dreieckes entschieden. Liegt im Zweidimensionalen dieser zwischen 90°und 270°, ist der Punkt innerhalb des Objekts [1]. Zur Veranschaulichung sind die folgenden grafischen Beispiele zweidimensional dargestellt. Im Dreidimensionalen liegen die Winkel zwischen Vektoren zwischen

[1] *Beweis:* Auf der Strecke zwischen dem Gitterpunkt $\mathcal{P}$ und seinem nächsten Oberflächenpunkt $\mathcal{P}_\mathcal{O}$ sind alle Punkte entsprechend Punkt $\mathcal{P}$ innerhalb oder außerhalb. Gibt es ein ϵ, das zwischen den beiden Punkten $\mathcal{P}$ und $\mathcal{P}_\mathcal{O}$ liegt und nicht genauso innerhalb oder außerhalb liegt, so ist $\mathcal{P}_\mathcal{O}$ nicht der nächste Oberflächenpunkt.

0°und 180°. Winkel zwischen 90°und 180°sind Ergebnisse für Punkte, die innerhalb des Objekts liegen.

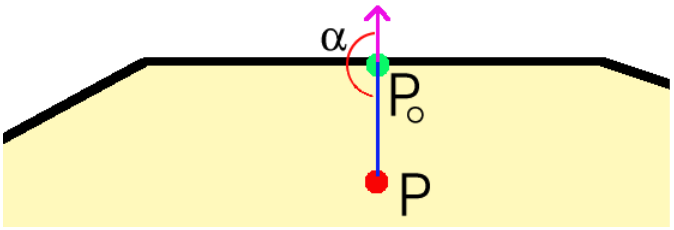

Abbildung 5.3: Der nächste Oberflächenpunkt liegt meist in einer Linie mit dem Gitterpunkt und der Richtung der Normalen.

Nun wird von dem aktuellen Gitterpunkt die Entfernung zu jedem Dreiecksmittelpunkt gemessen und die Kürzeste vermerkt.

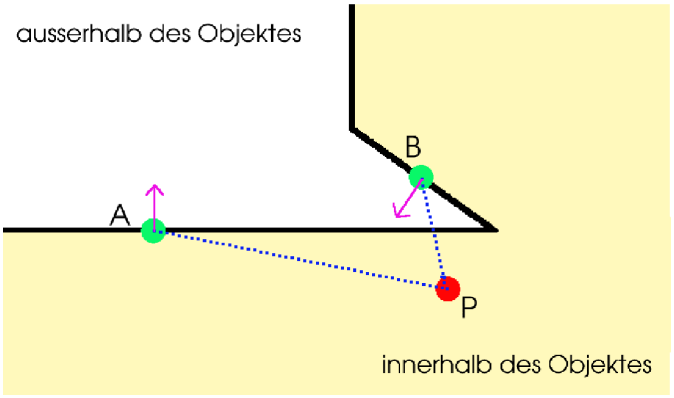

Abbildung 5.4: Das Dreieck mit dem am nächsten liegenden Mittelpunkt ist nicht unbedingt das Dreick, das den nächsten Oberflächenpunkt beinhaltet.

Wie in Abb. 5.4 verdeutlicht wird, liegt der nächste Dreiecksmittelpunkt nicht zwangsläufig im nächstliegenden Dreieck. Ein Feintest, der zu dem übergebenen Dreieck den nächsten Punkt auf der Fläche ermittelt, würde auf Dreieck B eine falsche Normale zurückgeben. Das Resultat wäre die Klassifikation *außerhalb* für Punkt $\mathcal{P}$. Wegen dieses Sonderfalles wird an dieser Stelle erst der nächste Eckpunkt

des Dreiecks ermittelt. Nun wird der Feintest auf allen Dreiecken durchgeführt, die diesen Punkt beinhalten. Diese Vorgehensweise ist recht zeitintensiv, wurde aber nicht weiter beschleunigt, da es sich um Vorberechnungen handelt, welche nicht zeitkritisch sind und bei denen die Priorität auf Präzision liegt.

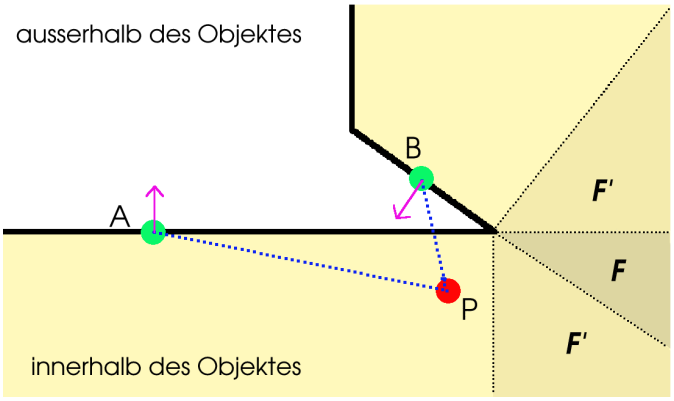

Abbildung 5.5: Die Fläche F hat als nächsten Oberflächenpunkt den Dreieckseckpunkt. Ihr kann keine Normale zugeordnet werden, aber der Punkt liegt für beide Dreiecke *innerhalb*. Die Flächen F' haben auch den Dreieckseckpunkt als nächsten Oberflächenpunkt, ihnen kann aber eine Oberfläche zugeordnet werden.

In Abb. 5.5 wird ein weiterer Sonderfall dargestellt. Wenn sich ein Punkt innerhalb der Fläche F befindet, ist eine Zuornung zu einer Normalen nicht eindeutig möglich. Dies hängt mit den Eigenschaften des Voronoi-Diagramms [24] zusammen. Im Falle, dass sich ein Punkt innerhalb der Flächen F' befindet, ist eine Zuordnung zu einer Normalen möglich, aber im Dreidimensionalen nicht schnell zu lösen. Wenn diese Situation eintritt, liegt der nächste Oberflächenpunkt nicht in einer Flucht mit der Normalen, wie in Abb. 5.3 beschrieben wurde. In solchen Fällen entscheidet das auf Raumwinkeln basierende Verfahren.

Raumwinkellösung

Für solche Sonderfälle und für die Situationen, in denen der Gitterpunkt sehr nahe der Oberfläche ist, ist eine unterstützende Methode hinzugefügt worden. Zur Berechnung des Raumwinkels wird folgende Formel verwendet:

$$\frac{1}{4\pi}\Omega = \frac{1}{4\pi}\sum_i \frac{\vec{s}_i * \vec{r}_i}{|\vec{r}_i|^3}. \tag{5.1}$$

Hierbei stehen der Vektor $\vec{r}$ für die Strecke Gitternetzpunkt $\rightarrow$ Oberflächenpunkt und $\vec{s}$ für einen flächengewichteten Vektor entlang der Normale des Dreiecks. Die Division durch 4π wurde beibehalten, um zu verdeutlichen, dass die Summe aller Vektoren $\vec{r}$ von einem Gitterpunkt $\mathcal{P}$ aus gesehen 4π ergibt (der Raumwinkel einer Umkugel ist gleich 4π). Das Ergebnis der Gleichung kann man wie folgt verwenden:

$$\frac{\Omega}{4\pi} \approx \begin{cases} 0, & \text{wenn } \mathcal{P} \text{ außerhalb des Objekts liegt} \\ 1, & \text{wenn } \mathcal{P} \text{ innerhalb des Objekts liegt} \end{cases}$$

Durch diese Vorgehensweise erhält man bei grenzwertigen Situationen eine Absicherung. Diese ist auch in solchen Fällen wichtig, wenn es Fehler im Modell gibt und zum Beispiel die Oberfläche sich überschneidende Dreiecke beinhaltet. Die besten Ergebnisse erhält man, wenn das Objekt das *hedgehog-theorem* [5] (engl.: *Igel*) erfüllt: Die Summe aller (nicht normierten) Normalen eines Objekts ergibt Null. In diesem Fall ist das Objekt meist *wasserdicht*: Es hat keinerlei Lücken und keine Überlappungen in der Tisselierung.

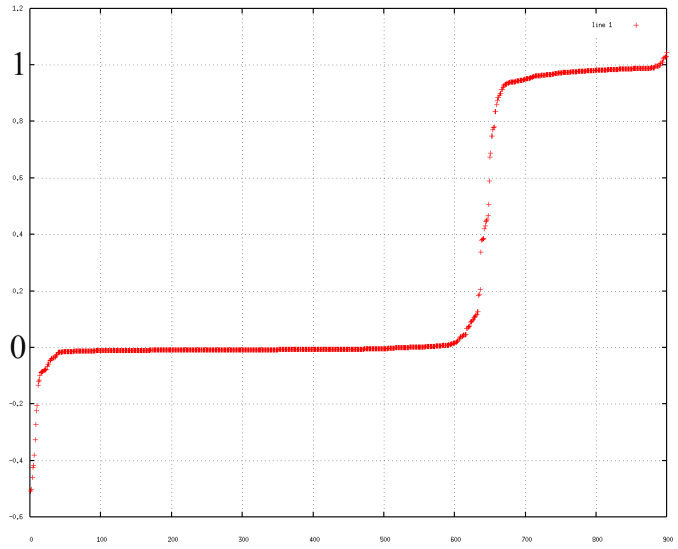

Abbildung 5.6: Die mit dem in 5.1.1 erläuterten Algorithmus entstehenden Werte zeigen, dass eine Entscheidung über den Raumwinkel getroffen werden kann. Die Punkte, die einen Wert um 0 haben, liegen außerhalb. Die Punkte, die einen Wert um 1 haben, liegen innerhalb. Ausreißer wie stärker negative oder zwischen 0 und 1 liegende Werte entstehen durch unsauber modellierte Objekte und Oberflächengrenzfälle. Diese werden durch die Vorgehensweise wie in 5.1.1 erläutert eindeutig zugeordnet.

Zusammenfassung. 5.1.1 Ein virtuelles Modell wird eingelesen und eine Bounding Box darum erstellt. Diese wird mit einem homogenen Punktgitter gefüllt. Für jeden Punkt wird errechnet, ob sich dieser innerhalb des Objekts befindet, welche Koordinaten der nächstgelegene Oberflächenpunkt hat (vgl. Ergebnis 6.1) und die Entfernung zwischen diesen beiden Punkten. Diese Informationen werden in eine Datei geschrieben, die später im Rahmen der online-Berechnungen ausgelesen wird. Das Erstellen dieser Datei für ein Objekt mit 20.000 Gitterpunkten dauert etwa eine Sekunde. Für die Interpolation der Koordinaten und der Oberflächenentfernung von 1.000.000 zufälligen Testpunkten innerhalb der Bounding Box des starren Objekts wurde eine Zeit von 0.97 Sekunden benötigt.

5.1.2 Modellaufbereitung

Medizinische Modelle liegen oft als Oberflächenmodelle vor, d.h. die Oberfläche ist als eine Ansammlung von Dreiecken definiert. Für die Berechnung der Deformation mit FEM wird eine Volumendarstellung benötigt, d.h. dass eine Innenstruktur der Objekte durch dreidimensionale Finite Elemente wie z.B. ein Tetraeder definiert sein muss. Nur so ist es möglich, den Objekten verschiedene Deformationseigenschaften innerhalb ihrer Struktur zuzuordnen. Um diese Forderung zu erfüllen wurde sich im Rahmen der vorliegenden Arbeit die Methodik aus Abschnitt 5.1.1 auch für die Verwendung der flexiblen Objekte zunutze gemacht. Diese werden, wie auch die starren Objekte, zunächst mit einer Bounding Box umhüllt und mit einer Gitterstruktur gefüllt. Im Unterschied zu der Vorgehensweise aus Abschnitt 5.1.1 wird im Folgenden aber nur unterschieden, ob der aktuelle Gitterpunkt innerhalb oder außerhalb des Objekts ist. Die Informationen, wo exakt der nächste Oberflächenpunkt liegt und wie groß die Entfernung dahin ist, sind hierbei nicht erforderlich. Wird er als *innerhalb* erkannt und klassifiziert, wird er in die Liste der Objektpunkte zu den Oberflächenpunkten hinzu aufgenommen. Das Ergebnis ist das Objekt in seiner ursprünglichen Form mit seinen Oberflächenpunkten ergänzt durch eine homogene Innenstruktur. Dieses Programm wurde im Rahmen dieser Arbeit entwickelt und unter dem Namen *pad* (engl.: *ausstopfen*) integriert. Nun folgt die Berechnung der Verbindungen unter den Punkten. Hierzu wird der Delaunay-Algorithmus [26] verwendet, welcher aufgrund der in [24] erläuterten Umkugelregel eine Tisselierung vornimmt. Dieser Algorithmus konstruiert ein Tetraedernetz, welches für die Verwendung von FEM optimal geeignet ist. Der Delaunay-Algorithmus besitzt allerdings die Eigenschaft, nur konvexe Modelle erstellen zu können. Da gerade medizinische Daten in Ihrer Struktur oft sehr komplex sind, musste eine Korrektur dieser Daten erfolgen. Hierzu wurden sich Ergebnisse einer früheren Arbeit [24] zunutze gemacht. Es wurde ein Algorithmus entwickelt, der Anhand von Homogenität und Objektform nicht korrekte, d.h. das Objekt entstellende und konkave Eigenschaften verdeckende Tetraeder entfernen konnte. Als Ergebnis dieser Vorgehensweise liegt nun ein Objekt vor, dass in seiner ursprünglichen Form nicht verfälscht ist und in ein Volumendatenmodell umgewandelt wurde. Aufgrund dieser Eigenschaft ist nun eine Zuordnung von verschiedenen Deformationseigenschaften für jedes einzelne finite Element möglich.

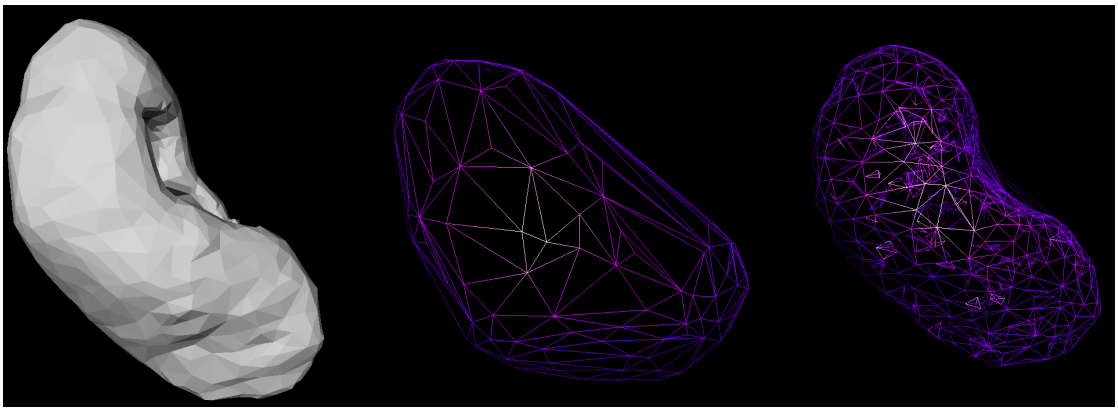

Abbildung 5.7: Im linken Bild ist das original Inventor Oberflächenmodell dargestellt. Wird dieses in ein Volumenmodell konvertiert und eine dreidimensionale Tisselierung (Tetraeder) durchgeführt, bleibt nur die konvexe Hülle sichtbar. Diese Fehler können mit den Ergebnissen aus [24] behoben werden, um ein realitätsnahes Objektverhalten darzustellen.

5.1.3 Deformation

In dem in dieser Arbeit vorgestellten Ansatz werden die Deformationsinformationen vorberechnet, um ein echtzeitfähiges System umzusetzen.

Vorberechnung

Wie im Abschnitt 2.1 bereits erläutert wurde, basiert die Deformation mit der Finiten Element Methode auf Objektknoten, die untereinander Kräfte ausüben. Diese Kräfte werden durch eine Matrix $\mathcal{K}$ von jedem Element auf jedes Element beschrieben. Die dünnbesetzte, symmetrische Matrix $\mathcal{K}$ besteht aus n 3×3 Matrizen, wobei n die Anzahl aller Elemente darstellt. Die Matrix $\mathcal{K}$ ist also $3n \times 3n$ groß. Weniger als 1% der Matrix ist ungleich Null. Diese Matrix kann effizient in Datenstrukturen gespeichert werden, die dem Elementverbund entsprechen. Die diagonalen Einträge $\mathcal{K}_{ii}$ sind in den Knotenpunkten des Verbundes gespeichert, während die nicht-diagonalen Elemente auf den Kanten gespeichert werden [20]. Jede Beziehung zwischen den Elementen wird durch eine 3×3 - Matrix beschrieben: Der Deformationsvektor mit seinen drei Komponenten ux, uy, uz und die drei Richtungen der

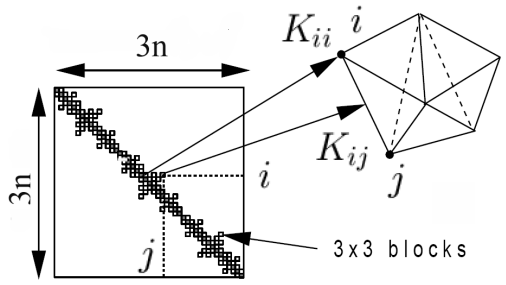

Abbildung 5.8: Speicherung der Steifigkeitsmatrix [20].

Kraft fx, fy, fz (vgl. Abschnitt 2.6, Greensche Funktionen). Jeder dieser Blöcke beinhaltet die proportionalen Koeffizienten der Kraft des j-ten Elements zur Verschiebung des i-ten Elementes. Die Vektoren u und f aus der Gleichung $\mathcal{K}u = f$ sind Vektoren der Länge $3n$ und haben die Struktur $(ux_1, uy_1, uz_1, ux_2, uy_2, uz_2...)$ und $(fx_1, fy_1, fz_1, fx_2, fy_2, fz_2...)$. Für Knoten, die nicht benachbart sind, sind diese Blöcke gleich Null.

Eine solche zur besseren Anschauung in 3×3 - Matrizen aufgeteilte Matrix könnte wie folgt aufgebaut sein:

$$\mathcal{K} = \left(\begin{array}{ccc|ccc|ccc} aa & ab & ac & ad & ae & af & ag & ah & ai \\ ab & bb & bc & bd & be & bf & bg & bh & bi \\ ac & bc & cc & cd & ce & cf & cg & ch & ci \\ \hline ad & bd & cd & dd & de & df & dg & dh & di \\ ae & be & ce & de & ee & ef & eg & eh & ei \\ af & bf & cf & df & ef & ff & fg & fh & fi \\ \hline ag & bg & cg & dg & eg & fg & gg & gh & gi \\ ah & bh & ch & dh & eh & fh & gh & hh & hi \\ ai & bi & ci & di & ei & fi & gi & hi & ii \end{array}\right) \tag{5.2}$$

Die Buchstaben sollen nur verdeutlichen, dass es sich um eine symmetrische Matrix handelt und haben sonst keine Bewandtnis. Auf Basis des Programmes *precomp*, welches Deformationen aufgrund von Interaktionselementen berechnet, wurden in dieser Arbeit erweiterte Programmausgaben entwickelt, so dass die eigens erstellte Umgebung *process* diese auslesen kann.

process-Klasse

Zunächst wird in dieser Umgebung die Datei *matrix.dat* ausgelesen, in welcher sich die von dem Programm *precomp* erstellte Matrik $\mathcal{K}$ befindet. Nun werden folgende Schritte ausgeführt:

Ankerpunkte löschen. Das Programm *precomp* erstellt in den Modellen Ankerelemente, welche in der Deformationsumgebung bisher als Interaktionselemente fungierten. Diese Einträge müssen bearbeitet werden, damit die Matrix später invertiert werden kann. Die original Matrix $\mathcal{K}$ ist degeneriert und damit nicht invertierbar, da in der Gleichung $\mathcal{K}u = f$ der Deformationsvektor u existiert, aber die Kraft f gleich Null ist. Diese *Deformationen* sind Translationen und Rotationen des gesamten Objekts. Um diese Verschiebungen zu verhindern, werden Regionen des Körpers in einem Teilraum fixiert. Dieser Teilraum kann verschoben und rotiert werden, ohne die Deformation des Objekts zu beeinflussen.

Um die Matrix invertieren zu können, müsste in diesen fixen Punkten die Verschiebung u auf Null gesetzt werden. Dies würde nicht bedeuten, dass f in diesen Punkten auch gleich Null wäre. Da die Anzahl der Unbekannten durch das setzen von $u = 0$ reduziert würde, müsste die Anzahl der nun unabhängigen Kräfte in f auch verringert werden. Hierzu würden alle Spalten und Zeilen der fixierten Elemente gelöscht. So würden die gelöschten Spalten mit $u = 0$ multipliziert und hätten keine Bedeutung mehr. Die gelöschten Zeilen ergäben eine Gleichung, welche die Reaktionskräfte in den fixierten Punkten definieren würde.

In dieser Arbeit wird dieser Effekt durch eine Entkopplung dieser Zeilen und Spalten erreicht, ohne die Größe der Matrix zu ändern und ohne die Gleichungen für die Reaktionskräfte aufstellen zu müssen. Diese beiden Prozeduren erreichen den selben Effekt, außer dass bei der in dieser Arbeit verwendeten Methode aufgrund der Einheitsmatrix falsche Verschiebungen ($u = f$) in den fixierten Punkten auftreten. Dies lässt sich durch das Löschen ($u = 0$) dieser Knoten korrigieren.

Die Umgebung *precomp* wurde so verändert, dass eine Datei *bflags.dat* ausgegeben wird, in der die Indizes der Ankerelemente angegeben wird. Hierbei stehen:

- *0* für *nicht zu einem Interaktionselement angehörend*
- *1* für *gehört zu Interaktionselement 1*
- *2* für *gehört zu Interaktionselement 2*

Diese Datei wird von der Umgebung *process* eingelesen und ausgewertet. Nun werden in der Matrix $\mathcal{K}$ alle Spalten und Zeilen, in denen sich Ankerelemente befinden, auf 0 gesetzt bzw. in eine Einheitsmatrix umgeschrieben, sollte sich die betreffende 3×3 - Matrix auf der Diagonalen befinden. Die in Formel 5.2 dargestellte Matrix würde sich wie folgt verändern, wenn das dritte Element ein Ankerelement ist:

$$\mathcal{K} = \begin{pmatrix} aa & ab & ac & ad & ae & af & 0 & 0 & 0 \\ ab & bb & bc & bd & be & bf & 0 & 0 & 0 \\ ac & bc & cc & cd & ce & cf & 0 & 0 & 0 \\ ad & bd & cd & dd & de & df & 0 & 0 & 0 \\ ae & be & ce & de & ee & ef & 0 & 0 & 0 \\ af & bf & cf & df & ef & ff & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.3}$$

Matrix invertieren. Diese Matrix muss zur weiteren Berechnung invertiert werden. Hierfür müssen bestimmte Randbedingungen erfüllt sein, damit die Matrix nicht degeneriert ist: Translation und infinitesimale Rotationen müssen verhindert werden. Hierzu werden ein oder zwei Regionen des Objekts im Raum fixiert. Diese lassen dich durch Deformation nicht beeinflussen. So kann der Benutzer zwar die Objekte beliebig im Raum verschieben, sämtliche Deformationen des Objekts orientieren sich aber an diesen fixen Interaktionselementen (vgl. Abschnitt *Ankerpunkte löschen*).

In dieser Arbeit wird hierzu zunächst die Funktion *gaussj* [27] aus den Numerical Recipes verwendet, welche eine Umsetzung des Gauss-Jordan Algorithmus' ist.

Dieser Algorithmus ist bekannt für seine zeitliche Ineffizienz, wurde aber wegen seiner Stabilität und wegen der zeitunkritischen Verwendung benutzt. Die invertierte Matrix $\mathcal{K}^{-1}$ ergibt sich nun wie folgt:

$$\mathcal{K}^{-1} = \begin{pmatrix} jj^{-1} & jk^{-1} & jl^{-1} & jm^{-1} & jn^{-1} & jo^{-1} & 0 & 0 & 0 \\ jk^{-1} & kk^{-1} & kl^{-1} & km^{-1} & kn^{-1} & ko^{-1} & 0 & 0 & 0 \\ jl^{-1} & kl^{-1} & ll^{-1} & lm^{-1} & ln^{-1} & lo^{-1} & 0 & 0 & 0 \\ jm^{-1} & km^{-1} & lm^{-1} & mm^{-1} & mn^{-1} & mo^{-1} & 0 & 0 & 0 \\ jn^{-1} & kn^{-1} & ln^{-1} & mn^{-1} & nn^{-1} & no^{-1} & 0 & 0 & 0 \\ jo^{-1} & ko^{-1} & lo^{-1} & mo^{-1} & no^{-1} & oo^{-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.4}$$

Das Invertieren der Matrix dauert etwa 15 Minuten für ein Objekt mit 600 Elementen (1.8 GHz Athlon / 1 GB Hauptspeicher). Für Objekte mit mehr als 3000 Elementen wird eine Berechnungszeit von über 5 Stunden benötigt.

Oberflächenpunkte extrahieren. Für die online-Deformation sind nur diejenigen Punkte zu Darstellung nötig, auf die Kraft ausgeübt werden kann. So müssen nur diese Punkte gespeichert werden, was das Einlesen beim Start des Simulators beschleunigt. Die Information, bei welchen Punkten es sich um Oberflächenelemente handelt, wird ähnlich wie die Interaktionselemente von der Umgebung *precomp* in eine Datei geschrieben. Diese wird ausgelesen und so ausgewertet, dass aus der invertierten Matrix $\mathcal{K}^{-1}$, die keine Interaktionselemente mehr enthält, eine Submatrix gebildet wird, welche nur noch Oberflächenelemente beinhaltet. Im Beispiel 5.5 sind das erste und das dritte Element der Matrix 5.4 Oberflächenpunkte, woraus sich die folgende Submatrix $(\mathcal{K}^{-1})'$ ergibt:

$$(\mathcal{K}^{-1})' = \begin{pmatrix} jj^{-1} & jk^{-1} & jl^{-1} & 0 & 0 & 0 \\ jk^{-1} & kk^{-1} & kl^{-1} & 0 & 0 & 0 \\ jl^{-1} & kl^{-1} & ll^{-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.5}$$

Übergabeformat

Die vorberechneten Daten müssen zusammengestellt werden, damit sie effizient an die Simulationsumgebung übergeben werden können. Hierzu wird die gesamte Information, die für das flexible Objekt vorberechnet wurde, in eine Datei geschrieben. Die Datei ist wie folgt aufgebaut:

1. Kopfzeile
   - Anzahl der Oberflächenpunkte und der Dreiecke
   - Angabe, ob die nachfolgende Matrix invertiert vorliegt oder nicht
   - Materialeigenschaften (vgl. Abschnitt 2.3)
2. Die in 5.1.3 erstellte Matrix sequentiell in einer Zeile
3. Die Interaktionsinformation in der Reihefolge der Oberflächenpunkte
4. Die Koordinaten der Oberflächenpunkte und deren Punktnormalen
5. Die Anzahl der Triangle Strips
6. Die einzelne Länge der Triangle Strips
7. Anzahl der Einträge in den Triangle Strips ohne die Trenneinträge (-1)
8. Darstellung der Triange Strips ohne Trenneinträge
9. Information für jeden Punkt, ob er ein Oberflächenpunkt ist
10. Farbinformation zur Deformationsveranschaulichung

5.2 Online-Berechnungen

Unter online-Berechnungen versteht man diejenigen Operationsschritte, deren Berechnungen erst in der Simulation selber erfolgen. Diese Werte können nicht vorberechnet werden und müssen in der Simulation berechnet werden, ohne die Echtzeitfähigkeit des Systems zu beschränken.

5.2.1 Auslesen der Gitterpunktinformationen

Die Datei, welche in Abschnitt 5.1.1 erstellt wurde, wird in der Simulationsumgebung ausgelesen. Zum effizienten Speicherzugriff wird jeder Gitterpunkt in ein `array` von `structs` *LatticePoint* gelesen. Das Auslesen der Datei beansprucht einen nicht unerheblichen Anteil Zeit beim Start der Simulationsumgebung. Ist die Datei allerdings einmal eingelesen, sind die Daten zum schnellen Zugriff im Hauptspeicher resident.

5.2.2 Interpolation

Über Eingabegeräte wird in der Simulationsumgebung durch den Benutzer bestimmt, an welcher Position sich die Objekte befinden. Als Eingabegeräte können alle Möglichkeiten der Simulationsumgebung angesprochen werden (3D-Stylus, Maus, Phantom, CubicMouse™, uvm.). Eine Erweiterung der im Rahmen dieser Arbeit entwickelten Umgebung könnte auch eine Kraftrückkopplung mit einem entsprechenden Interaktionsgerät (z.B. Phantom) ermöglichen. Zur besseren Anschauung wird hier zunächst genau ein dynamisches und ein statisches Objekt behandelt. Bei Kollision wird ein Punkt des dynamischen Objekts, der innerhalb der Bounding Box des statischen Objekts liegt, zwischen den umliegenden Eckpunkten linear interpoliert.

Es werden sowohl die Koordinaten des nächsten Oberflächenpunktes als auch die Distanz dorthin interpoliert. Hierzu werden temporär Objekte des `structs` *LatticePoint* erstellt. Die interpolierte Distanz und der dazugehörige Oberflächenpunkt werden der Simulation zur Berechnung der Deformation übergeben.

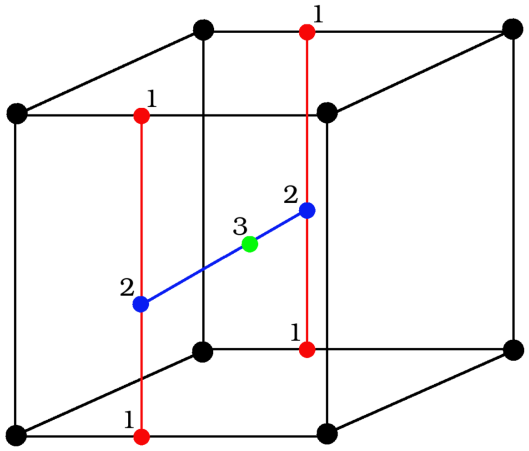

Abbildung 5.9: Die trilineare Interpolation besteht aus sieben linearen Interpolationen. Zunächst wird zwischen den Eckpunkten interpoliert (4 × rot), dann zwischen den neu errechneten Werten (2 × blau) und dazwischen (1 × grün).

Lineare Interpolation

In Abb. 5.10 ist es erforderlich, den Wert für Punkt $\mathcal{P}$ zu ermitteln, wenn man nur die Informationen der Punkte $\mathcal{A}$ und $\mathcal{B}$ hat. Der Wert von $\mathcal{A}$ ist gleich k, der von $\mathcal{B}$ ist gleich j. Die Abstände von $\mathcal{A}$ nach $\mathcal{P}$ und von $\mathcal{B}$ nach $\mathcal{P}$ sind mit u und v bekannt. $U + v$ ergibt 1. Der Wert von $\mathcal{P}$ ergibt sich nun aus

$$\mathcal{P} = \mathcal{A} * v + \mathcal{B} * u.$$

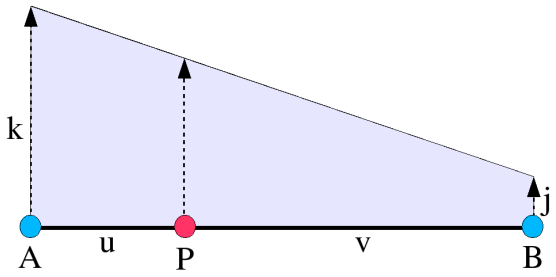

Abbildung 5.10: Lineare Interpolation

5.2.3 Gitterpunkte

Bei der linearen Interpolation muss auf eine ausreichende Anzahl Stützstellen geachtet werden. Der Interpolationsfehler ε wird um so größer, je weiter die Stützpunkte auseinander liegen. Als Fehler wird die Abweichung vom korrekt errechneten Wert bezeichnet. Wie in Abb. 5.11 dargestellt, liegen bei höherer Unterteilung die Interpolationswerte näher an den exakten Werten.

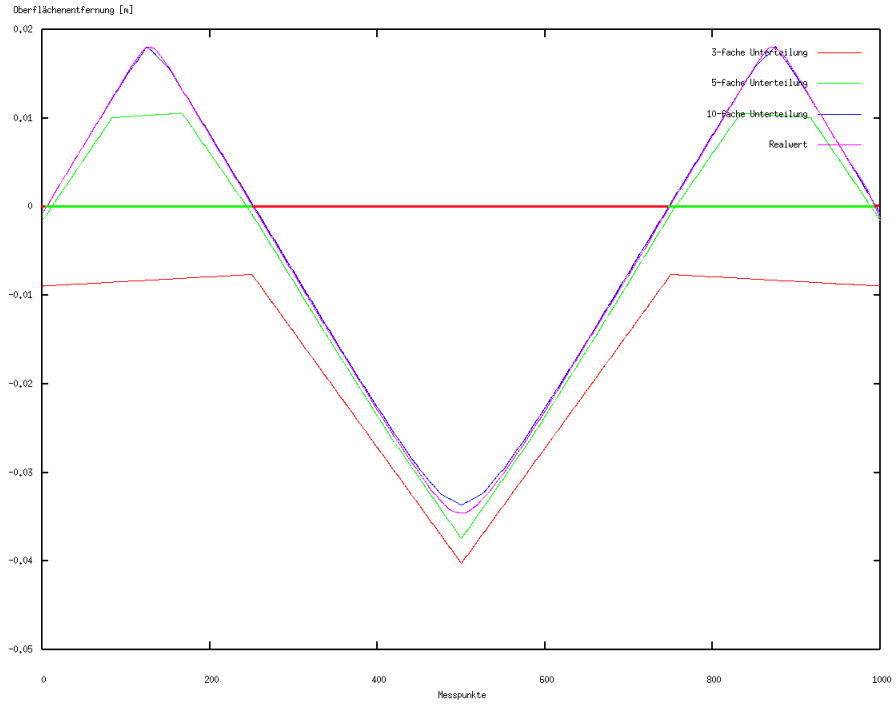

Abbildung 5.11: Die Anzahl der Interpolationsstützpunkte ist ausschlaggebend für die Genauigkeit. Hier sind die Abstände zur nächsten Oberfläche bei einem Strahl durch einen Torus dargestellt. Positive Werte liegen innerhalb, neagtive außerhalb des Objekts. Die Realentfernung der Punkte zur Oberfläche ist pink dargestellt. Eine Interpolation mit 10 Stützpunkten liefert eine gute Näherung, vor allem im Bereich der Übergänge *innerhalb-außerhalb*.

Finden der umgebenden Gitterpunkte

Beim Einlesen der Datei wird auf eine genaue Indizierung geachtet. Die Daten wurden innerhalb der Bounding Box so in die Gitterpunktdatei geschrieben, dass die Indizes der Gitterpunkte immer im selben Verältnis stehen (vgl. Abb. 5.12). Die entstandene Struktur entsteht durch das Ineinanderschachteln der drei `for`-Schleifen. Die erste läuft über x, die zweite über y und die dritte über z. Z ist dann einmal durchlaufen, wenn alle n Punkte pro Linie gesetzt wurden, wobei sich n aus der Anzahl der vom Benutzer angegebenen Unterteilungen und der kleinsten Dimension der Bounding Box ergeben.

Beim Einlesen der Datei wird nun vermerkt, bei welchem Index der erste y-Wert wechselt und der Index als *ybreak* gespeichert. Dies ist der anzahlmäßige Index-Abstand zur nächsten Zeile. Nun wird geprüft, bei welchem Index der x-Wert das erste mal wechselt und dieser Wert als *xbreak* gespeichert.

Wenn ein Punkt innerhalb der Bounding Box liegt, müssen zur Interpolation die acht Punkte gefunden werden, deren Quadrat diesen Punkt umschließt. Den Punkt in diesem Quadrat, welcher am nächsten zum Ursprung liegt, findet man nach der Vorberechnung

```
int tmpx = int((x-minx)/homdist);
int tmpy = int((y-miny)/homdist);
int tmpz = int((z-minz)/homdist);
```

über

```
int ulvIndex = tmpx*xbreak+tmpy*ybreak+tmpz;,
```

wobei `homdist` der gleichmäßige Abstand der Gitterpunkte ist. `UlvIndex` wird hier als der Index des *u*nten *l*inks *v*orne liegenden Punktes verwendet. Wenn man diesen Index des Gitterpunktes hat, kann nun mit

```
int urvIndex = ulvIndex+xbreak;
int olvIndex = ulvIndex+ybreak;
int orvIndex = ulvIndex+ybreak+xbreak;
int ulhIndex = ulvIndex+1;
int urhIndex = ulvIndex+xbreak+1;
int olhIndex = ulvIndex+ybreak+1;
int orhIndex = ulvIndex+xbreak+ybreak+1;
```

die anderen sieben umgebenden Punkte bestimmen. Zur Ermittlung dieser acht nächstliegenden Punkte bedarf es also lediglich drei Divisionen, drei Integer Subtraktionen und elf Integer Additionen. Es müssen keine Vektorlängen berechnet und keine Schleifen durchlaufen werden. Über diese Indizes kann man nun direkt die entsprechenden `structs` ansprechen und deren Informationen verrechnen.

In Abb. 5.12 ist zur besseren Anschauung der Abstand in x-Richtung erhöht worden. Da es sich um ein homogenes Gitter handelt, ist in der Realität der x-Abstand der Punkte genau so groß wie der y- und z- Abstand.

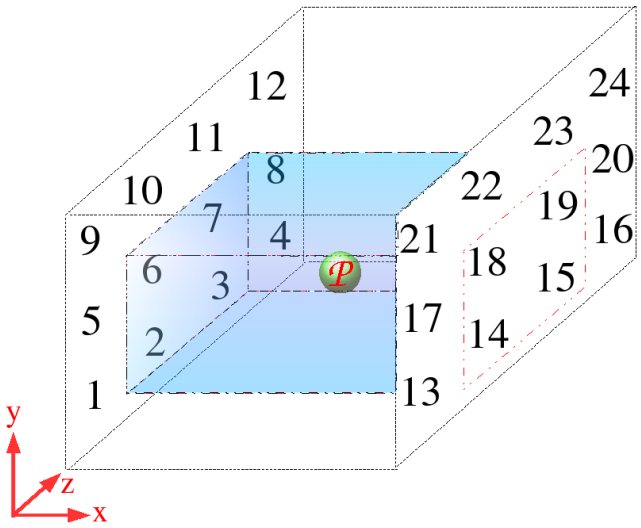

Abbildung 5.12: Durch strukturierte Indizierung ist das Zuordnen der Nachbarschaften effizient durchführbar. *Ybreak* wäre hier 4 und *xbreak* 12. Punkt $\mathcal{P}$ wird also von den Punkten 2, 3, 6, 7, 14, 15, 18 und 19 umschlossen.

6 Ergebnisse

In diesem Kapitel werden die Ergebnisse dieser Arbeit vorgestellt. Größtenteils besteht die Veranschaulichung aus Bildern mit angegebenen Bildwiederholungsraten oder Visualisierungen zur Darstellung der Genauigkeit des Verfahrens.

6.1 Gitterpunkte

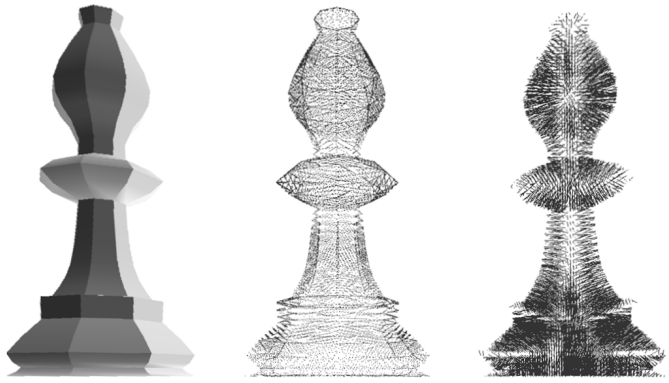

Abbildung 6.1: Ein konkaves Objekt, hier ein Schach-Läufer, zur Berechnung der Gitterpunktinformation. Links das Originalobjekt, mittig die Oberflächenpunkte, welche den Gitterpunkten am nächsten sind. In der rechten Abbildung sind Verbindungslinien zwischen den Gitterpunkten, die innerhalb des Objekts liegen und den Oberflächenpunkten dargestellt, auf welche bei der Kollision projiziert wird.

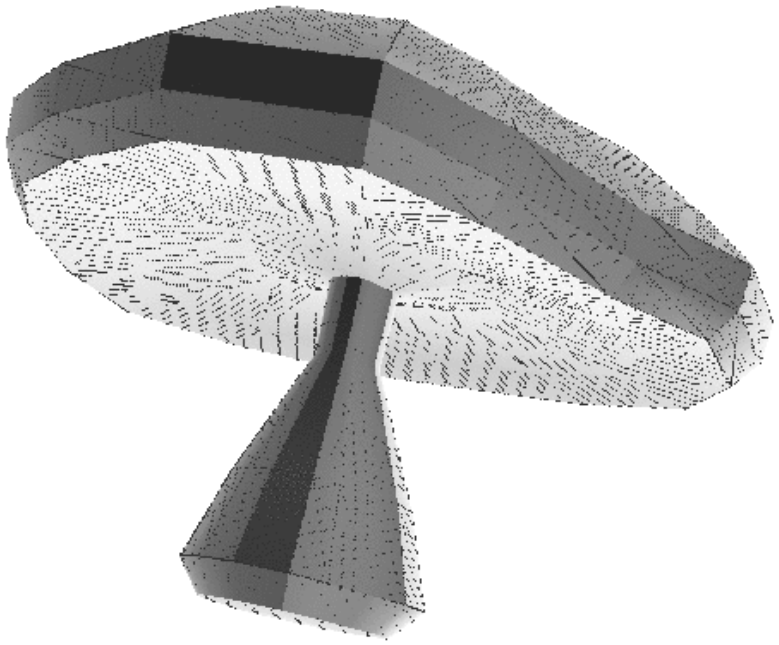

Abbildung 6.2: Stellt man die ermittelten Oberflächenpunkte zeitgleich mit dem Originalobjekt dar, so erkennt man die Genauigkeit des Verfahrens. Die Genauigkeit kann durch das Erhöhen der Unterteilungsanzahl stets verbessert werden. Das Verfahren ist von Mesheckpunkten unabhängig. Zur Berechnung des Verschiebungsfeldes wird zwischen den Oberflächenpunkten noch interpoliert.

6.2 Deformation

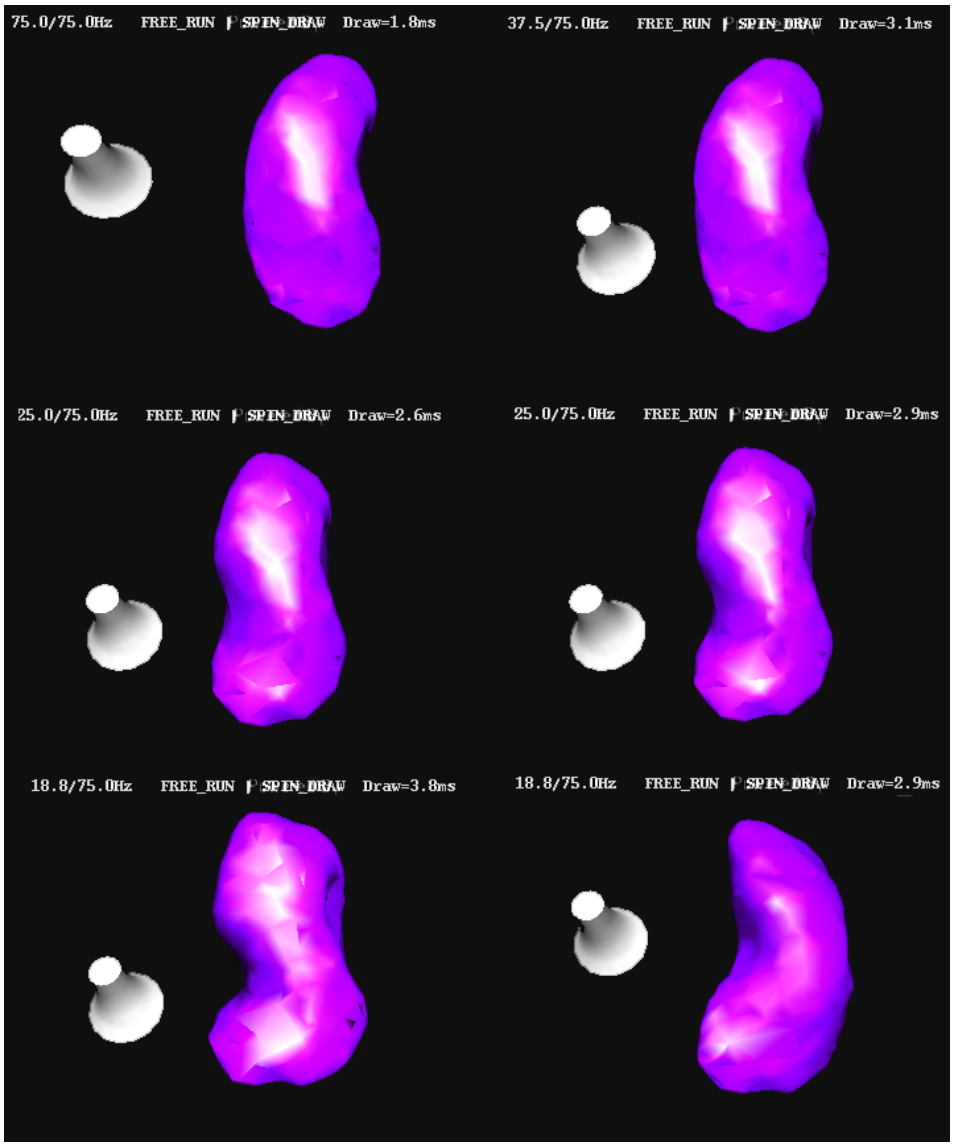

Abbildung 6.3: Die Deformation geht von einer Kraft aus, welche von dem starren Objekt in einer gaussglockenförmigen Verteilung ausgeht. Dieses Verfahren wurde zur Darstellung der Deformation ohne Kollisionserkennung integriert und hier anhand eines Nierenmodells mit 2743 Punkten dargestellt. Obwohl annähernd alle Knoten von der Deformation betroffen sind, kann die Bildwiederholrate den Anspruch der Echtzeitfähigkeit halten. Erklärung zum Einbruch der Bildwiederholrate in Abschnitt 7.2, Punkt **5.**.

6.3 Kollisionserkennung

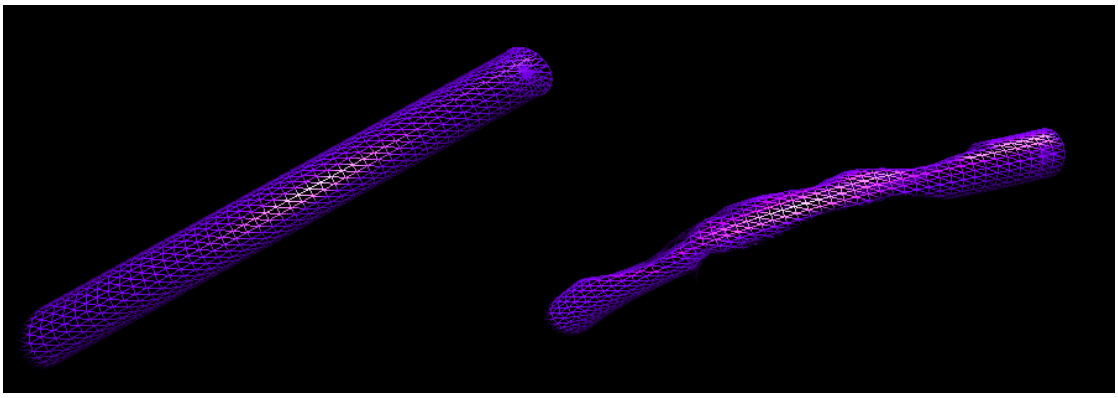

Abbildung 6.4: Zur Kontrolle der Kollisionserkennung wurde eine Funktion integriert, die plastische Deformation möglich macht. So können Eindringtiefen und die daraus resultierenden Objektmodifikationen nachhaltig sichtbar gemacht und Topologieänderungen verwirklicht werden

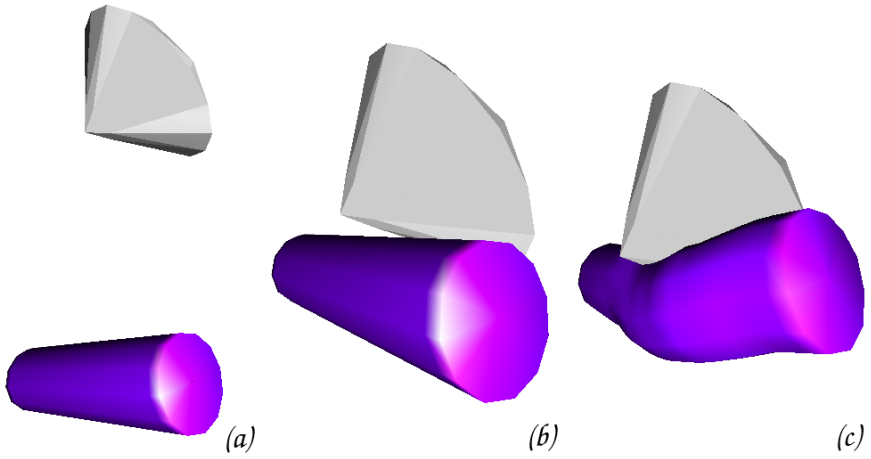

Abbildung 6.5: Kollisionserkennung und Deformation für ein aus 324 Elementen bestehendes flexibles Objekt (violett) und ein auf 8064 Gitterpunkten aufsetzendes starres Objekt. a) Kein Kontakt und Objekt außerhalb der Bounding Box: 75 fps, b) Kein Kontakt und Objekt innerhalb der Bounding Box: 75 fps, c) Kontakt und Deformation: 75 fps.

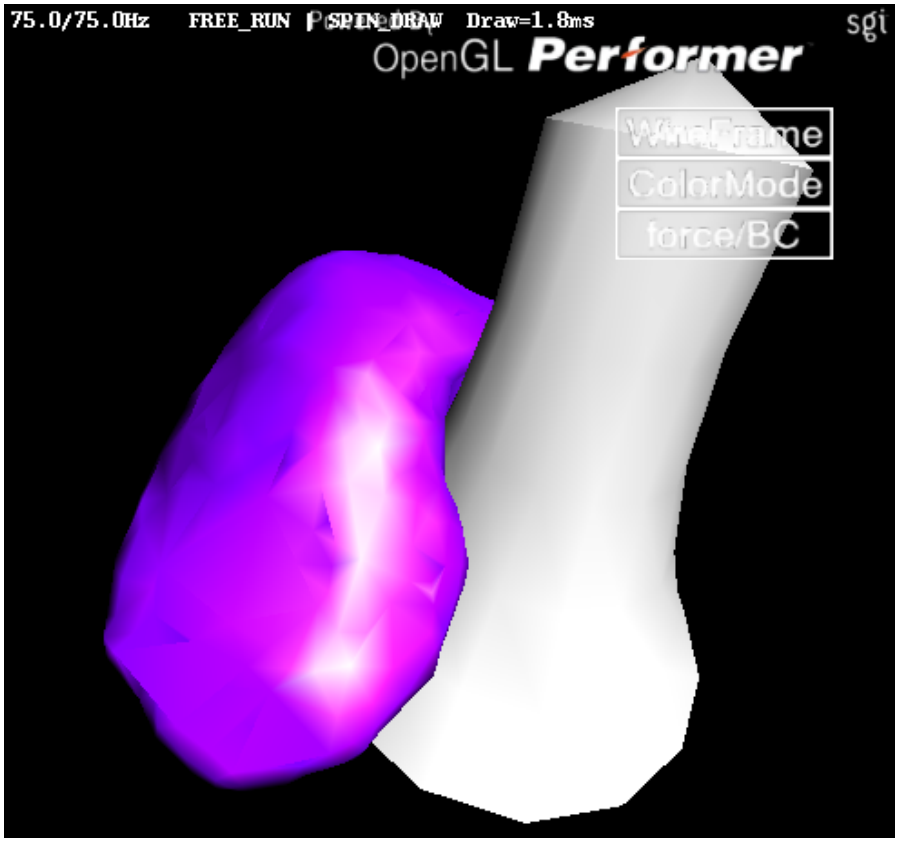

Abbildung 6.6: Selbst bei größeren Eindringtiefen kann das System den Anspruch auf Echtzeitfähigkeit halten (2800 Elemente großes Nierenmodell). Die angegebenen 75 Bilder pro Sekunde beziehen sich auf den erreichten Ruhezustand, beim Eindringen lagen die Wiederholraten zwischen 28 und 75 fps.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

In dieser Arbeit konnte ein echtzeitfähiges System zur Kollisionserkennung und Deformation vorgestellt werden. Anhand von vorberechneten Gitterpunktinformationen findet zunächst eine Kollisionserkennung statt. Im Falle einer Berührung wird die Eindringtiefe über alle Oberflächenpunkte bestimmt und die Kräfte errechnet, die auf das flexible Objekt ausgeübt werden. Sowohl das flexible als auch das starre Objekt können von beliebiger Form sein. Das System kann sehr hoch auflösende starre Objekte verarbeiten ($>$ 20.000 Punkte), was zu einer sehr genauen Bestimmung des Oberflächenpunktes führt, auf den das flexible Objekt bei Kollision abgebildet wird. Der bei der Interpolation entstehende Fehler ist sehr gering. Die Anzahl der Punkte des flexiblen Objekts ist speicherbedingt limitiert (vgl. Abschnitt 7.2: Ausblick). Bei einer maximal vorzuberechnenden Auflösung von etwa 5000 Punkten ist das System weiterhin echtzeitfähig.

Die im Rahmen dieser Arbeit gewonnenen Erkenntnisse lassen erschließen, dass mit Anwendung von Vorberechnungen eine Echtzeitkollisionserkennung und eine Deformation möglich ist. Hierzu werden lange Vorberechnungszeiten in Kauf genommen, die an die Speichergrenzen der heutigen Computersysteme gehen. Selbst für den Fall, dass das in Abschnitt *Ausblick*, Punkt **1.** angesprochene Problem gelöst wird, ist die Invertierung der Steifigkeitsmatrix immer noch sehr zeitaufwändig. Für kleine elastische Deformationen ist die lineare Näherung ausreichend.

Zum Durchführen einer nichtlinearen Deformation muss die Steifigkeitsmatrix geändert und wieder invertiert werden. Diese Operationen sind bei einer sinnvollen Objektgröße zum momentanen Zeitpunkt nicht in Echtzeit möglich. Ein zukunftsweisendes Simulationssystem muss zur Darstellung nichtlinearer Deformationseigenschaften fähig sein. Aus diesem Grunde ist nach Meinung des Autors bei der Weiterentwicklung von Deformationssimulatoren eher an die Entwicklung einer neuartigen Methodik zu denken als an eine Weiterführung dieses Ansatzes. Diese Bedenken beschränken sich ausschließlich auf die Deformation, die Verfahrensweise in der Kollisionserkennung hingegen wird als sehr interessanter Ansatz betrachtet.

7.2 Ausblick

Die in dieser Arbeit erstellten Inhalte können sinnvoll durch das Ergänzen von folgenden Ansätzen erweitert werden.

1. Für die flexiblen Objekte ist bei der Vorberechnung eine Invertierung der $\mathcal{K}$-Matrix erforderlich. Hierzu muss die gesamte Matrix im Speicher resident sein. Bei 1000 Elementen ist dies ein Speicheraufwand von 1000 * 1000 * 9 * 8(sizeof(double)). Dies entspricht einem Speicherbedarf von 72MB. Bei 10.000 Elementen wird bereits ein Speicherplatz von 7,2 GB benötigt, was von den heutigen Computersystemen nicht effizient verwaltet werden kann. Eine Behebung dieses Problems würde von großem Nutzen sein.

2. Die Bounding Box wird um das rigide Element in Form eines achsenparallel Würfels gelegt. Innerhalb dieser BB findet eine Kollisionserkennung nur aufgrund der Gitterpunktinformation statt. Bei stark konkaven Objekten wie z.B. einer Schüssel werden mit dem Eintreten in die Bounding Box von oben viele Gitterpunktinformationen abgefragt, obwohl man lange nicht den Boden berührt. Man spricht in diesem Fall vom so genannten *teapot in a stadium*-Problem. In dieser Arbeit wurde der Kollisionserkennungsweg diesbezüglich nicht geändert, da die Performanz ausreichend war. Beim Einsatz größerer Modelle kann eine solche Beschleunigung erforderlich sein.

3. Bei der Vorberechnung wird zu jedem Gitterpunkt die nächste Oberfläche angegeben. Genau in der Mitte des Objekts wechselt die Seite, auf die der Punkt des flexiblen Objekts abgebildet wird. Bei dünnen Objekten ist diese Schwelle nach kurzer Eindringtiefe erreicht. Die kollidierenden Punkte des flexiblen Objekts werden dann auf die Rückseite des starren Objekts abgebildet. So wird zum Einen das starre Objekt *verschluckt*, zum anderen kann die Eindringtiefe nicht mehr korrekt berechnet werden. Es entstehen Sprünge, wenn benachbarte Kollisionspunkte auf verschiedene Oberflächen projiziert werden. Bei einem Eindringen muss also verhindert werden, dass die Seite der Projektion wechselt. Ein Ansatz zur Behebung dieses Problems ist das Einfügen eines dreidimensionalen inneren Skeletts, welches von den Kollisionspunkten nicht passiert werden kann. So wird erreicht, dass diejenige Oberfläche, welche als erstes kollidiert, nicht gewechselt wird. Hieraus entstehen jedoch weitere Probleme:

   - Es darf keine Festlegung auf spezielle Oberflächen geben, damit das starre Objekt während der Kollision entlang des flexiblen Objekts *geschoben* und *gedreht* werden kann.
   - Die Kraftberechnung muss weiterhin korrekt erfolgen, ein Ansatz ist hier das Beibehalten einer ursprünglichen, spannungslosen Gestalt des flexiblen Objekts, welche nicht dargestellt wird, von der aber die Kraftberechnung ausgeht. Ein solcher Ansatz wurde in dieser Arbeit bereits realisiert, allerdings ist die berechnete Kraft bei dünnen starren Objekten meist gering.
   - Die Berechnungen müssen unempfindlich gegen die Eindringgeschwindigkeit sein. Wenn garantiert ist, dass die Gitterpunktgruppe, in der das Skelett definiert ist, in die Interpolation einbezogen wird, sind die o.g. Schritte mit wenigen Problemen zu realisieren. Da dies durch die jeweilige Wiederholfrequenz der Kollisionsabfrage jedoch nicht gegeben ist, ist eine Garantie nicht gewährleistet, da die Region der Skelettgitterpunkte zwischen den Abtastungen übersprungen werden kann.

4. Eine Kollisionserkennung und -behandlung findet nur in Bezug auf Punkte des flexiblen Objekts statt. Bei einem stark vereinfachten bzw. optimierten

Objekt kommt es zu langen Verbindungskanten ohne Stützpunkt. Durchdringt das starre Objekt eine solchen Kante, erfolgt im Rahmen der hier vorgelegten Arbeit keine Kollision. Ein Kollisionstest, der Objektkanten berücksichtigt und dabei den Anspruch auf Echtzeitfähigkeit nicht verliert, wäre eine sinnvolle Erweiterung. Eine weiter Lösungsmöglichkeit wäre das dynamische Erhöhen der Punktanzahl in der Umgebung des KollisionsObjekts. So könnten größere Objekte mit geringer Punktzahl verwendet werden, was dem in **1.** angesprochenen Problem entgegenkäme.

5. Wie in Abb. 6.3 zu erkennen ist, wird die Bildwiederholrate geringer, obwohl der Geometrieumfang und die Beleuchtung gleich bleiben. Dies liegt an der Softwareumgebung, auf die das System aufgebaut ist. Performer™ unterstützt multiprocessing, aber kein multithreading. Es findet also ein sequentieller Programmablauf statt, in dem der Draw-Prozess erst nach Beendigung des Calculate-Prozesses aufgerufen wird. Man könnte diesen Draw-Prozess entkoppeln, in dem man einen Ringbuffer (flux) eröffnet. Daraufhin könnte man allerdings nicht mehr sehen, wie lange die Berechnung des Calculate-Prozesses dauert. Eine garantiert hohe Bildwiederholrate kombiniert mit einer Anzeige, wie lange der Calculate-Prozess dauert, würde den optischen Eindruck des Systems verbessern.

6. Das Einbinden in ein stereoskopisches, immersives Display (i-Cone, Cave) würde die Bedienung intuitiver und einfacher gestalten.

7. Das Hinzufügen eines haptischen Eingabe- und Steuergerätes mit Kraftrückgabe (z.B. Phantom) wäre einfach einzubinden und würde die Anwendung realitätsnaher erscheinen lassen.

A Anhang

A.1 Begriffe

Punktwolke — Eine Ansammlung von Punkten im Raum, jeweils durch x, y, und z-Koordinate definiert.

Konvex — Als konvex (gewölbt, gerundet) bezeichnet man die Form eines Körpers, dessen Oberfläche nach außen gewölbt ist. Definition: Eine Fläche oder ein Körper sind konvex, wenn für je zwei darin enthaltene Punkte auch alle Punkte der Verbindungslinie zum Objekt gehören [2].

Konkav — Konkav ist eine fachsprachliche Bezeichnung für nach innen gewölbt, ausgehöhlt, gekrümmt. Eine Oberfläche eines Körpers wird als konkav bezeichnet, wenn sie eine Wölbung nach innen hat [2].

Immersion — Der Begriff der Immersion meint das Eintauchen in eine künstliche Welt durch Auflösung der räumlichen Grenzen [2]. Die Immersion wird durch die Ausgabegeräte und die verwendete Virtuelle Realität beeinflusst.

Boundary Element Methode

Die Boundary-Element-Methode (BEM) verwendet analytische Re-Formulierungen partieller Differentialgleichungen, die nur die Oberflächenvariablen enthält. So wird bei der BEM nur die Oberfläche der betrachteten Struktur diskretisiert, nicht jedoch deren Volumen. Materialeigenschaften innerhalb des Volumens werden nicht berücksichtigt [28][29].

Bounding Box

Eine Bounding Box umschließt ein Objekt vollständig, so dass sich keine Teile des Objekts außerhalb der Bounding Box befinden.

online

In dieser Arbeit wird der Begriff *online* verwendet, wenn diejenigen Berechnungen behandelt werden, welche in Echtzeit in der Simulationsumgebung durchgeführt werden.

offline

In dieser Arbeit wird der Begriff *offline* verwendet, wenn diejenigen Berechnungen behandelt werden, welche nicht echtzeitfähig sind und vorberechnet werden können.

Voxel

Ein Voxel bezeichnet eine quaderförmige Zelle innerhalb eines regelmäßig aufgeteilten Quaders oder unbegrenzten Raumes. Der Begriff wird vor allem in der 3D-Computergrafik verwendet und setzt sich aus den Wörtern Volume und Pixel zusammen [2].

Triangle-Strip

Eine platzsparende Form der Speicherung von Dreiecken, wobei Punktinformation mehrfach ausgewertet wird. Diese Form der Speicherung ist in der Computergrafik sehr üblich.

Virtuelle Realität Als Virtuelle Realität wird im Computer simulierte Wirklichkeit bezeichnet. Unter virtueller, also scheinbarer Realität (engl.: virtual reality; VR) versteht man eine vom Computer simulierte, als dreidimensional erscheinende stereoskopische Umgebung, in die sich der Benutzer *hineinbegeben* kann [2].

A.2 AVANGO

Das Software Framework Avango wird seit 1996 in der Abteilung Virtual Environments des Fraunhofer Instituts für Medienkommunikation (IMK.VE) entwickelt und dient zur Erstellung verteilter, interaktiver VE Anwendungen. Es unterstützt eine Vielzahl von Display-Konfigurationen und Eingabegeräten, zu denen z.B. unterschiedliche Trackingsysteme oder 3D Interaktionsgeräte zählen. Um so viele menschliche Sinne wie möglich anzusprechen, kann zur Ausgabe Spezialhardware wie Vibrationsböden, Duftzerstäubungssysteme oder 3D-Soundsysteme verwendet werden. Die Anforderungen an ein solches Entwicklungssystem für virtuelle Umgebungen sind: Performance, Flexibilität, schnelle Anwendungsentwicklung, Erweiterbarkeit und Portierbarkeit. Als Basis dient das OpenGL Performer Toolkit der Firma SGI. Darauf baut das objektorientierte Avango Framework auf, das eine Programmierschnittstelle (API) in der Sprache C++ zur Verfügung stellt und die Erstellung von anwendungsspezifischen Klassen erlaubt. Zusätzlich zur C++ API bietet Avango eine vollständige Anbindung der Interpreter-Sprache Scheme, von der aus alle high-level Avango Objekte erstellt und manipuliert werden können. Komplexe und performance-kritische Funktionalitäten werden durch Ableitungen und Erweiterungen bestehender Avango-Klassen in C++ implementiert, während die typische Avango Anwendung selbst aus einer Sammlung von Scheme-Skripten besteht, die Avango-Objekte instanziieren, ihre Methoden anwenden, Werte von Feldern setzen und die Beziehungen zwischen Objekten definieren. Die Komponenten der Avango-Entwicklungsumgebung werden in Abb. A.1 dargestellt. Die

C++ API erlaubt die Definition zweier verschiedener Kategorien von Objektklassen: *Nodes* bieten eine objektorientierte Scene-Graph API, die die Repräsentation und die Darstellung komplexer Geometrien ermöglicht, während *Sensors* Avango's Schnittstellen zur realen Welt sind und benutzt werden, um externe Gerätedaten in eine Anwendung zu importieren. Alle Avango-Objekte werden als sogenannte Fieldcontainer bezeichnet, welche den Status von Objekten als eine Sammlung von Feldern repräsentieren.

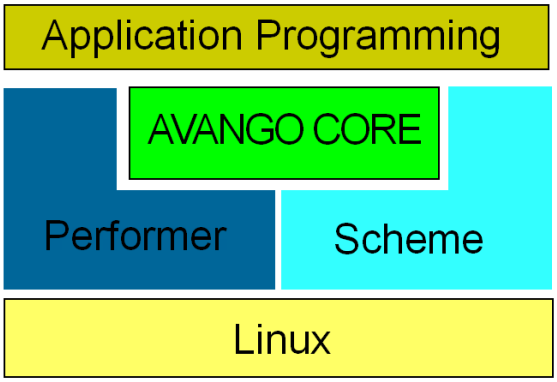

Abbildung A.1: Avango-Framework

Abbildungsverzeichnis

Literaturverzeichnis

[1] O.C. Zienkiewcz, R.L. Taylor
The Finite Element Method
vol. 1, Edition 5, Butterworth-Heinemann, Oxford, 2000.

[2] *Das Net-Lexikon*
URL: www.net-lexikon.de

[3] Bro-Nielsen M., Cotin S.
Real-Time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation Eurographics'96, Vol.15 ,1996 No.3, C57-C66 (note: in Eq.(23) the cylindric interchange of the indices should be followed by change of the sign).

[4] Wikipedia Internet-Enzyklopädie
URL:http://de.wikipedia.org/wiki/Hauptseite

[5] Klimenko S., Nikitina L., Nikitin I.,
Flexible Materials in Avango™ Virtual Environment Framework
Computer Graphics International, 2003. Proceedings, 2003, pp. 84- 89.

[6] L.D. Landau, E.M. Lifschitz
Theory of Elasticity
Volume VII of Theoretical Physiys, Moscow, Nauka, 1970.

[7] Picibono G., Delingette H., Nicholas H.-A.
Non-Linear Anisotropic Elasticity for Real-Time Surgery Simulation,
INRIA Research Report 4028, October, 2000.

[8] Y. Zhuang, J.F. Canny,
Real-Time global deformations,
In Algorithmic and Computational Robotics, pp97-107, Natrick MA, 2001. A.K. Peters.

[9] Hirota, G., Fisher, S. and Lin, M.,
Simulation of Non-penetrating Elastic Bodies Using Distance Fields,
UNC Technical Report TR00-018, 2000.

[10] M. Lin, A.Gottschalk,
Collision Detection between Geometrical Models
A survey, Proc. IMA Conference on Mathematics of Surfaces, 1998.

[11] McNeely W.A., Puterbaugh K.D., Troy J.J.
Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling
Proceedings of the 26th annual conference on Computer Graphics and interactive techniques, pp: 401 - 408, 1999

[12] Renz M., Preusche C., Pötke M., Kriegel H.-P., Hirzinger G.
Stable Haptic Interaction with Virtual Environments Using an Adapted Voxmap-Pointshell Algorithm
Eurographics Conference, Birmingham, UK, 2001.

[13] Garcia-Alonso, A., Serrano, N., und Flaquer J.,
Solving the Collision Detection Problem
IEEE Computer Graphics and Applications, vol. 14, no. 3, pp. 36-43, 1994.

[14] Kaufman A., Cohen D., Yagle R.;
Volume Graphics
IEEE Computer, 26(7), pp. 51-64. 1993.

[15] Logan, I.P. et al.
Virtual Environment Knee Arthroscopy Training System
Society for Computer Simulation, Simulation Series, vol. 28, no. 4, pp. 17-22, 1996.

[16] R. J. Adams, B. Hannaford,
Stable Haptic Interaction with Virtual Environments

Proc. IEEE Transactions on Robotics and Automation, vol. 15, no. 3, June 1999, pp. 465-474.

[17] S. Gibson
Using Linked Volumes to Model Object Collisions, Deformation, Cutting, Carving and Joining
MERL - Mitsubishi Electric Research Laboratory , Cambridge, June 2000, IEEE Trans. on Visualization and Compuer Graphics, Vol. 5, No. 4, 1999, pp. 333-348.

[18] S. Gibson,
Beyond volume rendering: visualisation, haptic exploration and physical modeling of element-based objects
Visualisation in Scientific Computing, pp. 10-24. Springer-Verlag 1995.

[19] Morten Bro-Nielsen, PhD,
Fast Finite Elements for Surgery Simulation
HT Medical Inc., Rockville, Maryland, USA; Department of Mathematical Modelling, Technical University of Denmark, Denmark, 1997.
www.mortenbronielsen.net/documents/mmvr5_paper.pdf

[20] Nikitin I., L. Nikitina, P.Frolow, G. Goebbels, M. Goebel, S. Klimenko, G.M.Nielson
Real-time simulation of elastic objects in Virtual Environments using finite element method and precomputed Green's Functions
Eighth Eurographics Workshop on Virtual Environments, pp. 047–052, 2002.

[21] James D., Pai D.
Artdefo - Accurate Real Time Deformable Objects
Computer Graphics, vol. 33, pp.65-72, 1999.

[22] U. Meier, C. Monserrat, N.-C. Parr, F.J. Garcia, J.A. Gil,
Realtime Simulation of Minimally Invasive Surgery with Cutting Based on Boundary Element Methods
Lecture Notes in Computer Science, 2001 V.2208, p.1263.

[23] Sven Kürten
Simulation von elastischem Gewebeverhalten unter Echtzeitsapekten
Diplomarbeit, Universität Bonn, 2003.

[24] Christian Wienss,
Implementation eines Filters zur Korrektur der dreidimensionalen Delaunay Triangulation und das Einbinden der Modelle in einen Deformationssimulator
Studienarbeit Feb. 2004
URL:http://geri.uni-koblenz.de/Studienarbeiten/SA_Christian_Wienss.pdf

[25] W. Wang, Y.K. Choi, B. Chan, M.S. Kim, and J.Y. Wang,
Efficient collision detection for ellipsoids using separating planes. Computing, vol. 72, 2004, pp. 235-246.

[26] Apel J.,
Delaunay Triangulation
TU-Ilmanau, URL: www.stud.tu-ilmenau.de/japel/vortrag/gdv/delaunay.htm

[27] *Numerical Recipes*
URL: www.nr.com

[28] Internetseite der Fraunhofer IMK
URL:http://www.imk.fhg.de

[29] Internetseite der Uni Stuttgart
URL: http://www.mecha.uni-stuttgart.de