Interaktive Echtzeitsimulation deformierbarer Oberflächen für Trainingssysteme in der Augenchirurgie

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Dipl.-Math.
Kathrin Weber
aus Mainz

Mannheim, 2009

Dekan: Professor Dr. Felix Freiling, Universität Mannheim
Referent: Professor Dr. Reinhard Männer, Universität Heidelberg
Korreferent: Professor Dr. Peter Fischer, Universität Heidelberg

Tag der mündlichen Prüfung: 17.12.2009

für meine Eltern

Zusammenfassung

Diese Arbeit beschreibt Simulations-Algorithmen für virtuelle Augenoperationen und leistet somit einen Beitrag zur Verbesserung der ophthalmochirurgischen Aus- und Weiterbildung.
Die operative Behandlung einer Augenerkrankung erfordert häufig, dass der Chirurg membranartiges, flächig angewachsenes Gewebe aus dem Augeninneren entfernt. Da die Manipulation von Membranen den Schwerpunkt vieler augenchirurgischer Eingriffe darstellt, konzentriert sich vorliegende Arbeit auf die Simulation deformierbarer Oberflächen. Um eine Membran räumlich zu diskretisieren, wird deren Oberfläche trianguliert und das resultierende Drahtgittermodell dient als Basis für ein Feder-Masse-System, das die Deformation der Membran im Verlauf der Simulation berechnet.
Da eine reale Membran üblicherweise fragmentiert, wenn der Chirurg versucht, sie vom übrigen Augengewebe zu lösen, muss auch die virtuelle Membran reißen können. Diese Arbeit formuliert Bruchkriterien, die Ort, Richtung und Länge einer Rissausbreitung voraussagen und eine physikalisch plausible Riss-Simulation garantieren. Aufbauend auf diesen Kriterien werden zwei Reiß-Algorithmen definiert, die sich darin unterscheiden, wie sie das Feder-Masse-Gitter adaptieren, um den Riss topologisch umzusetzen.
Um eine anhaftende Membran vom Untergrund abzulösen, benutzt ein Augenchirurg – unter anderem – nadelähnliche Instrumente. Daher beschreibt vorliegende Arbeit algorithmische Lösungen für eine Interaktion zwischen virtuellen Nadel-Instrumenten und simulierter Membran. Das Grundgerüst der Interaktion besteht aus einer geeigneten Kollisionserkennung und einer reibungsfreien Kollisionsantwort. Als Erweiterung des Grundgerüstes verfügt die Kollisionsantwort zusätzlich über Modelle für Gleit- und Haftreibung, die wesentlich dazu beitragen, dass der Benutzer eine Membran realistisch manipulieren kann.
Die im Rahmen vorliegender Arbeit entwickelten Algorithmen wurden in drei verschiedene Trainingsmodule des kommerziellen Augenoperationssimulators EYESi integriert, der von der Firma VRmagic vertrieben wird. Zwei dieser Module sind bereits seit längerem Teil der VRmagic-Produktpalette und tragen somit schon heute zur Aus- und Weiterbildung von Augenchirurgen bei.

Danksagung

Zuallererst möchte ich Herrn Prof. Dr. Reinhard Männer meinen Dank aussprechen, der meine Doktorarbeit betreute und dessen Tür mir bei Fragen – auch wenn der Terminkalender mal voll war – stets offen stand. Herrn Prof. Dr. Peter Fischer danke ich dafür, dass er sich als Korreferent zur Verfügung gestellt hat.

Ganz besonderer Dank gebührt der Firma VRmagic, ohne deren Kooperation mit der Universität Mannheim diese Doktorarbeit nicht möglich gewesen wäre. Die tolle Arbeitsatmosphäre im *VRmagic Core Development Center* und meine Integration in das EYESi-Entwicklerteam haben dazu beigetragen, mich nie „nur“ als Doktorandin sondern immer auch als VRmagic-Mitarbeiterin zu fühlen. Ich danke allen „VRmagicians“, die mich bei meiner Arbeit unterstützt haben. Hervorheben möchte ich Clemens Wagner und Johannes Grimm, die immer ein offenes Ohr für mich hatten und sich die Zeit genommen haben, meine Dissertation korrekturzulesen. Thomas Ruf danke ich dafür, dass er mir mit seiner enormen Fachkompetenz über die eine oder andere technische Hürde hinweggeholfen hat.

Nicht zu vergessen meine Kollegen von ViPA – der Arbeitsgruppe des Instituts für Computergestütze Medizin –, die mich während meiner Doktorarbeit begleitet haben und mit denen ich über Probleme und Lösungsansätze diskutieren konnte. Im Besonderen danke ich Stephan Diederich, der meine komplette Zusammenschrift korrekturgelesen hat.

Für die Bewältigung sämtlicher verwaltungstechnischer Angelegenheiten danke ich Andrea Seeger und Christiane Glasbrenner.

Es war mir eine Freude, mit so vielen kompetenten, motivierten, netten und hilfsbereiten Menschen zusammenarbeiten zu können. Ich beende diese Dissertation mit dem guten Gefühl, sowohl wissenschaftliche Arbeit als auch einen sinnvollen Beitrag zur augenchirurgischen Ausbildung geleistet zu haben.

Felix, bei dir mache ich es kurz: Danke!

Inhaltsverzeichnis

Kapitel 1

Einleitung

With today's knowledge and technology, up to 80% of global blindness is preventable or treatable.

WHO Report 2009 [134]

Intraokulare Operationen gehören zu den schwierigsten chirurgischen Eingriffen. Die Strukturen im Auge sind extrem empfindlich und um sie nicht zu verletzen, muss der Chirurg seine Instrumente mit einer Präzision im Submillimeterbereich kontrollieren. Erschwerend für die Hand-Auge-Koordination kommt hinzu, dass der Eingriff unter einem Stereomikroskop durchgeführt wird.
In vielen medizinischen Bereichen ist es üblich, dass auszubildende Ärzte eine Behandlungsmethode direkt am Patienten üben. Diese gängige Praxis ist prinzipiell problematisch und speziell in der Augenchirurgie kann die mangelhafte Technik eines Anfängers zu schwerwiegenden Komplikationen führen.
Daher gehören sogenannte *Wetlabs* zu den üblichen Methoden einer augenchirurgischen Ausbildung: Die Augen toter Tiere werden präpariert und in einer Vorrichtung fixiert, um als Ersatz für den realen Patienten zu dienen. Diese Art der Ausbildung hat jedoch mehrere Nachteile: Tieraugen unterscheiden sich anatomisch von menschlichen Augen und totes Gewebe hat andere physikalische Eigenschaften als lebendes Gewebe. Einige Augenerkrankungen treten bei Tieren nicht oder nur selten auf. Schlachttiere sind im Allgemeinen vollkommen gesund und Tiere mit Augenerkrankungen gezielt zu züchten ist ethisch bedenklich. Selbst wenn ein präpariertes Auge zur Verfügung steht, das ein bestimmtes Krankheitsbild zeigt, kann dieses nur einmal als chirurgisches Versuchsobjekt dienen und ist anschließend unbrauchbar. An Tieraugen ist die Übung chirurgischer Eingriffe daher nur eingeschränkt möglich.
Für eine optimale Aus- und Weiterbildung von Augenchirurgen werden Trainingssysteme benötigt, an denen die Behandlung beliebiger Krankheitsbilder erlernt

und zu Übungszwecken beliebig oft wiederholt werden kann. Diese Anforderungen können computergestützte Simulatoren erfüllen: In einer *Virtuellen Realität* (VR) lassen sich Krankheitsbilder aller Art computergrafisch erzeugen und ein Trainingsprogramm des Simulators kann so oft neu gestartet werden, bis der Benutzer das gewünschte Ergebnis erzielt. Darüber hinaus ist es möglich, dass ein Simulator Aufgaben eines Ausbilders übernimmt: Er kann den Benutzer über Sprach- oder Textausgaben anleiten, auf Fehler hinweisen und abschließend bewerten, wie gut der Eingriff gelungen ist.
Simulatorgestützte Ausbildung hat enormes Potential, das allerdings nur dann ausgeschöpft werden kann, wenn der Simulator den Benutzer in eine überzeugende VR eintauchen lässt. Voraussetzung hierfür ist, dass ein Operationssimulator über eine Mensch-Maschine-Schnittstelle verfügt, die den Einsatz chirurgischer Instrumente möglichst naturgetreu nachbildet. Hard- und Software müssen die Sinne des Benutzers auf eine Art stimulieren, die ihn vergessen lässt, dass er eine virtuelle und keine reale Operation durchführt.
Diese Anforderungen realisiert der von der Firma VRmagic entwickelte Augenoperationssimulator EYESi, der weltweit für die augenchirurgische Aus- und Weiterbildung eingesetzt wird. Die EYESi-Hardware bildet – unter anderem – den Patientenkopf, das Stereomikroskop und die Instrumente nach. Die Software von EYESi umfasst sogenannte Trainingsmodule, von denen jedes einen bestimmten chirurgischen Eingriff simuliert. Die Palette an angebotenen Modulen wird ständig erweitert und mit jedem neuen Modul wächst der Bedarf an neuen Simulations-Algorithmen. Diese steigenden Anforderungen an die EYESi-Simulationen bilden den Ausgangspunkt für vorliegende Arbeit.

1.1 Schwerpunkte der Arbeit

Im Verlauf einer Augenoperation besteht häufig die Notwendigkeit, dass der Chirurg mit seinen Instrumenten membranartiges Gewebe bearbeitet, das flächig am Augeninneren anhaftet. Die Manipulation solcher Membrane ist ein wesentlicher Aspekt vieler augenchirurgischer Eingriffe: Wenn eine *Epiretinale Membran* (ERM) auf der Netzhaut wuchert und das Sehvermögen einschränkt, wird diese im Rahmen eines sogenannten ERM-Peelings vollständig aus dem Auge entfernt. Es kommt jedoch auch vor, dass Teile nicht-pathologischer Membrane beseitigt werden, um darunter liegendes Gewebe für die nächsten Schritte der Operation freizulegen. Ein Beispiel ist das Peeling der *Inneren Grenzmembran* (IGM bzw. ILM für *Internal Limiting Membrane*), welche die oberste Schicht der Netzhaut definiert.
Die weltweit am häufigsten durchgeführte Augenoperation ist die Behandlung eines *Katarakts* (*Grauen Stars*). Ein Teilschritt der Katarakt-Operation heißt *Kapsulorhexis* und definiert, wie der *Kapselsack* zu öffnen ist, der die (krankhaft

getrübte) Linse umschließt: Der Augenchirurg benutzt eine Pinzette oder/und ein nadelartiges Instrument, um ein Stück Membran aus dem Kapselsack heraus zu reißen. Die entstehende Öffnung muss kreisrund sein, zentriert auf der Linse sitzen und über einen bestimmten Radius verfügen, damit der Kapselsack – nach Entfernung der getrübten Linse – als stabiler Behälter für die Kunstlinse dienen kann.
Alle drei Beispiele chirurgischer Eingriffe basieren auf Interaktionen mit Membranen, die im Verlauf des Eingriffs reißen können bzw. gerissen werden müssen. Um effizient zu arbeiten und Komplikationen zu vermeiden, muss ein angehender Chirurg lernen, die Rissausbreitung in einer Membran durch gezielte Instrument-Interaktionen zu steuern. Besonders hohe Anforderungen an die Kontrolle über die Rissausbreitung stellt die Kapsulorhexis: Der Riss im Kapselsack muss einer exakt vorgegebenen Kreisbahn folgen.
Für eine Kapsulorhexis oder für ein Membran-Peeling verwendet der Augenchirurg – unter anderem – nadelähnliche Instrumente. Sie sind geeignet, um eine Membran anzuritzen oder um ein Stück Membran abzupellen, das sehr stark mit dem Untergrund verwachsen ist. Allerdings verlangt der Gebrauch von Nadel-Instrumenten Erfahrung und Feingefühl, da die Nadelspitze bei zu geringer Kraftausübung von der Membran abrutscht und bei zu hoher Kraftausübung Verletzungen verursachen kann.

Vorliegende Arbeit definiert Simulations-Algorithmen, mit denen sich virtuelle Operationsszenarien für Kapsulorhexis, ILM-Peeling und ERM-Peeling realisieren lassen. Der gerade beschriebene medizinische Hintergrund motiviert, welche Schwerpunkte die Arbeit setzt:
Eine virtuelle Augenoperation ist eine interaktive Echtzeitanwendung und sollte als solche mit einer Bildwiederholrate von mindestens 25fps laufen. Die Einhaltung der Echtzeitbedingung hatte bei sämtlichen Entwicklungsarbeiten oberste Priorität.
Kernanforderung aller Operationsszenarien ist die Manipulation von Membranen. Die Arbeit konzentriert sich daher auf die Simulation deformierbarer Oberflächen. Der Begriff „deformierbare Oberfläche“ deutet an, dass für das Membran-Modell eine räumliche Diskretisierung gewählt wird, welche die Dicke der Membran vernachlässigt. Eine volumetrische Diskretisierung – beispielsweise in Form eines Tetraeder-Gitters – würde die Anzahl notwendiger Diskretisierungspunkte erhöhen und die Membran-Simulation somit verlangsamen.
Ein physikalisch basiertes Deformationsmodell schafft die Grundlage für realistische Gewebe-Reaktionen einer virtuellen Membran. Das Membran-Modell hat die Aufgabe, interne Verzerrungskräfte in Echtzeit zu berechnen und externe Kräfte als Eingabe entgegen zu nehmen. Die Gesamtheit aller wirkenden Kräfte kontrolliert das Deformationsverhalten der Membran.

Den Implementierungen der Operationsszenarien liegt ein gemeinsames Framework zugrunde, das die wesentlichen Komponenten einer Membran-Simulation definiert. Neben dem Deformationsmodell gehören zu diesen Komponenten: ein numerisches Integrationsverfahren (das die Simulation zeitlich diskretisiert), eine Kollisionserkennung (welche die Kontakte zwischen Membran und anderen Simulations-Objekten detektiert) und eine Kollisionsantwort (die auf die Membran wirkende Kräfte berechnet, um Kollisionen wieder aufzulösen).
In Bezug auf die Kollisionsbehandlung (Kollisionserkennung inklusive -antwort) legt dieses Framework nur fest, welche Konzepte prinzipiell verwendet werden. Der Umgang mit Kollisionen zwischen Membran und einem Nadel-Instrument wird als separates Thema behandelt. Denn Nadel-Interaktionen stellen spezielle, besonders hohe Anforderungen an die Kollisionserkennung und -antwort: Eine Nadelspitze kann in die Membran eindringen, ohne einen der Diskretisierungspunkte auf der Membran zu berühren. Daher muss die Kollisionserkennung über eine sehr hohe Genauigkeit verfügen. Die Kollisionsantwort der Nadelspitze muss die detektierten Kollisionen nicht nur auflösen, sondern gleichzeitig eine realistische Interaktion mit der Membran ermöglichen. Damit sich die Membran kontrolliert manipulieren lässt, muss die Kollisionsantwort Modelle definieren, die ein Verhaken der Membran an der Nadelspitze abbilden können.
Neben der Realisierung von Nadel-Interaktionen bildet die Entwicklung von Reiß-Algorithmen einen weiteren Schwerpunkt der Arbeit. Ein Reiß-Algorithmus wird in zwei unabhängige Komponenten zerlegt: Zunächst entscheiden Bruchkriterien, ob die Membran reißt, wo und in welche Richtung sie reißt und wie weit sich der Riss ausbreiten soll. Die Definition geeigneter Bruchkriterien ist die Voraussetzung für eine physikalisch plausible Riss-Simulation. Sobald die Rissausbreitung berechnet ist, wird diese in die räumliche Diskretisierung der Membran eingefügt. Die notwendigen topologischen Operationen zusammen mit den Bruchkriterien definieren den vollständigen Reiß-Algorithmus – der speziell für eine virtuelle Kapsulorhexis hohe Anforderungen zu erfüllen hat: Der Chirurg muss den simulierten Kapselsack so reißen können, dass der Riss einer exakt vorgegeben Bahn folgt. Somit muss der Algorithmus vollkommene Kontrolle über die Rissausbreitung erlauben und den Riss topologisch genau so umsetzen, wie es die Bruchkriterien vorgeben. Die zu Beginn der Simulation gewählte räumliche Diskretisierung der Membran darf die topologische Umsetzung des Risses nicht einschränken.

Die neuen, getesteten Algorithmen wurden in EYESi-Trainingsmodule für Kapsulorhexis, ILM-Peeling und ERM-Peeling integriert. Mit dieser Integration waren vielfältige Modul-Entwicklungsarbeiten verbunden (die aus Platzgründen allerdings nicht vollständig beschrieben werden).
Ein zu Beginn der Arbeit existierender Kapsulorhexis-Prototyp wurde komplett überarbeitet. Das neue Kapsulorhexis-Modul wurde – nach eingehender Prüfung

durch erfahrene Chirurgen – in die VRmagic-Produktpalette aufgenommen und wird seitdem zusammen mit den übrigen EYESi-Trainingsmodulen verkauft. Im ILM-Modul wurden die bisher eingesetzten Algorithmen für Reißen und Nadel-Membran-Interaktionen ausgetauscht. Das ebenfalls bereits existierende Modul für ein ERM-Peeling war eines der ersten von VRmagic entwickelten EYESi-Trainingsmodule. Da es auf inzwischen veralteten Algorithmen beruht, hat VRmagic ein umfassendes Re-Design beschlossen, das im Rahmen dieser Arbeit begonnen wurde. Der Prototyp für das neue ERM-Modul ist in wesentlichen Teilen voll funktionsfähig und basiert auf den neu entwickelten Instrument-Membran-Interaktionen.

1.2 Überblick über die Arbeit

Plattform für sämtliche Entwicklungen dieser Arbeit war der Augenoperationssimulator EYESi. Um einen Eindruck davon zu vermitteln, wie EYESi den Benutzer in eine VR eintauchen lässt, umreißt Kapitel 2 Aufbau und Funktionsumfang des Simulators. Grundlagen über die Simulation deformierbarer Objekte sind in Kapitel 3 zu finden. Darauf aufbauend legt Kapitel 4 ein Simulations-Framework fest, das die Komponenten „Deformationsmodell“, „Numerische Integration“ und „Kollisionsbehandlung“ konkretisiert. Erweiterungen des Deformationsmodells – beispielsweise für das Anhaften und Ablösen einer Membran – werden ebenfalls in diesem Kapitel behandelt. In Kapitel 5 werden zwei Algorithmen für das Reißen von Membranen entwickelt. Beide basieren auf den gleichen Bruchkriterien, unterscheiden sich jedoch darin, wie sie einen Riss topologisch umsetzen. In einem entkernten EYESi-Trainingsmodul wird getestet, inwiefern die Reiß-Algorithmen den Anforderungen entsprechen. Kapitel 6 formuliert Algorithmen für die Interaktion zwischen Membran und einem Nadel-Instrument. Als Basis dient ein Modell für reibungsfreie Kontakte. Modelle für Gleit- und Haftreibung erweitern diese Basis und ermöglichen eine kontrollierte Manipulation der Membran. Auch die Interaktions-Algorithmen werden in einem entkernten EYESi-Modul auf ihre Funktionstüchtigkeit geprüft. Kapitel 7 beschreibt die EYESi-Trainingsmodule für Kapsulorhexis, ILM-Peeling und ERM-Peeling, die mit Hilfe der neuen Simulations-Algorithmen realisiert wurden. Kapitel 8 schließt die Arbeit mit einer Diskussion und einem Ausblick ab.

Kapitel 2

Der Augenoperationssimulator EYESi

EYESi (*Eye Surgery Simulator*) ist ein kommerzieller Trainingssimulator für die Ausbildung in der Augenchirurgie, der von der Firma VRmagic vertrieben und ständig weiterentwickelt wird. Er bietet sowohl angehenden als auch erfahrenen Chirurgen die Möglichkeit, ihre Techniken außerhalb des Operationssaals zu üben und trotzdem das Gefühl zu haben, eine reale Operation durchzuführen.
Seit der Gründung des Unternehmens im Jahr 2001 ist VRmagic zum weltweit führenden Anbieter von VR-Simulatoren für die augenchirurgische Aus- und Weiterbildung aufgestiegen.

Operationen am menschlichen Auge gehören zu den anspruchsvollsten chirurgischen Eingriffen und werden, wie in Abb. 2.1 zu sehen, unter einem Stereomikroskop durchgeführt. Anatomisch lässt sich das Auge in einen vorderen und einen hinteren Bereich unterteilen. Der „vordere Augenabschnitt“ setzt sich aus Bindehaut, Hornhaut (Cornea), Linse und Iris zusammen. Der „hintere Augenabschnitt“ liegt jenseits der Linse und besteht aus dem Glaskörper (der klaren, gelartigen Füllung des Auges), der lichtempfindlichen Netzhaut (Retina) und der darunter liegenden, ernährenden Aderhaut. Behandlungen im hinteren Bereich gehören zur sogenannten *vitreoretinalen* Chirurgie. Schematisch zeigt Abb. 2.2 einen solchen Eingriff. Hier ist sowohl eine Lichtquelle (links) als auch ein Instrument für die Gewebe-Interaktion (rechts) in das Auge eingeführt. Letzteres kann beispielsweise ein mit einem oszillierenden Messer ausgestattetes Saug-Schneide-Instrument (*Vitrektor*) sein, mit dem sich Gewebe aus dem Auge entfernen lässt. Instrument-Parameter wie Saugstärke oder Schneiderate sowie Einstellungen des Stereomikroskops kann der Arzt über Fußpedale steuern. Für weitere Konfigurationen steht meist ein Touchscreen zur Verfügung.

Abbildung 2.1: Chirurgie am Stereomikroskop

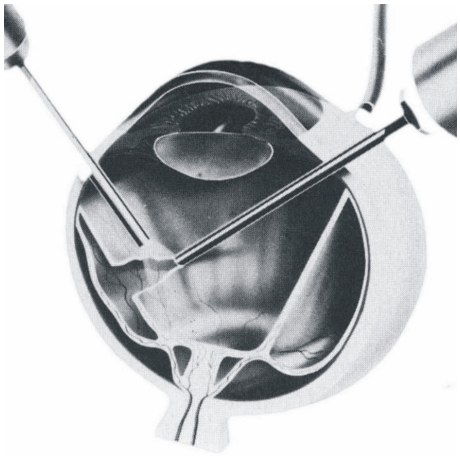


Abbildung 2.2: Vitreoretinale Behandlung (Quelle: Freyler [43])

Die folgenden beiden Abschnitte umreißen, wie die EYESi-Hardware im Zusammenspiel mit der EYESi-Software eine VR für Eingriffe am menschlichen Auge erzeugt. Der abschließende Abschnitt beschreibt, wie EYESi weltweit für die augenchirurgische Ausbildung eingesetzt wird.

2.1 Die EYESi-Hardware

Abb. 2.3 zeigt den Simulator EYESi mit all seinen Benutzerschnittstellen. Der Tisch ist auf einer höhenverstellbaren Hubsäule montiert und hält die „Blue Box“, in der sich die Hardware eines Standard-PCs verbirgt. In der Nachbildung des Mikroskops befinden sich zwei OLEDs (*Organic Light Emitting Diode*) mit einer Auflösung von je 800×600 Farbpixeln. Durch Okulare blickt der Arzt auf die OLEDs und sieht das computergenerierte, stereomikroskopische Bild der Operationsumgebung. Auf dem schwarzen, rechts am Boden liegenden Fuß-Panel befinden sich Schalter und Joysticks, mit denen sich Einstellungen des Mikroskops wie Fokus und Zoom steuern lassen. Die grauen Fußpedale des linken Panels können beispielsweise die Saugstärke eines Instruments kontrollieren. Auf dem Touchscreen ist je nach Modus entweder dieselbe virtuelle Szene wie auf den OLEDs zu sehen oder aber eine GUI (*Graphical User Interface*), über die der Arzt z. B. den Schwierigkeitsgrad einer Trainingsaufgabe wählen kann. Auf dem EYESi-Tisch liegt das Kunststoff-Modell eines Kopfes. (Aus haptischen Gründen ist es wichtig, dass dieses Modell die tatsächliche Form eines Kopfes nachbildet, denn in der Realität kommt es vor, dass sich der Arzt auf der Stirn des Patienten abstützt. Ins-

Abbildung 2.3: EYESi

Abbildung 2.4: EYESi-Kopf

besondere schränkt die Geometrie des Gesichts die Instrument-Bewegungen ein.) In der Augenhöhle des EYESi-Kopfes ist eine Halbkugel aufgehängt, die als mechanisches Augenmodell dient. Seine Rotationsfreiheitsgrade entsprechen denen eines echten Auges, Federn simulieren die rückstellenden Kräfte des Augenmuskels. Wie in Abb. 2.4 zu sehen, verfügt das Augenmodell über eine feste Anzahl an Einstichlöchern. Durch diese kann der Arzt die Nachbildungen echter Operationsinstrumente einführen. Die virtuelle Szene auf den OLEDs muss dann mit den Bewegungen der Instrumente und des mechanischen Auges in Übereinstimmung gebracht werden. Für diese Aufgabe wird ein *Optisches Tracking* eingesetzt, dessen Aufbau Abb. 2.5 zeigt. Die Halbkugel des Augenmodells hängt über der Konstruktion, die vollständig in einem EYESi-Kopf untergebracht ist. Um die Lichtquelle am Boden sind drei FPGA (*Field Programmable Gate Array*) Kameras angeordnet, welche die eingeführten Instrumente und das mechanische Auge beobachten. Um deren Bewegungen *tracken* zu können, sind an der Halbkugel und an den Instrument-Spitzen Farbmarkierungen angebracht. Eigentlich sind zwei Kameras ausreichend, um aus 2D-Bildern die 3D-Koordinate einer Farbmarkierung zu rekonstruieren. Mit Hilfe einer dritten Kamera jedoch werden Verdeckungsprobleme eingeschränkt. Die Position eines Instruments ist durch die Position seiner Farbmarkierung an der

Abbildung 2.5: Kamerablock

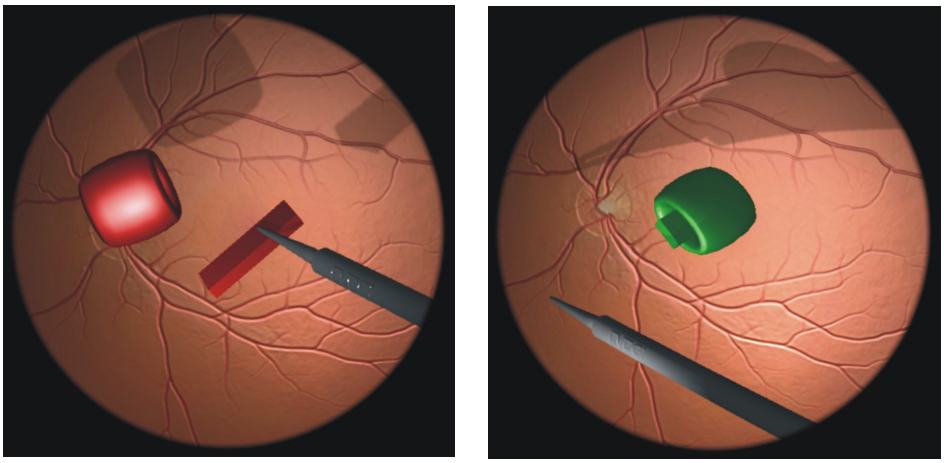
Spitze und durch ein zugehöriges Einstichloch eindeutig bestimmt. Um die Rotation des Instruments um seine Längsachse zu ermitteln, wird zusätzlich ein magnetischer Sensor verwendet, der sich im Handteil des Instruments befindet.

2.2 Die EYESi-Software

Für den Benutzer unterteilt sich die EYESi-Software in eine GUI-Komponente und eine VR-Komponente:
Die Übungsaufgaben, die EYESi anbietet, werden als *Trainingsmodule* bezeichnet und lassen sich über die GUI starten. Sie stellt ein umfangreiches Menü zur Verfügung, durch das der Arzt mit Hilfe des Touchscreens navigiert. In der GUI muss der Arzt z. B. auch festlegen, welches Instrument er für die Gewebe-Interaktion benutzen möchte. Der Teil des Instruments, den der Arzt in das mechanische Auge einführt, besteht schließlich nur aus einem dünnen, geraden Metallstück, dessen Ende mit einem Farbpunkt markiert ist. Erst nach der GUI-Auswahl zwischen Pinzette, Vitrektor, Nadel, Kanüle usw. kann dem Instrument in der virtuellen Szene eine computergrafische Darstellung und eine Funktion zugewiesen werden.
Während ein Modul läuft, berechnet die VR-Komponente all diejenigen Bildinhalte, welche den Eingriff am Auge darstellen.

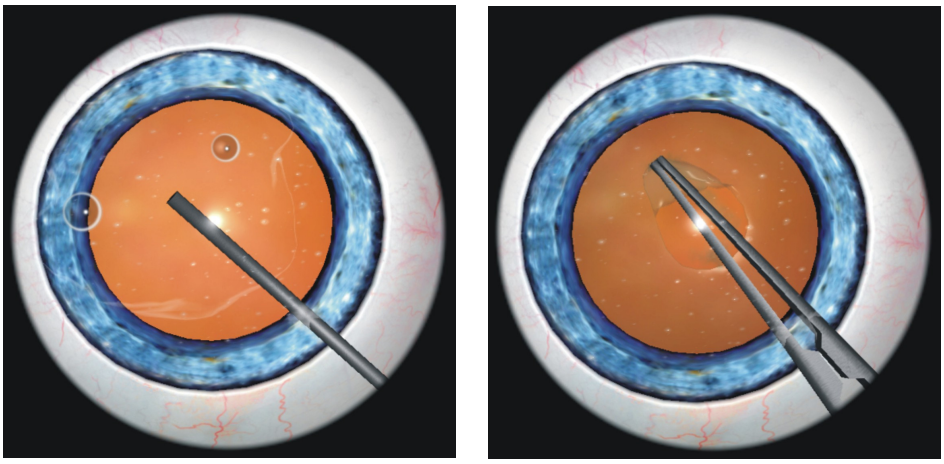
EYESi unterscheidet zwischen „abstrakten“ Trainingsmodulen und solchen, die einen chirurgischen Eingriff simulieren:
In einem abstrakten Trainingsmodul muss der Arzt Objekte manipulieren, die keine Entsprechung in einem realen Augen haben. Ein Beispiel ist das Szenario in Abb. 2.6, in dem ein Quader und ein Torus mit elliptischer Öffnung ineinander geschoben werden müssen. Die Aufgabe ist erst erfüllt, wenn die Objekt-Einfärbungen von rot zu grün wechseln. Mit Hilfe abstrakter Trainingsmodule kann der Arzt gezielt bestimmte Fähigkeiten verbessern, die eine Grundvoraussetzung für echte operative Eingriffe sind. So muss der Arzt z. B. lernen, mit der eingeschränkten Bewegungsfreiheit der Instrumente zurecht zu kommen. Schließlich kann ein eingeführtes Instrument nur um seine eigene Achse und um das entsprechende Einstichloch rotieren.
Nicht-abstrakte Trainingsmodule dagegen versuchen, eine Operation oder Teilschritte einer Operation so realistisch wie möglich nachzubilden. Abb. 2.7(a) zeigt, wie mit einer Kanüle eine klare Flüssigkeit in den vorderen Augenabschnitt eingespritzt wird, um die Druckverhältnisse im Inneren des Auges zu stabilisieren. In Abb. 2.7(b) ist zu sehen, wie anschließend im Rahmen einer *Kapsulorhexis* (siehe Abschnitt 7.1) ein Stück Membran von der Oberfläche der Linse entfernt wird.

In der Realität betreffen chirurgische Eingriffe entweder den vorderen oder den hinteren Augenabschnitt. Dementsprechend gliedern sich auch die Trainingsmodule von EYESi. Das Modul in Abb. 2.6 z. B. stellt den hinteren Augenabschnitt mit Sicht auf die Netzhaut dar. Abb. 2.7 zeigt Screenshots eines Moduls für den vorderen Augenabschnitt. Hier erscheint der Bereich innerhalb der Iris rötlich-orange, da das Auge von einer Lichtquelle ausgeleuchtet wird, deren Strahlen von der Netzhaut diffus reflektiert werden.



(a) (b)

Abbildung 2.6: Abstraktes EYESi-Trainingsmodul



(a) (b)

Abbildung 2.7: Simulation chirurgischer Eingriffe

2.3 EYESi in der chirurgischen Ausbildung

Die vorherigen Abschnitte haben sich darauf beschränkt, die Funktionsweise von EYESi zu umreißen. Von besonderer Bedeutung für die Motivation und den Wert vorliegender Arbeit ist natürlich der Nutzen von EYESi für die chirurgische Ausbildung und dessen Akzeptanz bei erfahrenen Augenchirurgen. Schließlich sind die in den nachfolgenden Kapiteln beschriebenen Algorithmen für EYESi entwickelt worden.

Der erste Prototyp von EYESi wurde 2001 auf dem Kongress der Deutschen Ophthalmologischen Gesellschaft (DOG) vorgestellt. Im selben Jahr erfolgte die Gründung der VRmagic GmbH, die bis heute weltweit an die einhundert Geräte verkauft hat. (Seit 2006 gibt es EYESi auf allen Kontinenten.) Abnehmer sind Universitäten, Universitätskliniken oder allgemein Ausbildungseinrichtungen für angehende Augenchirurgen. Ein Kunde von VRmagic ist beispielsweise die internationale Non-Profit-Organisation ORBIS, die sich das Ziel gesetzt hat, Operationen gegen Erblindung weltweit möglich zu machen. Im Jahr 2005 wurden sowohl das ORBIS-Trainingsflugzeug als auch verschiedene Länderprogramme mit EYESi-Simulatoren ausgestattet, um Augenärzte in Entwicklungsländern in der operativen Entfernung des Grauen Star zu schulen.
Mit EYESi hat VRmagic sogenannte *Drylabs* als neues Format in der ophthalmochirurgischen Ausbildung etabliert. Drylabs, wie in Abb. 2.8 zu sehen, sind EYESi-Trainingskurse, in denen die Teilnehmer sowohl von erfahrenen Augenchirurgen als auch von EYESi-Instruktoren (meist VRmagic-Mitarbeitern) betreut werden. Das erste Drylab fand 2003 auf dem 6. Internationalen Vitreoretinalen Symposium in Frankfurt-Marburg statt. Im selben Jahr entschied die DOG, simulatorgestützte Ausbildung zum Standard zu machen. (In den USA wurde

Abbildung 2.8: Der Einsatz von EYESi im Rahmen zweier Drylabs

der Einsatz von Simulatoren 2007 in die Richtlinien für die augenkundliche Ausbildung aufgenommen.) Um Drylabs auf ihrem jährlichen Kongress anzubieten, mietet die DOG EYESi-Geräte von VRmagic. Seit 2006 veranstaltet VRmagic das Europäische Vitreoretinale Drylab in Mannheim, in dem Augenärzte aus ganz Europa an EYESi-Simulatoren geschult werden.

Diese Entwicklungsgeschichte zeigt, wie EYESi weltweit für Schulungen eingesetzt wird und ausbildende Ärzte diesen Einsatz befürworten und vorantreiben. Wesentlichen Anteil am Erfolg von EYESi haben natürlich dessen realitätsnahe Simulationen, die kontinuierlich weiterentwickelt werden. Dass Chirurgen EYESi als Schulungsgerät akzeptieren, liegt jedoch auch daran, dass EYESi Aufgaben eines Ausbilders übernimmt: Zu jeder Trainingsaufgabe bietet die EYESi-GUI Informationsmaterial über deren medizinischen Hintergrund. Sowohl zu Beginn als auch während des virtuellen Eingriffs leitet EYESi den Benutzer an, welche Schritte wann und auf welche Art durchzuführen sind. Damit der Benutzer beurteilen kann, wie gut er eine Aufgabe bewältigt hat, verfügt jedes Trainingsmodul über ein Bewertungssystem, das für erfolgreich abgeschlossene Teilaufgaben Punkte vergibt und für Fehler, wie z. B. Verletzungen der Netzhaut, Punkte abzieht. Da die Schwierigkeit eines realen Eingriffs von Patient zu Patient schwankt, bietet EYESi jede Trainingsaufgabe in mehreren Schwierigkeitsstufen an. (Wegen dieser Rahmen-Anforderungen an ein EYESi-Trainingsmodul wurden auch für die Module, die im Verlauf vorliegender Arbeit entstanden und in Kapitel 7 beschrieben sind, verschiedene Schwierigkeitsgrade und ein Bewertungssystem implementiert.)

Kapitel 3

Simulation deformierbarer Objekte

Die Publikationen zum Thema „Simulation deformierbarer Objekte“ beschreiben entweder geometrische oder physikalische *Deformationsmodelle*. Erstere basieren z. B. auf parametrischen Kurven und Oberflächen oder *Free-Form Deformations* (FFD)[1]. Hier jedoch wollen wir uns auf physikalische Ansätze beschränken. Die Forschung in diesem Bereich reicht bis in das Jahr 1987 zurück, in dem Lasseter [69] und Terzopoulos et al. [118] ihre Pionierarbeiten veröffentlichten. Die seitdem vorgeschlagenen physikalischen Modelle lassen sich in *Lagrange-Methoden* und *Euler-Methoden* unterteilen. Eine Euler-Methode definiert eine Menge stationärer Punkte, in denen Daten gehalten werden, die sich über die Zeit verändern. Soll beispielsweise simuliert werden, wie sich Tinte in einem Wasserglas verteilt, wird das Glas von einem starren, dreidimensionalen Gitter überdeckt. Eigenschaften wie Geschwindigkeit, Druck und Dichte werden nur an den Gitter-Punkten ausgewertet und die Dichte-Verteilung dient als Grundlage für die computergrafische Darstellung der Szene. Da Euler-Methoden vorzugsweise in der Simulation von Flüssigkeiten und Gasen zum Einsatz kommen, werden sie hier nicht näher beschrieben[2]. Ein Lagrange-Modell besteht ebenfalls aus einer Menge Information tragender Punkte, auch *Partikel* genannt, diese allerdings verändern ihren Aufenthaltsort über die Zeit. Sie stellen die räumliche Diskretisierung des zu simulierenden, kontinuierlichen Objekts dar und ihre momentane Verteilung beschreibt die momentane Transformation und Deformation des Körpers.
Es existieren sehr verschiedene Ansätze, doch alle erfüllen denselben Zweck: Als Eingabe für das Modell dienen Partikel-Informationen wie Ruheposition und aktuelle Position. Die Ausgabe liefert interne Deformationskräfte, welche im Au-

[1] Zum Thema FFD sei der Leser auf Barr [7] oder Sederberg und Parry [106] verwiesen.

[2] Die Publikationen von Foster und Metaxas [40, 41, 42] haben die Euler-Methode in der Fluid-Simulation populär gemacht.

genblick auf den Körper wirken.
Die Bewegung des Körpers wird erst durch die *Numerische Integration* erzeugt. Sie diskretisiert die Simulation zeitlich und ist (im Allgemeinen) vollkommen unabhängig vom gewählten Deformationsmodell. Basierend auf den Kräften, die zu einem diskreten Zeitpunkt t wirken, berechnet der Integrationsschritt neue Partikel-Positionen zum Zeitpunkt $t + h$. Die *Schrittweite* h ist ein Parameter des Integrationsverfahrens. Durch die Abfolge sich zeitlich verändernder Partikel-Verteilungen entsteht die Animation des deformierbaren Körpers.
Zu beachten ist, dass das Deformationsmodell ausschließlich interne Kräfte berechnet, die durch die momentane Verzerrung und Verspannung des Körpers hervorgerufen werden. Im Allgemeinen interagiert der Körper jedoch mit anderen Objekten der Simulationsumgebung. Sobald er mit einem dieser Objekte in Kontakt kommt, muss die *Kollisionserkennung* dieses Ereignis detektieren. Die Kollisionserkennung ist eine Komponente des Simulations-Frameworks, die üblicherweise zwischen zwei aufeinander folgenden Integrationsschritten zum Einsatz kommt. Im direkten Anschluss an die Kollisionserkennung muss die *Kollisionsantwort* externe Kräfte für den Körper berechnen, welche einer Kollision im Rahmen des nächsten Integrationsschritts entgegenwirken.
Das gesamte Simulations-Framework besteht demnach im Wesentlichen aus einer Schleife, in der folgende Komponenten iterativ aufgerufen werden:

- *Kollisionserkennung:* Alle Kollisionen des Körpers mit anderen Objekten der Simulationsumgebung werden detektiert.
- *Kollisionsantwort:* Auf Basis der Kollisionen werden externe Kräfte berechnet, die auf den Körper wirken.
- *Deformationsmodell:* Ausgehend von der Deformation des Körpers werden interne Kräfte berechnet.
- *Numerische Integration:* Externe und interne Kräfte führen im Rahmen des Integrationsschritts zu einer neuen Transformation und Deformation des Körpers.

In den folgenden Abschnitten werden diese Komponenten ausführlicher beschrieben.

3.1 Physikalisch basierte Deformationsmodelle

Das Thema „Physikalisch basierte Deformationsmodelle“ ist sehr komplex. Der folgende Abschnitt soll daher nur einen groben Überblick liefern. Einen sehr guten Einstieg in die Materie bieten die „State of the Art“-Berichte von Gibson

und Mirtich [51] und Nealen et al. [86]. Die weitaus aktuellere Publikation von Nealen et al. aus dem Jahr 2006 unterteilt die Lagrange-Modelle in *Mesh Based Methods* und *Mesh Free Methods*. Die räumliche Diskretisierung eines kontinuierlichen Objekts wurde bisher als Ansammlung von Partikeln beschrieben. Im Rahmen der Simulation deformierbarer Festkörper basiert diese Diskretisierung jedoch üblicherweise auf einer Gitter-Struktur. Das als *Mesh* bezeichnete Gitter definiert die Nachbarschaftsbeziehungen zwischen den einzelnen Diskretisierungspunkten, auch *Mesh-Knoten* genannt. Um eine Tischdecke zu simulieren, bietet sich z. B. ein Dreiecks-Mesh an, in dem sich benachbarte Dreiecke entweder nur einen Knoten oder zwei Knoten und eine Kante teilen. Manche Simulationen laufen auf *regelmäßigen* Gittern, welche die Berechnung der Deformation beschleunigen können. Der Begriff „regelmäßig“ meint hier, dass Abstände benachbarter Knoten im undeformierten Mesh entlang der Koordinatenachsen konstant sind. Mesh-freie Methoden sind in der Computergrafik erst später populär geworden, insbesondere im Zusammenhang mit der Simulation von Flüssigkeiten und schmelzenden Objekten.

Kontinuumsmechanik Ausgangspunkt vieler Deformationsmodelle ist eine *Partielle Differentialgleichung* (PDGL). Die Kontinuumsmechanik liefert diese PDGL, in der ein deformierbarer Körper als Kontinuum beschrieben wird und deren Lösungsfunktion sowohl räumlich als auch zeitlich kontinuierlich ist. Das Deformationsmodell besteht dann im Wesentlichen aus der PDGL und einem numerischen Lösungsverfahren, das diese räumlich diskretisiert. (Nicht zu verwechseln mit der numerischen Integration, welche das Problem zeitlich diskretisiert.)
„Werkzeuge“ der Kontinuumsmechanik sind Tensoren, welche die Verzerrung und die Verspannung eines Körpers an einem beliebigen Materialpunkt berechnen. Angenommen, eine Funktion ist gegeben, die jedem Punkt des undeformierten Körpers einen Verschiebungsvektor zuordnet und dadurch eine Deformation definiert. Mit Hilfe partieller Raum-Ableitungen dieser Verschiebungsfunktion misst der *Verzerrungstensor* V Längen- und Winkelverzerrungen eines infinitesimalen Quaders. Die Komponenten des *Spannungstensors* S erfassen die *Normal-* und *Schubspannungen*, die an einem Materialpunkt wirken. Beide Tensoren sind symmetrisch und von zweiter Stufe, entsprechen also einer 3×3 Matrix. Das *Elastizitätsgesetz* stellt vereinfachend einen linearen Zusammenhang zwischen den Komponenten von V und S her.
Abschnitt 5.4.1 kommt noch einmal auf die Kontinuumsmechanik zurück und gewährt einen etwas ausführlicheren Einblick.

Finite Elemente Methode Die Methode der Finiten Elemente[3] (FEM) hat ihren Ursprung in den Ingenieurwissenschaften und ist wohl der populärste Ansatz für die numerische Lösung von PDGLs auf unregelmäßigen Gittern.
Sei $\vec{m}$ ein Materialpunkt im undeformierten Körper und $\vec{x}(\vec{m}, t)$ dessen Koordinate zum Zeitpunkt t. Die Kontinuumsmechanik liefert die zu lösende Bewegungsgleichung als PDGL:

$$\rho \ddot{\vec{x}} = \nabla \cdot S + \vec{f}$$

ρ entspricht der Dichte des Materials und $\vec{f}$ einer äußeren Kraft. Der Divergenzoperator transformiert den 3×3 Spannungstensor S in einen 3D Vektor, der die innere Kraft an einem Materialpunkt beschreibt.
Der deformierbare Körper sei nun durch Finite Elemente diskretisiert, beispielsweise durch ein Tetraeder-Mesh mit Knoten-Positionen $\vec{x}_i(t)$. Die Suche nach der räumlich kontinuierlichen Funktion $\vec{x}(\vec{m}, t)$ wird somit auf die Suche nach diskreten Positionen $\vec{x}_i(t)$ beschränkt. Pro Element wird nun eine Approximation von $\vec{x}(\vec{m}, t)$ gewählt. Eine solche *Ansatzfunktion* entspricht einer Linearkombination von Interpolationsfunktionen $\vec{b}_i$. Die Koeffizienten der Linearkombination stimmen im einfachsten Fall mit Knoten-Positionen $\vec{x}_i(t)$ überein:

$$\vec{x}(\vec{m}, t) \approx \sum_i \vec{x}_i(t)\vec{b}_i(\vec{m})$$

Der Index i bezeichnet hier die Mesh-Knoten desjenigen Elements, in dem $\vec{m}$ enthalten ist. Die Wahl der Interpolationsfunktionen $\vec{b}_i$ ist Teil der Modell-Definition. Ersetzt man $\vec{x}(\vec{m}, t)$ in der PDGL durch diese Elementansätze, entsteht ein System algebraischer Gleichungen für die $\vec{x}_i(t)$, das numerisch gelöst werden kann.
Die FEM ist sehr rechenintensiv, weshalb viele Publikationen ([95], [31], [75]) auf eine einfache Form dieser Methode zurückgreifen. Hier werden Mesh-Knoten als Punktmassen und Finite Elemente als generalisierte Federn aufgefasst.
Kontinuumsmechanische Ansätze sind anspruchsvoll, da im Allgemeinen Verzerrungen nicht-linear von den Materialpunkt-Verschiebungen und Verspannungen nicht-linear von den Verzerrungen abhängen. Diese Zusammenhänge können zwar linearisiert werden, so wie in Debunne et al. [30, 29], besonders die Linearisierung des Verzerrungstensors jedoch ist kritisch. Denn dadurch verliert der Tensor seine Rotations-Invarianz, was im Falle großer Deformationen zur Berechnung von Phantom-Kräften führt. Für dieses Problem schlagen Müller et al. [75, 76] und Capell et al. [21] Lösungen vor, die es erlauben, auch unter dem Einfluss starker Verformungen auf linearen Gleichungen zu rechnen.

FEM-verwandte Methoden Es existieren einige Methoden, die mit der FEM verwandt sind, in der Animation deformierbarer Objekte jedoch eine eher unter-

[3] Ein Standardwerk zu diesem Thema ist das Buch von Bathe [8].

geordnete Rolle spielen:
Die *Finite Differenzen Methode* (FDM) legt der räumlichen Diskretisierung eines Körpers ein regelmäßiges Gitter zugrunde und als Näherung für partielle Raum-Ableitungen dienen Differenzenquotienten. Terzopoulos et al. [118] benutzen diesen Ansatz für die Simulation geometrisch einfacher Körper. Verglichen mit der FEM ist eine FDM leicht zu implementieren, es ist jedoch schwierig, den Rand eines beliebigen Körpers durch ein regelmäßiges Gitter zu approximieren.
Ausgangspunkt für die *Finite Volumen Methode* (FVM) ist eine zu lösende Erhaltungsgleichung in integraler Form. Das Verfahren wandelt Volumen-Integrale einer PDGL, die einen Divergenz-Term enthalten, in Oberflächen-Integrale um. Wie die FEM ist die FVM für unregelmäßige Gitter geeignet. Teran et al. [115] setzen sie ein, um Muskelkontraktionen zu modellieren.
Die *Randelementmethode* (REM) diskretisiert nur die Oberfläche eines volumetrischen Objekts, so dass sämtliche unbekannte Zustandsgrößen auf dessen Rand liegen. Dadurch reduziert die Methode die Dimension des gestellten Problems, sie kann jedoch keine Inhomogenitäten im Inneren des Körpers modellieren. Eingesetzt wird die Methode z. B. in den Arbeiten von James und Pai [63, 64, 65].

Mesh-freie Methoden Mesh-freie Methoden sind geeignet für die Diskretisierung volumetrischer Objekte, deren Deformation sehr stark von ihrer Ausgangsform abweichen kann. Mit einer Knetmasse beispielsweise sind beliebige plastische Verformungen möglich. Besonders in Bezug auf die Simulation von Fluiden ist die Mesh-freie Diskretisierung durch Partikel intuitiv. Für deformierbare Oberflächen bieten sich eher Mesh-basierte Verfahren an, trotzdem sollen die Mesh-freien hier der Vollständigkeit halber erwähnt werden:
Eine einfache Möglichkeit, Flüssigkeiten zu simulieren, besteht in der Anwendung des *Lennard-Jones-Potentials*, das aus der Atomphysik stammt und die Wechselwirkung zwischen ungeladenen, nicht chemisch aneinander gebundenen Atomen approximiert. Partikel mit sehr kleinem Abstand üben abstoßende Kräfte aufeinander aus, die mit zunehmendem Abstand in Anziehungskräfte übergehen und über sehr große Entfernungen verschwindend gering werden. Dieser Verlauf für die Partikel-Kräfte wird z. B. in den Arbeiten von Tonnesen [122] und Steele et al. [112] benutzt.
Eine effiziente, robuste und leicht zu implementierende Methode für die Animation stark deformierbarer Festkörper schlagen Müller et al. [77] vor: *Meshless Deformations Based on Shape Matching*. In jedem Zeitschritt wird die initiale, unverzerrte Partikel-Verteilung über die Methode der kleinsten Quadrate in die aktuelle Verteilung eingepasst, um für jedes Partikel eine Ziel-Position zu definieren. Anschließend drückt ein geometrisches, unbedingt stabiles Integrationsverfahren die Partikel in Richtung ihrer Ziel-Positionen.

Mesh-freie Methoden zur Lösung Partieller Differentialgleichungen[4] sind das Pendant zu FEM, FDM, FVM und REM. Hier werden Ansatzfunktionen für die Umgebung eines Partikels definiert. Nachbarschaftsbeziehungen zwischen Partikeln ändern sich und Umgebungen nahe beieinander liegender Partikel überlappen. (Im Gegensatz dazu definiert eine FEM Ansatzfunktionen für topologisch fixierte, sich nicht überschneidende Elemente.) Im Bereich Computergrafik waren Müller et al. [78] die ersten, die einen Mesh-freien, kontinuumsmechanischen Ansatz für die Simulation deformierbarer Festkörper vorgeschlagen haben.

Feder-Masse-Systeme Neben der FEM sind Feder-Masse-Systeme die populärsten Modelle zur Simulation deformierbarer Festkörper. Einem Feder-Masse-System liegt keine PDGL zugrunde, vielmehr beginnt die Modell-Definition mit der räumlichen Diskretisierung des Körpers durch Punktmassen, die durch masselose Federn miteinander verknüpft sind. Jeder Mesh-Knoten $\vec{p}_i$ verfügt über eine konstante Masse m_i, einen Ortsvektor $\vec{x}_i(t)$ und einen Geschwindigkeitsvektor $\vec{v}_i(t)$. Für eine Feder des Gitters, die $\vec{p}_i$ mit $\vec{p}_j$ verbindet, ist eine Ruhelänge l_{ij} und eine Federkonstante c_{ij} definiert. Durch die $\vec{x}_i(t)$ und $\vec{v}_i(t)$ ist der System-Zustand zum Zeitpunkt t vollständig beschrieben.
Seien $\vec{p}_i$ und $\vec{p}_j$ durch eine Feder verbunden. Mit $\vec{x}_{ij} = \vec{x}_j - \vec{x}_i$ ergibt sich für die auf den Knoten $\vec{p}_i$ wirkende (linear elastische) Federkraft $\vec{f}_i$:

$$\vec{f}_i = c_{ij}(|\vec{x}_{ij}| - l_{ij})\frac{\vec{x}_{ij}}{|\vec{x}_{ij}|}$$

Um die interne Gesamtkraft $\vec{F}_i$ auf $\vec{p}_i$ zu bestimmen, werden die Federkräfte aller an $\vec{p}_i$ hängenden Federn aufsummiert. Die Bewegungsgleichung $\vec{F}_i = m_i \cdot \ddot{\vec{x}}_i$ ist die einzige DGL, die im Rahmen einer Feder-Masse-Simulation auftritt und wird durch das Integrationsverfahren gelöst.
In einigen Arbeiten ([14], [6], [119]) wird das bisher beschriebene kanonische Modell auf eine allgemeinere Basis gestellt. Hier werden üblicherweise zunächst *weiche* Randbedingungen formuliert, welche die Mesh-Knoten im Verlauf der Simulation *tendenziell* einhalten sollen. Auf Basis einer Randbedingung wird ein zu minimierender Energie-Term E definiert. Knoten-Kräfte $\vec{f}_i$ entstehen durch Ableitung dieses Terms nach den Knoten-Positionen:

$$\vec{f}_i = -\frac{\partial E}{\partial \vec{x}_i}$$

Ist die Einhaltung von Federlängen l_{ij} die einzige Randbedingung, stimmt der allgemeine Ansatz mit dem kanonischen Modell überein. Teschner et al. [119] z. B.

[4] Einen Überblick bieten Fries und Matthies [44].

stellen zusätzlich Energie-Terme für die Flächenerhaltung von Dreiecken und für die Volumenerhaltung von Tetraedern auf.
Während man eine FEM als „physikalisch korrekt“ bezeichnen kann, trifft auf ein Feder-Masse-System eher die Beschreibung „physikalisch plausibel“ zu. Denn die Gleichungen der Kontinuumsmechanik enthalten Materialkonstanten, deren Werte für eine Vielzahl von Werkstoffen experimentell ermittelt wurden und in der entsprechenden Literatur nachgeschlagen werden können. Ein Feder-Masse-System dagegen erfordert ein Parameter-Tuning, da Federkonstanten keine Entsprechung in echten Materialien haben. Breen et al. [14] jedoch modellieren in ihrer Arbeit Kleidungsstücke und vertreten die Meinung, dass ein Feder-Masse-System hierfür eher geeignet ist als eine FEM[5]. Schließlich stellen Textilien kein Kontinuum dar, sondern ein Geflecht aus verwebten Fasern. Tatsächlich haben Feder-Masse-Systeme nach dieser Publikation die Literatur über Kleidungssimulation lange dominiert.

3.2 Numerische Integration

Es existieren *explizite* und *implizite* Methoden für die numerische Integration. Anhand des expliziten und des impliziten *Euler-Verfahrens* werden diese beiden Klassen hier umrissen[6].

Problemstellung Ausgangspunkt ist ein *Anfangswertproblem* (AWP), gegeben durch folgende *Gewöhnliche Differentialgleichung erster Ordnung*:

$$\dot{y}(t) = g\left(y(t), t\right) \qquad y(t_0) = y_0$$

Die Funktion g sei bekannt. Da die gesuchte Funktion y nicht nur die DGL erfüllen, sondern auch den Anfangswert y_0 zum Zeitpunkt t_0 annehmen soll, wird diese Aufgabenstellung als „Anfangswertproblem“ bezeichnet. Die DGL heißt „gewöhnlich“, da in ihr keine partiellen Ableitungen auftauchen und sie ist von „erster Ordnung“, da sie nur die erste und keine höheren Ableitungen von y enthält.
Nur in einfachen Fällen ist es möglich, die Funktion y analytisch zu bestimmen, daher ist im Allgemeinen ein numerisches Verfahren notwendig, mit dem sich DGLs näherungsweise lösen lassen. Ein Verfahren für ein AWP definiert eine *Schrittweite* h und ermittelt iterativ Funktionswerte $y(t + h)$ ausgehend vom bekannten Startwert $y(t_0)$.

[5] Etzmuß et al. [37] z. B. modellieren Kleidung durch eine FEM.

[6] Die Ausführungen orientieren sich an den SIGGRAPH 2001 Kursunterlagen von Witkin und Baraff [133].

Explizites Euler-Verfahren Wir nehmen an, dass die Funktion y beliebig oft differenzierbar ist und betrachten deren Taylor-Entwicklung um einen Punkt t (beliebig aber fest)[7]:

$$y(t+h) = y(t) + h\,\dot{y}(t) + \frac{h^2}{2!}\,\ddot{y}(t) + \cdots + \frac{h^n}{n!}\,y^{(n)}(t) + \cdots$$

Das einfachste numerische Verfahren, das explizite Euler-Verfahren, geht aus dieser Taylor-Reihe hervor, indem man sie nach der ersten Ableitung von y abbricht:

$$y(t+h) = y(t) + h\,\dot{y}(t)$$

Da $\dot{y}(t)$ über die DGL definiert ist, können die Werte $y(t+h)$ Schritt für Schritt ermittelt werden. Die explizite Berechenbarkeit der gesamten rechten Seite gibt dem Verfahren seinen Namenszusatz. Explizite Methoden haben gemeinsam, $y(t+h)$ allein auf Basis bereits bekannter Werte zu bestimmen und lassen sich daher ohne großen Aufwand implementieren.

Anwendung des expliziten Euler-Verfahrens Gegeben sei ein Deformationsmodell, welches das zu simulierende Objekt durch eine Menge von Partikeln räumlich diskretisiert. Jedes Partikel verfüge über eine konstante Masse m, einen zeitabhängigen Ortsvektor $\vec{x}(t) \in \mathbb{R}^3$ und einen zeitabhängigen Geschwindigkeitsvektor $\vec{v}(t) \in \mathbb{R}^3$. Orts- und Geschwindigkeitsvektoren aller Partikel seien zum aktuellen Zeitpunkt t gegeben. Das Deformationsmodell und die Methoden für die Kollisionsbehandlung liefern für jedes Partikel eine Kraft $\vec{f}(t)$, die momentan auf das Partikel wirkt. Die Bewegung einer Punktmasse wird durch die Gleichung $\vec{f} = m \cdot \vec{a}$ bzw. $\ddot{\vec{x}} = \vec{f}/m$ bestimmt. Diese entspricht einer Gewöhnlichen DGL zweiter Ordnung, die wir in eine Gewöhnliche DGL erster Ordnung transformieren müssen, um das explizite Euler-Verfahren anwenden zu können:

$$\begin{pmatrix} \dot{\vec{x}}(t) \\ \dot{\vec{v}}(t) \end{pmatrix} = \begin{pmatrix} \vec{v}(t) \\ \vec{f}(t)/m \end{pmatrix}$$

Ein expliziter Euler-Schritt für ein einzelnes Partikel hat somit die Form:

$$\begin{pmatrix} \vec{x}(t+h) \\ \vec{v}(t+h) \end{pmatrix} = \begin{pmatrix} \vec{x}(t) \\ \vec{v}(t) \end{pmatrix} + h \cdot \begin{pmatrix} \vec{v}(t) \\ \vec{f}(t)/m \end{pmatrix}$$

Um den Integrationsschritt für alle n Partikel des Systems zu definieren, werden sämtliche Orts- und Geschwindigkeitsvektoren in einem $6n$-dimensionalen Vektor zusammengefasst.

[7] Ist y beliebig oft differenzierbar, bedeutet dies zunächst nur, dass die Taylor-Reihe definiert ist. Ob diese Potenzreihe jedoch einen Konvergenzradius ungleich Null hat und ob sie auf ihrem Konvergenzbereich tatsächlich gegen y konvergiert, hängt von den Eigenschaften von y ab.

Implizites Euler-Verfahren Gehen wir der Einfachheit halber davon aus, dass g nicht explizit von der Zeit t abhängt:

$$\dot{y}(t) = g\left(y(t)\right) \qquad y(t_0) = y_0$$

Das bisher beschriebene Euler-Verfahren ermittelt den Wert $y(t_0 + h) = y_1$ dann folgendermaßen:

$$y_1 = y_0 + h \cdot g(y_0)$$

Wie bereits erwähnt, liegt hier eine explizite Integrationsmethode vor, da sich die rechte Seite direkt aus dem Zustand zum Zeitpunkt t_0 ergibt. Für explizite Verfahren gilt jedoch im Allgemeinen, dass bereits geringe Verlängerungen der Schrittweite h zur Divergenz der Iteration führen können. Implizite Methoden wirken diesem Problem entgegen, sind jedoch rechenaufwendiger. Das implizite Euler-Verfahren definiert die Suche nach y_1 durch:

$$y_1 = y_0 + h \cdot g(y_1)$$

Hier lässt sich die rechte Seite nicht mehr explizit berechnen. Die Idee ist, einen Zustand y_1 zu finden, der direkt zu y_0 zurückführt, wenn man einen expliziten Euler-Schritt rückwärts vornimmt:

$$y_0 = y_1 - h \cdot g(y_1)$$

Da g im Allgemeinen nicht-linear ist, wird der Ausdruck $g(y_1)$ durch eine lineare Approximation ersetzt. Mit $\Delta y = y_1 - y_0$ liest sich der implizite Euler-Schritt wie folgt:

$$y_0 + \Delta y = y_0 + h \cdot g(y_0 + \Delta y) \quad \text{bzw.} \quad \Delta y = h \cdot g(y_0 + \Delta y)$$

Als Näherung für $g(y_0 + \Delta y)$ dient die abgebrochene Taylor-Entwicklung von g:

$$g(y_0 + \Delta y) \approx g(y_0) + g'(y_0) \cdot \Delta y$$

Durch die Ableitung des Vektors $g(y_0)$ ergibt sich die Matrix $g'(y_0)$. Die Approximation liefert diejenige für Δy:

$$\Delta y = h \cdot (g(y_0) + g'(y_0) \cdot \Delta y) \quad \text{bzw.} \quad \Delta y - h \cdot g'(y_0) \cdot \Delta y = h \cdot g(y_0)$$

Mit der Bezeichnung I für die Einheitsmatrix folgt:

$$\left(\frac{1}{h} I - g'(y_0)\right) \Delta y = g(y_0) \quad \text{bzw.} \quad \Delta y = \left(\frac{1}{h} I - g'(y_0)\right)^{-1} g(y_0)$$

Nun lässt sich $y_1 = y_0 + \Delta y$ berechnen. Der gesuchte Wert y_1 ist demnach implizit durch das hier konstruierte lineare Gleichungssystem gegeben, das in jedem Iterationsschritt gelöst werden muss.

Güte der numerischen Lösung Die Näherungslösung des expliziten Euler-Verfahrens kann nur dann mit der exakten Lösung der DGL übereinstimmen, falls y linear ist. Denn nur dann sind die von der Taylor-Reihe abgeschnittenen, höheren Ableitungen tatsächlich Null. Der erste in der Taylor-Entwicklung vernachlässigte Term dominiert den Fehler, den das Verfahren erzeugt. Da die explizite Euler-Methode nur die erste Ableitung von y berücksichtigt, hat es *Fehlerordnung* bzw. *Konsistenzordnung* 1. Weitere Kriterien für die Güte eines numerischen Verfahrens sind *Konvergenzordnung* und *Stabilität*. Die „Konvergenzordnung“ ist ein Maß für die Geschwindigkeit, mit der sich die Glieder einer konvergenten Folge dem Grenzwert nähern. Aus der „Stabilität“ einer Methode lässt sich ableiten, wie klein die Schrittweite h gewählt werden muss, damit sich Rundungsfehler nicht aufschaukeln und Konvergenz sichergestellt ist.
Das implizite Euler-Verfahren verfügt genau wie das explizite über Konsistenz- und Konvergenzordnung 1, hat jedoch den Vorteil, bei beliebigen Schrittweiten stabil zu bleiben.

3.3 Behandlung von Kollisionen

Die Kollisionsbehandlung zerfällt in die beiden Komponenten Kollisionserkennung und Kollisionsantwort. Im Allgemeinen beschränkt sich die Kollisionserkennung nicht darauf zu ermitteln, ob zwei Körper A und B miteinander kollidieren oder nicht. Sie liefert ebenfalls Daten wie „Kollisionsort“ (Welche Teilgeometrien von A und B kollidieren?), „Kollisionstiefe“ oder „Oberflächennormale am Kollisionspunkt“. Denn erst mit Hilfe solcher Detail-Informationen kann die Kollisionsantwort effizient Objekt-Durchdringungen verhindern bzw. wieder auflösen.

3.3.1 Kollisionserkennung

Angenommen, in einer Simulationsumgebung befinden sich n bewegliche Objekte. Falls kein geeigneter Algorithmus für die Kollisionserkennung implementiert ist, müssen $\binom{n}{2}$ paarweise Überschneidungstests durchgeführt werden. Daher wurden eine Reihe von Verfahren entwickelt, die diesen Rechenaufwand reduzieren. Der folgende, kleine Überblick beschränkt sich auf rein geometrische Algorithmen und lässt bildbasierte und stochastische Ansätze aus. Eine ausführliche Diskussion ist im „State of the Art“-Bericht von Teschner et al. [120] zu finden.

Bounding Volume Hierarchy Zu den effizientesten Datenstrukturen für die Kollisionserkennung gehören die Bounding Volume Hierarchies (BVH)[8]. Eine

[8]Für eine ausführliche Beschreibung von BVHs sei der Leser auf das Buch von Zachmann und Langetepe [136] verwiesen.

BVH besteht aus einer Baumstruktur, deren Wurzel das vollständige, zu simulierende Objekt repräsentiert. Den Knoten des Baums sind Hüllkörper zugeordnet, die jeweils einen Teilbereich des Objekts umschließen. Ausgehend von der Wurzel werden die Hüllkörper verfeinert, die Blätter des Baums entsprechen üblicherweise den Primitiven, aus denen sich das Objekt zusammensetzt. Diese Primitive können beispielsweise Dreiecke oder allgemein Polygone sein. Somit definieren die Ebenen des Baums Hüllkörper-Zerlegungen des Objekts mit unterschiedlichen Auflösungen.
Zur Definition einer BVH gehört die Wahl von Hüllkörper-Typen, die geeignet sind, Teilgeometrien des Objekts zu approximieren. Üblich sind z. B. Kugeln, Zylinder, AABBs ([124]), OBBs ([55]) oder k-DOPs ([67]). AABB steht (selbsterklärend) für *Axis-Aligned Bounding Box* und OBB für *Oriented Bounding Box*. k-DOPs (*Oriented Discrete Polytopes*) sind die Erweiterung von OBBs (diese entsprechen 6-DOPs) und lassen beliebig viele Grenzflächen für den Umriss des Hüllkörpers zu.
Um zwei eventuell miteinander kollidierende Objekte A und B zu überprüfen, muss sich der Algorithmus Schritt für Schritt entlang beider BVHs „herunterarbeiten“ und dabei paarweise Hüllkörper von A und B auf Überschneidung testen. Das Ergebnis der Suche sind Paare kollidierender Primitive.
Werden BVHs für deformierbare Objekte eingesetzt, müssen deren Hüllkörper in jedem Zeitschritt angepasst werden. Hierfür existieren jedoch sehr effiziente Algorithmen.

Broad Phase, Narrow Phase Hubbard [62] und Ganovelli et al. [48] unterscheiden zwischen einer Broad Phase und einer Narrow Phase der Kollisionserkennung. Die Broad Phase entspricht einer groben Vorsortierung, die mit Hilfe nichtrechenintensiver Operationen ermittelt, welche Objekte, die einen Körper A umgeben, mit Sicherheit nicht mit A kollidieren. Nur die nicht aussortierten Objekte gelangen in die Narrow Phase und werden durch exakte Überschneidungstests überprüft.
Eine BVH von A beispielsweise kann als Narrow Phase aufgefasst werden. Die BVH kann um eine Broad Phase erweitert werden, indem zusätzliche, geometrisch einfache Hüllkörper definiert werden, die einen Mindestabstand d zur Oberfläche von A einhalten. (Die Hüllkörper der BVH dagegen umschließen die entsprechenden Teilbereiche von A so eng wie möglich.) Nur diejenigen Objekte, die in die Hüllkörper der Broad Phase eindringen, gelangen in die Narrow Phase. Wird der Parameter d in Abhängigkeit der maximalen Objekt-Geschwindigkeiten gewählt, die im Verlauf der Simulation zu erwarten sind, ist es ausreichend, die Broad Phase nur in größeren Zeitabständen zu berechnen.

Spatial Subdivision Eine Spatial Subdivision zerlegt den Raum, in dem sich die simulierten Körper bewegen können, in Unterräume. Diese Raumaufteilung wird zu Beginn der Simulation festgelegt und kann auf regelmäßigen Gittern ([123], [137]) oder rekursiven, hierarchischen Zerlegungen ([74]) basieren. Zu dem Zeitpunkt, zu dem die Kollisionserkennung durchgeführt werden soll, werden alle Objekte in die Unterräume einsortiert. Exakte Überschneidungstests müssen dann nur noch für diejenigen Objekt-Paare berechnet werden, die sich in denselben oder benachbarten Unterräumen befinden.
Die Ausdehnung der Unterräume muss in Abhängigkeit der Objekt-Abmessungen gewählt werden: Angenommen, in einer Simulationsumgebung bewegen sich n nicht-deformierbare Kugeln mit Radius r. Sinnvollerweise unterteilt eine Spatial Subdivision den Raum in diesem Fall in würfelförmige Zellen der Breite, Tiefe und Höhe $2r$. Denn eine Kugel, deren Mittelpunkt sich in einer bestimmten Zelle befindet, kann dann höchstens mit denjenigen Kugeln kollidieren, deren Mittelpunkte in derselben oder den 26 direkt angrenzenden Zellen liegen. (Natürlich sind die Objekt-Geometrien und damit die Wahl einer geeigneten Raum-Zerlegung im Allgemeinen komplizierter.)

Distance Fields In der Literatur werden Distance Fields manchmal auch als *Distance Volumes* ([15]) oder *Distance Functions* ([18]) bezeichnet. Ein Distance Field ist eine Funktion $d : \mathbb{R}^3 \to \mathbb{R}$ und definiert die Oberfläche eines Körpers A implizit durch die Nullstellenmenge $S = \{\vec{p} \in \mathbb{R}^3 \mid d(\vec{p}) = 0\}$. Gilt $d(\vec{p}) \neq 0$, gibt das Vorzeichen von $d(\vec{p})$ an, ob sich $\vec{p}$ innerhalb oder außerhalb von A befindet. $|d(\vec{p})|$ entspricht dem Abstand von $\vec{p}$ zur Oberfläche von A.
Im Allgemeinen ist die kontinuierliche Funktion d nicht bekannt und muss in diskreter Form konstruiert werden. Legt man der virtuellen Welt ein regelmäßiges Gitter zugrunde, wird d für alle Gitterpunkte explizit berechnet, an beliebigen Stellen $\vec{p}$ zwischen den Gitterpunkten erhält man $d(\vec{p})$ durch Interpolation. Im Verlauf der Simulation liefert das Distance Field Abstände zwischen der Oberfläche von A und den Diskretisierungspunkten umliegender Objekte.
Ist A ein deformierbarer Körper, muss d nach jedem Integrationsschritt aktualisiert werden. Da dies sehr rechenintensiv ist, sind Distance Fields für deformierbare Körper in interaktiven Umgebungen nur bedingt geeignet. Für nicht-deformierbare Festkörper dagegen müssen Distance Fields nur einmal, vor Beginn der Simulation konstruiert werden. Während der Simulation beschränkt sich ihre Aktualisierung dann auf Translationen und Rotationen.

Continuous Collision Detection Dieser Ansatz fällt etwas aus der Reihe, da er nicht der Beschleunigung, sondern der Qualitätssteigerung einer Kollisionserkennung dient. Normalerweise frieren Algorithmen für die Kollisionserkennung

die Simulationsumgebung zu einem diskreten Zeitpunkt ein, um nach Objekt-Durchdringungen zu suchen. Werden zum Zeitpunkt t und zum Zeitpunkt $t + h$ keine Kollisionen entdeckt, ist es jedoch trotzdem möglich, dass Kollisionen im offenen Zeitintervall $(t, t + h)$ stattgefunden haben. Ansätze für eine Continuous Collision Detection behandeln dieses Problem, indem sie Objekt-Bewegungen im Zeitraum $(t, t + h)$ interpolieren. Betrachtet man beispielsweise eine nicht-deformierbare Kugel mit Radius r, so definiert die lineare Interpolation der Kugel-Bewegung einen abgerundeten Zylinder, der die Kugel-Position zum Zeitpunkt t mit derjenigen zum Zeitpunkt $t + h$ verbindet. Eine Kollision zwischen zwei Kugeln wird ausgeschlossen, falls sich die beiden entsprechenden Zylinder nicht überschneiden.
Eine Continuous Collision Detection kann z. B. in Kombination mit einer BVH definiert werden ([17], [104]). Hierbei decken Hüllkörper die Volumina ab, die durch die lineare Bewegung von Objekt-Primitiven zwischen zwei Zeitschritten definiert sind.

3.3.2 Kollisionsantwort

Prinzipiell hat die Kollisionsantwort dafür Sorge zu tragen, dass geometrische Randbedingungen der Simulation eingehalten werden. Im Wesentlichen sind diese durch die Forderung formuliert, dass kollidierende Körper sich nicht gegenseitig durchdringen dürfen[9]. Im Bereich „Simulation deformierbarer Objekte“ existiert kaum Literatur, die sich explizit mit dem Thema Kollisionsantwort befasst. Daher haben sich bisher nur sehr wenige, klar unterscheidbare Konzepte herauskristallisiert, die im Folgenden beispielhaft beschrieben werden.

Die Algorithmen für die Kollisionsantwort lassen sich grob in *Penalty Based Methods* und *Constraint Based Methods* unterteilen. Der Begriff „Penalty“ deutet an, dass die Kollisionsantwort die Verletzung einer Randbedingung zunächst zulässt. Anschließend wirkt die „Bestrafung“ dem ungültigen Systemzustand entgegen, eventuell jedoch ohne zu garantieren, dass dieser instantan vollständig behoben wird. Constraint Based Methods dagegen beschränken die Bewegung eines Körpers im Voraus, so dass geometrische Randbedingungen erst gar nicht verletzt werden können. Solche Methoden sind meistens dann zwingend erforderlich, wenn Objekt-Interaktionen simuliert werden, die in der Realität durch Berührungsflächen und Kontaktkräfte definiert sind.
Betrachten wir z. B. einen Würfel, der auf eine schiefe Ebene gelegt wird und diese nach dem Loslassen reibungsfrei hinabgleiten soll. Da das numerische Inte-

[9] Da sich in der Realität Volumina kollidierender Körper *niemals* überschneiden, entspricht diese Forderung einer *harten* Randbedingung. Die *tendenzielle* Einhaltung von Federnulllängen in einem Feder-Masse-System dagegen ist eine *weiche* Randbedingung.

grationsverfahren keine Kenntnis von geometrischen Randbedingungen hat, lässt die Schwerkraft den Würfel im nächsten Integrationsschritt in die schiefe Ebene einsinken. Eine klassische Penalty Based Method würde anschließend eine Kraft auf den Würfel definieren, mit Richtung orthogonal zur schiefen Ebene und Größe proportional zur Eindringtiefe. Das Ergebnis wäre kein geradliniges Gleiten, sondern ein stufenförmiges Holpern des Würfels entlang der schiefen Ebene. Für dieses Beispiel lässt sich leicht eine Constraint Based Method definieren, die erst gar nicht zulässt, dass der Integrationsschritt in einem ungültigen Zustand endet: Die auf den Würfel wirkende Schwerkraft lässt sich entlang der schiefen Ebene in eine orthogonale und eine tangentiale Komponente zerlegen. Ergänzt man die Eingabe des Integrationsverfahrens um einen Kraftvektor, welcher der negativen orthogonalen Komponente entspricht, kann sich der Würfel nur parallel zur schiefen Ebene bewegen.
Eine Constraint Based Method benötigt den exakten Zeitpunkt, zu dem sich zwei Objekte berühren. Erst wenn die Kontaktfläche zu diesem Zeitpunkt bekannt ist, können die Objekt-Bewegungen sinnvoll eingeschränkt werden. Das beschriebene Beispiel ist einfach umzusetzen, da der Kontakt zwischen Würfel und Ebene für den gesamten Verlauf der Simulation vordefiniert ist. Normalerweise muss der genaue Kontakt-Zeitpunkt $t_\mathrm{k} \in (t-h, t)$ berechnet werden, nachdem eine Kollision zum Zeitpunkt t festgestellt wurde. Anschließend muss die gesamte Simulationsumgebung in den Zustand zum Zeitpunkt t_k zurückgesetzt werden. Penalty Based Methods sind nicht nur leichter zu implementieren, sondern auch effizienter in Bezug auf die Rechenzeit, da sie den Zustand zum Zeitpunkt t „akzeptieren“ und das Problem im Verlauf der nächsten Integrationsschritte lösen.

Darüber hinaus unterscheiden sich die Methoden für die Kollisionsantwort darin, auf welcher „Ebene“ sie versuchen, die geometrischen Randbedingungen aufrecht zu erhalten bzw. wieder herzustellen: Die Ansätze können auf der Berechnung von Verschiebungen, Impulsen und/oder Kräften basieren. Genau genommen könnte man noch die impulsbasierten Ansätze von den geschwindigkeitsbasierten abgrenzen und die kraftbasierten von den beschleunigungsbasierten, je nachdem ob die Simulationsumgebung als abgeschlossenes System betrachtet wird, in dem der Gesamtimpuls erhalten bleibt und die Summe aller wirkenden Kräfte immer Null ist.

Meistens vermischen sich die hier genannten Konzepte sehr stark, was sich auch in der verwendeten Terminologie widerspiegelt. Betrachten wir noch einmal den Würfel auf der schiefen Ebene, diesmal mit folgender Definition für die Kollisionsantwort: Der eingesunkene Würfel wird orthogonal zur Ebene auf deren Oberfläche verschoben, zusätzlich wird sein Geschwindigkeitsvektor auf die Ebene projiziert. Position und Geschwindigkeit des Würfels haben somit gültige Wer-

te und unter der Voraussetzung, dass von nun an keine Kräfte mehr auf den Würfel wirken, sind zukünftige Kollisionen ausgeschlossen. Natürlich jedoch ist der Würfel nach wie vor der Schwerkraft ausgesetzt, weshalb dieser Ansatz erneute Kollisionen nicht verhindert. Trotzdem wird ein solcher Algorithmus wegen der expliziten Beschränkung des Geschwindigkeitsvektors als Constraint Based bezeichnet. (In interaktiven Echtzeitumgebungen mit deformierbaren Objekten wird auf die Berechnung von Kontakt-Zeitpunkten im vergangenen Zeitintervall $(t - h, t)$ meistens vollständig verzichtet.)

Kapitel 4

Simulation von Membranen im menschlichen Auge

Für die vorliegende Arbeit sollen Membrane im menschlichen Auge simuliert werden, mit denen der Benutzer über Instrumente interagieren kann und die unter Zug-Belastung reißen. Bevor die Reiß- und Interaktions-Algorithmen definiert werden können, muss ein Framework für die Simulation festgelegt werden, das die notwendigen Komponenten „Deformationsmodell“, „Numerische Integration“, „Kollisionserkennung“ und „Kollisionsantwort“ konkretisiert. Dieses Simulationsgerüst wird im Folgenden beschrieben. Anschließend werden spezielle Aspekte der Membran-Simulation behandelt. Hierunter fällt z. B. das Anhaften und Ablösen einer Membran.

4.1 Membrane als Feder-Masse-System

Für die Simulation von Weichgewebe in medizinischen Anwendungen werden meistens entweder Feder-Masse-Systeme oder FEM-Implementierungen verwendet. Beispiele für den Einsatz von Feder-Masse-Systemen sind die Arbeiten von Kühnapfel et al. [68], Brown et al. [20] oder Mollemans et al. [83]. FEM-Modelle sind z. B. in Cotin et al. [25, 26] und Picinbono et al. [97, 98] zu finden.
Der prinzipielle Vorteil einer FEM besteht darin, dass sie das Potential für physikalisch korrekte Simulationen bietet. Die Betonung liegt hier auf dem Wort „Potential“, denn die Simulationsergebnisse können nur dann physikalisch korrekt sein, wenn das FEM-Modell die Realität korrekt abbildet. Wenn das Modell jedoch – beispielsweise aus Rechenzeitgründen – stark vereinfacht wird, stellt sich die Frage, inwiefern der physikalische Ansatz tatsächlich „bessere“ Resultate als ein Feder-Masse-System garantiert. Besonders in interaktiven Echtzeitumgebungen fällt die Wahl für ein Deformationsmodell daher häufig auf Feder-Masse-

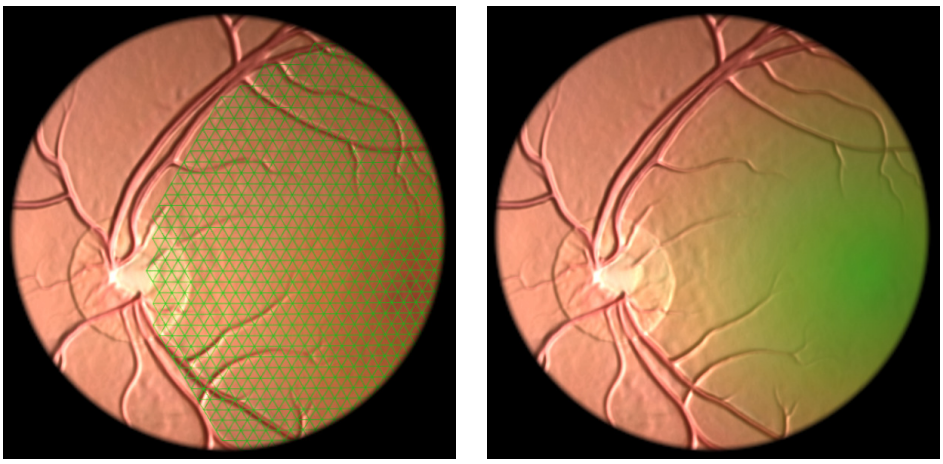
Systeme, die wegen ihrer Einfachheit prinzipiell weniger Rechenzeit beanspruchen als eine FEM.

Da sich Feder-Masse-Systeme für die Simulation von Membranen in EYESi-Trainingsmodulen bereits bewährt haben, wird auch für vorliegende Arbeit auf diesen Ansatz zurückgegriffen: Das Modell besteht aus einem Dreiecks-Mesh, in dem alle Mesh-Knoten über dieselbe Masse m und alle Federn über dieselbe Federkonstante c verfügen. Die Mesh-Topologie ist durch folgende Konsistenzbedingungen definiert:

- Topologisch benachbarte Dreiecke sind entweder über eine gemeinsame Spitze oder über eine gemeinsame Seite miteinander verbunden.
- Genau dann gehören alle drei Seiten eines Dreiecks zum Rand des Meshs, wenn zu diesem Dreieck kein Nachbar existiert. (Es ist also nicht erlaubt, dass ein Dreieck nur mit einer seiner Spitzen am Mesh-Rand hängt.)
- Jede Dreiecksseite entspricht einer Feder.
- Jede Feder verbindet zwei Mesh-Knoten.
- Jeder Mesh-Knoten verfügt über eine Ruheposition. Die Ruhelänge einer Feder muss mit dem Abstand der beiden entsprechenden Ruhepositionen übereinstimmen.

In Abb. 4.1(a) ist beispielhaft das Gitter-Modell einer Membran zu sehen. Ihre flächige Darstellung wird im einfachsten Fall dadurch erreicht, dass jedem Mesh-Knoten ein Farbwert zugeordnet wird. Die Farbwerte innerhalb eines Dreiecks entstehen durch Interpolation. Die meisten EYESi-Membrane verfügen inzwischen jedoch über Texturen. In beiden Fällen gilt, dass die Dreiecke des Simulationsmodells mit den Dreiecken der computergrafischen Darstellung übereinstimmen.

Die zuletzt genannte Konsistenzbedingung ist speziell für die in Kapitel 5 beschriebenen Reiß-Algorithmen formuliert und bedarf einer Erläuterung:
Um einen wachsenden Riss topologisch im Mesh zu realisieren, müssen immer wieder existierende Mesh-Elemente angepasst bzw. neue erzeugt werden. Jedes Mal, wenn topologische Operationen eine Feder betreffen, muss dieser anschließend eine neue Ruhelänge zugewiesen werden. Daher muss bekannt sein, welchen Abstand die beiden Feder-Knoten in der undeformierten Membran haben.
Dies setzt natürlich voraus, dass der Reiß-Algorithmus für die Konsistenz von Ruhepositionen sorgt, wenn er existierende Knoten adaptiert oder dem Mesh neue hinzufügt. Da sämtliche Mesh-Anpassungen in Kapitel 5 auf baryzentrischen Koordinaten beruhen, kann diese Konsistenz jedoch problemlos sichergestellt werden. (Wird beispielsweise ein neuer Knoten innerhalb eines existierenden



(a) (b)

Abbildung 4.1: Gitter-Modell und EYESi-Darstellung einer Membran

Dreiecks eingefügt, so bilden die Ruhepositionen der Dreiecksknoten das baryzentrische Koordinatensystem, in dem die Ruheposition des neuen Knotens definiert werden kann.)
Die Konsistenz von Ruhepositionen wird bereits an dieser Stelle explizit gefordert, da man im Allgemeinen nicht davon ausgehen kann, dass diese Bedingung nach der Mesh-Generierung erfüllt ist: Im Rahmen der Mesh-Initialisierung wird jedem Knoten eine Startposition zugewiesen. Anschließend erhält jede Feder üblicherweise eine Ruhelänge, die mit dem Abstand zweier Startpositionen übereinstimmt. In dieser Konfiguration entsprechen die Startpositionen den gesuchten Ruhepositionen. Soll das Mesh jedoch bereits zu Beginn der Simulation unter einer Vorspannung stehen, kann dies erreicht werden, indem die Ruhelängen ausgewählter Federn mit einem Faktor < 1 multipliziert werden. Die Menge aller Startpositionen beschreibt das Mesh dann nicht mehr in einem verzerrungsfreien Zustand.

4.2 Auswahl eines numerischen Integrationsverfahrens

In Abschnitt 3.2 wurde der prinzipielle Unterschied zwischen expliziten und impliziten Integrationsmethoden anhand des expliziten und des impliziten Euler-Verfahrens erläutert. Im Bereich Computergrafik finden sich besonders auf dem Gebiet der Kleidungssimulation Publikationen, die sich speziell mit dem Thema

Numerische Integration befassen ([126], [127], [60]). Unbedingt stabile, implizite Methoden werden hier häufig favorisiert, da sie den hohen Rechenaufwand für das Lösen der Gleichungssysteme durch eine große Schrittweite h wieder aufwiegen können. Es ist dann üblich, zwischen zwei gerenderten Bildern einer virtuellen Szene nur einen einzigen Integrationsschritt durchzuführen. Zu beachten ist, dass für die Lösung des Gleichungssystems eine Matrix invertiert werden muss, deren Aufbau von der Konnektivität der Punktmassen in einem Feder-Masse-System abhängt. Es ist nur dann möglich, diese Matrix-Inversion offline, also im Voraus zu berechnen, wenn die Mesh-Topologie im Verlauf der Simulation unverändert bleibt. Im Zusammenspiel mit einem Reiß-Algorithmus geht der Rechenaufwand daher trotz großer Schrittweite wieder stark in die Höhe. Darüber hinaus kann gerade der große Zeitschritt Probleme verursachen, da Objekte im Zeitraum $[t, t+h]$ sehr tief ineinander eindringen oder sogar tunneln können. Die Verwendung einer Continuous Collision Detection und/oder einer Constraint Based Collision Response ist zwar eine Lösung, allerdings eine sehr aufwändige. Aus diesen Gründen fällt hier die Entscheidung gegen ein implizites Verfahren.

Das von Leonhard Euler 1768 vorgestellte explizite Euler-Verfahren ist die einfachste numerische Methode zur Lösung eines Anfangswertproblems. Augustin Louis Cauchy benutzte es, um einige Eindeutigkeitsresultate für Gewöhnliche Differentialgleichungen zu beweisen. Für die Praxis ist das explizite Euler-Verfahren wegen seiner schlechten numerischen Eigenschaften jedoch weniger geeignet. Effizientere explizite Verfahren lassen sich beispielsweise dadurch herleiten, dass in die Berechnung des nächsten Schritts mehr als nur einer der bereits bekannten Stützpunkte einbezogen wird. Diese Vorgehensweise führt zur Klasse der *Mehrschrittverfahren*, zu der auch die *Verlet-Methode* (Verlet [125]) gehört. Sie wurde speziell für die Lösung Gewöhnlicher DGLs zweiter Ordnung entwickelt, um Newtons's Bewegungsgleichung zu integrieren. Daher bietet sie sich für Deformationsmodelle an, die aus einer Ansammlung von Punktmassen bestehen. Das Verlet-Verfahren oder eine seiner Varianten (z. B. *Leap-Frog-Verfahren*) wird sehr häufig für die Simulation deformierbarer Objekte eingesetzt, beispielsweise in den Arbeiten von Hauth et al. [61], Fuhrmann et al. [46], Kačić-Alesić et al. [66] und vielen anderen.
In seiner Grundform aktualisiert das Verlet-Verfahren die Position $\vec{x}(t)$ einer Punktmasse folgendermaßen:

$$\vec{x}(t+h) = 2\vec{x}(t) - \vec{x}(t-h) + \vec{a}(t)h^2$$

Die Verwendung von $\vec{x}(t)$ und $\vec{x}(t-h)$ auf der rechten Seite kennzeichnet den Verlet-Algorithmus als Mehrschrittverfahren. In dieser Form hat er jedoch den Nachteil, keine Geschwindigkeiten zu bestimmen, die eventuell für weitere Berechnungen in der Simulation notwendig sind. Daher wird für vorliegende Arbeit

auf den *Velocity-Verlet* zurückgegriffen:

$$\vec{x}(t+h) = \vec{x}(t) + \vec{v}(t)h + \frac{1}{2}\vec{a}(t)h^2$$

$$\vec{v}(t+h) = \vec{v}(t) + \frac{\vec{a}(t) + \vec{a}(t+h)}{2}h$$

Der Velocity-Verlet-Algorithmus ist stabiler als das explizite Euler-Verfahren und hat Konsistenz- und Konvergenzordnung 2.

Jeder reale Körper verliert während einer Deformation Energie. Dementsprechend muss das Feder-Masse-System gedämpft werden, da es andernfalls oszillieren würde. Zu diesem Zweck ist es üblich, für jeden Mesh-Knoten eine geschwindigkeitsabhängige, innere Reibungskraft zu berechnen[1]. Allerdings gilt, dass jede zusätzliche Kraft – auch eine Dämpfungskraft –, die in den expliziten Integrationsschritt einfließt, die Gefahr numerischer Instabilität erhöht. (Einzig ausgenommen sind Kräfte, welche sich darauf beschränken, Komponenten bereits berechneter Kräfte auszulöschen.) Eine einfache Möglichkeit, dem System Energie zu entziehen und gleichzeitig dessen Stabilität zu erhöhen, besteht darin, die Dämpfung in das numerische Verfahren zu integrieren und jede neu berechnete Geschwindigkeit mit einem Faktor < 1 zu multiplizieren. Diese Vorgehensweise ist nicht nur numerisch sicher, sondern auch „rechnerisch günstig“. Fuhrmann [45] z. B. schlägt diese Art der Dämpfung vor, die hier in Kombination mit dem Velocity-Verlet verwendet wird.

4.3 Prinzipielle Behandlung von Kollisionen

In den in Kapitel 7 beschriebenen EYESi-Anwendungen werden nur Kollisionen zwischen einer Membran und anderen Objekten der Operationsumgebung berücksichtigt. D. h. Selbstkollisionen der Membran werden vernachlässigt.
Von besonderer Bedeutung für vorliegende Arbeit ist die Wechselwirkung zwischen einer Membran und einem eingeführten Instrument. Der folgende Abschnitt beschreibt die algorithmische Basis für die Behandlung dieser Kollisionen. Auf spezielle Aspekte der Kollisionserkennung und -antwort für die chirurgische Instrument-Interaktion geht Kapitel 6 ein.

Kollisionserkennung In Abschnitt 3.3 wurden einige Methoden für die Kollisionserkennung vorgestellt.
Auf den Einsatz einer Continuous Collision Detection wird hier aus Rechenzeitgründen verzichtet. Insbesondere ist die Wahrscheinlichkeit für Tunneleffekte

[1] In Nealen et al. [86] z. B. ist beschrieben, wie innere Reibungskräfte definiert werden können.

beim Einsatz eines expliziten numerischen Verfahrens mit kleinen Schrittweiten relativ gering.
Wenn in einer Simulationsumgebung verschiedenartige Objekte miteinander kollidieren können, eignen sich Distance Fields nur für die nicht-deformierbaren Festkörper mit komplexer Geometrie. Da Instrumente in der Augenchirurgie einfache Formen haben – schließlich müssen sie durch kleine Einstichlöcher am Auge passen –, lohnt sich für deren virtuelle Gegenstücke die Definition von Distance Fields nicht.
Eine Spatial Subdivision bietet sich besonders für die Behandlung von Kollisionen zwischen zwei oder mehr deformierbaren Objekten an. In die Unterräume der Spatial Subdivision werden dann die Mesh-Primitive der Simulationsgitter einsortiert.
Für die Kollisionserkennung zwischen deformierbarer Membran und starren Instrumenten fällt die Wahl auf den Einsatz von Hüllkörpern. Als Hüllkörper-Typen dienen Kugeln und Zylinder. Ein Instrument wird entweder durch einen einzigen Hüllkörper (z. B. möglich bei einer stabförmigen Lampe) oder durch eine Hüllkörper-Kette approximiert. Da die Objekt-Geometrien einfach sind und die Gesamtanzahl notwendiger Hüllkörper gering ist, kann auf die Definition von Hüllkörper-Hierarchien (BVHs) verzichtet werden.
Um eine Narrow Phase zu definieren, werden Hüllkörper gewählt, die ein eingeführtes Instrument so eng wie möglich umschließen. Die Hüllkörper der Broad Phase stimmen mit denjenigen der Narrow Phase überein, außer dass deren Radien um die Breite d der Broad Phase vergrößert sind.
Welche Mesh-Primitive als Basis für die Kollisionserkennung dienen, hängt einerseits davon ab, welche Dimensionen ein Hüllkörper im Vergleich mit der Mesh-Auflösung hat und andererseits davon, wie „wichtig“ eine exakte Kollisionsbehandlung an der jeweiligen Stelle ist. Für die Wechselwirkung zwischen Membran und Instrument-Rumpf reicht es oftmals aus, nur Positionen von Mesh-Knoten zu prüfen, um Durchdringungen rechtzeitig zu erkennen. Wenn jedoch die Spitze eines nadelähnlichen Instruments mit der Membran interagiert, kann es passieren, dass diese in ein Mesh-Dreieck eindringt bevor Kollisionen mit Mesh-Knoten auftreten. In diesem Fall beruht die Kollisionserkennung (der Narrow Phase) daher auf Abstandsberechnungen zwischen Mesh-Dreiecken und Hüllkörpern.

Kollisionsantwort Constraint-basierte Methoden sind sehr aufwändig, insbesondere wenn mehrere Randbedingungen gleichzeitig aufrechterhalten werden müssen. Daher wird hier ein Penalty-basierter Ansatz verwendet, für den im Übrigen die kleinen Schrittweiten des expliziten Integrationsverfahrens von Vorteil sind, da diese bewirken, dass Kollisionen bereits bei geringen Eindringtiefen erkannt werden.

Die Kollisionsantwort soll ausschließlich Kräfte definieren und auf Adaptionen von Geschwindigkeiten oder Positionen verzichten. Schließlich sind es auch in der Realität Kräfte, die Durchdringungen zwischen Objekten verhindern. Für jedes kollidierende Mesh-Primitiv wird eine Penalty-Kraft berechnet, die dann auf die Knoten des Primitivs verteilt wird. Daher ergeben sich auch keine Probleme, falls die Simulation nachträglich um neue geometrische Randbedingungen erweitert wird. Jede Randbedingung resultiert in Antwortkräften, die sich in den Mesh-Knoten addieren.
Die Richtung einer Antwortkraft zeigt den kürzesten Weg aus dem kollidierenden Hüllkörper (bei reibungsfreier Wechselwirkung). Ihre Größe wird proportional zur Eindringtiefe des Mesh-Primitivs gewählt.

Ablauf eines Simulationsschritts Um zu garantieren, dass der Benutzer einer Echtzeitanwendung Bewegungen als kontinuierlich wahrnimmt, muss diese mit einer Bildwiederholrate von mindestens 25Hz (= 25fps = 25 *frames per second*) laufen. EYESi-Trainingsmodule sind auf 30Hz getaktet, somit stehen für die Generierung eines einzelnen Bildes insgesamt ca. 0.033 Sekunden Rechenzeit zur Verfügung. Alle Berechnungen, die notwendig sind, um den Zustand des Feder-Masse-Modells zum Zeitpunkt $t+0.033$s zu ermitteln, werden hier im sogenannten *Simulationsschritt* zusammengefasst. Da für die numerische Integration mit dem Velocity-Verlet ein explizites Verfahren verwendet wird, ist es erforderlich, die „Simulationsschrittweite“ von 0.033s in n Integrationsschritte der Schrittweite $h = 0.033\text{s}/n$ zu unterteilen. Um h und damit die Gefahr für numerische Instabilität zu minimieren, wird n in Abhängigkeit der zur Verfügung stehenden Rechenzeit maximal gewählt. Insbesondere steigern kleine Schrittweiten die Qualität der hier verwendeten Methoden für Kollisionserkennung und -antwort.
Vor jedem der n Integrationsschritte müssen alle auf die Membran wirkenden Kräfte neu berechnet werden. Daher muss auch die Kollisionserkennung für ein Instrument n-mal pro Simulationsschritt durchgeführt werden. Allerdings liefert das EYESi-Tracking-System nur einmal pro Frame Informationen über die momentane Position eines eingeführten Instruments. Seine Rotation und Translation werden in Form einer 4×4 Matrix übergeben, die als *Modelview* (MV) bezeichnet wird. Mit Hilfe der aktuellen MV und der MV des letzten Frames wird eine Instrument-Transformation im Verlauf der n Integrationsschritte linear interpoliert.
Wie ein kompletter Simulationsschritt aufgebaut ist, veranschaulicht Algorithmus 1. Der Übersicht halber setzt dieser Pseudo-Code voraus, dass sich die gesamte Kollisionserkennung auf die Interaktion zwischen Membran und einem einzelnen Instrument beschränkt.
Wie Zeile 2 zu entnehmen, wird die Broad Phase eines Instruments nur einmal vor

Beginn der Integrationsschleife berechnet. Für die exakten Überschneidungstests der Narrow Phase innerhalb der Schleife (Zeile 19) wird die MV des Instruments linear interpoliert (Zeile 17). Die in Zeile 18 beginnende Schleife hat die Laufvariable e, die für „Mesh-Element“ steht. Ist e eine Feder oder ein Dreieck, so ist es Aufgabe der Kollisionsantwort, Penalty-Kräfte auf die Knoten von e zu verteilen. Daher die Bezeichnung „nodeForces(e)“ in Zeile 21. Wenn in Zeile 27 die numerische Integration durchgeführt wird, müssen alle wirkenden Kräfte in den

Algorithm 1 Calculate one simulation step per frame

```
 1: newToolMV ← getTrackedToolMV()
 2: broadPhase ← broadPhaseCollisionDetection(oldToolMV,mesh)
 3:
 4: for integrationStep = 1 to numberOfIntegrationSteps do
 5:    // Set all node forces to zero
 6:    for all nodes n ∈ mesh do
 7:       force(n) ← (0, 0, 0)
 8:    end for
 9:
10:    // Calculate internal forces
11:    for all nodes n ∈ mesh do
12:       force(n) ← getForceFromDeformationModel(n)
13:    end for
14:
15:    // Calculate external forces
16:    factor ← integrationStep/numberOfIntegrationSteps
17:    interpolatedToolMV ← oldToolMV + (newToolMV − oldToolMV) · factor
18:    for all elements e ∈ broadPhase do
19:       collisionInfo ← narrowPhaseCollisionDetection(interpolatedToolMV,e)
20:       if collisionInfo not empty then
21:          nodeForces(e) ← nodeForces(e) + collisionResponse(collisionInfo)
22:       end if
23:    end for
24:
25:    // Calculate one integration step
26:    stepSize ← 0.033s/numberOfIntegrationSteps
27:    velocityVerlet(stepSize)
28: end for
29:
30: oldToolMV ← newToolMV
31: renderVirtualScene()
```

Mesh-Knoten aufaddiert und abgespeichert sein. Der Simulationsschritt ist mit Zeile 30 beendet, Zeile 31 soll lediglich andeuten, wie sich dieser in die komplette Berechnung eines Frames einfügt.

4.4 Spezielle Aspekte der Membran-Simulation

In den vorherigen Abschnitten wurde eine algorithmische Basis für die Simulation zusammengestellt. Die Grundform des Feder-Masse-Modells wird nun um einige Komponenten erweitert.

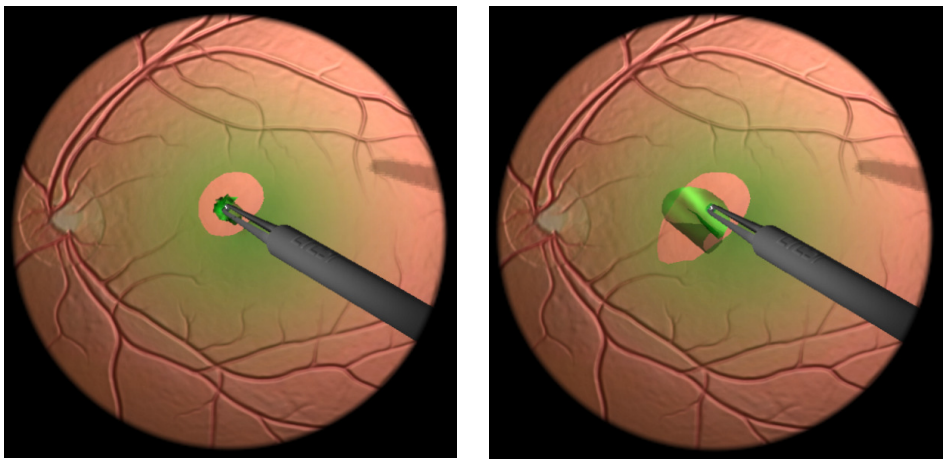
4.4.1 Modellierung der Biege-Steifigkeit

Die Federkonstanten des Feder-Masse-Systems definieren die Zug-Festigkeit oder Zug-Steifigkeit der Membran. Zwei einzelne Federn, die über einen gemeinsamen Knoten miteinander verbunden sind, können widerstandsfrei jeden beliebigen Winkel einschließen. In seiner bisherigen Form bildet das Feder-Masse-System demnach keine Biege-Steifigkeit ab. Einige der Membrane im menschlichen Auge verfügen jedoch über eine für den Chirurgen spürbare Biege-Steifigkeit, weshalb das Membran-Modell in der Lage sein muss, auch diese Eigenschaft abzubilden.

Im Wesentlichen existieren zwei verschiedene Konzepte für die Simulation von Biege-Steifigkeit:
Die erste Möglichkeit besteht darin, dem Feder-Masse-Gitter zusätzliche Struktur-Federn hinzuzufügen. Teilen sich zwei Dreiecke eine gemeinsame Feder, kann eine Biege-Feder z. B. die beiden Knoten verbinden, die nicht zur gemeinsamen Feder gehören. Derartige Ansätze sind beispielsweise in den Arbeiten von Provot [102] und Eberhardt et al. [34] zu finden. Als Nachteil dieser Methode wird häufig angeführt, dass Biege-Federn nicht nur die Biege-Steifigkeit sondern immer auch die Zug-Steifigkeit des simulierten Materials beeinflussen.
Daher wurden Modelle wie das von Bridson et al. [18] vorgeschlagen, welche die Krümmung einer Oberfläche lokal approximieren und Biege-Kräfte normal zur Oberfläche ableiten, um der Krümmung entgegen zu wirken[2]. Verglichen mit dem ersten Ansatz sind diese jedoch sehr rechenaufwendig.

Für vorliegende Arbeit wird – aus Rechenzeitgründen – auf den Einsatz von Biege-Federn zurückgegriffen, falls die Biege-Steifigkeit einer Membran nicht vernachlässigt werden kann. Basis für die Definition eines Biege-Elements sind zwei Knoten, welche durch zwei Federn eines dritten Knotens miteinander verkettet sind. Diese beiden Knoten werden im Rahmen der Mesh-Generierung genau dann

[2] Einen Überblick über Krümmungs-basierte Modelle bieten Thomaszewski und Wacker [121].



(a) (b)

Abbildung 4.2: Eine Membran einmal ohne und einmal mit Biege-Elementen

durch eine Biege-Feder verknüpft, falls sie nicht zu ein und demselben Dreieck gehören. Die Ruhepositionen der Knoten definieren die Federnulllänge des Biege-Elements. Während der Mesh-Initialisierung wird darauf geachtet, dass keine zwei Biege-Elemente dieselben zwei Knoten miteinander verbinden.

In Abb. 4.2 ist zu sehen, wie sich die Biege-Federn auf die Deformation einer Membran auswirken. Eine Instrument-Bewegung, in deren Verlauf eine Pinzette ein Stück Membran greift und von der Retina herunterzieht, wurde aufgezeichnet und zweimal abgespielt: einmal ohne und einmal mit Biege-Elementen. Während sich die Membran in Abb. 4.2(a) auf kleinem Raum zusammengefaltet hat, haben dies die Biege-Elemente in Abb. 4.2(b) verhindert. (Die beiden herausgerissenen Membran-Stücke unterscheiden sich in ihrer Geometrie, da die Biege-Elemente natürlich die Spannungsverteilung im Mesh beeinflussen.)

4.4.2 Anhaften und Ablösen einer Membran

Die Membrane, die im Verlauf eines chirurgischen Eingriffs entfernt werden, haften an anderen Strukturen des Auges. Erst eine gezielte Instrument-Interaktion löst die Membran vom Untergrund. Daher muss auch das Anhaften und Ablösen einer virtuellen Membran modelliert werden.

Um Mesh-Knoten an einer bestimmten Position zu fixieren, werden häufig Federn mit Ruhelänge $l_0 = 0$ benutzt, die in der Literatur manchmal als *Homing*-Federn bezeichnet werden. Zhang et al. [138] und LeDuc et al. [70] verwenden diese spezi-

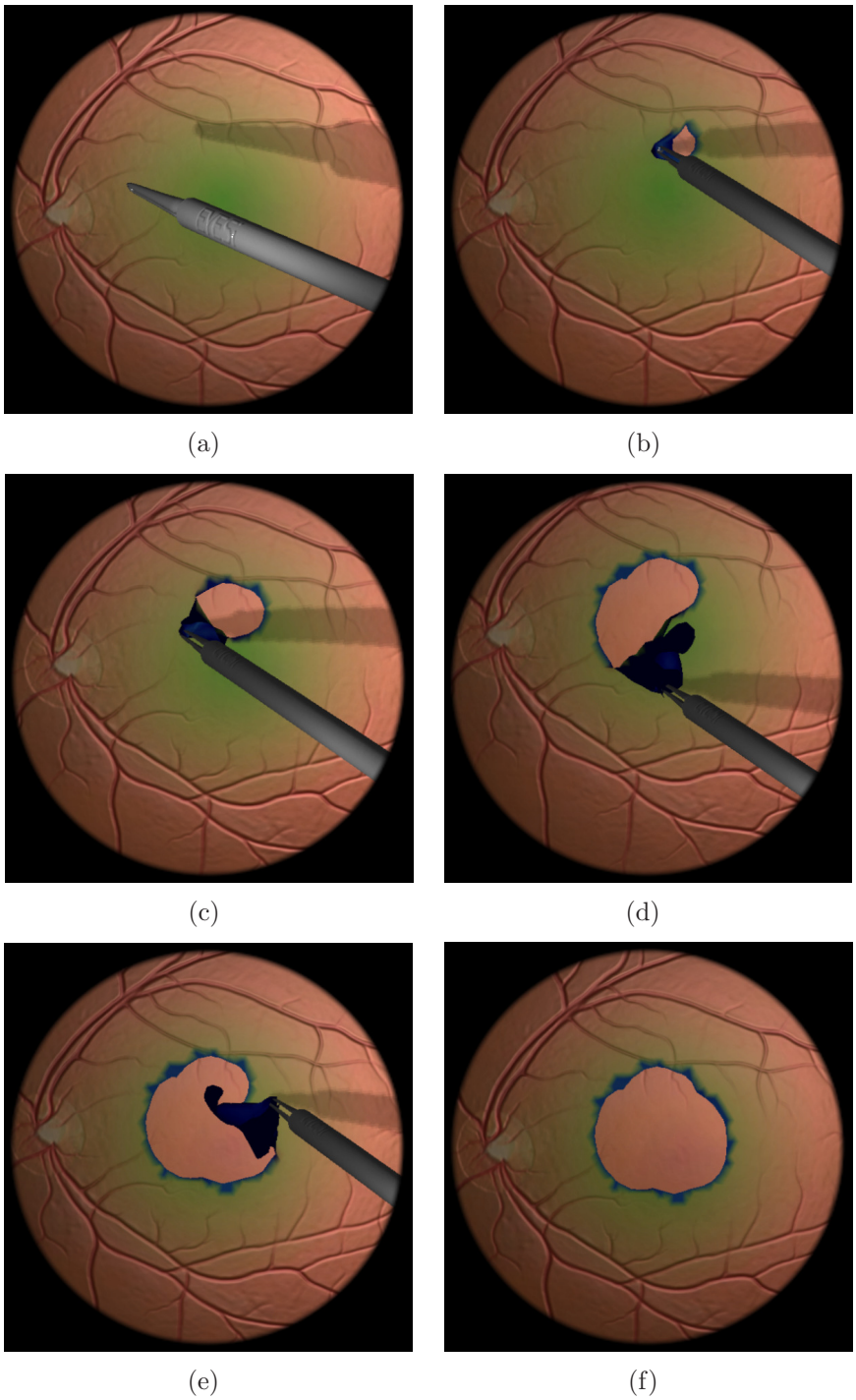
ellen Federn, um volumetrische Objekte durch ein Oberflächen-Mesh modellieren zu können. Die Homing-Kräfte dienen hier also der Form-Stabilität des jeweiligen Objekts. Chang et al. [22] setzen Homing-Federn in der Haar-Simulation ein, damit benachbarte Haarsträhnen aneinander haften. Bridson et al. [18] lassen verschwitzte Kleidungsstücke an der Haut eines Avatars kleben.
Homing-Kräfte – die wie alle Kräfte in jedem Integrationsschritt aktualisiert werden müssen – haben die positive Eigenschaft, weniger Rechenoperationen zu beanspruchen als „normale“ Federkräfte. Für eine Feder mit $l_0 > 0$ muss eine Quadratwurzel berechnet werden, um deren momentane, absolute Auslenkung zu bestimmen. Für die Berechnung einer Homing-Kraft dagegen muss nur der Vektor, der die aktuelle Position eines Knotens mit dessen Homing-Position verbindet, mit einer Homing-Federkonstante multipliziert werden.

Für die Anwendungen in Kapitel 7 erhält jeder Knoten im Rahmen der Mesh-Initialisierung eine Homing-Position, die im einfachsten Fall mit dessen Ruheposition identisch ist. Gehört der Untergrund, an dem die Membran anhaftet, zu einem ebenfalls deformierbaren Objekt der EYESi-Umgebung, werden die Homing-Positionen im Verlauf der Simulation an die Deformation der Objekt-Oberfläche angepasst.
Zusätzlich zu seiner Homing-Position verfügt jeder Knoten über eine initiale Homing-Konstante. Sobald sich ein Knoten ablösen soll, wird seine Homing-Konstante im Verlauf mehrerer Iterationen dekrementiert bis sie den Wert Null erreicht. Das stufenweise Herabsetzten der Konstante verhindert, dass sich ein Knoten sprunghaft vom Untergrund löst.
Als Ablöse-Kriterium dient der Abstand des Knotens von seiner Homing-Position. Denn dieser ist ein Maß für die Summe aller Kräfte, welche den Knoten von seiner Homing-Position wegziehen.
Die Funktion „detachMembrane()“, die das Ablösen realisiert, wird nur einmal pro Frame nach Beendigung der Integrationsschleife (Zeilen 4 bis 28 in Algorithmus 1 auf Seite 35) aufgerufen. Denn bevor entschieden wird, welche Knoten sich ablösen, sollen sich lokale Verzerrungen und Verspannungen möglichst gleichmäßig über das Mesh verteilen können.
Algorithmus 2 veranschaulicht die Implementation der Funktion „detachMembrane()“ in Form von Pseudo-Code:
Ein Knoten gilt als abgelöst, falls seine momentane Homing-Konstante Null (Zeile 4) oder echt kleiner als seine initiale Homing-Konstante (Zeile 7) ist. In letzterem Fall beschränkt sich die Funktion darauf, die Homing-Konstante des Knotens weiter zu dekrementieren (Zeile 8).
In den Zeilen 12 bis 24 wird entschieden, ob der jeweilige Knoten n zur Ablösung überhaupt in Frage kommt. Hintergrund ist der, dass ein Chirurg in der Realität Membrane ausgehend von einer Rand- oder Ablösekante abzieht bzw. ab-

pellt. Nur so lässt sich eine Membran entfernen, ohne stark zu fragmentieren. Daher erlaubt Algorithmus 2 nur dann, Knoten n abzulösen, wenn dieser entweder auf dem Rand der Membran liegt (Zeile 14) oder zu einer Ablösekante gehört. Letzteres trifft zu, falls n über eine Feder mit einem Knoten verbunden ist, dessen Homing-Konstante bereits bis auf Null herabgesetzt wurde (Zeile 18). Möchte

Algorithm 2 Implementation of function detachMembrane()

```
1:  // Detach membrane node-wise
2:  for all nodes n ∈ mesh do
3:    // Node n already detached?
4:    if homingConstant(n) = 0 then
5:      continue with next iteration
6:    end if
7:    if homingConstant(n) < initialConstant(n) then
8:      homingConstant(n) ← max(homingConstant(n) − decrement, 0)
9:      continue with next iteration
10:   end if
11:
12:   // Is it allowed to detach node n?
13:   forbidDetachment ← true
14:   if isEdgeNode(n) then
15:     forbidDetachment ← false
16:   end if
17:   for all springs s with {n, otherNode} ∈ s do
18:     if homingConstant(otherNode)=0 then
19:       forbidDetachment ← false
20:     end if
21:   end for
22:   if forbidDetachment then
23:     continue with next iteration
24:   end if
25:
26:   // Detach node n?
27:   vector ← homingPosition(n) − currentPosition(n)
28:   squaredDistance ← vector · vector
29:   squaredThreshold ← minimalDistanceForDetachment^2
30:   if squaredDistance > squaredThreshold then
31:     homingConstant(n) ← max(homingConstant(n) − decrement, 0)
32:   end if
33: end for
```



(a) (b)

(c) (d)

(e) (f)

Abbildung 4.3: Anhaften und Ablösen einer Membran

der Chirurg seinen virtuellen Eingriff an einer beliebigen Stelle der Membran beginnen, muss er diese zunächst mit einem geeigneten Instrument anritzen. Die Reiß-Algorithmen aus Kapitel 5 ermöglichen diese Interaktion und erzeugen neue Randknoten, von denen ausgehend die Membran anschließend abgelöst werden kann.
In den Zeilen 26 bis 32 schließlich wird entschieden, ob Knoten n tatsächlich abgelöst werden soll. Überschreitet der Abstand zwischen Knoten n und seiner Homing-Position einen vorgegebenen Grenzwert, wird seine Homing-Konstante zum ersten Mal dekrementiert.

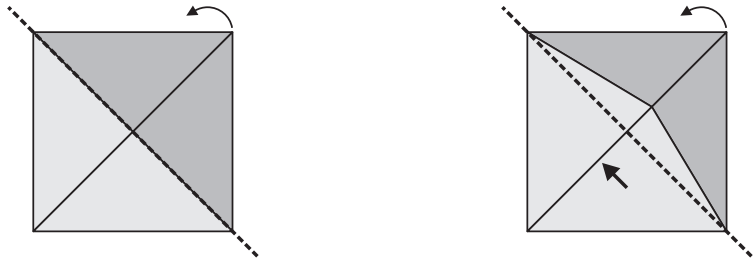
Abb. 4.3 veranschaulicht die Funktionsweise der Methode „detachMembrane()“. Alle abgelösten Knoten der Membran sind blau eingefärbt. In Abb. 4.3(f) ist zu sehen, dass sich von der zurückgebliebenen Membran nur die Risskante vom Untergrund gelöst hat und alle anderen Mesh-Knoten noch anhaften.

4.4.3 Das Locking-Problem

Locking (das Blockieren) beschreibt die Abhängigkeit zwischen der Topologie eines Meshs und seinen Bewegungs-Freiheitsgraden. Eine deformierbare Oberfläche, die durch ein Dreiecks-Mesh (ohne Biege-Elemente) simuliert wird, kann sich nur dann entlang einer vorgegebenen geraden Linie falten, falls in der Mesh-Topologie eine Kette von Dreiecksseiten genau auf dieser Linie liegt[3]. Eine beliebig gewählte Faltung erzeugt Verzerrungen, deren Ausmaße von der Mesh-Topologie abhängen. Je gröber die Mesh-Auflösung, desto stärker spürbar ist dieser Effekt.
In bestimmten Operationssituationen ist die Eigenschaft einer simulierten Membran, der Ausbildung einer klaren Falzkante zu widerstehen, hinderlich bzw. unrealistisch: Der Chirurg benutzt ein nadelartiges Instrument, um ein bereits abgelöstes Stück Membran entlang der Ablösefront umzuklappen und flach auf den Untergrund zu legen. Wiederholt hakt er sich in die Membran ein, übt dabei jedes mal leichten Druck aus um nicht abzurutschen und schiebt das Membran-Stück voran. In der Realität bleibt die Membran flach liegen, während der Arzt das Instrument aus- und wieder einhakt. Eine simulierte Membran jedoch wird sich entlang der Ablösekante wieder aufrichten.

Eine mögliche Lösung des Locking-Problems stammt aus der FEM-Simulation und sieht die Verwendung *nicht-konformer* Elemente vor (Brenner und Scott [16]). In einem nicht-konformen Mesh dürfen die Übergänge zwischen benachbarten Elementen geometrisch unstetig sein. Dadurch wird verhindert, dass sich Elemente in ihrer Bewegung unnatürlich gegenseitig behindern. English und Bridson [35] greifen für die Textil-Simulation auf diesen Ansatz zurück, damit Stoffe

[3] Liu et al. [71] z. B. analysieren das Locking-Problem anhand eines Dreiecks-Meshs.



(a) (b)

Abbildung 4.4: Das Locking-Problem

ein realistisches Faltenmuster ausbilden können: Im Mesh benachbarte Dreiecke sind über die Mittelpunkte ihrer Seiten verknüpft. Somit müssen benachbarte Dreiecke nicht an ihren Eckpunkten übereinstimmen, wodurch im Verlauf der Simulation Löcher im Mesh entstehen. Ein nicht-konformes Simulations-Mesh ist demnach nicht für die grafische Darstellung geeignet, weshalb English und Bridson ihr Modell zusätzlich mit einem konformen Mesh koppeln müssen.
Eine weitaus einfachere Lösung ist in der Arbeit von Choi und Ko [23] zu finden, die sich ebenfalls mit Kleidungssimulation befasst. Die Dreieckskanten in ihrem Modell verfügen über eine hohe Zug-Steifigkeit, während Kompressionen nur sehr geringe Rückstellkräfte erzeugen. Abb. 4.4 veranschaulicht, wie dieser Ansatz dem Locking-Problem entgegenwirkt. Zwischen den hellgrauen und dunkelgrauen Dreiecken soll sich eine Falzkante ausbilden, indem die beiden dunkelgrauen Dreiecke um die gestrichelte Linie rotieren. In Abb. 4.4(a) ist dies ohne Verzerrungen möglich, da alle drei Knoten, die sowohl zu hellen als auch zu dunklen Dreiecken gehören, auf der gestrichelten Linie liegen. In Abb. 4.4(b) dagegen führt die Rotation zu einer Verkürzung der durch den Pfeil markierten Kante. Wenn sich diese Kante jedoch leicht oder sogar vollkommen widerstandslos komprimieren lässt, wird sie die Faltung nicht blockieren.

Um das Ausbilden einer Falzkante (bei der Interaktion mit Membranen ohne Biege-Elemente) zu unterstützen, wird hier in Anlehnung an die Arbeit von Choi und Ko Algorithmus 3 vorgeschlagen. Dieser beschreibt die Implementation der Funktion „supportFoldedEdge()“, die im Rahmen eines Simulationsschritts im direkten Anschluss an die Funktion „detachMembrane()“ aufgerufen wird. Die Idee ist, *blockierende* Federn – also solche, die der Ausbildung einer Falzkante entgegenwirken – temporär zu deaktivieren. Da der Chirurg eine Membran üblicherweise entlang einer Ablösefront umzulegen versucht, wird die Suche nach blockierenden Federn auf die Umgebung dieser Fronten eingeschränkt. Wenn sich

Algorithm 3 Implementation of function supportFoldedEdge()

```
1: // Mark all springs as non-locking
2: for all springs s ∈ mesh do
3:    lockingFlag(s) ← false
4: end for
5:
6: // Find and deactivate locking springs
7: for all nodes n ∈ mesh do
8:    // Is node n part of a folded edge?
9:    if homingConstant(n) = 0 then
10:      continue with next iteration
11:   end if
12:   isPartOfFoldedEdge ← false
13:   for all springs s with {n, otherNode} ∈ s do
14:      if homingConstant(otherNode) = 0 then
15:         isPartOfFoldedEdge ← true
16:      end if
17:   end for
18:   if isPartOfFoldedEdge = false then
19:      continue with next iteration
20:   end if
21:
22:   // Deactivate compressed springs
23:   for all springs s with n ∈ s do
24:      if isCompressed(s) then
25:         lockingFlag(s) ← true
26:         constant(s) ← 0
27:      end if
28:   end for
29: end for
30:
31: // Reactivate springs
32: for all springs s ∈ mesh do
33:   if lockingFlag(s) OR constant(s) = initialConstant(s) then
34:      continue with next iteration
35:   end if
36:   constant(s) ← min [constant(s) + increment, initialConstant(s)]
37: end for
```

ein umgelegtes Stück Membran wieder aufrichtet, sind dafür diejenigen Federn verantwortlich, die zuvor durch die Faltung komprimiert wurden. Daher werden genau die Federn deaktiviert, die zum Zeitpunkt des Aufrufs von „supportFoldedEdge()“ gestaucht sind. Deaktivierte Federn, die nicht mehr komprimiert sind oder nicht mehr zur Umgebung einer Ablösefront gehören, werden wieder aktiviert. Die sowohl räumliche als auch zeitliche Beschränkung des Deaktivierens garantiert, dass „supportFoldedEdge()“ keine unerwünschten Nebeneffekte auf das prinzipielle Deformationsverhalten der Membran hat.
Die Funktion markiert zunächst alle Federn der Membran als *nicht-blockierend* (Zeilen 1 bis 4 in Algorithmus 3). Anschließend werden sämtliche Knoten der Membran durchlaufen (Zeilen 6 bis 29). In dieser Schleife wird zunächst entschieden, ob Knoten n zu einer Ablösekante gehört (Zeilen 8 bis 20). Voraussetzung hierfür ist, dass n selbst noch anhaftet oder gerade dabei ist sich abzulösen (Zeilen 9 bis 11). Knoten n wird als Teil einer Ablösefront aufgefasst, falls er über eine Feder mit einem Knoten verbunden ist, der bereits vollständig abgelöst ist (Zeilen 12 bis 17). Liegt n auf einer Ablösekante, werden anschließend alle an n hängenden Federn überprüft (Zeilen 22 bis 28): Jede momentan komprimierte Feder wird als *blockierend* markiert und deaktiviert, indem ihre Federkonstante auf Null gesetzt wird. Abschließend durchläuft Algorithmus 3 noch einmal sämtliche Federn des Meshs, um deaktivierte aber nicht mehr blockierende Federn wieder zu aktivieren (Zeilen 31 bis 37).

Abb. 4.5 veranschaulicht die Wirkung von „supportFoldedEdge()“. Zunächst wurde eine Instrument-Bewegung aufgezeichnet, in deren Verlauf eine Pinzette ein Membran-Stück von der Linse herunterzieht und loslässt, bevor es vollständig vom Rest der Membran abreißt. Anschließend wurde die Instrument-Bewegung zweimal abgespielt, einmal ohne und einmal mit Aufruf der Methode „supportFoldedEdge()“. (Da die transparente Membran schlecht zu erkennen ist, wurden einige ihrer Konturen nachgezeichnet: Die weiße Linie markiert die Risskante, die weiß-gestrichelte die Ablösefront und die schwarze den Umriss der abgelösten Membran-Lasche.) In Abb. 4.5(d) ist zu sehen, dass sich die Membran nach Abschluss der Interaktion entlang der Ablösekante wieder aufgerichtet hat. Abb. 4.5(d') dagegen zeigt, wie die Funktion „supportFoldedEdge()“ dafür sorgt, dass das umgeklappte Stück Membran flach auf der Linse liegen bleibt.

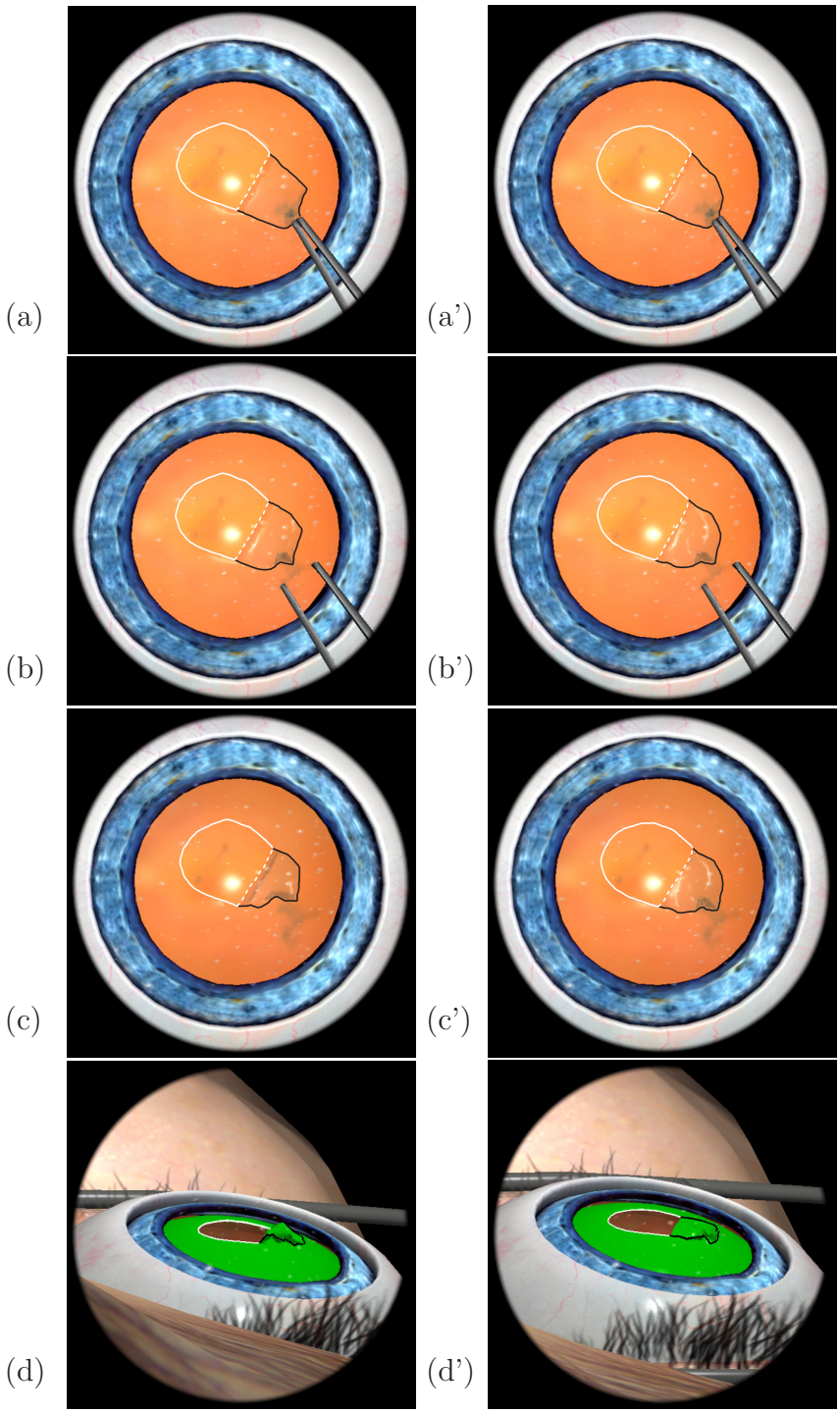


Abbildung 4.5: Ausbildung einer Falzkante: links ohne und rechts mit Hilfe der Funktion „supportFoldedEdge()“

Kapitel 5

Reißen von Membranen

In vielen virtuellen Umgebungen ist es nicht notwendig, dass simulierte Objekte reißen oder brechen können. Denkt man beispielsweise an eine Simulation für die Anprobe von Kleidungsstücken, ist nur gefordert, dass sich die Textilien realistisch an den Körper des Avatars anschmiegen. Bietet eine interaktive Simulationsumgebung dem Benutzer jedoch die Möglichkeit, Objekte beliebig großen Belastungen auszusetzen, müssen diese zerreißen, wenn die Materialgrenzen erreicht sind.
Für einen chirurgischen Eingriff gilt im Allgemeinen, dass sich der Arzt zunächst Zugang zum eigentlichen Operationsgebiet verschaffen muss, um anschließend Gewebe von erkrankten Organen zu entfernen. In Simulatoren für die medizinische Ausbildung ist es daher fast immer notwendig, simulierte Objekte auch fragmentieren zu können.

Der „Medizinische Hintergrund“ in Abschnitt 5.1 erläutert, weshalb ein Simulator für die augenchirurgische Ausbildung ohne einen Reiß-Algorithmus nicht auskommt. Abschnitt 5.2 enthält eine Liste von Anforderungen, die ein solcher Algorithmus erfüllen sollte. Abschnitt 5.3 stellt eine Reihe von Publikationen vor, in denen virtuelle Objekte zerrissen oder zerbrochen werden. Die zugrunde liegende Aufgabenstellung, sich fortpflanzende Risse zu simulieren, lässt sich in zwei Teilprobleme zerlegen. Zunächst einmal muss mit Hilfe geeigneter Bruchkriterien entschieden werden, wo und wie sich ein Riss ausbreitet. Ist diese Information z. B. in Form einer Rissebene gegeben, muss der Rissfortschritt anschließend topologisch in der Mesh-Struktur umgesetzt werden. Die beiden im Rahmen der vorliegenden Arbeit entwickelten Reiß-Algorithmen zerfallen daher jeweils in zwei voneinander unabhängige Komponenten, die in Abschnitt 5.4 und Abschnitt 5.5 beschrieben werden. Beide Reiß-Algorithmen ermitteln die Rissausbreitung, wie in Abschnitt 5.4 beschrieben. Sie unterscheiden sich jedoch darin, wie sie den Riss topologisch umsetzen (Abschnitt 5.5). Die topologischen Anpassungen des zweiten Reiß-Algorithmus gehen weder aus den Publikationen hervor,

die in Abschnitt 5.3 aufgelistet sind, noch wurde ein komplett neues Verfahren entwickelt. Vielmehr wurde ein Algorithmus adaptiert, der für virtuelle Schneidevorgänge vorgesehen ist. Da das Schneiden simulierter Objekte ein eigenständiges Forschungsgebiet ist, enthält Abschnitt 5.5 ein separates Unterkapitel, das die relevanten Publikationen vorstellt. Abschnitt 5.6 präsentiert Ergebnisse anhand von Test-Anwendungen, Abschnitt 5.7 schließt das Kapitel zusammenfassend ab.

5.1 Medizinischer Hintergrund

In der Ophthalmologie kommt es sehr häufig vor, dass membranartige Strukturen aus dem Auge entfernt werden müssen.
Ein Beispiel ist die *Kapsulorhexis* – ein Teilschritt im Rahmen einer Katarakt-Operation: Die Linse des menschlichen Auges ist von einer Membran umgeben, genannt *Kapselsack* oder *Linsenkapsel*. Der Chirurg muss den Kapselsack öffnen, indem er ein kleines, rundes Stück aus der Membran herausreißt. In Abb. 5.1 ist zu sehen, wie dieser Eingriff mit einer Pinzette durchgeführt wird. Da eine zentrierte, kreisrunde Öffnung des Kapselsacks entstehen soll, muss der Chirurg die Rissausbreitung mit viel Feingefühl kontrollieren.
Eingriffe im hinteren Augenabschnitt erfordern häufig, Membrane relativ großflächig von der Retina zu entfernen. Ein Beispiel sind pathologische Membrane, die auf der Netzhaut wuchern und die Sehkraft beeinträchtigen. Darüber hinaus kommt es vor, dass ein Teil der *Inneren Grenzmembran* – abgekürzt *ILM* für *Internal Limiting Membrane* – von der Retina abgezogen werden muss, um diese

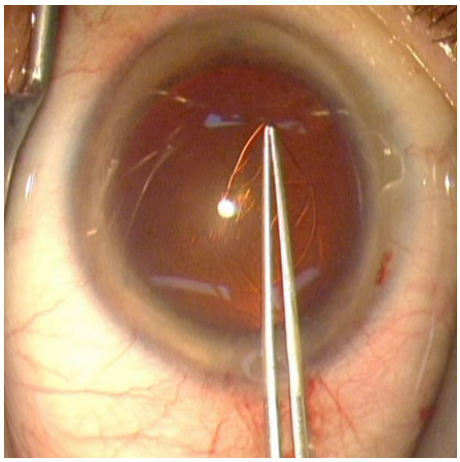


Abbildung 5.1: Öffnen des Kapselsacks mit einer Pinzette

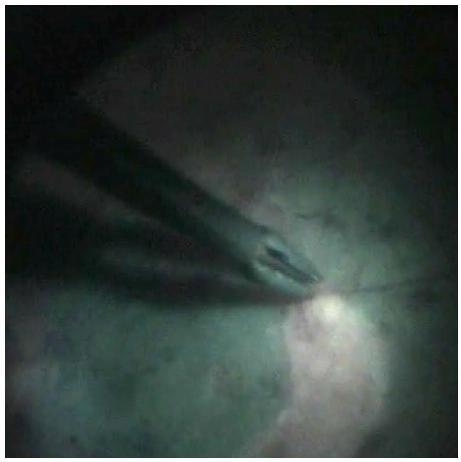


Abbildung 5.2: Ein Teil der ILM wird entfernt

für nachfolgende Behandlungen freizulegen. Abb. 5.2 zeigt einen solchen Eingriff an einer vom Chirurgen grün eingefärbten ILM. Das Ablösen einer Membran vom umliegenden Gewebe wird meist dadurch erschwert, dass diese leicht fragmentiert. Die Herausforderung besteht also darin, so vorzugehen, dass sich die Membran in möglichst großen Stücken entfernen lässt.

5.2 Anforderungen an den Algorithmus

Der vorliegenden Arbeit liegt (unter anderem) die Motivation zugrunde, einen Reiß-Algorithmus zu entwickeln, der in verschiedenen Modulen von EYESi zur Anwendung kommen soll. Das Kapsulorhexis-Modul, welches die Öffnung des Kapselsacks während einer Katarakt-Operation trainiert, stellt die höchsten Anforderungen an einen Reiß-Algorithmus. Denn hier fokussiert der Arzt permanent auf die Spitze eines einzelnen Risses. In einem solchen Szenario sollte ein Reiß-Algorithmus nachfolgende Anforderungen erfüllen:

1. Der Reiß-Algorithmus muss für den Einsatz in einer Echtzeitanwendung geeignet sein.

2. Für eine realistische Simulation ist es notwendig, die Rissfortpflanzung physikalisch, d. h. auf Basis von Spannungen oder Verzerrungen zu berechnen. Es existieren Ansätze wie z. B. der von Neff et al. [87, 88], die eine Rissausbreitung ausschließlich mit Hilfe eines deskriptiven Modells animieren. Hierbei besteht jedoch die Gefahr, dass durch Belastung entstandene Spannungen beim Reißen nicht oder nicht ausreichend abgebaut werden. Beim Zug an einem Membran-Stück würde sich dies in einer unrealistischen Dehnung der Membran bemerkbar machen.

3. Der Arzt soll einen Riss in der Größenordnung von Millimeterbruchteilen kontrolliert fortsetzen können. Daher darf es keine ruckartigen Wechsel zwischen Spannungsaufbau und Rissausbreitung geben und die Rissausbreitung muss in eine beliebige Richtung möglich sein.

4. Die entstehende Risskante soll glatt sein und keine topologischen Artefakte, wie z. B. Zacken oder Löcher, zeigen. Ein herausgerissenen Membran-Stück muss sich wie ein Puzzleteil wieder in die zurückgebliebene Membran einfügen lassen.

5. Um einen Riss darzustellen, ist es grundsätzlich notwendig, Veränderungen an der Mesh-Topologie vorzunehmen. Wenn eine Membran sehr stark fragmentiert wird und infolge der topologischen Adaptionen zu viele neue Mesh-Elemente entstehen, kann dies zu Rechenzeitproblemen führen. Der Al-

gorithmus soll daher die Anzahl neuer Elemente entlang der Risskante möglichst klein halten.

6. Die topologischen Änderungen entlang der Risskante dürfen die in Abschnitt 4.1 festgelegten Konsistenzbedingungen nicht verletzen.

5.3 Vergleichbare Arbeiten

Möchte man die Publikationen zum Thema Reißen in verschiedene Kategorien einsortieren, kann man z. B. der Einteilung von Tabiei und Wu [114] folgen. Diese berücksichtigt, wie ein Rissfortschritt topologisch realisiert wird: Die einfachste Methode, ein Objekt zu fragmentieren, besteht darin, Simulationselemente komplett aus der Mesh-Struktur zu entfernen. Am anspruchsvollsten sind Verfahren, welche die Mesh-Elemente kontinuierlich adaptieren und verfeinern, um den erwünschten Rissverlauf zu approximieren. Dazwischen sind diejenigen anzuordnen, die sich im Rahmen der topologischen Anpassungen darauf beschränken, Mesh-Elemente entlang ihrer Grenzen zu trennen.
Im Folgenden wird eine Reihe von gitterbasierten Verfahren in diese drei Kategorien eingeteilt. Auf die Diskussion gitterloser Ansätze wird an dieser Stelle verzichtet – der interessierte Leser sei z. B. auf die Arbeiten von Wicke et al. [132] und Pauly et al. [96] verwiesen. Abschließend beschreibt ein separater Abschnitt, welche Reiß-Algorithmen bisher in EYESi zur Anwendung kommen.

5.3.1 Reißen durch Löschen von Elementen

Das Löschen zugbelasteter Mesh-Elemente stellt einen Reiß-Ansatz dar, der vorzugsweise in frühen Arbeiten zu finden ist, die zum Forschungsthema „Simulation deformierbarer Objekte“ publiziert wurden.

Bereits 1988 beschreiben Terzopoulos et al. [117, 116] die Simulation von Brüchen in ihrer wegweisenden Arbeit über die Modellierung von Deformationen. Ein Bruch entsteht dort, wo der Abstand benachbarter Knoten einen Grenzwert überschreitet. Die Verbindung zwischen diesen Knoten wird nicht mehr gerendert und die dazugehörigen Elastizitätsparameter werden auf Null gesetzt. Die betroffenen Elemente werden zwar nicht explizit aus der Simulationsstruktur entfernt, das Anpassen der Modellparameter führt jedoch im Rahmen der Deformationsberechnung zu einem äquivalenten Ergebnis.
Ähnlich gehen Norton et al. [93] vor, die ein volumetrisches Objekt als kubisches Feder-Masse-System animieren und dieses unter dem Einfluss großer Verzerrungen zerbrechen lassen. Sobald innerhalb eines Kubus die Dehnung einer Feder

einen Grenzwert überschreitet, wird dieser als „gebrochen“ markiert. Um topologische Inkonsistenzen zu verhindern, müssen gebrochene Kuben vollständig aus dem Modell entfernt werden.

Das Löschen von Mesh-Elementen führt zu einem kontinuierlichen Materialverlust entlang der Risskante und zu einer sprunghaften Rissausbreitung, wobei die Größe eines „Sprungs“ von den Abmessungen des gelöschten Elements abhängt. Dieser Ansatz kann die in Abschnitt 5.2 formulierten Anforderungen 3 und 4 nicht erfüllen.

5.3.2 Reißen entlang von Elementgrenzen

Soll ein simuliertes Objekt fragmentieren, ohne dass Fläche bzw. Volumen verloren geht, ist es topologisch am einfachsten, die Risskante entlang von Elementgrenzen zu definieren.

Mazarak et al. [73] simulieren starre Objekte, beispielsweise Gebäude aus Beton, die unter dem Einfluss einer Druckwelle in sich zusammenfallen. Das Modell basiert auf einem Voxel-Gitter, wobei die Seiten benachbarter Voxel über starre Federn verbunden sind. Aus der mathematischen Beschreibung der Druckwelle wird die Belastung der Verbindungselemente berechnet. Überschreitet die Belastung einen Grenzwert, wird die Verbindung gelöst.
Smith et al. [110, 111] simulieren spröde zerbrechende Materialien, denen ein Tetraeder-Mesh zugrunde liegt. Zwei Tetraeder, die über eine gemeinsame Seite verfügen, sind über ein Linienelement verbunden, dessen Länge einer Zwangsbedingung unterliegt. In Abhängigkeit der Kraft, die notwendig ist, um eine Zwangsbedingung aufrecht zu erhalten, wird entschieden, ob die zugehörige Verbindung getrennt wird.
Boux de Casson und Laugier [28] benutzen ein auf Dreiecken basierendes Feder-Masse-Gitter, um deformierbare Oberflächen wie z. B. eine Fahne entlang der Dreieckskanten zu zerreißen. Das momentane Ende eines Risses stimmt daher immer mit einem Mesh-Knoten überein. An diesem Rissknoten pflanzt sich der Riss fort, wenn eine am Rissknoten hängende Feder eine vorgegebene Dehnungsgrenze überschreitet. Aufgetrennt wird das Mesh dann entlang derjenigen Feder am Rissknoten, welche am ehesten senkrecht zur überdehnten Feder steht.
Grimm [57] simuliert deformierbare Oberflächen ebenfalls mit Hilfe eines dreiecksbasierten Feder-Masse-Systems. Das Problem, einen Riss topologisch zu realisieren, ist wie in [28] gelöst. Die Ansätze unterscheiden sich jedoch in der Definition von Bruchkriterien: Um zu entscheiden, ob und wo das Mesh reißt, berechnet Grimm [57] für jeden Mesh-Knoten einen Spannungswert, der sich aus den Spannungen direkt anhängender Federn ergibt. Ein Knoten reißt, wenn seine Spannung

eine Schwelle überschreitet, deren Höhe von der Position des Knotens im Mesh abhängt. Der Ansatz unterscheidet zwischen inneren Knoten, Randknoten und solchen, die eine Rissspitze darstellen. Die Schwellwerte innerer Knoten sind am höchsten, die Schwellwerte von Rissspitzen am niedrigsten. Ist die Spannung eines Knotens ausreichend hoch, wird diejenige Feder am Knoten aufgetrennt, die über die geringste relative Länge verfügt.
Müller et al. [79] lassen starre Materialien brechen, die sie zwischen zwei Kollisionsereignissen als undeformierbare Festkörper simulieren. Während einer Kollision dagegen berechnet eine statische, nicht-lineare Finite Elemente Analyse Spannungstensoren, mit denen sich eine lokal definierte Bruchebene bestimmen lässt. (Als Bruchkriterium dient das *Kriterium der maximalen Hauptspannung*, auf das Abschnitt 5.4 eingeht.) Tetraeder, welche die Bruchebene schneiden, werden entlang ihrer Grenzen getrennt. In [80] erweitern Müller et al. diese Arbeit, um komplexe Geometrien zu simulieren und zu fragmentieren, die in einem regelmäßigen Gitter aus Kuben eingebettet sind.

Im Gegensatz zum „Reißen durch Löschen von Elementen“ führt das „Reißen entlang von Elementgrenzen“ zu keinen topologischen Artefakten. Auch dieser Ansatz allerdings kann Anforderung 3 nicht erfüllen: Die Abmessungen der Elementgrenzen bestimmen die Ausmaße einer Rissausbreitung und die Orientierung der Elementgrenzen bestimmt die möglichen Rissrichtungen.
Die von Grimm [57] vorgeschlagenen Bruchkriterien jedoch lassen sich auf einen Algorithmus übertragen, der unabhängig von Elementgrenzen reißt. Abschnitt 5.4 wird daher noch einmal auf diesen Ansatz eingehen.

5.3.3 Reißen unabhängig von Elementgrenzen

Die folgenden Arbeiten haben gemeinsam, Ort und Richtung eines Risses zu definieren, indem das *Kriterium der maximalen Hauptspannung*, das *Kriterium der maximalen Hauptverzerrung* oder eine Modifikation dieser Kriterien angewendet wird. Da Abschnitt 5.4 auf diese Thematik eingeht, wird an dieser Stelle auf Erläuterungen verzichtet.

1999 veröffentlichten O’Brien und Hodgins [95] ihre wegweisende Arbeit über die Animation dreidimensionaler Brüche in spröden Materialien auf Basis eines linearen FEM-Modells. Für die Auswertung der Spannungsverteilung im Mesh schlagen O’Brien und Hodgins einen speziellen Tensor vor, den sie *Separation Tensor* nennen. Dieser liefert einen Rissknoten und eine Rissebene, welche den Rissknoten enthält. Alle am Rissknoten hängenden Tetraeder, welche die Rissebene schneiden, werden exakt entlang der Ebene in feinere Tetraeder zerlegt. Dieser Ansatz wurde von O’Brien et al. [94] für das Reißen duktiler Materialien erweitert

und von Yngve et al. [135] für die Animation von Explosionen eingesetzt.
Molino et al. [81, 82] führen ein Verfahren namens *Virtueller-Knoten-Algorithmus* ein[1], das Tetraeder-Gitter entlang stückweise linearer Pfade zerlegt. Anstatt ein gebrochenes Simulationselement zu unterteilen, wird dieses je nach Komplexität des Bruchs ein- oder mehrmals dupliziert. Nur die grafische Repräsentation des Materials wird fragmentiert und auf die ursprünglichen und neuen Elemente verteilt. Ein Tetraeder des initialen Meshs kann allerdings nur in maximal vier Bruchstücke zerfallen. Losasso et al. [72] z. B. greifen auf diesen Algorithmus zurück, um schmelzende und verbrennende Feststoffe zu simulieren.
Ähnlich wie Müller et al. [79] realisieren Bao et al. [5] das Brechen steifer Objekte durch einen Wechsel zwischen Festkörper-Simulation und Finiter Elemente Analyse. Die topologischen Veränderungen an der Bruchstelle führt der Virtuelle-Knoten-Algorithmus von Molino et al. durch. Um realistischere Rissmuster zu erzeugen, schlagen Bao et al. ein modifiziertes Kriterium der maximalen Hauptspannung vor.
Die Publikation von Gingold et al. [53] aus dem Jahr 2004 ist (nach Angabe der Autoren) die erste, die einen Algorithmus vorschlägt, der triangulierte Oberflächen (z. B. Glühbirnen) unabhängig von Dreieckskanten zerbricht. Das Kriterium der maximalen Hauptverzerrung liefert die Bruchrichtung. Ein durch drei Punkte definierter Polygonzug bestimmt eine atomare Rissausbreitung. Anfangs- und Endpunkt liegen auf existierenden Dreiecksknoten, der mittlere Punkt liegt auf einer existierenden Dreieckskante. Stimmt ein Liniensegment des Polygonzugs annähernd mit einer Dreieckskante überein, wird die Mesh-Topologie adaptiert und die Kante auf das Segment verschoben. Dadurch reduziert sich die Anzahl neu zu erzeugender Dreiecke.

In den genannten Ansätzen sind die Rissrichtungen unabhängig von Elementgrenzen. Allerdings endet ein Riss niemals im Inneren eines vorab existierenden Elements, sondern breitet sich von Elementrand zu Elementrand aus. Die Rissausbreitung ist demnach nicht vollständig unabhängig von Elementgrenzen und Anforderung 3 aus Abschnitt 5.2 nur teilweise erfüllt. Keiner der Algorithmen wurde in einer Echtzeitanwendung getestet (Anforderung 1) und Ansatz [53] ist der einzige, der die Anzahl neu generierter Elemente zu minimieren versucht (Anforderung 5). Die Suche nach einem geeigneten topologischen Verfahren wird daher in Abschnitt 5.5 auf Publikationen über Schneide-Algorithmen ausgedehnt. Auf die Berechnung von Bruchrichtungen in obigen Arbeiten jedoch wird Abschnitt 5.4 noch einmal eingehen.

[1] Wie in Areias et al. [4] und Bao et al. [5] nachzulesen, kann der Virtuelle-Knoten-Algorithmus als Verallgemeinerung der *Extended Finite Element Method* (XFEM) betrachtet werden, welche in Daux et al. [27] beschrieben wird.

5.3.4 Bisherige Reiß-Algorithmen in EYESi

Schon seit längerem existieren für den Operationssimulator EYESi Trainingsmodule, in denen als Feder-Masse-System modelliertes Gewebe reißen kann:

In einem der ersten Module, die für EYESi entwickelt wurden, muss eine auf der Netzhaut wuchernde Membran aus dem Auge entfernt werden (siehe auch Abschnitt 7.3). Überschreitet die Dehnung einer Feder einen Grenzwert, wird diese als „gebrochen“ markiert. Anschließend werden die Feder und die Dreiecke, welche diese Feder enthalten, im Simulationsgitter gelöscht.
Das ILM-Modul (siehe auch Abschnitt 7.2) erfordert, dass ein Teilstück der *Inneren Grenzmembran* entfernt wird. Der eingesetzte Reiß-Algorithmus ist der von Grimm [57], welcher das Dreiecks-Mesh entlang der Federn auftrennt und dabei die Ausgangsfläche der Membran erhält.
Inzwischen wurde der Ansatz aus [57] von Grimm auf das Reißen dreidimensionaler Objekte erweitert. Mit diesem Algorithmus wird eine als Tetraeder-Mesh modellierte Linse fragmentiert, um sie anschließend in kleinen Stücken aus dem Auge entfernen zu können. (Diese Arbeit wurde bisher noch nicht veröffentlicht.)

Die Nachteile der eingesetzten Methoden wurden bereits erläutert. Um für virtuelle Membrane eine realistischere Rissausbreitung zu ermöglichen, werden im Folgenden zwei neue Reiß-Algorithmen vorgeschlagen.

5.4 Bestimmung der Rissausbreitung

Dieser Abschnitt definiert Bruchkriterien, die bestimmen, wann bzw. wo ein Dreiecks-Mesh reißt und wohin sich der Riss ausbreitet. Gesucht ist ein Algorithmus, der eine Rissrichtung unabhängig von Elementgrenzen berechnet. Die einzigen Verfahren aus Abschnitt 5.3, die diese Anforderung erfüllen, sind diejenigen, welche das *Kriterium der maximalen Hauptspannung* oder das *Kriterium der maximalen Hauptverzerrung* anwenden. Um diese Kriterien erläutern zu können, folgt zunächst ein kurzer Einblick in die Kontinuumsmechanik.

5.4.1 Einblick in die Kontinuumsmechanik

Die Kontinuumsmechanik[2] beschreibt die Mechanik deformierbarer Medien. Der mikroskopische Aufbau der Materie wird vernachlässigt und das Objekt der Betrachtung als Kontinuum aufgefasst. Auf dieser Grundlage werden Themen wie Elastizität, Plastizität, Viskoelastizität, Materialermüdung und -versagen behan-

[2]Fachliteratur bieten z. B. Chung [24] oder Bonet und Wood [13].

delt. Die Elastizitätstheorie beruht auf dem verallgemeinerten Hooke'schen Gesetz, das einen linearen, tensoriellen Zusammenhang zwischen einer Verspannung und der daraus resultierenden Verzerrung des Kontinuums herstellt.

5.4.1.1 Beschreibung von Verspannungen

Gegeben sei ein volumetrischer Körper, der einer äußeren Belastung ausgesetzt ist. Nun soll der Spannungszustand in einem beliebigen Punkt $\vec{p}$ des Körpers beschrieben werden. Dazu stelle man sich eine Schnittfläche A durch $\vec{p}$ mit beliebiger Orientierung vor. Auf ein Flächenelement ΔA, welches $\vec{p}$ enthält, wirkt eine Schnittkraft $\Delta\vec{f}$. Der Quotient $\Delta\vec{f}/\Delta A$ entspricht der mittleren Spannung für das Flächenelement und Grenzwertbildung führt zum *Spannungsvektor* $\vec{t}$:

$$\vec{t} = \lim_{\Delta A \to 0} \frac{\Delta \vec{f}}{\Delta A} = \frac{\mathrm{d}\vec{f}}{\mathrm{d}A}$$

Die Komponente von $\vec{t}$ normal zur Schnittfläche heißt *Normalspannung* σ, die Tangentialkomponente heißt *Schubspannung* τ und zerfällt in der Schnittebene wiederum in zwei Komponenten. Man kann zeigen, dass drei Spannungsvektoren, die zu drei senkrecht aufeinander stehenden Schnittflächen gehören, den Spannungszustand in $\vec{p}$ vollständig beschreiben. Diese Vektoren liefern drei Normalspannungen $(\sigma_x, \sigma_y, \sigma_z)$ sowie sechs Schubspannungen $(\tau_{xy}, \tau_{xz}, \tau_{yx}, \tau_{yz}, \tau_{zx}, \tau_{zy})$, für die allerdings gilt: $\tau_{xy} = \tau_{yx}$, $\tau_{xz} = \tau_{zx}$ und $\tau_{yz} = \tau_{zy}$. Angeordnet in einer symmetrischen Matrix definieren diese Komponenten den Spannungstensor S:

$$S = \begin{pmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_z \end{pmatrix}$$

5.4.1.2 Beschreibung von Verzerrungen

Betrachtet wird ein Körper, der einer Belastung ausgesetzt ist und sich unter dieser verformt. Gegeben sei eine Funktion $\vec{d}(\vec{p}) = (u(\vec{p}), v(\vec{p}), w(\vec{p}))$, die jedem Materialpunkt $\vec{p} = (x, y, z)$ einen Verschiebungsvektor zuordnet. Dieser Verschiebungsvektor $\vec{d}(\vec{p})$ entspricht der durch die Deformation verursachten Materialpunkt-Verrückung. Nun stelle man sich einen infinitesimalen Quader vor, der einen beliebigen aber festen Punkt $\vec{p}$ enthält und entlang der Koordinatenachsen ausgerichtet ist. Um den Verzerrungszustand in $\vec{p}$ zu messen, untersucht man, wie sich unter der Verschiebungsfunktion $\vec{d}(\vec{p})$ Kantenlängen und Winkel des Quaders ändern. Gesucht sind Längenverzerrungen ϵ_x, ϵ_y, ϵ_z und Winkelverzerrungen γ_{xy}, γ_{xz}, γ_{yz}, welche durch die partiellen Ableitungen der Verschiebungsfunktion nach

den Raumkoordinaten beschrieben werden können. Geht man vereinfachend davon aus, dass der Körper nur kleinen Deformationen unterworfen ist, ergibt sich:

$$\epsilon_x = \frac{\delta u}{\delta x} \quad \epsilon_y = \frac{\delta v}{\delta y} \quad \epsilon_z = \frac{\delta w}{\delta z}$$

$$\gamma_{xy} = \frac{\delta u}{\delta y} + \frac{\delta v}{\delta x} \quad \gamma_{xz} = \frac{\delta u}{\delta z} + \frac{\delta w}{\delta x} \quad \gamma_{yz} = \frac{\delta v}{\delta z} + \frac{\delta w}{\delta y}$$

Die Dehnungen sowie die halben Winkeländerungen bilden die Komponenten des symmetrischen Verzerrungstensors V:

$$V = \begin{pmatrix} \epsilon_x & \frac{1}{2}\gamma_{xy} & \frac{1}{2}\gamma_{xz} \\ \frac{1}{2}\gamma_{xy} & \epsilon_y & \frac{1}{2}\gamma_{yz} \\ \frac{1}{2}\gamma_{xz} & \frac{1}{2}\gamma_{yz} & \epsilon_z \end{pmatrix}$$

5.4.1.3 Elastizitätsgesetz

Das Elastizitätsgesetz ist eine Verallgemeinerung des Hooke'schen Gesetzes und stellt einen linearen Zusammenhang zwischen Verspannung und Verzerrung her. Hier beschränken wir uns auf Materialien, die sowohl *homogen* als auch *isotrop* sind. Ein homogenes Material hat an jeder Stelle die gleichen Eigenschaften, ein isotropes Material hat in jede Richtung die gleichen Eigenschaften. Für isotrope Medien reduziert sich die Anzahl unabhängiger Konstanten, welche die Materialeigenschaften beschreiben, von einundzwanzig auf zwei. Diese heißen *Elastizitätsmodul* E und *Querkontraktionszahl* ν. Das Elastizitätsgesetz verknüpft die Komponenten des Verzerrungs- und des Spannungstensors folgendermaßen:

$$\epsilon_x = \frac{1}{E}\left[\sigma_x - \nu\left(\sigma_y + \sigma_z\right)\right] \quad \epsilon_y = \frac{1}{E}\left[\sigma_y - \nu\left(\sigma_z + \sigma_x\right)\right] \quad \epsilon_z = \frac{1}{E}\left[\sigma_z - \nu\left(\sigma_x + \sigma_y\right)\right]$$

$$\gamma_{xy} = \frac{2(1+\nu)}{E}\tau_{xy} \qquad \gamma_{xz} = \frac{2(1+\nu)}{E}\tau_{xz} \qquad \gamma_{yz} = \frac{2(1+\nu)}{E}\tau_{yz}$$

Dieser lineare Zusammenhang gilt im Allgemeinen nur für kleine Belastungen. Überschreiten die Spannungen die *Proportionalitätsgrenze*, wachsen die Verzerrungen überproportional. Ein Material, das dem Hooke'schen Gesetz genügt, wird als *linear-elastisch* bzw. *elastisch* bezeichnet.

5.4.2 Richtung der Rissausbreitung

Mit Hilfe der eingeführten Größen für Verspannung und Verzerrung werden im Folgenden das *Kriterium der maximalen Hauptspannung* und das *Kriterium der maximalen Hauptverzerrung* erläutert. Denn mit diesen Kriterien soll die Berechnung einer Rissrichtung für dreiecksbasierte Feder-Masse-Gitter motiviert werden.

5.4.2.1 Motivation

Die Komponenten des Spannungstensors S entsprechen Normal- und Schubspannungen, die in drei senkrecht aufeinander stehenden Schnittflächen wirken. Prinzipiell gilt, dass ein infinitesimaler Quader durch positive Normalspannungen auf Zug und durch negative Normalspannungen auf Druck beansprucht wird. Man kann zeigen, dass bei geeigneter Orientierung der Schnittflächen alle Schubspannungen verschwinden und die Normalspannungen ihre Extremalwerte annehmen. Die zugehörigen Schnittrichtungen werden *Hauptrichtungen* genannt, die extremalen Normalspannungen heißen *Hauptspannungen*. Das Kriterium der maximalen Hauptspannung – auch als *Rankine-Kriterium*[3] bekannt – besagt, dass ein Materialversagen dort auftritt, wo die maximale Hauptspannung einen materialabhängigen Grenzwert überschreitet. Die Flächennormale der lokal definierten Bruchebene ist durch die zur maximalen Hauptspannung gehörenden Hauptrichtung gegeben. Übertragen auf ein zweidimensionales Problem bedeutet dies, dass sich der Riss in Richtung der minimalen Hauptspannung ausbreitet.
Entsprechend zum Spannungstensor verfügt auch der Verzerrungstensor über drei senkrecht aufeinander stehende Hauptrichtungen, für die alle Winkelverzerrungen verschwinden und die Längenverzerrungen ihre Extremalwerte annehmen. Das Kriterium der maximalen Hauptverzerrung ist vollkommen analog zum Kriterium der maximalen Hauptspannung formuliert.
Im Allgemeinen stimmen Hauptverzerrungsrichtungen und Hauptspannungsrichtungen nicht überein. Für ein isotropes, elastisches Material jedoch gilt, dass die Hauptrichtungen des Verzerrungstensors mit denen des Spannungstensors zusammenfallen.[4]

Die Anwendung der beschriebenen Kriterien setzt voraus, dass einem simulierten Körper ein Modell zugrunde liegt, das die Berechnung von Spannungs- und Verzerrungstensoren unterstützt. FEM-Modelle erfüllen diese Bedingung, Feder-Masse-Modelle dagegen nicht. Allerdings ist den Kriterien eine einfache Heuristik zu entnehmen, die auch auf Feder-Masse-Modelle übertragbar ist: Wenn ein (zweidimensionales) Objekt reißt, dann in die Richtung minimaler (negativster) Verzerrung oder Spannung.

5.4.2.2 Umsetzung

Ausgangspunkt der Überlegungen ist ein Riss wie in Abb. 5.3 zu sehen, dessen Ende durch einen einzelnen Knoten, den *Risskeim*, markiert ist. Die Dreiecke, welche direkt am Risskeim hängen, bilden die *Rissregion*. Die gesuchte Rissrichtung

[3] siehe Rankine [103]
[4] siehe z. B. Schnell et al. [105], Seite 63

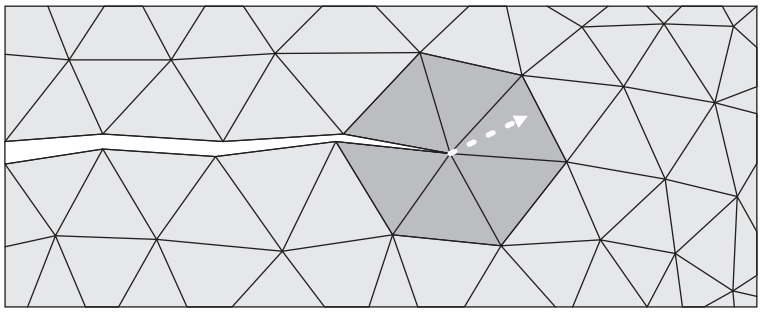


Abbildung 5.3: Gesucht ist eine Rissrichtung ausgehend vom Risskeim

(in Abb. 5.3 dargestellt durch einen weißen Pfeil) liegt in einem dieser Dreiecke und verweist vom Risskeim auf einen Punkt der gegenüberliegenden Dreieckskante. Im Folgenden wird ein Dreieck der Rissregion mit Eckpunkten $\vec{a}$, $\vec{b}$ und $\vec{c}$ betrachtet, wobei $\vec{a}$ der Risskeim sei. Abb. 5.4 zeigt ein solches Dreieck sowohl unverzerrt (angedeutet durch Knoten-Indizes „u“) als auch verzerrt (angedeutet durch Knoten-Indizes „v“). Die Verzerrung entspreche der durch die Simulation verursachten, aktuellen Deformation. Translation und Rotation der beiden Dreiecke sind hier nicht von Bedeutung. Der Punkt $\vec{d}$ auf der Dreieckskante $\vec{c} - \vec{b}$ ist sowohl im unverzerrten als auch im verzerrten Dreieck durch eine baryzentrische Koordinate $t \in [0, 1]$ eindeutig definiert:

$$\begin{aligned} \vec{d}_\mathrm{u} &= (1-t)\,\vec{b}_\mathrm{u} + t\,\vec{c}_\mathrm{u} &&= \vec{b}_\mathrm{u} + t\,(\vec{c}_\mathrm{u} - \vec{b}_\mathrm{u}) \\ \vec{d}_\mathrm{v} &= (1-t)\,\vec{b}_\mathrm{v} + t\,\vec{c}_\mathrm{v} &&= \vec{b}_\mathrm{v} + t\,(\vec{c}_\mathrm{v} - \vec{b}_\mathrm{v}) \end{aligned}$$

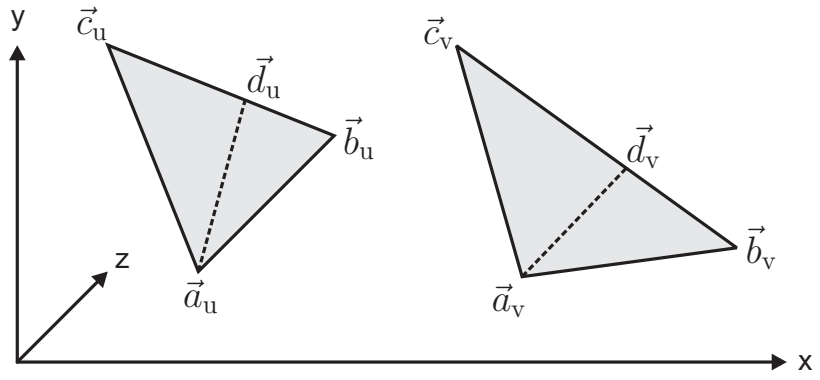


Abbildung 5.4: Unverzerrter und verzerrter Zustand eines Dreiecks

In Anlehnung an das Kriterium der maximalen Hauptverzerrung ist diejenige baryzentrische Koordinate $t_{\min}$ gesucht, für welche die Dehnung zwischen Kantenpunkt $\vec{d}$ und Risskeim $\vec{a}$ minimal wird. Gesucht ist also das Minimum der Funktion $f(t)$, die jedem $t \in [0,1]$ die relative Länge des entsprechenden Liniensegments zuordnet:

$$f(t_{\min}) \;=\; \frac{|\vec{b}_{\mathrm{v}} + t_{\min}\,(\vec{c}_{\mathrm{v}} - \vec{b}_{\mathrm{v}}) - \vec{a}_{\mathrm{v}}|}{|\vec{b}_{\mathrm{u}} + t_{\min}\,(\vec{c}_{\mathrm{u}} - \vec{b}_{\mathrm{u}}) - \vec{a}_{\mathrm{u}}|} \;=\; \min\left\{\frac{|\vec{d}_{\mathrm{v}} - \vec{a}_{\mathrm{v}}|}{|\vec{d}_{\mathrm{u}} - \vec{a}_{\mathrm{u}}|}\right\}_{t\in[0,1]}$$

Das Minimum existiert und ist folgendermaßen beschränkt:

$$\frac{\min\left\{|\vec{d}_{\mathrm{v}} - \vec{a}_{\mathrm{v}}|\right\}_{t\in[0,1]}}{\max\left\{|\vec{d}_{\mathrm{u}} - \vec{a}_{\mathrm{u}}|\right\}_{t\in[0,1]}} \le \min\left\{\frac{|\vec{d}_{\mathrm{v}} - \vec{a}_{\mathrm{v}}|}{|\vec{d}_{\mathrm{u}} - \vec{a}_{\mathrm{u}}|}\right\}_{t\in[0,1]} \le \frac{\max\left\{|\vec{d}_{\mathrm{v}} - \vec{a}_{\mathrm{v}}|\right\}_{t\in[0,1]}}{\min\left\{|\vec{d}_{\mathrm{u}} - \vec{a}_{\mathrm{u}}|\right\}_{t\in[0,1]}}$$

Sei nun

$$g(t) = f^2(t) = \frac{v(t)}{u(t)} = \frac{|\vec{b}_{\mathrm{v}} + t\,(\vec{c}_{\mathrm{v}} - \vec{b}_{\mathrm{v}}) - \vec{a}_{\mathrm{v}}|^2}{|\vec{b}_{\mathrm{u}} + t\,(\vec{c}_{\mathrm{u}} - \vec{b}_{\mathrm{u}}) - \vec{a}_{\mathrm{u}}|^2} \quad \forall t \in \mathbb{R}$$

$g(t)$ nimmt seine Extremalwerte an denselben Stellen an wie $f(t)$. Um eine Kurvendiskussion zu führen und Kandidaten für die Minimumstelle zu finden, müssen die Nullstellen der Ableitung $g'(t)$ gefunden werden:

$$g'(t) \;=\; \left(\frac{v(t)}{u(t)}\right)' \;=\; \frac{v'(t)u(t) - v(t)u'(t)}{u^2(t)} \;\stackrel{!}{=}\; 0$$

Multipliziert man $v'u - vu'$ aus, lassen sich Terme dritter Ordnung kürzen und für die Nullstellen t_i von g' ergibt sich folgende quadratische Gleichung (in der ein Ausdruck der Form $\vec{v}_1\vec{v}_2$ bzw. $\vec{v}^2$ für das Standardskalarprodukt $\langle\vec{v}_1, \vec{v}_2\rangle$ bzw. $\langle\vec{v}, \vec{v}\rangle$ steht):

$$\begin{aligned} 0 \;\stackrel{!}{=}\; & \left[(\vec{b}_{\mathrm{u}} - \vec{a}_{\mathrm{u}})(\vec{c}_{\mathrm{u}} - \vec{b}_{\mathrm{u}}) \cdot (\vec{c}_{\mathrm{v}} - \vec{b}_{\mathrm{v}})^2 - (\vec{b}_{\mathrm{v}} - \vec{a}_{\mathrm{v}})(\vec{c}_{\mathrm{v}} - \vec{b}_{\mathrm{v}}) \cdot (\vec{c}_{\mathrm{u}} - \vec{b}_{\mathrm{u}})^2\right] \cdot t^2 \\ & + \left[(\vec{c}_{\mathrm{v}} - \vec{b}_{\mathrm{v}})^2 \cdot (\vec{b}_{\mathrm{u}} - \vec{a}_{\mathrm{u}})^2 - (\vec{c}_{\mathrm{u}} - \vec{b}_{\mathrm{u}})^2 \cdot (\vec{b}_{\mathrm{v}} - \vec{a}_{\mathrm{v}})^2\right] \cdot t \\ & + \left[(\vec{b}_{\mathrm{v}} - \vec{a}_{\mathrm{v}})(\vec{c}_{\mathrm{v}} - \vec{b}_{\mathrm{v}}) \cdot (\vec{b}_{\mathrm{u}} - \vec{a}_{\mathrm{u}})^2 - (\vec{b}_{\mathrm{u}} - \vec{a}_{\mathrm{u}})(\vec{c}_{\mathrm{u}} - \vec{b}_{\mathrm{u}}) \cdot (\vec{b}_{\mathrm{v}} - \vec{a}_{\mathrm{v}})^2\right] \end{aligned}$$

Um $t_{\min}$ zu identifizieren, werden die Werte $f(t_i)$ sowohl untereinander als auch mit den Randwerten $f(0)$ und $f(1)$ verglichen. Eine Nullstelle t_i mit $t_i \notin [0,1]$ wird verworfen.
Ist $t_{\min}$ für alle Dreiecke der Rissregion bestimmt, wird dasjenige als *Rissdreieck* gewählt, in welchem die geringste Dehnung vorhanden ist. Als Ergebnis liefert die Berechnung der Rissrichtung demnach ein Dreieck der Rissregion mit zugehöriger baryzentrischer Koordinate t.

5.4.3 Ort der Rissausbreitung

Vor Berechnung einer Rissrichtung muss zunächst ermittelt werden, ob ein Knoten im Mesh vorhanden ist, an dem ein neuer Riss entstehen oder sich ein existierender Riss fortsetzen soll. Zu diesem Zweck wird jedem Knoten k ein Spannungswert $s(k)$ zugeordnet, der von den Verzerrungen aller an k hängenden Federn f_i abhängt:

$$s(k) = \sum_i \frac{c_i}{2} |l(f_i) - l_0(f_i)|$$

Hierbei ist $l_0(f_i)$ die Federnulllänge von f_i, $l(f_i)$ deren aktuelle Länge und c_i deren Federkonstante.
Wie in Grimm [57] vorgeschlagen und in Abb. 5.5 dargestellt, wird zwischen drei verschiedenen Knoten-Typen unterschieden. Ein *innerer Knoten* (weiß) zeichnet sich dadurch aus, dass jede an ihm hängende Feder die gemeinsame Seite zweier Dreiecke definiert. Ein *Randknoten* (schwarz und grau) dagegen verfügt über genau zwei Federn, die jeweils zu nur einem Dreieck gehören. Der grau unterlegte *Risskeim* ist ein besonderer Randknoten: Abb. 5.5 zeigt das Mesh in einem deformierten Zustand. Die Pfeile deuten an, dass es zu jedem schwarzen Knoten entlang der Risskante einen korrespondierenden schwarzen Knoten gibt, der über dieselbe Ruheposition verfügt. Entspannt sich das Mesh im Rahmen der Simulation, legen sich korrespondierende Randknoten aufeinander und schließen den Riss. Ein Risskeim verfügt über keinen korrespondierenden Knoten. (Die in Abschnitt 5.5.2 und 5.5.3 beschriebenen Ansätze werden diese Topologie entlang der Risskante aufrecht erhalten.)

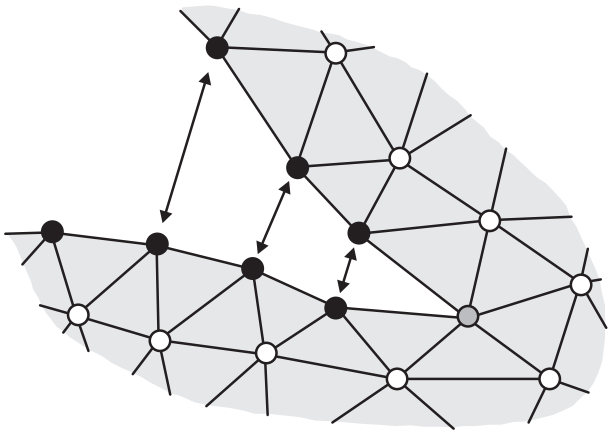


Abbildung 5.5: Innere Knoten (weiß), Randknoten (schwarz und grau), Risskeim (grau)

In Abhängigkeit des Knoten-Typs ist jedem Mesh-Knoten einer von drei Schwellwerten zugeordnet:

$$minInnerStress,\ \ minEdgeStress,\ \ minSproutStress\ \ \in \mathbb{R}^+$$

Diese definieren die Spannung, welche mindestens notwendig ist, damit der jeweilige Knoten reißt. Um das gewünschte Reiß-Verhalten abzubilden, gilt:

$$minSproutStress \ll minEdgeStress \ll minInnerStress$$

Die Suche nach einem *Rissknoten* wird in jedem Simulationsschritt durchgeführt und lokalisiert zu jedem Knoten-Typ denjenigen Knoten, der seinen Schwellwert am stärksten überschreitet. Bei der anschließenden Auswahl (zwischen maximal drei Knoten) wird ein Risskeim immer einem Randknoten und ein Randknoten immer einem inneren Knoten vorgezogen. Überschreitet z. B. mindestens ein Risskeim seinen Schwellwert, werden Randknoten und innere Knoten als Kandidaten verworfen, egal wie hoch deren Spannungen sind.

5.4.4 Länge der Rissausbreitung

In Abschnitt 5.4.2 wurde eine Rissrichtung ausgehend von einem Risskeim berechnet. Ausgehend von einem Randknoten oder einem inneren Knoten kann eine Rissrichtung natürlich vollkommen analog bestimmt werden. Am jeweiligen Rissknoten entsteht dann ein neuer Riss. Die Behandlung neuer Risse wird in Abschnitt 5.5.4 separat beschrieben und benötigt keine Berechnung einer Risslänge. Die in Abschnitt 5.5.2 und 5.5.3 vorgeschlagenen topologischen Verfahren jedoch brauchen die Information, wie weit sich ein Riss ausgehend vom Risskeim ausbreiten soll.

Speziell für Risskeime werden eine minimale Risslänge $minTearLength$ und eine maximale Risslänge $maxTearLength$ global definiert. $maxTearLength$ entspricht größenordnungsmäßig der durchschnittlichen Ruhelänge aller Federn im Mesh. Zusätzlich zu $minSproutStress$ wird ein Grenzwert $maxSproutStress$ festgelegt, auf den eine Knoten-Spannung $sproutStress$ bei Überschreitung herunter gesetzt wird. Die gesuchte Risslänge $tearLength$ wird proportional zur Spannung bestimmt:

$$tearLength \ = \ minTearLength + \lambda(maxTearLength - minTearLength)$$

$$\lambda = \frac{sproutStress - minSproutStress}{maxSproutStress - minSproutStress} \in [0,1]$$

Der Wert von $tearLength$ wird an das topologische Verfahren aus Abschnitt 5.5.2 übergeben. Für den Algorithmus aus Abschnitt 5.5.3 muss $tearLength$ eventuell

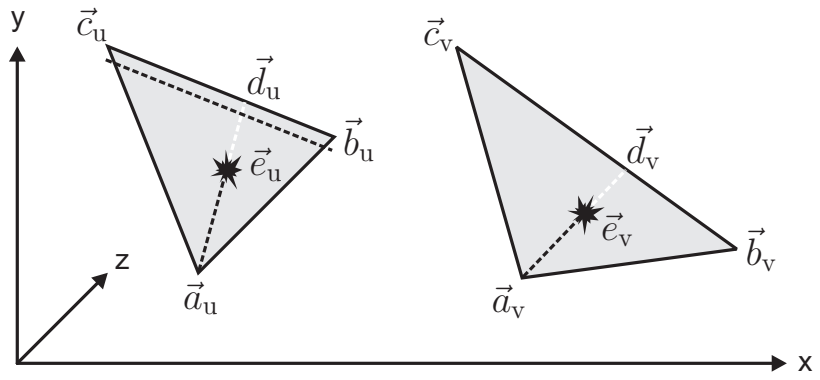


Abbildung 5.6: Neues Rissende innerhalb eines Rissdreiecks

noch einmal verkürzt werden: Betrachten wir ein Rissdreieck (wiederum in unverzerrtem und verzerrtem Zustand) so wie in Abb. 5.6 zu sehen. Um ein neues Rissende $\vec{e}_\mathrm{u}$ innerhalb des Rissdreiecks zu erhalten, muss $tearLength$ nach oben durch $|\vec{d}_\mathrm{u} - \vec{a}_\mathrm{u}|$ beschränkt sein. Für den Ansatz aus Abschnitt 5.5.3 ist wichtig, dass $\vec{e}_\mathrm{u}$ nicht zu nah an der Kante $\vec{c}_\mathrm{u} - \vec{b}_\mathrm{u}$ liegt. Wie durch die gestrichelte, zu $\vec{c}_\mathrm{u} - \vec{b}_\mathrm{u}$ parallele Linie angedeutet, wird $tearLength$ daher durch $|\vec{d}_\mathrm{u} - \vec{a}_\mathrm{u}| - \epsilon$ nach oben begrenzt, mit $0 < \epsilon \ll 1$. Die endgültigen Ergebnisse dieser Berechnung sind somit die baryzentrischen Koordinaten von $\vec{e}_\mathrm{u}$ und damit von $\vec{e}_\mathrm{v}$.

5.5 Topologische Umsetzung eines Risses

Wie in Abschnitt 5.3 angekündigt, werden im Folgenden Arbeiten zum Thema Schneiden diskutiert, um die Suche nach einem geeigneten topologischen Verfahren fortzusetzen. Anschließend werden zwei Algorithmen vorgeschlagen, die das Feder-Masse-Gitter in der Umgebung eines Risskeims an den kontinuierlich wachsenden Riss anpassen. Der letzte Abschnitt beschreibt, wie das Entstehen und Beenden von Rissen topologisch behandelt wird.

5.5.1 Vergleichbare Arbeiten zum Thema Schneiden

Auch im Bereich der Schneide-Algorithmen sind Ansätze zu finden, welche das gewünschte Verhalten modellieren, indem Mesh-Elemente komplett aus dem Simulationsgitter gelöscht werden (Bro-Nielsen [19], Cotin et al. [26, 32]). Standard jedoch sind Methoden, die das Mesh lokal an die Trajektorie der Instrument-

Bewegung anpassen. In Abhängigkeit davon, zu welchem Zeitpunkt die Mesh-Adaptionen stattfinden, wird ein Schneide-Algorithmus als *nicht-progressiv*, *semi-progressiv* oder als *progressiv* bezeichnet: Ein Verfahren wird als nicht-progressiv eingestuft, falls das Instrument zunächst mehrere Mesh-Elemente durchquert, bevor ein Schnitt topologisch dargestellt wird. Ein semi-progressiver Ansatz adaptiert ein geschnittenes Element, sobald das Instrument dieses verlässt. Solange sich das Schneide-Instrument innerhalb des Elements bewegt, wird der Schnitt nicht aktualisiert. Progressive Methoden dagegen passen den Schnitt zu jedem diskreten Zeitpunkt an die Instrument-Bewegung an.
Alle im Folgenden beschriebenen Algorithmen definieren Schnittrichtungen unabhängig von der Mesh-Topologie. Es wird beurteilt, inwiefern die jeweiligen Ansätze als topologisches Verfahren für einen Reiß-Algorithmus dienen können.

5.5.1.1 Nicht-Progressives Schneiden

Nicht-Progressive Ansätze sind eher selten, da sie im Allgemeinen nur für solche Anwendungen geeignet sind, in denen die Schnittfläche erst dann für den Benutzer sichtbar wird, wenn sie eine gewisse Größe erreicht oder das Objekt vollständig durchtrennt hat.

Serby et al. [108] definieren den Schnitt in einer triangulierten Oberfläche durch ein Liniensegment. Knoten in der Nähe des Schnitts werden auf das Liniensegment verschoben, so dass eine Kette von Dreieckskanten das Segment exakt überlagert. Anschließend wird das Mesh entlang dieser Kanten aufgetrennt.
Steinemann et al. [113] schlagen eine Mischung aus Mesh-Verfeinerung und Anpassung existierender Elemente vor, um ein Tetraeder-Gitter zu zerschneiden. Zunächst wird ein Dreiecks-Mesh generiert, das die komplette Schnittfläche approximiert. Knoten in der Nähe der Schnittfläche werden auf diese projiziert, mittig geschnittene Tetraeder werden verfeinert. Anschließend werden Tetraeder entlang ihrer Grenzen getrennt.

Nicht-progressive Algorithmen können am ehesten für das Reißen spröder Materialien eingesetzt werden. Denn in diesen breiten sich Risse mit sehr hohen Geschwindigkeiten aus.

Bemerkung Ansatz [113] unterscheidet explizit zwischen Mesh-Verfeinerung und Mesh-Anpassung. Mit „Anpassung“ oder „Adaption“ sind häufig topologische Operationen gemeint, die in der Literatur auch als *Node Snapping* bezeichnet werden: Ein existierendes Element wird einer gewünschten Topologie angepasst, indem Element-Knoten auf Soll-Positionen versetzt werden. Die Knoten „schnappen ein“.

5.5.1.2 Semi-Progressives Schneiden

Semi-Progressive Ansätze zeichnen sich im Allgemeinen dadurch aus, dass eine Kollisionserkennung detektiert, wann und auf welche Art das Instrument in ein Element eindringt und wann und auf welche Art das Instrument das Element wieder verlässt. Anschließend wird der Schnitt realisiert.

Bielser et al. [12] definieren Schnitt-Muster für die prinzipiellen Möglichkeiten, einen Tetraeder mit einem Liniensegment zu schneiden und sortieren diese in eine Look-Up-Table ein. Jeder geschnittene Tetraeder wird in siebzehn Sub-Tetraeder zerlegt, denn die so erzeugte Topologie kann alle fünf definierten Schnitt-Muster abbilden. In der Folgearbeit [11] generieren Bielser und Gross nur noch diejenigen Sub-Tetraeder, die notwendig sind, um das tatsächlich zutreffende Schnitt-Muster topologisch umzusetzen.
Auch Ganovelli et al. [47] bilden Tetraeder-Schnitte auf vordefinierte Schnitt-Konfigurationen ab. Im Gegensatz zu [12] verfeinern sie zusätzlich direkte Nachbarn des geschnittenen Tetraeders, um Mesh-Inkonsistenzen zu verhindern. Als Erweiterung dieser Arbeit schlagen Ganovelli und O'Sullivan [49] eine Methode namens *Edge Collapse* vor, die stark degenerierte Tetraeder aus der Mesh-Struktur entfernt ohne Löcher zu hinterlassen.
Sifakis et al. [109] beschreiben einen Algorithmus, der als Eingabe eine beliebige triangulierte Oberfläche erhält, welche den Schnitt definiert und inkrementell erweitert werden kann. „Beliebig“ heißt in diesem Zusammenhang, dass sich die triangulierte Oberfläche z. B. selbst schneiden darf. Der Teil des Ansatzes, der die endgültigen topologischen Veränderungen entlang der Schnittfläche vornimmt, ist eine Erweiterung des Virtuellen-Knoten-Algorithmus auf unbegrenzt viele Tetraeder-Schnitte. Allerdings wird ein Schnitt erst dann sichtbar, wenn er einen Tetraeder vollständig durchtrennt.
Pietroni et al. [100] betten ein deformierbares Objekt in ein regelmäßiges Gitter ein. Ihr Ansatz für virtuelles, interaktives Schneiden namens *Splitting Cubes* kann als Version des Virtuellen-Knoten-Algorithmus für Kuben betrachtet werden.

Wird ein semi-progressiver Ansatz für einen Reiß-Algorithmus eingesetzt, muss der Riss pro Zeitschritt mindestens ein Mesh-Element vollständig durchqueren. Die Länge einer Rissausbreitung hängt somit von der Mesh-Auflösung ab.

5.5.1.3 Progressives Schneiden

Besonders bei progressiven Schneide-Algorithmen besteht die Gefahr, dass unerwünscht viele neue Elemente entlang des Schnitts entstehen. Hier muss ein vernünftiges Maß zwischen Mesh-Verfeinerung und Approximation der Schnittkante gefunden werden.

Als Basis-Algorithmus dient Mor [84, 85] ein semi-progressiver Ansatz, der einen Tetraeder je nach Schnitt-Muster in mindestens fünf und maximal neun Tetraeder zerlegt. Für progressives Schneiden sind elf Konfigurationen definiert, mit denen temporäre Schnitte abgebildet werden können. Solange sich das Instrument innerhalb des Tetraeders bewegt, wird der Schnitt durch eine Abfolge von Löschen und Generieren temporärer Mesh-Verfeinerungen dargestellt. Verlässt das Instrument den Tetraeder, entscheidet der semi-progressive Algorithmus über die dauerhafte Unterteilung des Elements.
Die Arbeit von Bielser et al. [9, 10] ist die Erweiterung von [12] und [11] auf progressives Schneiden. Im Gegensatz zu [84] wird ein Tetraeder inkrementell verfeinert, wobei versucht wird, die Anzahl der Lösch-Operationen zu minimieren. Der ursprüngliche Tetraeder wird durch bis zu siebzehn neue Tetraeder ersetzt.
Wicke et al. [131] führen eine FEM-Simulation ein, die ein Objekt räumlich durch beliebige konvexe Polyeder-Elemente diskretisiert. Der Algorithmus ist dadurch nicht auf einen bestimmten Element-Typ festgelegt. Da infolge eines Schneidevorgangs wiederum beliebige konvexe Polyeder-Elemente entstehen dürfen, sind topologische Anpassungen in diesem Modell sehr einfach umzusetzen. Dieser Ansatz erlaubt progressives Schneiden, kann jedoch für reine Dreiecks- oder Tetraeder-Gitter nicht eingesetzt werden.
Nienhuys und van der Stappen [91] behandeln interaktives Schneiden in triangulierten Oberflächen.[5] Ein Liniensegment approximiert das Skalpell und das aktuelle Schnittende ist stets durch einen Mesh-Knoten definiert, der exakt auf dem Liniensegment liegt und mit dem Skalpell durch das Mesh bewegt wird. Dieser ausgezeichnete Knoten wird *aktiver Knoten* genannt, die Dreiecke am aktiven Knoten bilden die *aktive Region*. Hinter dem aktiven Knoten entstehen neue Dreiecke entlang der Schnittkante, vor dem aktiven Knoten werden Dreiecke gelöscht, sofern es die Mesh-Topologie zulässt. Lokal wird die Mesh-Qualität erhöht, indem Dreieckskanten in der Umgebung des aktiven Knotens *flippen*: Teilen sich zwei Dreiecke eine gemeinsame Kante, so verbindet diese die beiden gemeinsamen Knoten. Nach dem Flippen verbindet diese Kante die beiden anderen der insgesamt vier Knoten.

Die Arbeiten von Mor und Bielser et al. bzw. alle genannten Arbeiten, die auf der Definition von Schnitt-Mustern basieren, betrachten einen geschnittenen Tetraeder separat und zerlegen diesen in Sub-Tetraeder, welche den ursprünglichen Tetraeder exakt ausfüllen. Übertragen auf ein Dreiecks-Mesh bedeutet dies, dass aus einem geschnittenen Element im Allgemeinen mehr als zwei Sub-Elemente entstehen. Denn aus einem geschnittenen Dreieck können nur dann genau zwei

[5] Etwas ausführlicher ist der Algorithmus im Technischen Bericht [90] und in der Doktorschrift [89] beschrieben.

neue Dreiecke hervorgehen, wenn der Schnitt durch ein einziges Liniensegment gegeben ist und genau durch eine der Dreiecksspitzen verläuft. Dieser kontinuierliche Zuwachs an Mesh-Elementen kann leicht zu Rechenzeitproblemen führen. In der Arbeit von Nienhuys und van der Stappen existieren keine „geschnittenen Dreiecke“. Vielmehr passt sich die gesamte aktive Region kontinuierlich an die Bewegung des aktiven Knotens an, wobei nur in geringem Maße neue Dreiecke entstehen. Insbesondere stimmt die Topologie um einen aktiven Knoten mit der in Abschnitt 5.4.3 beschriebenen Topologie um einen Risskeim überein: Der aktive Knoten ist stets mit genau zwei Randknoten verbunden, die durch gemeinsame Ruhepositionen miteinander korrespondieren. Daher greift Abschnitt 5.5.3 auf diesen Ansatz zurück, um einen Risskeim durch das Mesh zu bewegen.

5.5.2 1. Ansatz: Reißen mit konstanter Dreiecksanzahl

Der hier beschriebene, erste Ansatz für eine Rissumsetzung ist eine Erweiterung des topologischen Konzepts von Boux de Casson und Laugier [28] bzw. von Grimm [57]. Diese trennen das Mesh entlang existierender Dreieckskanten auf, so dass ausschließlich neue Knoten und neue Federn entlang der Risskante entstehen. Die Anzahl an Dreiecken bleibt konstant, die Risskante allerdings ist unter Umständen sehr zackig. Darüber hinaus führt der diskrete Ansatz zu einem ruckartigen Wechsel zwischen Spannungsaufbau und Rissfortschritt. Durch eine Kombination aus Node Snapping und Animation der Rissausbreitung entlang einer adaptierten Kante wird dieses topologische Verfahren zu einem pseudo-kontinuierlichen Ansatz erweitert.

Bemerkung Alle notwendigen Mesh-Adaptionen werden zunächst durch Anpassung von Knoten-Ruhepositionen vorgenommen. Mit Hilfe baryzentrischer Koordinaten werden diese Adaptionen anschließend auf das deformierte Mesh übertragen. Daher wird im Folgenden darauf verzichtet, zwischen unverzerrten und verzerrten Zuständen zu unterscheiden.

Sei ein Rissdreieck mit Eckpunkten $\vec{a}$, $\vec{b}$ und $\vec{c}$ gegeben, wobei $\vec{a}$ mit dem Risskeim übereinstimme. Die Rissrichtung ist nach Abschnitt 5.4.2 durch $\vec{d} = \vec{b} + t(\vec{c} - \vec{b})$ mit $t \in [0, 1]$ definiert. In Abb. 5.7(a) ist der auf der Kante des Rissdreiecks liegende Punkt $\vec{d}$ durch ein Kreuz markiert. Wie in Abb. 5.7(b) zu sehen, wird nun entweder die Position von $\vec{b}$ oder die Position von $\vec{c}$ an $\vec{d}$ angepasst. Gewählt wird derjenige der beiden Knoten, welcher näher an $\vec{d}$ liegt. Somit dreht sich eine der an $\vec{a}$ hängenden Federn in Richtung der gewünschten Rissausbreitung.
Im weiteren Verlauf der Simulation soll sich das Rissende von $\vec{a}$ nach $\vec{d}$ ausbreiten. Betrachten wir die Bilderfolge in Abb. 5.8. Die Topologie in Abb. 5.8(a) stimmt mit derjenigen in Abb. 5.7(b) überein. Jedesmal wenn die Spannung am Risskeim

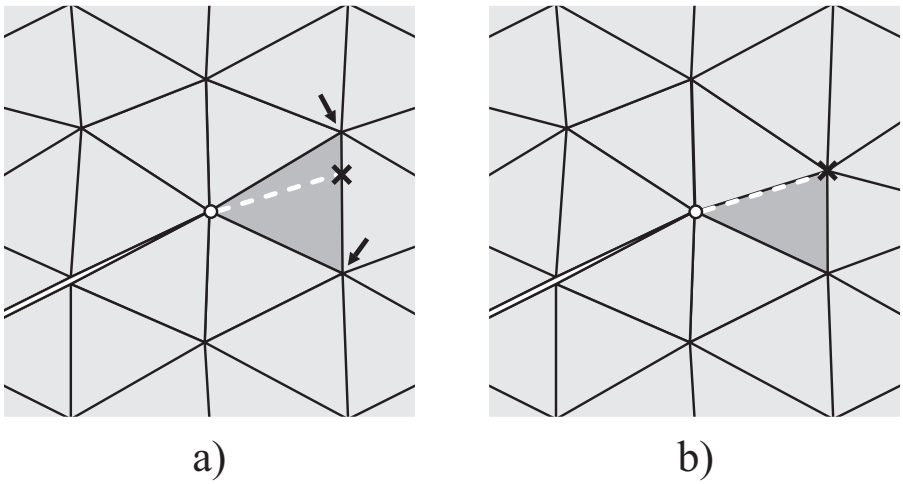


Abbildung 5.7: Adaption einer Kante an die Rissausbreitung

den Schwellwert für Reißen überschreitet, wird die Position von $\vec{a}$ um $tearLength$ (siehe Abschnitt 5.4.4) in Richtung $\vec{d}$ verschoben:

$$\vec{a} := \vec{a} + tearLength \cdot \frac{\vec{d} - \vec{a}}{|\vec{d} - \vec{a}|}$$

Im Falle von $tearLength \geq |\vec{d} - \vec{a}|$ wird $\vec{a}$ exakt auf die Position von $\vec{d}$ versetzt. Um das „Wandern“ von $\vec{a}$ topologisch konsistent darzustellen, werden – wie in Abb. 5.8(b) zu sehen – die beiden Dreiecke an der Kante $(\vec{d} - \vec{a})$ temporär in vier Dreiecke zerlegt. Somit kann sich der Riss ähnlich einem Reißverschluss entlang der Kante öffnen. Abb. 5.8(d) zeigt, wie die vier temporären Dreiecke wieder auf zwei reduziert werden, nachdem der Risskeim seine Zielposition erreicht hat.

Da neue Dreiecke nur temporär in das Gitter eingefügt werden, bleibt die Gesamtanzahl an Dreiecken im Mesh konstant. (Dies gilt natürlich nur, falls ein begonnener Riss irgendwann auf einer Position $\vec{d}$ stehen bleibt oder irgendwann den Rand des Gitters erreicht, wo er beendet werden kann.)
Wenn ein Algorithmus wie dieser mit temporären Konfigurationen arbeitet, die nicht dafür vorgesehen sind, dauerhaft im Mesh zu verbleiben, ist die Behandlung mehrerer Risse prinzipiell mit einem gewissen Verwaltungsaufwand verbunden. Für vorliegenden Fall reicht es jedoch aus, sämtliche Informationen über eine temporäre Konfiguration lokal im entsprechenden Risskeim abzuspeichern. Der Aufbau einer globalen Verwaltungsstruktur ist nicht notwendig.
Ausführlicher wird der vorgeschlagene Ansatz in Abschnitt 5.6 untersucht und diskutiert.

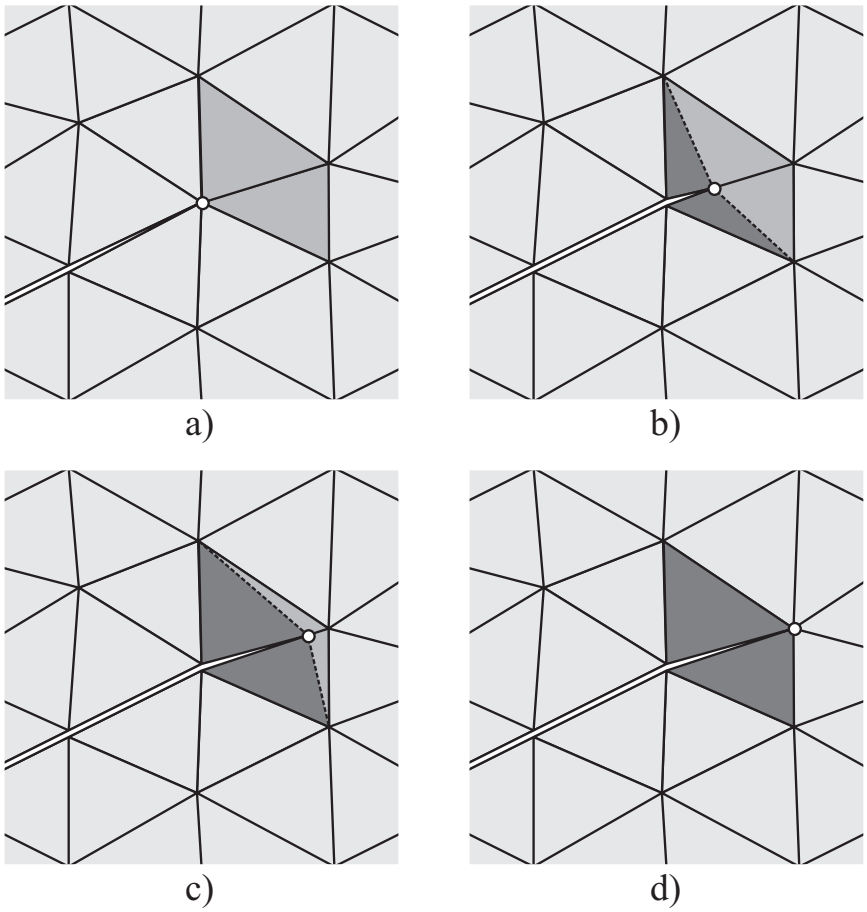


Abbildung 5.8: Animation des Risses entlang einer Kante

5.5.3 2. Ansatz: Reißen durch progressives Schneiden

In diesem Abschnitt wird der interaktive Schneide-Algorithmus von Nienhuys und van der Stappen [91] für die topologische Umsetzung eines Risses adaptiert. Der Ansatz setzt sich im Wesentlichen aus folgenden Komponenten zusammen:
(Die in [91] verwendeten Begriffe „aktiver Knoten“, „aktive Region“ und „Schnittkante“ wurden durch „Risskeim“, „Rissregion“ und „Risskante“ ersetzt.)

- Bewegung des Risskeims innerhalb der Rissregion
- Löschen von Knoten in der Nähe des Risskeims
- Teilung einer zu langen Risskante
- Flippen von Dreieckskanten in der Umgebung des Risskeims

Bemerkung Die prinzipielle Vorgehensweise in Bezug auf Mesh-Adaptionen ist dieselbe wie im 1. Ansatz: Alle notwendigen Operationen werden zunächst auf den Ruhepositionen der Knoten ausgeführt und anschließend durch baryzentrische Koordinaten auf das deformierte Mesh übertragen.

Ob eine betrachtete Dreieckskante flippen soll, wird mit Hilfe des *Delaunay-Kriteriums* entschieden. Abb. 5.9 veranschaulicht dieses Kriterium und die Flip-Operation. Um die Kante eines deformierbaren Gitters zu überprüfen, müssen die zwei an dieser Kante hängenden Dreiecke zunächst entfaltet werden, so dass sie in einer gemeinsamen Ebene liegen. Eine Kante erfüllt das Delaunay-Kriterium genau dann nicht, wenn die Umkreise der beiden anliegenden Dreiecke jeweils alle vier Dreiecksspitzen einschließen. Nach dem Flippen enthält jeder Umkreis nur noch die drei Knoten des zugehörigen Dreiecks und das Delaunay-Kriterium ist erfüllt. Diese Operation verbessert die Mesh-Qualität, da sie den minimalen Winkel innerhalb der beiden Dreiecke erhöht und somit wohlgeformte Triangulierungen erzeugt.

Bewegung des Risskeims Die Dreiecke am Risskeim in Abb. 5.10(a) bilden die Rissregion. Angenommen, die Berechnung der Rissausbreitung liefert das dunkelgraue Dreieck als Rissdreieck und in diesem die schwarz markierte Position als neues Rissende. Der Risskeim wird auf diese Position versetzt, so wie in Abb. 5.10(b) zu sehen.
Am Risskeim hängen zwei Randknoten, welche sich eine gemeinsame Ruheposition teilen. Sei $\vec{r}$ die Ruheposition der Randknoten und $\vec{a}$ diejenige des Risskeims. Man beachte, dass sich durch das Versetzen des Risskeims Orientierung und Länge von $\vec{a} - \vec{r}$ ändern. Der Algorithmus von Nienhuys und van der Stappen sieht vor, dass der aktive Knoten stets exakt auf dem Liniensegment liegen muss, welches das virtuelle Skalpell abstrahiert. Da das Liniensegment beliebige Richtungsänderungen vollführen kann und der aktive Knoten dessen Bewegungen folgt, sind auch solche Verschiebungen des aktiven Knotens möglich, welche

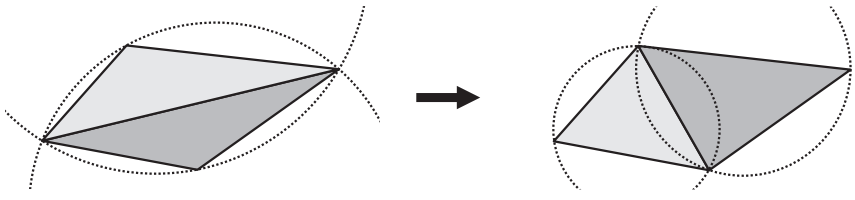


Abbildung 5.9: Kanten werden geflippt, wenn sie das Delaunay-Kriterium nicht erfüllen

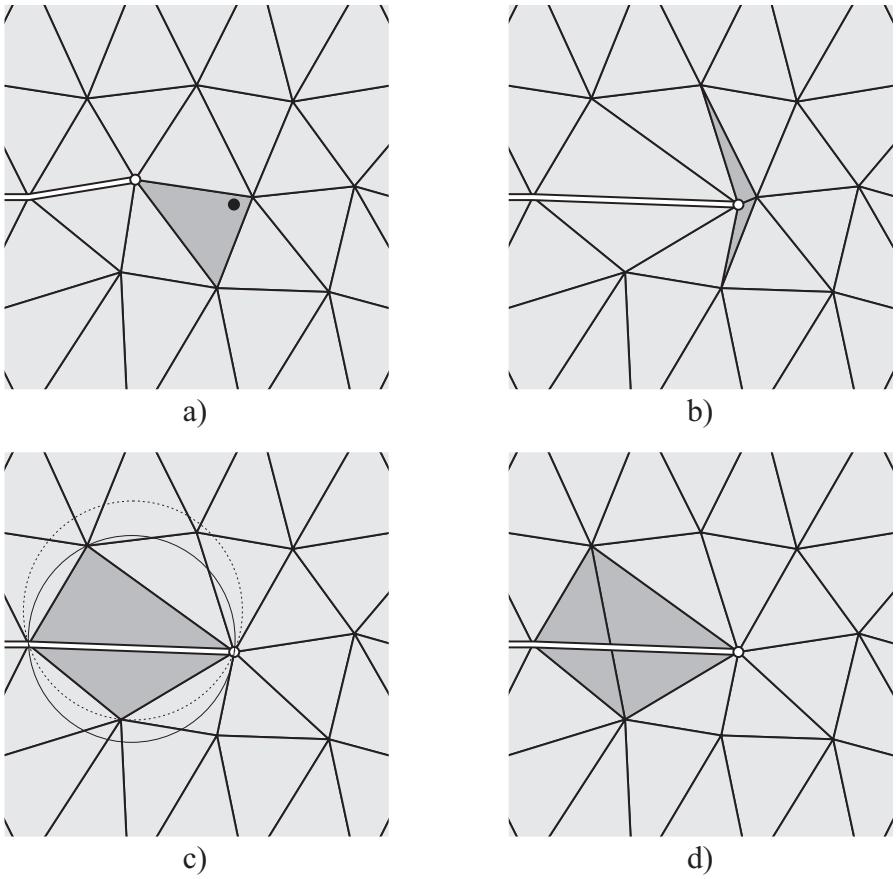


Abbildung 5.10: Bewegung des Risskeims (a → b), Löschen eines nahen Knotens (b → c) und Teilung der Risskante (c → d)

die Schnittkante $\vec{a} - \vec{r}$ wieder verkürzen. Dies sieht unnatürlich aus und da die Bewegung eines Risskeims an keine Instrument-Position gekoppelt ist, wird die Verkürzung einer Risskante verhindert. Um die Zielposition des Risskeims nicht nachträglich anpassen zu müssen, wird das Problem bereits in der Berechnung der Rissausbreitung behandelt: Für die Rissrichtung kommen nur solche Vektoren in Frage, welche mit der Risskante $\vec{a} - \vec{r}$ einen spitzen Winkel einschließen. Somit wird im Voraus erzwungen, dass der Risskeim „vorwärts“ wandert.

Löschen von Knoten In Abb. 5.10(b) ist zu sehen, wie der Risskeim einem anderen Knoten der Rissregion sehr nahe kommt. Unterschreitet der Abstand dieser beiden Knoten einen vordefinierten Grenzwert, wird der nahe Knoten auf die aktuelle Position des Risskeims versetzt, wie in Abb. 5.10(c) gezeigt. Dadurch

degenerieren die in Abb. 5.10(b) dunkelgrau markierten Dreiecke zu Liniensegmenten und können mitsamt dem nahen Knoten aus dem Gitter entfernt werden ohne Löcher zu hinterlassen.

Teilung der Risskante Nienhuys und van der Stappen entscheiden mit Hilfe des Delaunay-Kriteriums, wann die sich hinter dem aktiven Knoten verlängernde Schnittkante verfeinert wird. Auf die in Abb. 5.10(c) dunkelgrau eingefärbten Dreiecke kann das Delaunay-Kriterium angewendet werden, obwohl diese sich keine gemeinsame Kante teilen. Schließlich verfügen die beiden am Risskeim hängenden Randknoten über dieselbe Ruheposition. Die beiden Umkreise zeigen an, dass das Delaunay-Kriterium verletzt ist, weshalb die Risskante – wie in Abb. 5.10(d) zu sehen – geteilt wird. Es entstehen zwei neue Dreiecke entlang der Risskante.
Zu beachten ist, dass die Mesh-Triangulierung beeinflusst, wann das Delaunay-Kriterium verletzt und die Risskante unterteilt wird. Die Auflösung der Risskante ist damit von der Auflösung des Gitters abhängig, weshalb der ursprüngliche Schneide-Algorithmus mit einem zusätzlichen Grenzwert-Parameter ausgestattet wird. Überschreitet die Länge der Risskante diesen Grenzwert, wird sie geteilt. Somit ist eine bestimmte Auflösung der Risskante garantiert.

Flippen von Dreieckskanten Nienhuys und van der Stappen beschreiben das Flippen als Operation mit der Funktion, die Mesh-Qualität lokal zu verbessern. In Abhängigkeit der anfänglichen Mesh-Qualität bietet es sich daher an, das Flippen großräumig um den Risskeim herum vorzunehmen. Dieser topologische Eingriff hat jedoch den Nachteil, die Kräfteverteilung im Mesh zu verändern. Eine (im Allgemeinen deformierte) Feder verschwindet zunächst und übt anschließend auf zwei andere Knoten Kräfte aus. Wie Abb. 5.11 veranschaulicht, kommt hinzu, dass das Flippen die Krümmung einer in 3D deformierbaren Oberfläche lokal umkehrt. Für den Reiß-Algorithmus soll das Flippen daher so weit wie möglich eingeschränkt werden und es stellt sich die Frage, inwiefern diese Operation für den Algorithmus zwingend notwendig ist. Betrachten wir hierzu die Bilderfolge

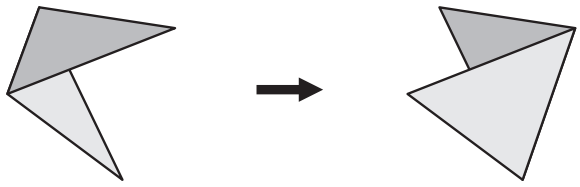


Abbildung 5.11: Flippen einer Kante in 3D

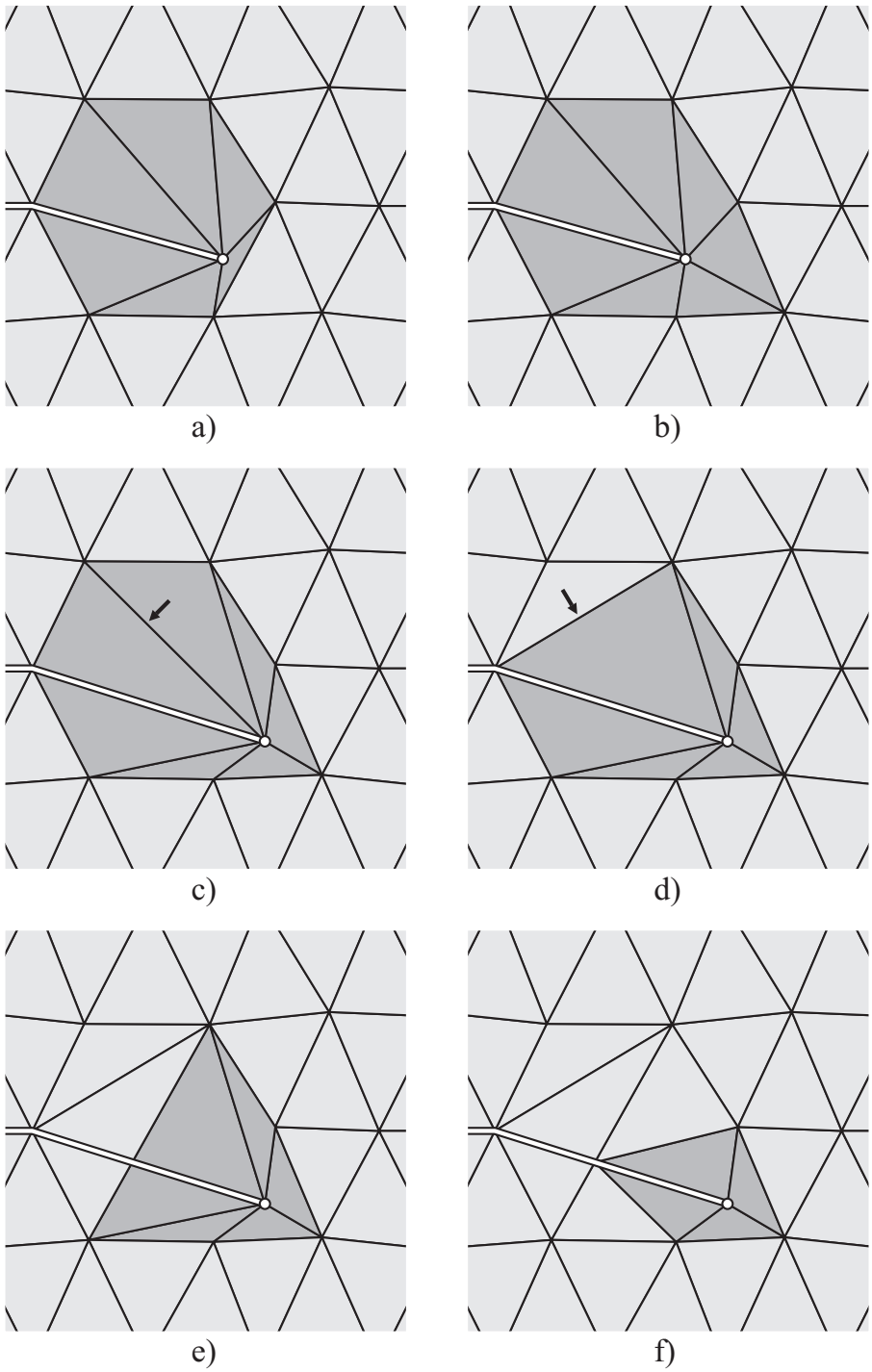


Abbildung 5.12: Wandern der Rissregion durch Flippen

in Abb. 5.12. Bild (a) zeigt, wie sich der Risskeim einer Kante nähert, ohne dass sein Abstand zu benachbarten Knoten den Grenzwert für die Lösch-Operation unterschreitet. In Abschnitt 5.4.4 wurde festgelegt, dass ein neu berechnetes Rissende einen Mindestabstand ϵ zu der Kante einhalten soll, welche dem Risskeim im Rissdreieck gegenüber liegt. Der vorliegende Fall ist die Erklärung für dieses Vorgehen. Denn bevor sich der Riss weiter ausbreiten kann, muss die den Weg versperrende Kante wegflippen, wie in Abb. 5.12(b) dargestellt. Somit öffnet sich die Rissregion vor dem Risskeim und dieser kann seine Bewegung fortsetzen, wie in Abb. 5.12(c) zu sehen. Durch die weitere Rissausbreitung werden die Dreieckskanten hinter dem Risskeim in die Länge gezogen, insbesondere diejenige, welche in Bild (c) durch einen Pfeil markiert ist. Da diese Kante in Bild (d) flippt, wird das Dreieck, in welchem der Pfeil enthalten ist, nach hinten aus der Rissregion heraus geschoben. Somit verringert sich die Anzahl an Dreiecken innerhalb der Rissregion von sieben auf sechs. Anschließend erfüllt die Risskante das Delaunay-Kriterium nicht mehr und teilt sich, wie in Abb. 5.12(e) zu sehen. Im abschließenden Bild (f) flippen zwei weitere Kanten, so dass sich die Rissregion auf insgesamt vier Dreiecke reduziert.
Die Bilderfolge zeigt, dass die Rissregion durch das Flippen überhaupt erst in der Lage ist, sich zusammen mit dem Risskeim durch das Mesh zu schieben. Die Rissregion muss in Richtung der Risskeim-Bewegung neue Dreiecke aufnehmen und hinter dem Risskeim Dreiecke aus der Rissregion ausschließen. Daher werden für den Reiß-Algorithmus genau diejenigen Kanten, welche zu Dreiecken der aktuellen Rissregion gehören, auf das Delaunay-Kriterium überprüft und bei Bedarf geflippt.

5.5.4 Entstehen und Beenden von Rissen

Die Abschnitte 5.5.2 und 5.5.3 behandeln die Ausbreitung eines bereits existierenden Risses. Im Folgenden wird beschrieben, wie neue Risskeime entstehen und wie Risse beendet werden, wenn sie den Mesh-Rand erreichen.

Betrachten wir zunächst den Fall, dass der in Abschnitt 5.4.3 umrissene Algorithmus einen Randknoten als Rissknoten auswählt. An diesem Randknoten werden ein Rissdreieck und eine Rissrichtung bestimmt. Genau wie in Abschnitt 5.5.2 wird eine Kante des Rissdreiecks an die berechnete Rissrichtung angepasst. Abb. 5.13(a) zeigt einen solchen Randknoten mit bereits adaptierter Dreieckskante. Die dunkelgraue, gestrichelte Linie zeigt die ursprüngliche Lage dieser Kante. Das Gitter wird nun entlang der adaptierten Kante komplett aufgetrennt, so wie in Abb. 5.13(b) zu sehen. Hierbei werden der Rissknoten und die angepasste Kante verdoppelt und an einem ursprünglich inneren Knoten entsteht ein neuer Risskeim.

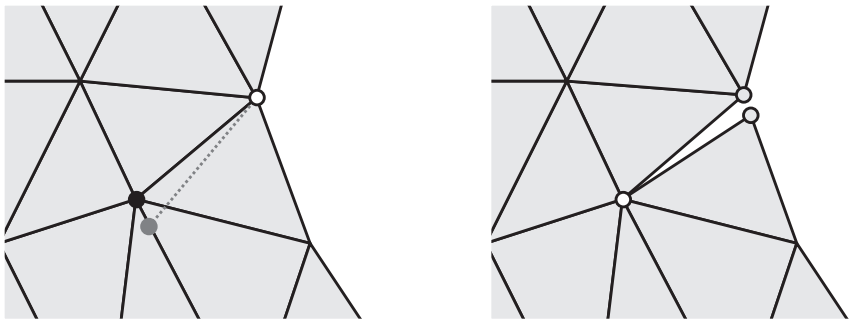


Abbildung 5.13: Ein neuer Riss entsteht am Rand des Gitters

Zu beachten ist, dass ein Randknoten nur dann als Rissknoten in Frage kommt, wenn an ihm mindestens zwei Dreiecke hängen. Denn das Mesh kann nur entlang einer Kante aufgetrennt werden, an der sich zwei Dreiecke befinden. Insbesondere wird die Berechnung der Rissrichtung an einem Randknoten derart eingeschränkt, dass anschließend als anzupassende Kante mit Sicherheit keine Kante des Mesh-Rands gewählt wird.

Ist der Rissknoten ein innerer Mesh-Knoten, ist die Vorgehensweise sehr ähnlich. Wiederum werden am Rissknoten zunächst Rissdreieck und Rissrichtung ermittelt, um anschließend eine der Dreieckskanten an die Rissrichtung anzupassen. In Abb. 5.14(a) ist der innere Rissknoten weiß markiert und über die bereits adaptierte Kante mit dem schwarzen Knoten verbunden. Damit sich das Gitter entlang dieser Kante öffnen kann, muss eine Mesh-Verfeinerung vorgenommen

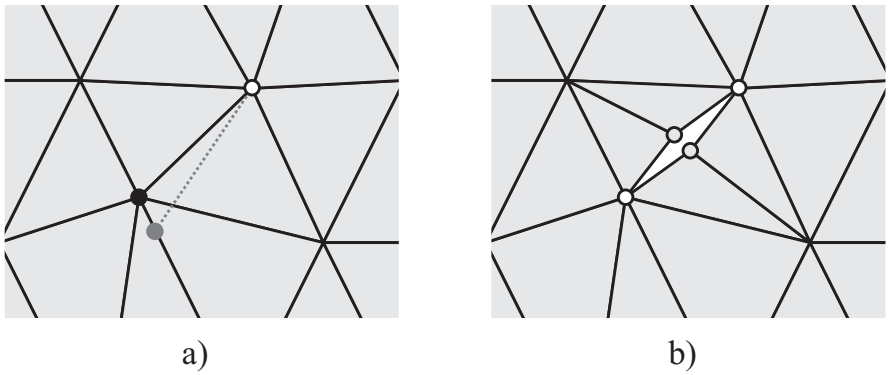


Abbildung 5.14: Ein neuer Riss entsteht im Inneren des Gitters

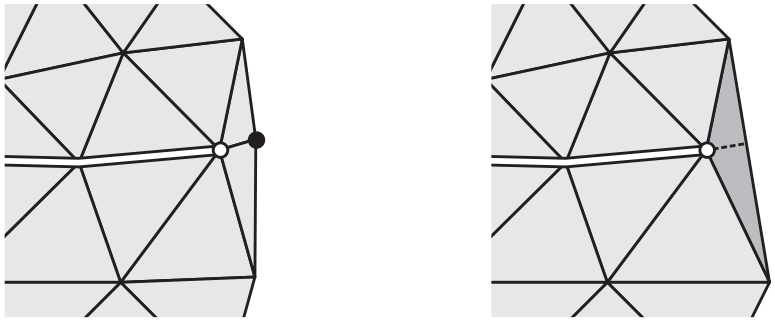


Abbildung 5.15: Ein Risskeim erreicht den Rand des Gitters

werden, wie in Abb. 5.14(b) dargestellt. Die beiden neuen Knoten werden mittig auf der angepassten Kante eingefügt, an den Enden der Kante entstehen zwei neue Risskeime.

Für den in Abschnitt 5.5.2 beschriebenen Ansatz ist es nicht erforderlich, eine spezielle Behandlung für das Beenden von Rissen zu definieren. Der Algorithmus wählt eine zu adaptierende Dreieckskante, entlang derer die Rissausbreitung animiert wird. Endet diese Kante in einem Randknoten, so wird das Mesh durch Verdopplung dieses Randknotens automatisch zerteilt.
Sei nun ein Risskeim gegeben, den der in Abschnitt 5.5.3 vorgestellte Ansatz durch das Gitter bewegt. Abb. 5.15(a) zeigt, wie sich dieser Risskeim dem schwarz markierten Randknoten nähert. Unterschreitet der Abstand zwischen Risskeim und Randknoten einen Grenzwert, wird der Riss beendet. Das Mesh wird entlang der Kante aufgetrennt, welche den Risskeim mit dem Randknoten verbindet.
In Abb. 5.15(b) ist zu sehen, wie sich der Risskeim Richtung Mesh-Rand bewegt, ohne in die unmittelbare Umgebung eines Randknotens zu kommen. Der Riss wird beendet, wenn der Abstand zwischen Risskeim und Randkante einen Grenzwert unterschreitet. Indem das dunkelgrau dargestellte Dreieck in zwei kleinere unterteilt wird, kann der Riss senkrecht zur Randkante abgeschlossen werden.

5.6 Ergebnisse

Die beiden vorgeschlagenen Reiß-Algorithmen sind mit den in Abschnitt 4.1 festgelegten Konsistenzbedingungen kompatibel. Der Ort der Rissausbreitung wird auf Basis von Spannungen, die Richtung der Rissausbreitung auf Basis von Verzerrungen ermittelt. Die in Abschnitt 5.2 formulierten Anforderungen 2 und 6

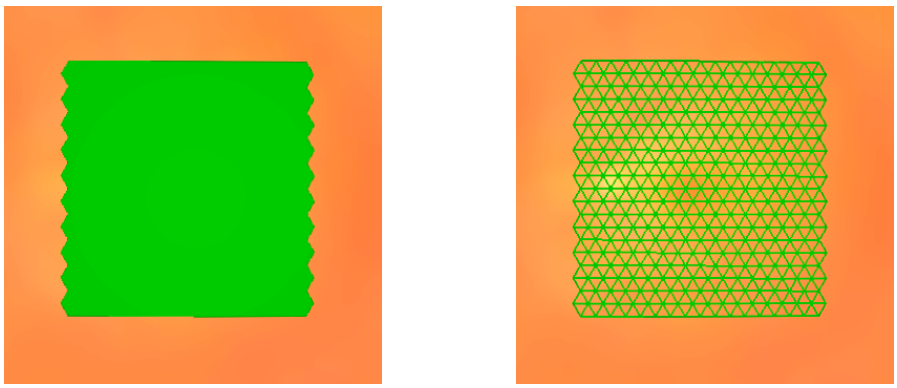


Abbildung 5.16: Das zu reißende Mesh der Test-Umgebung

sind daher erfüllt. Im Folgenden wird untersucht, inwiefern die Reiß-Algorithmen auch den übrigen Anforderungen genügen. Hierbei werden sie mit dem bisher in EYESi eingesetzten Reiß-Algorithmus verglichen, der das Mesh entlang von Dreieckskanten auftrennt (Grimm [57]).

Bei der Test-Umgebung handelt es sich um ein entkerntes EYESi-Trainingsmodul, das prinzipiell folgendermaßen konfiguriert bzw. parametrisiert ist:

- Für die Mesh-Simulation gelten folgende Angaben:
  - Das Mesh besteht aus 367 Knoten, 1026 Federn (durchschnittliche Ruhelänge: 0.5mm) und 660 Dreiecken. Zu sehen ist es in Abb. 5.16.
  - Jeder Mesh-Knoten verfügt über die Masse $m = 0.01\text{kg}$.
  - Jede Feder hat die Federkonstante $k_\text{F} = 6500\text{N/m}$.
  - Jeder Simulationsschritt zerfällt in 25 Integrationsschritte.
- Für die Reiß-Algorithmen gelten folgende Angaben:
  - Im Anschluss an den Simulationsschritt wird nach einem Rissknoten gesucht. Ist die Spannung im Mesh ausreichend hoch und die Suche erfolgreich, wird die Methode ausgeführt, welche den jeweiligen Reiß-Algorithmus implementiert. Eine Reiß-Methode wird demnach maximal einmal pro Frame aufgerufen.
  - Für alle drei eingesetzten Reiß-Algorithmen werden dieselben Werte für *minInnerStress*, *minEdgeStress* und *minSproutStress* verwendet.

– In der Definition der beiden neuen Reiß-Algorithmen wurde festgelegt, dass die Länge $tearLength$ einer Rissausbreitung in Abhängigkeit der Spannung am Rissknoten berechnet wird. Für folgende Anwendungen gilt $minTearLength = 0.001$mm und $maxTearLength = 0.3$mm.

Falls die Test-Umgebung im Folgenden von diesen Angaben abweicht, wird an entsprechender Stelle darauf hingewiesen.

5.6.1 Laufzeit

Die Messungen wurden auf einem Rechner mit Intel(R) Core(TM)2 Quad CPU Q9300 und 2.5GHz unter Windows XP durchgeführt. Um Störungen zu minimieren, wurden alle unnötigen Dienste angehalten und das Programm für die Laufzeit-Messung als einzige Anwendung gestartet.

Um die Laufzeiten der Reiß-Algorithmen miteinander zu vergleichen, wurde das in Abb. 5.16 zu sehende Mesh mit jedem der drei Ansätze jeweils 100 mal vollständig auseinander gerissen. Das Gitter ist in der Mitte des oberen Rands entlang zweier Federn aufgetrennt, so dass sich bereits zu Beginn der Simulation eine Soll-Riss-Stelle in Form eines Risskeims im Mesh befindet. Die Knoten des rechten und des linken Rands werden nicht simuliert, sondern in jedem Frame ein Stück auseinander geschoben, so dass das Gitter der Länge nach durchreißt. In jeder der jeweils 100 Iterationen wird diese automatische Knoten-Verschiebung leicht variiert, so dass auch der Riss seinen Weg durch das Mesh variiert.

Die Abbildungen 5.17 bis 5.19 zeigen für jeden der drei Reiß-Algorithmen den ersten der jeweils 100 Versuche. In Abb. 5.17 beschreibt der Riss eine Linkskurve, der deutlich anzusehen ist, dass die Rissausbreitung an die Topologie des Gitters gebunden ist. Die Linkskurve ist ebenfalls, allerdings glatter, mit dem pseudo-kontinuierlichen Ansatz aus Abschnitt 5.5.2 zu erkennen (Abb. 5.18). Dieser kann die Richtung der Rissausbreitung beliebig adaptieren, jedoch stets nur zu dem Zeitpunkt, zu dem der Rissknoten das Ende einer gedrehten Kante erreicht hat. In Abb. 5.19 fällt auf, dass der kontinuierliche Reiß-Algorithmus aus Abschnitt 5.5.3 den Riss auf direktem Weg von oben nach unten durch das Mesh führt. Dies liegt daran, dass der kontinuierliche Ansatz die Rissrichtung nach jedem Simulationsschritt anpassen kann.

Der ersten Zeile von Tabelle 5.1 ist zu entnehmen, wieviele Methoden-Aufrufe pro Algorithmus insgesamt notwendig waren, um das Mesh 100 mal vollständig durchzureißen. Der kantenbasierte Ansatz erweitert den Riss pro Aufruf um eine komplette Federlänge, wodurch sich der Wert $1800 = (20 - 2) \cdot 100$ ergibt. Denn das Mesh besteht aus 20 Reihen Dreiecken, wobei die ersten beiden Reihen bereits

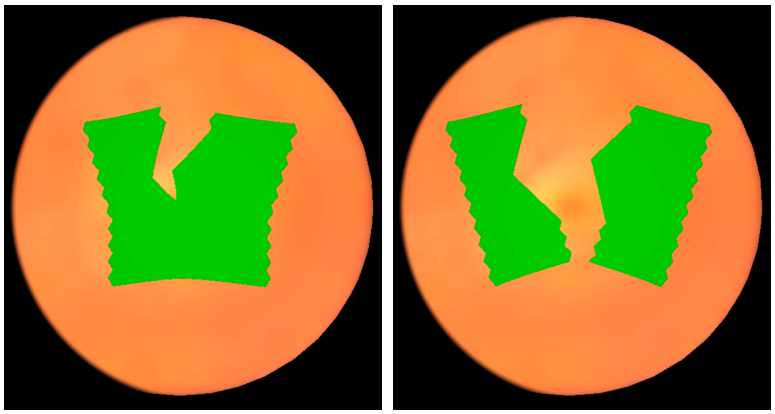


Abbildung 5.17: kantenbasiertes Reißen

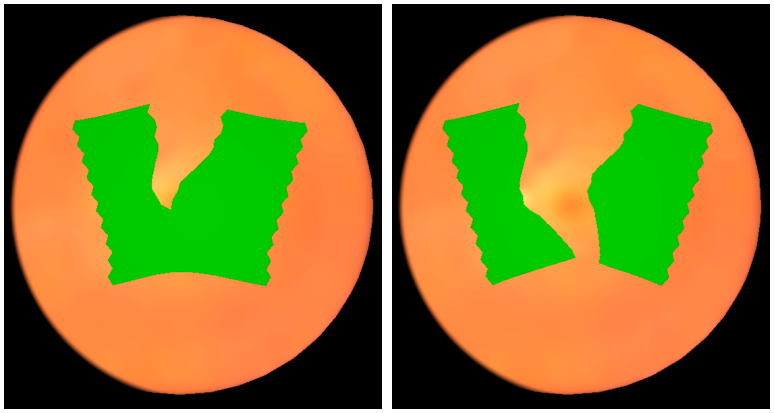


Abbildung 5.18: pseudo-kontinuierliches Reißen

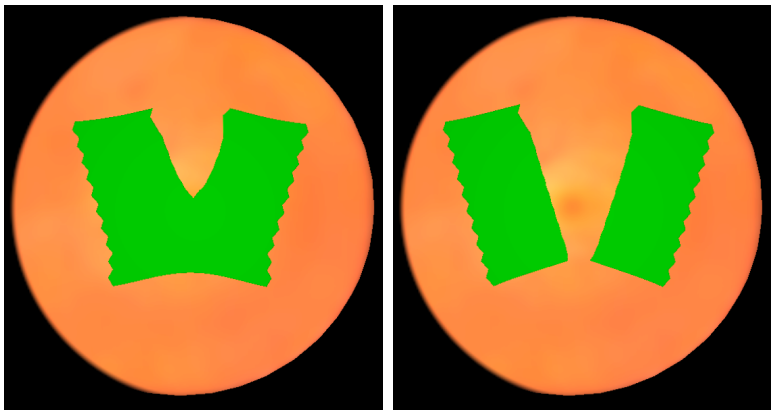


Abbildung 5.19: kontinuierliches Reißen

	kantenbasiertes Reißen	pseudo-kontinuierliches Reißen	kontinuierliches Reißen
Methoden-Aufrufe insgesamt	1800	34399	32448
Rechenzeit insgesamt	128.03ms	573.22ms	2386.90ms
mittlere Rechenzeit pro Aufruf	0.071ms	0.017ms	0.074ms
maximale Rechenzeit pro Aufruf	0.105ms	0.183ms	0.245ms

Tabelle 5.1: Für die Messungen wurde das Mesh mit jedem der drei Algorithmen jeweils 100 mal komplett durchgerissen

zu Beginn der Simulation über eine Kerbe verfügen. Mit 34399 und 32448 liegen die Werte für die beiden anderen Algorithmen weitaus höher, da sie den Riss pro Aufruf nur um eine Länge zwischen $minTearLength$ und $maxTearLength$ vergrößern. (Der pseudo-kontinuierliche Ansatz erreicht mehr Aufrufe als der kontinuierliche, da er den Riss auf einer eher kurvigen Strecke durch das Mesh führt.) Für sämtliche Methoden-Aufrufe benötigte der kantenbasierte Algorithmus aufsummiert 128.03ms, der pseudo-kontinuierliche 573.22ms und der kontinuierliche 2386.90ms. Unterm Strich beansprucht der kontinuierliche Ansatz somit fast 19 mal so viel Rechenzeit wie der kantenbasierte Ansatz. Ausschlaggebend jedoch ist die pro Methoden-Aufruf zu veranschlagende Rechenzeit. Wie Tabelle 5.1 zu entnehmen, beansprucht das kantenbasierte Reißen pro Aufruf eine Rechenzeit von durchschnittlich 0.071ms. Für den pseudo-kontinuierlichen Ansatz liegt dieser Wert bei 0.017ms und für den kontinuierlichen Ansatz bei 0.074ms. Im Schnitt liegen kantenbasiertes und kontinuierliches Reißen somit gleichauf. Da die maximal gemessenen Rechenzeiten für einen einzelnen Methoden-Aufruf (letzte Zeile in Tabelle 5.1) alle deutlich unter 1ms liegen, sind alle drei Algorithmen für den Einsatz in einer Echtzeitanwendung geeignet und können auch mehrmals pro Frame aufgerufen werden.

5.6.2 Die Reiß-Algorithmen im Vergleich

Für die Laufzeit-Messungen wurde das Mesh durch eine automatisierte Knoten-Verschiebung auseinander gerissen. Im Folgenden soll illustriert werden, wie gut sich die Rissausbreitung mit den verschiedenen Reiß-Algorithmen kontrollieren lässt, wenn der Benutzer mit Hilfe einer Pinzette versucht, den Riss auf einer vorgegebenen Bahn zu führen.
Abb. 5.20 zeigt das Mesh der Test-Anwendung, auf dem ein Kreis eingezeichnet ist. In der Gitter-Darstellung rechts ist zu sehen, dass es im Mesh keine Kette von Dreieckskanten gibt, die mit dem Kreis zusammenfällt. Das durch den Kreis eingeschlossene Material soll in einem zusammenhängenden Stück aus dem Mesh heraus gerissen werden, wobei der Riss möglichst genau der Kreislinie folgen soll. Um mit Hilfe der Pinzette die notwendigen Spannungen aufbauen zu können, ist das Gitter an seinen Rändern fixiert.
In den Abbildungen 5.21 bis 5.23 ist der Versuch mit allen drei Reiß-Algorithmen zu sehen: Das kantenbasierte Reißen schneidet unzufriedenstellend ab, da es die Kreislinie nur so gut approximieren kann, wie dies eine Folge von Federn bereits im Voraus tut. Mit Hilfe des pseudo-kontinuierlichen Ansatzes wurde der Kreis sichtbar besser angenähert, die Risskante verläuft allerdings recht ungleichmäßig. Der kontinuierliche Ansatz schließlich liefert das optisch beste Resultat.
Welche Ergebnisse mit den beiden neuen Algorithmen erreicht werden, hängt natürlich wesentlich davon ab, wie häufig und mit wieviel Geduld der Eingriff wiederholt wird. Mit ausreichend Übung ist es wahrscheinlich möglich, mit dem pseudo-kontinuierlichen Ansatz ähnlich gute Resultate zu erzielen wie mit dem kontinuierlichen. Der persönliche Eindruck der Autorin jedoch ist, dass der Umgang mit dem pseudo-kontinuierlichen Algorithmus weitaus schwieriger ist: Die

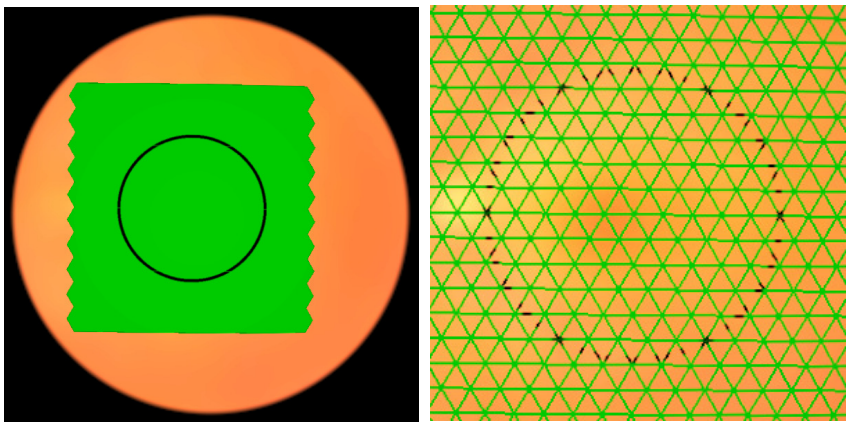


Abbildung 5.20: Aus dem Mesh soll ein kreisrundes Stück heraus gerissen werden

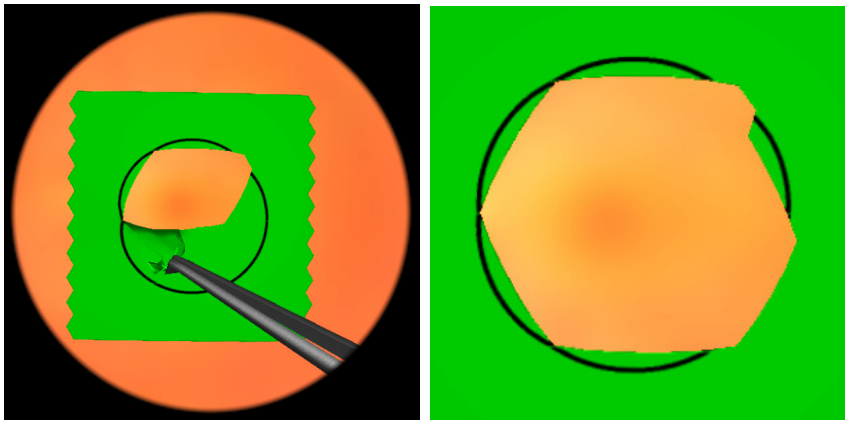


Abbildung 5.21: kantenbasiertes Reißen

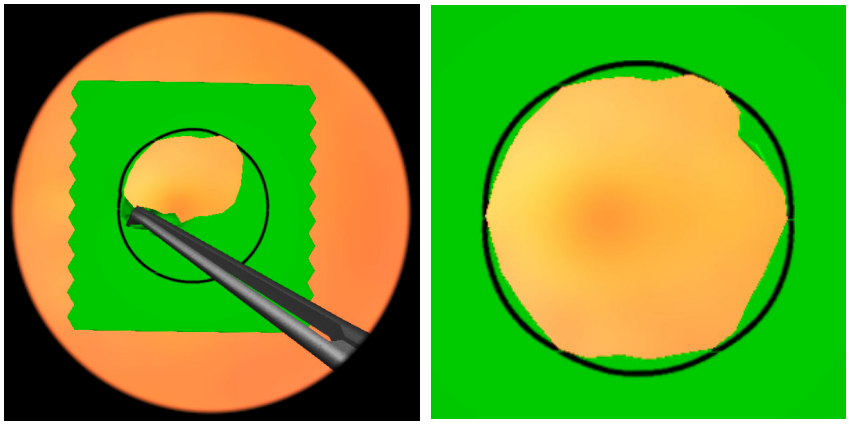


Abbildung 5.22: pseudo-kontinuierliches Reißen

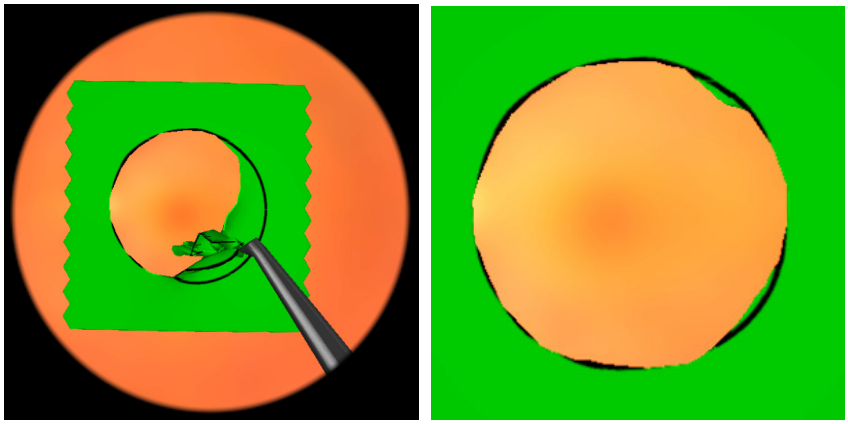


Abbildung 5.23: kontinuierliches Reißen

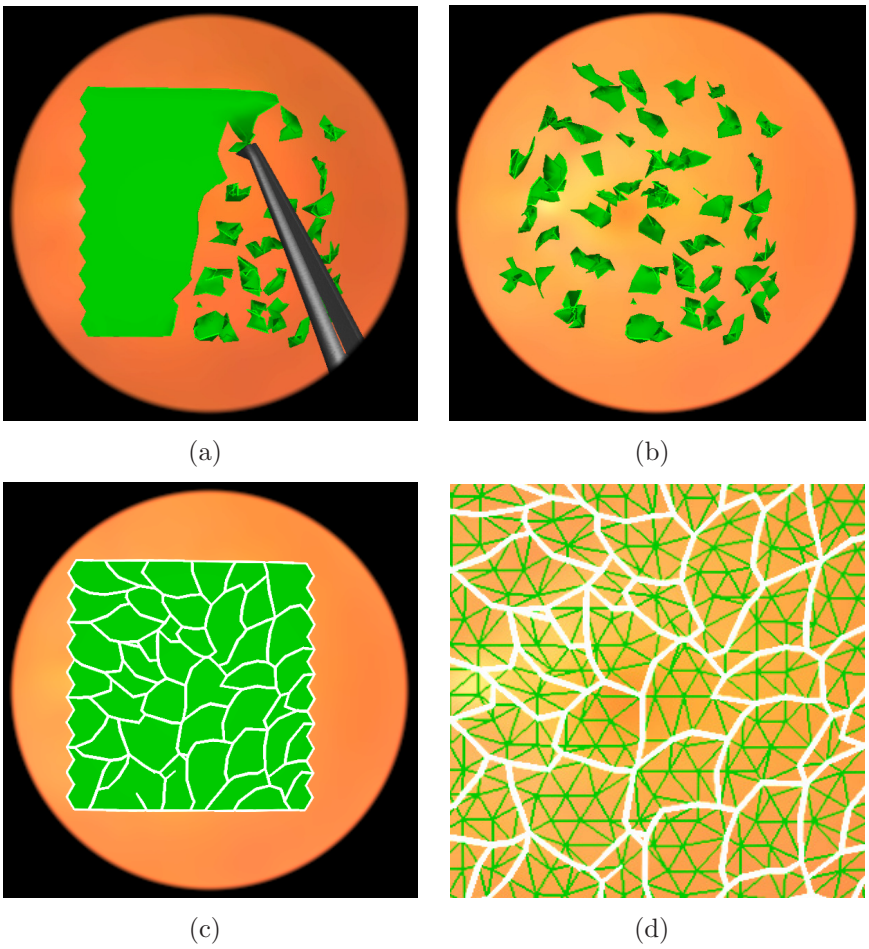
Rissrichtung passt sich erst dann der neuen Zug-Richtung an, wenn der Riss das Ende einer adaptierten Dreieckskante erreicht hat. Diese zeitliche Verzögerung kann dazu führen, dass der Benutzer die Zug-Richtung „übersteuert“ und der Riss seine Richtung anschließend unerwartet stark ändert. Da der kontinuierliche Ansatz die Rissrichtung im Zuge jedes einzelnen Risswachstums anpasst, erlaubt dieser eine bessere Kontrolle über die Rissausbreitung.

5.6.3 Die Re-Triangulierung mit Ansatz 2

Kantenbasiertes und pseudo-kontinuierliches Reißen halten die Anzahl an Dreiecken im Mesh konstant. Um die Risskante zu erzeugen, müssen nur Knoten und Federn verdoppelt werden. Wie das kontinuierliche Reißen die Mesh-Topologie verändert, wird im Folgenden anhand eines Beispiels veranschaulicht:
Das Mesh entspricht dem aus vorherigem Abschnitt und ist wiederum an seinen Rändern fixiert. Von diesem werden mit einer Pinzette kleine Stücke abgerissen bis es vollständig fragmentiert ist. In den Abbildungen 5.24(a) und 5.24(b) ist dieser Vorgang zu sehen. In Abb. 5.24(c) wurden die einzelnen Bruchstücke wie ein Puzzle wieder zum ursprünglichen Mesh zusammengesetzt. Hier ist zu sehen, dass der Reiß-Algorithmus keine Löcher oder sonstigen Artefakte entlang der Risskante erzeugt. Einen Ausschnitt der neu entstandenen Mesh-Topologie zeigt Abb. 5.24(d).
Zu Beginn der Simulation besteht das Mesh aus 367 Knoten, 1026 Federn und 660 Dreiecken. Wie sich die Anzahl an Elementen verändert hat, nachdem das Mesh vollständig fragmentiert wurde, ist Tabelle 5.2 zu entnehmen. So wurde die Anzahl an Dreiecken beispielsweise von 660 um 104 auf 764 erhöht. Ausschlaggebend für die Rechenzeitkosten der Feder-Masse-Simulation jedoch ist die Menge an Federn im Mesh. Im Verlauf des Reißens wurden 614 Federn erzeugt und 216 gelöscht, so dass sich die Anzahl an Federn um ca. 38.8% auf insgesamt 1424 erhöht hat. Da kein Vergleichs-Algorithmus implementiert wurde, wird an dieser Stelle auf die Arbeit von Nienhuys und van der Stappen [90, 91] verwiesen. Diese stellen den interaktiven Schneide-Algorithmus – auf dem der kontinuierliche Reiß-Algorithmus aufbaut – einem klassischen Subdivision-Algorithmus ge-

	Knoten	Federn	Dreiecke
Startanzahl	367	1026	660
Endanzahl	708	1424	764
erzeugt	413	614	248
gelöscht	72	216	144

Tabelle 5.2: Änderung der Anzahl an Mesh-Elementen durch Reißen



(a) (b)

(c) (d)

Abbildung 5.24: Änderung der Mesh-Topologie mit dem kontinuierlichen Reiß-Algorithmus

genüber, der Mesh-Elemente weder adaptiert noch löscht sondern ausschließlich verfeinert. Dieser Vergleich zeigt, dass ein reiner Subdivision-Algorithmus die Anzahl an Mesh-Elementen sehr viel stärker wachsen lässt. (Das virtuelle Skalpell wird durch ein Test-Mesh gezogen, das aus 50 Dreiecken besteht. Der Ansatz von Nienhuys und van der Stappen erhöht die Anzahl an Dreiecken um 10, der Subdivision-Algorithmus dagegen um 62.)

5.7 Zusammenfassung

In diesem Kapitel wurden zwei Algorithmen erarbeitet, mit denen deformierbare Oberflächen physikalisch plausibel gerissen werden können. Um eine Rissausbreitung zu berechnen, greifen beide Ansätze auf dieselben Bruchkriterien zurück. Sie unterscheiden sich jedoch darin, wie sie einen Riss topologisch umsetzen.
Laufzeit-Messungen haben ergeben, dass beide Reiß-Algorithmen für die Berechnung eines Rissfortschritts weniger als eine Viertel-Millisekunde Rechenzeit beanspruchen und für den Einsatz in einer Echtzeitanwendung geeignet sind.
Im Gegensatz zum bisher in EYESi verwendeten „Reißen entlang von Dreieckskanten“, kann sich der Riss unabhängig von der Mesh-Struktur ausbreiten und hinterlässt eine glatte Risskante, welche die dreiecksbasierte Topologie des Simulationsgitters verbirgt. Ein Risskeim ist stets mit zwei Randknoten verbunden, die durch identische Ruhepositionen miteinander korrespondieren. Dies garantiert, dass die topologische Umsetzung des Risses zwei parallel verlaufende Risskanten erzeugt.
Die Bruchkriterien basieren auf Verzerrungs- und Spannungsauswertungen, die in einer physikalisch plausiblen Rissausbreitung resultieren. Wenn der Benutzer mit einem Instrument an der Membran zieht, kann er den Riss auf einer vorgegeben Bahn führen. Dieses Ergebnis ist von besonderer Bedeutung, da ein Chirurg in der Lage sein muss, die simulierte Membran kontrolliert zu reißen. Der pseudo-kontinuierliche Ansatz allerdings berechnet stets erst dann eine neue Rissrichtung, wenn der Risskeim das Ende einer adaptierten Feder erreicht hat. Diese zeitliche Verzögerung erschwert die Kontrolle des Benutzers über die Rissausbreitung. Der kontinuierliche Reiß-Algorithmus dagegen erlaubt, die Rissrichtung im Zuge jedes einzelnen Risswachstums an den gewünschten Verlauf anzupassen. In der Test-Umgebung, welche eine Kreisbahn als Rissverlauf vorgibt, wurden daher mit dem kontinuierlichen Ansatz die besten Ergebnisse erzielt. Der bisher in EYESi verwendete, kantenbasierte Ansatz kann eine vorgegebene Rissbahn nur so gut approximieren, wie dies eine Folge von Federn im Mesh bereits im Voraus tut.
Die Rechenzeitkosten der Feder-Masse-Simulation hängen von der Anzahl an Federn im Mesh ab. Verglichen mit einem reinen Subdivision-Algorithmus – der Mesh-Elemente verfeinert, bis deren Grenzen die gewünschte Topologie beschreiben – generieren die beiden hier vorgeschlagenen Algorithmen neue Federn nur in geringem Maße: Eine temporäre Topologie des pseudo-kontinuierlichen Ansatzes erhöht die Anzahl an Federn im Mesh zunächst um Vier. Sobald sich der Riss um eine Federlänge vergrößert hat, werden drei dieser Federn wieder gelöscht. Der kontinuierliche Reiß-Algorithmus hat dieselben topologischen Eigenschaften wie der interaktive Schneide-Algorithmus von Nienhuys und van der Stappen [90, 91], der durch geeignete Mesh-Operationen – wie z. B. Kanten-Flippen – die Anzahl neu zu erzeugender Elemente minimiert.

Kapitel 6

Interaktionen zwischen Instrument und Membran

Zu jeder Operation gehören spezielle Instrumente, die dem Arzt den chirurgischen Eingriff überhaupt erst ermöglichen. Ob der Eingriff ohne Komplikationen gelingt, hängt wesentlich davon ab, wie geübt der Chirurg im Umgang mit dem jeweiligen Instrument ist. An einem VR-Trainingssystem kann ein Arzt seine Fähigkeiten nur dann verbessern, wenn die virtuellen Instrumente und das simulierte Weichgewebe realistisch aufeinander reagieren. Bei der Entwicklung einer Simulation für die medizinische Ausbildung ist daher die geeignete Implementierung einer Instrument-Interaktion grundsätzlich von besonderer Bedeutung.

Abschnitt 6.1 umreißt den medizinischen Hintergrund der in diesem Kapitel beschriebenen Implementierungen. Abschnitt 6.2 listet eine Reihe von Anforderungen auf, die eine virtuelle Instrument-Membran-Interaktion erfüllen sollte. In Abschnitt 6.3 werden Publikationen vorgestellt, die für die vorliegende Arbeit relevant sind. Anschließend folgt die Definition einer reibungsfreien Instrument-Membran-Interaktion (Abschnitt 6.4), die als Grundgerüst für die Gleitreibungs- und Haftreibungsmodelle dient (Abschnitt 6.5). Ergebnisse werden in Abschnitt 6.6 präsentiert und in Abschnitt 6.7 zusammengefasst.

6.1 Medizinischer Hintergrund

Die in der Ophthalmologie gebräuchlichen Instrumente sind sehr dünn und feingliedrig, da sie durch möglichst kleine Einschnitte in das Auge eingeführt werden müssen. Üblich sind z. B. Scheren, Pinzetten, Saug-Schneide-Geräte oder Laser-Instrumente für die Behandlung von Netzhautlöchern. Das vorliegende Kapitel konzentriert sich auf den Einsatz von nadelartigen Instrumenten, mit denen der

Abbildung 6.1: Peeling der ILM mit einem Schaber

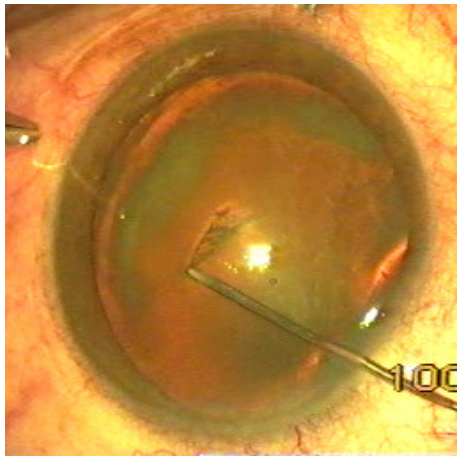


Abbildung 6.2: Anritzen des Kapselsacks mit einem Zystotom

Arzt flächig angewachsene Membrane im Auge manipuliert. Der Abschnitt „Medizinischer Hintergrund“ aus vorherigem Kapitel hat bereits einige chirurgische Eingriffe angesprochen, in denen Membrane aus dem Auge entfernt werden: Kapsulorhexis (Öffnen des Kapselsacks, der die Linse umschließt), Peeling der ILM (teilweise Entfernung der *Internal Limiting Membrane*), Peeling einer pathologischen Membran (komplette Entfernung einer auf der Netzhaut wuchernden Membran). In all diesen Eingriffen können nadelartige Instrumente zum Einsatz kommen:
Um Gewebe im Rahmen eines Membran-Peelings abzutragen, schiebt der Chirurg die Ablösekante der Membran mit der Instrument-Spitze in kleinen Bewegungen voran, wobei er darauf achten muss, die Netzhaut nicht zu verletzen. Abb. 6.1 zeigt den Beginn eines ILM-Peelings mit grün eingefärbter ILM. Manche der verwendeten Nadeln oder Schaber sind mit Diamantstaub behandelt und verfügen daher über eine extrem raue Oberfläche, die ein ständiges Abrutschen von der Membran verhindern soll.
In Abb. 6.2 ist zu sehen, wie der Kapselsack der Linse angeritzt wird, um eine Kapsulorhexis durchzuführen. Das eingesetzte Instrument ist ein *Zystotom* – eine Nadel mit umgeknickter, abgeflachter Spitze.
Für die genannten Eingriffe können Nadel-Instrumente und/oder Pinzetten verwendet werden. Welches Instrument ein Chirurg benutzt, ist zum einen eine Kostenfrage und zum anderen eine Frage der persönlichen Vorliebe.

6.2 Anforderungen an die Interaktionen

Die Besonderheit der hier behandelten Instrument-Membran-Interaktionen besteht darin, dass die eigentliche Interaktion sehr eng mit der Kollisionsbehandlung an der Instrument-Spitze verbunden ist. In Bezug auf Pinzetten, Saug-Instrumente oder Laser lassen sich instrumentspezifische Interaktion (Greifen, Absaugen, Lasern) und Kollisionsbehandlung am Instrument-Körper relativ klar voneinander trennen. Beim Lasern eines Netzhautlochs berühren sich Instrument und Gewebe sogar überhaupt nicht. Eine Nadel verfügt über keinen expliziten Interaktionsmechanismus, den der Benutzer wahlweise an- und ausschalten kann. Die Interaktion mit einem nadelartigen Instrument definiert sich vielmehr zu einem großen Teil implizit über die Kollisionsbehandlung.
Zielsetzung für dieses Kapitel ist es, eine Membran-Interaktion mit einem Instrument zu realisieren, dessen Spitze durch einen schmalen Zylinder geometrisch approximiert werden kann. Die Interaktion wird auf eine Wechselwirkung zwischen dem Mesh und einem undeformierbaren Zylinder reduziert und soll folgende Anforderungen erfüllen:

1. Die Interaktion muss für den Einsatz in einer Echtzeitanwendung geeignet sein.

2. Verglichen mit den Abmessungen der Mesh-Primitive ist der Durchmesser einer Instrument-Spitze sehr klein. Daher ist zwischen Mesh und Zylinder eine dreiecksbasierte Kollisionserkennung erforderlich, um alle Instrument-Membran-Kontakte ausreichend genau zu erfassen.

3. Die Wechselwirkung zwischen Membran und Zylinder soll sich einheitlich in das in Abschnitt 4.3 beschriebene Framework einfügen. Die Kollisionsantwort des Zylinders soll daher rein kraftbasiert sein.

4. Der Benutzer muss die Membran mit dem Instrument führen können, ohne permanent abzurutschen. Eine Hauptanforderung an die Interaktion besteht deshalb darin, ein geeignetes Modell für Reibung zu implementieren. Von besonderer Bedeutung ist die Realisierung einer Haftreibung, da erst diese ermöglicht, die zunächst anhaftende Membran abzulösen und vor dem Instrument herzuschieben.

5. Zunächst soll ein Modell für reibungsfreie Kontakte definiert werden. Dieses soll als algorithmische Basis für eine Interaktion dienen, in der Gleitreibung und/oder Haftreibung wahlweise an- und ausgeschaltet werden können.

6. Die Interaktion muss im Zusammenspiel mit den in Kapitel 5 beschriebenen Reiß-Algorithmen ein realistisches Membran-Verhalten erzeugen. Wenn das

Instrument ausgehend von einem bereits vorhandenen Riss oder Einschnitt durch die Membran geführt wird, muss die Kollisionsantwort der Interaktion dafür sorgen, dass sich das Mesh entlang der Instrument-Bewegung auftrennt.

6.3 Vergleichbare Arbeiten

Interaktionen zwischen dünnen Instrumenten und Weichgewebe sind typisch für minimal-invasive Eingriffe. Denn die Instrumente müssen einen möglichst kleinen Durchmesser haben, um durch möglichst kleine Einschnitte in den Körper eingeführt werden zu können. Die Abschnitte 6.3.1 und 6.3.2 stellen Publikationen vor, die sich mit der Entwicklung von Laparoskopie- und Brachytherapie-Simulatoren befassen und das Thema Instrument-Gewebe-Interaktion behandeln. Abschnitt 6.3.3 diskutiert Publikationen aus dem Bereich Textil-Simulation, da die Modelle für Reibung zwischen Stoff und Festkörper hilfreich sind für die Modellierung von Reibung zwischen Membran und Instrument. Abschließend beschreibt Abschnitt 6.3.4 Instrument-Interaktionen, wie sie bisher in EYESi realisiert wurden.

6.3.1 Laparoskopie-Simulation

Im Rahmen der laparoskopischen Chirurgie werden mit Hilfe eines optischen Instruments Eingriffe innerhalb der Bauchhöhle vorgenommen. Die häufigsten Interaktionen sind solche zwischen einem starren, stabförmigen Instrument und deformierbarem Gewebe, wie z. B. dem Magen.

Picinbono et al. [99] approximieren das jeweilige Instrument geometrisch durch einen Zylinder. Die Knoten kollidierender Dreiecke werden auf eine Ebene projiziert, die an den Zylinder angelegt wird. Die Normale der Projektionsebene entsteht durch Mittelwertbildung aus den Normalen der kollidierenden Dreiecke. Diese Kollisionsantwort wird im Anschluss an jeden Integrationsschritt durchgeführt. Da sich die Knoten während des Integrationsschritts unabhängig von der Instrument-Interaktion frei bewegen können, gleitet das Gewebe reibungsfrei entlang der Instrument-Oberfläche.
Forest et al. [39] verwenden ebenfalls einen Zylinder als Interaktionsobjekt. In einem mehrstufigen Prozess werden Verschiebungen für kollidierende Knoten, Kanten und Dreiecke so berechnet, dass die Mesh-Primitive den Zylinder auf einem möglichst kurzen Weg verlassen. Die Kollisionsbehandlung wird genau einmal pro Frame durchgeführt, und zwar nach Berechnung der Gewebe-Deformation und vor dem Rendern der aktuellen Szene.

In der Arbeit von Garcia-Perez et al. [50] ist die geometrische Approximation der Instrumente prinzipiell frei wählbar. Im Rahmen der Kollisionsantwort entscheidet eine Fuzzy-Logik über die Richtung notwendiger Knoten-Verschiebungen. Als Eingabe für das Fuzzy-System dienen ein Vektor, der die Instrument-Bewegung beschreibt, und ein Vektor, der die Oberflächennormale am Kollisionspunkt approximiert. Das Resultat ist eine Linearkombination dieser Vektoren, deren Koeffizienten gewichten, inwiefern die aktuelle Instrument-Bewegung als „Eindringen“, „Heraus ziehen“ oder als „Gleiten über die Oberfläche“ zu interpretieren ist. Wie in den beiden vorher genannten Arbeiten entsteht eine reibungsfreie Interaktion durch den ständigen Wechsel zwischen Simulieren und Verschieben der Mesh-Knoten.

Die beschriebenen Kollisionsantworten modellieren keine Reibung (Anforderung 4) und sind nicht kraft- sondern verschiebungsbasiert (Anforderung 3). Natürlich kann jeder verschiebungsbasierte Ansatz in einen kraftbasierten umformuliert werden, indem Knoten-Kräfte in Richtung der Verschiebung und proportional zur Länge der Verschiebung definiert werden. Allerdings enthalten alle genannten Algorithmen Komponenten, die voraussetzen, dass das Dreiecks-Mesh im Verlauf der gesamten Simulation die Oberfläche eines volumetrischen Objekts – z. B. des Magens – beschreibt. Die in der vorliegenden Arbeit modellierten Membrane haften nur zu Beginn der Simulation an der Oberfläche von umliegendem Gewebe. Ist die Membran einmal abgelöst, können obige Kollisionsantworten nicht mehr verwendet werden.
Bemerkenswert an der Arbeit von Garcia-Perez et al. [50] ist, dass die Instrument-Bewegung explizit in die Kollisionsantwort einfließt. Es wird sich zeigen, dass die in Abschnitt 6.5.2 beschriebene Haftreibung derart von der Instrument-Bewegung abhängt, dass es nicht notwendig ist, diese explizit als Parameter zu übergeben.

6.3.2 Brachytherapie-Simulation

Die Brachytherapie ist eine minimal-invasive Behandlung des Prostatakrebses. Weit verbreitet ist das Verfahren der Seed-Implantation, bei dem millimetergroße radioaktive Körner über Hohlnadeln in das vom Krebs befallene Gebiet eingeschossen werden. Da leicht Verletzungen entstehen können, wenn der Chirurg die Hohlnadel einführt, besteht hier der Bedarf an virtuellen Trainingsumgebungen.

Es existieren einige Publikationen, wie die von Nienhuys et al. [92], Alterovitz et al. [3] und Goksel et al. [54], die sich mit der Einführung einer Nadel in Weichgewebe befassen. Die Arbeiten unterscheiden sich z. B. darin, dass die Nadel mal als starres, mal als flexibles Objekt simuliert wird, oder dass das gesamte Modell mal zwei- und mal dreidimensional ist. Relevant für vorliegende Problemstellung

ist, wie die Wechselwirkung zwischen Nadel und Weichgewebe modelliert wird. Alle genannten Arbeiten realisieren die Reibung zwischen Instrument und Gewebe als *Stick-Slip-Effekt*. Dieser Begriff bezeichnet das ruckartige Gleiten von gegeneinander bewegten Körpern. Der Stick-Slip-Effekt tritt üblicherweise auf, wenn die Haftreibung größer ist als die Gleitreibung und führt zu einer schnellen Bewegungsfolge aus Haften, Verspannen, Trennen und Abgleiten. Die Implementierungen der Stick-Slip-Reibung in [92], [3] und [54] basieren auf dem Modell von DiMaio et al. [33] und unterscheiden sich nur geringfügig. Im Wesentlichen sind folgende Modell-Komponenten repräsentativ: Grundsätzlich sind diejenigen Mesh-Knoten von der Interaktion betroffen, die sich in direkter Umgebung der Nadel befinden. Ein endlicher Zustandsautomat ordnet jedem Interaktionsknoten entweder den Zustand *stick* oder den Zustand *slip* zu. Der Zustand eines Knotens wechselt von *stick* zu *slip*, wenn die aus der Mesh-Deformation resultierende Kraft auf den Knoten einen vordefinierten Grenzwert überschreitet. Umgekehrt geht ein Knoten vom Zustand *slip* in den Zustand *stick* über, wenn die Geschwindigkeit des Knotens relativ zum Instrument einen vordefinierten Grenzwert unterschreitet. Solange sich ein Knoten im Zustand *stick* befindet, wird dieser auf seiner aktuellen Position relativ zur Nadel fixiert. Ein Knoten im Zustand *slip* darf sich ausschließlich parallel zur Instrument-Achse bewegen.

Die prinzipielle Idee, Reibung zu realisieren, besteht hier darin, einen Knoten solange auf einer Relativposition zu fixieren, bis ihn interne Verzerrungskräfte von der Instrument-Oberfläche „abreißen“. Dieser Ansatz wird in Abschnitt 6.5.2 aufgegriffen, um die erwünschte Haftreibung zu definieren. Allerdings vernachlässigt obiges Stick-Slip-Modell das Volumen der Nadel, weshalb keine Kollisionsantwort orthogonal zur Instrument-Achse berechnet wird. Da die Nadel während einer Brachytherapie in Richtung ihrer Achse in das volumetrische Gewebe geschoben wird, ist diese Vereinfachung unproblematisch. Das für die vorliegende Arbeit entwickelte Reibungsmodell jedoch soll und muss die Durchdringung zwischen Mesh und Instrument-Körper berücksichtigen.

6.3.3 Vergleichbare Arbeiten in der Textil-Simulation

In den bisher genannten Arbeiten interagiert das virtuelle Instrument mit volumetrischen Objekten. Die Modellierung frei deformierbarer Oberflächen ist zentrales Thema der Textil- und Kleidungssimulation, weshalb dieses Forschungsgebiet auch für vorliegende Arbeit relevant ist. Von Interesse ist hier die Realisierung von Reibung zwischen Textilien und kollidierenden Körpern.

Um Reibung zu beschreiben, wird in der Kleidungssimulation üblicherweise auf das Coulomb'sche Gesetz zurückgegriffen. Dieses ist eine grobe Näherung und

besagt, dass am Kontaktpunkt zweier bewegter Körper eine tangentiale Reibungskraft $\vec{f}_\mathrm{r}$ wirkt, die proportional zur Normalkraft $\vec{f}_\mathrm{n}$ ist. Die Proportionalitätskonstante μ hängt von den aneinander reibenden Materialien ab und wird als Reibungskoeffizient bezeichnet.
Basierend auf diesem Zusammenhang definieren Fuhrmann [45] und Harmon et al. [59] Reibungsmodelle, die sich wiederum an das Modell von Bridson et al. [17] anlehnen. Charakteristisch ist, dass diese Arbeiten den Reibungseffekt nicht kraftbasiert beschreiben. Bridson et al. [17] und Harmon et al. [59] wählen einen impulsbasierten Ansatz und Fuhrmann [45] definiert ein verschiebungsbasiertes Modell. Die Adaptionen von Geschwindigkeiten bzw. Positionen werden in jedem Integrationsschritt für alle kollidierenden Mesh-Knoten vorgenommen. Für die impulsbasierte Lösung ist eine Veränderung $\Delta\vec{v}_\mathrm{t}$ der Tangentialgeschwindigkeit $\vec{v}_\mathrm{t}$ gesucht und für die verschiebungsbasierte Lösung eine tangentiale Veränderung $\Delta\vec{x}_\mathrm{t}$ der momentanen Knoten-Position $\vec{x}$. Die Adaptionen werden so gewählt, dass $|\Delta\vec{v}_\mathrm{t}|$ bzw. $|\Delta\vec{x}_\mathrm{t}|$ proportional zur Normalkraft $\vec{f}_\mathrm{n}$ sind, die an der aktuellen Knoten-Position wirkt. $\vec{f}_\mathrm{n}$ wiederum kann proportional zur Kollisionstiefe des Knotens ermittelt werden.

In Anlehnung an genannte Arbeiten wird in Abschnitt 6.5.1 eine Gleitreibung definiert, die ebenfalls auf dem Coulomb'schen Gesetz beruht. Diese jedoch wird kraftbasiert modelliert, um den in Abschnitt 6.2 genannten Anforderungen zu genügen.

6.3.4 Bisherige Interaktionen in EYESi

Zu Beginn der vorliegenden Arbeit existierten im EYESi-Framework bereits Implementierungen für den Gebrauch von Nadeln und Schabern. Um zu erläutern, weshalb diese den Anforderungen aus Abschnitt 6.2 nicht genügen, wird der Interaktionsmechanismus im Folgenden kurz zusammengefasst.

Die bisherige EYESi-Interaktion besteht aus folgender Iteration, die einmal pro Frame durchgeführt wird:

1. Das Instrument speichert die aktuellen Positionen aller Mesh-Knoten relativ zum Instrument-Körper.

2. Die Gewebe-Deformation wird im Rahmen des Simulationsschritts berechnet.

3. Das Instrument ermittelt alle Dreiecke, die seine Hüllkörper (OBBs) schneiden und extrahiert die kollidierenden Knoten aus den kollidierenden Dreiecken.

4. Alle kollidierenden Knoten werden auf die gespeicherten, alten Relativpositionen gesetzt.

5. Die virtuelle Szene wird gerendert.

In diesen Ansatz ist Reibung nicht explizit integriert. Da ein Knoten jedoch so lange auf seiner alten Relativposition zum Instrument fixiert wird, bis er nicht mehr kollidiert, erzeugt der Algorithmus automatisch einen Haftreibungseffekt. Eine solche Reibung ist zwar erwünscht, sollte aber parametrisierbar sein.
Der größte Nachteil dieser verschiebungsbasierten Methode besteht darin, dass sie nicht vollkommen zuverlässig arbeitet. Obiger Algorithmus geht davon aus, dass nach Schritt 4 alle Kollisionen aufgelöst sind. Doch wenn Knoten kollidierender Dreiecke verschoben werden, können Kanten von bis dahin nicht kollidierenden Dreiecken in den Instrument-Körper gezogen werden. Wenn dies passiert, beschreiben die Knoten-Positionen, die im anschließenden Schritt 1 gespeichert werden, keinen kollisionsfreien Zustand mehr. Die Folge ist ein unnatürliches Kleben der Membran, das in ungünstigen Fällen nur durch mehrere, schnelle Instrument-Bewegungen aufgelöst werden kann.

6.4 Reibungsfreie Interaktion

Die Wechselwirkung zwischen Membran und Instrument-Spitze wird auf eine Interaktion zwischen Mesh-Dreiecken und einem undeformierbaren, abgerundeten Zylinder reduziert. (Ist im weiteren Verlauf des Kapitels die Rede von einem Zylinder, ist grundsätzlich ein Zylinder mit abgerundeten Enden gemeint.)
Im Folgenden werden die Kollisionserkennung und die reibungsfreie Kollisionsantwort zwischen Mesh und Zylinder definiert, die als Basis für weitere Entwicklungen dienen.

6.4.1 Kollisionserkennung

Da die Instrument-Spitze mit einem Mesh-Dreieck kollidieren kann, ohne Knoten oder Kanten des Dreiecks zu berühren, wird für den Zylinder eine dreiecksbasierte Kollisionserkennung formuliert. Demnach besteht der Kollisionstest zwischen Mesh und Zylinder aus einer Reihe atomarer Tests, von denen jeder den Abstand zwischen einem einzelnen Dreieck und dem Zylinder überprüft.

Ein Zylinder ist definiert durch ein Liniensegment, das zusätzlich mit einem Radius r ausgestattet ist. Das Liniensegment selbst ist durch seine Endpunkte $\vec{u}$ und $\vec{v}$ gegeben. Um die Kollision zwischen einem Zylinder und einem Dreieck mit Eckpunkten $\vec{a}$, $\vec{b}$ und $\vec{c}$ zu erkennen, muss dessen Abstand zum Liniensegment

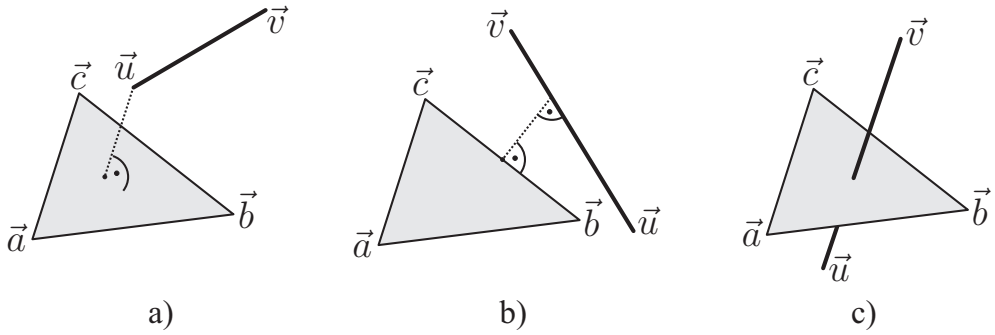


Abbildung 6.3: Ein Liniensegment nähert sich einem Dreieck

des Zylinders berechnet werden. Gesucht sind ein Punkt $\vec{s}$ auf dem Liniensegment und ein Punkt $\vec{d}$ auf dem Dreieck mit minimalem Abstand.
In Abb. 6.3 ist dargestellt, auf welche drei Arten sich ein Liniensegment einem Dreieck nähern kann. Sonderfälle, wie beispielsweise die parallele Ausrichtung von Liniensegment und Dreiecksebene, seien hier ausgeschlossen.
Abb. 6.3(a) zeigt eine Konfiguration, in der Liniensegment und Dreieck sich nicht gegenseitig durchdringen. Angenommen, der Punkt $\vec{d}$ liegt im Inneren und nicht auf dem Rand des Dreiecks. Der gesuchte Punkt $\vec{s}$ entspricht demnach einem Punkt des Liniensegments, dessen Projektion auf die Dreiecksebene mit $\vec{d}$ übereinstimmt. Daher muss $\vec{s} = \vec{u}$ oder $\vec{s} = \vec{v}$ gelten. Denn wenn man davon ausgeht, dass Liniensegment und Dreieck nicht parallel zueinander liegen, kann man von jedem Segmentpunkt aus, der zwischen $\vec{u}$ und $\vec{v}$ liegt, auf dem Segment näher Richtung Dreiecksoberfläche wandern bis man schließlich einen Endpunkt des Liniensegments erreicht. Es sei denn, die Projektion des wandernden Punktes gelangt bis an den Rand des Dreiecks. In diesem Fall jedoch muss auch $\vec{d}$ auf dem Rand des Dreiecks liegen, wie in Abb. 6.3(b) zu sehen. Wenn $\vec{d}$ mit einem Randpunkt des Dreiecks übereinstimmt, sind nicht nur die Enden sondern alle Punkte des Segments Kandidaten für $\vec{s}$. Der Abstand zwischen $\vec{d}$ und $\vec{s}$ entspricht in Abb. 6.3(b) dem Abstand zwischen Liniensegment $\vec{c}-\vec{b}$ und Liniensegment $\vec{v}-\vec{u}$.
Abb. 6.3(c) illustriert die letzte der drei möglichen Konfigurationen: Liniensegment und Dreieck durchdringen sich, so dass $\vec{s} = \vec{d}$ gilt.
Aus diesen Betrachtungen ergibt sich, dass folgende Tests durchgeführt werden müssen, um den Abstand zwischen einem Dreieck und einem Liniensegment zu berechnen:

- Test auf Durchdringung
- Test des Liniensegments gegen alle drei Dreieckskanten

- Test der beiden Segmentenden gegen die Dreiecksfläche

Die insgesamt sechs Tests[1] liefern jeweils ein Punktepaar $(\vec{d}, \vec{s})$. Eine Kollision zwischen Zylinder und Dreieck ist genau dann gegeben, wenn $|\vec{d} - \vec{s}|$ für mindestens ein Punktepaar $(\vec{d}, \vec{s})$ den Radius r des Zylinders unterschreitet. Die für die Kollisionsantwort benötigten Vektoren $\vec{d}$ und $\vec{s}$ sind diejenigen mit minimalem Abstand $|\vec{d} - \vec{s}| > 0$. Zu beachten ist, dass im Falle einer Durchdringung nicht das Punktepaar mit $\vec{d} = \vec{s}$ gespeichert wird. Die Begründung hierfür folgt im Abschnitt über die Kollisionsantwort.

Der Kollisionstest zwischen Zylinder und Mesh-Dreieck, bzw. die Berechnung nächstgelegener Punkte für Liniensegment und Mesh-Dreieck ist sehr rechenintensiv. Die der Gewebe-Deformation vorgeschaltete Broad Phase filtert daher solche Dreiecke, die sich nicht in der Nähe des Zylinders befinden. Im Rahmen der Broad Phase werden die Abstände aller Mesh-Knoten und die Abstände aller Dreiecksschwerpunkte zum Liniensegment des Zylinders berechnet. Ein Dreieck gelangt in die Narrow Phase, falls der Abstand seines Schwerpunkts oder der Abstand einer seiner drei Mesh-Knoten einen vorgegebenen Schwellwert unterschreitet. Der exakte, aber aufwändige Kollisionstest wird dann nur noch für die Dreiecke der Narrow Phase durchgeführt.

6.4.2 Kollisionsantwort

Im Folgenden wird eine reibungsfreie Kollisionsantwort für ein kollidierendes Dreieck mit Knoten $\vec{a}$, $\vec{b}$ und $\vec{c}$ definiert. Um gleichzeitig notwendige Informationen für die in Abschnitt 6.5.2 modellierte Haftreibung bereitzustellen, müssen für die drei Knoten des Dreiecks Ziel-Positionen $\vec{a}_\mathrm{t}$, $\vec{b}_\mathrm{t}$ und $\vec{c}_\mathrm{t}$ (Index „t“ für „target“) ermittelt werden. Diese Ziel-Positionen entstehen, indem man das Dreieck auf dem kürzesten Weg aus dem Zylinder mit Radius r herausschiebt.

Betrachten wir zunächst eine Kollision, die keine Durchdringung zwischen Dreieck und Liniensegment des Zylinders darstellt. Die Kollisionserkennung liefert ein Punktepaar $(\vec{d}, \vec{s})$ mit $\vec{d} \neq \vec{s}$, wobei $\vec{d} - \vec{s}$ senkrecht zur Oberfläche des Zylinders steht. In Abb. 6.4(a) z. B. liegt $\vec{d}$ auf dem Rand des Dreiecks. Das gesamte Dreieck wird in Richtung von $\vec{d} - \vec{s}$ aus dem Zylinder geschoben. Die Länge der Verschiebung entspricht der Kollisionstiefe und beträgt $r - |\vec{d} - \vec{s}|$. Wie am Beispiel in Abb. 6.4(b) zu sehen, landet $\vec{d}$ auf der Oberfläche des Zylinders und stellt den einzigen Kontaktpunkt zwischen Dreieck und Zylinder dar. Dieser Zustand wird bereits als kollisionsfrei betrachtet, alternativ kann die Verschiebung um einen

[1] In Bezug auf die Anzahl an Rechenoperationen optimierter Code für derartige Tests ist beispielsweise in Ericson [36] zu finden.

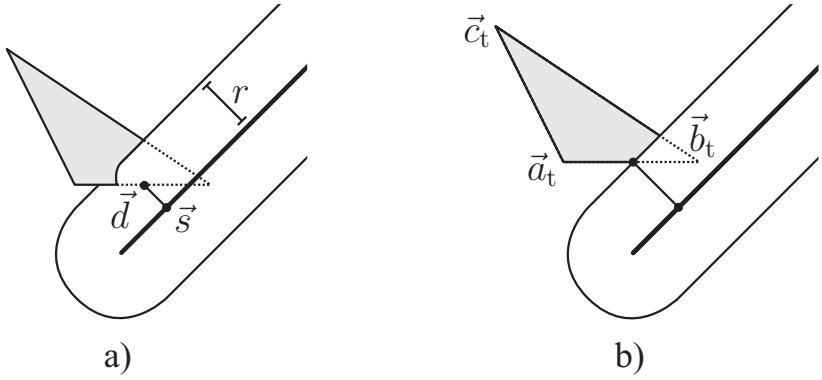


Abbildung 6.4: Ein Dreieck wird auf kürzestem Weg aus dem Zylinder geschoben

Faktor ϵ mit $0 < \epsilon \ll 1$ vergrößert werden.
Im Falle einer Durchdringung zwischen Liniensegment und Dreieck muss letzteres zunächst aus dem Segment herausgeschoben werden, bevor die Kollision mit dem Zylinder aufgelöst wird. Wie in Abb. 6.5 zu sehen, hängt die Richtung dieser Verschiebung nicht von der Lage des Durchstoßpunkts ab. Daher wurde im vorherigen Abschnitt 6.4.1 festgelegt, dass die Kollisionserkennung das Punktepaar $(\vec{d}, \vec{s})$ mit minimalem Abstand echt größer Null als Ergebnis liefert. Das Dreieck muss das Segment entweder über eine seiner drei Seiten (Abb. 6.5(b)) oder über eines der beiden Segmentenden (Abb. 6.5(a)) verlassen. In beiden Fällen ist die gesuchte Verschiebung durch $\vec{s} - \vec{d}$ gegeben. Um das Dreieck aus dem Zylinder zu entfernen, muss diese Verschiebung um den Radius r verlängert werden. Im Falle einer Durchdringung entspricht die Kollisionstiefe demnach nicht $r - |\vec{d} - \vec{s}|$ sondern $r + |\vec{s} - \vec{d}|$.

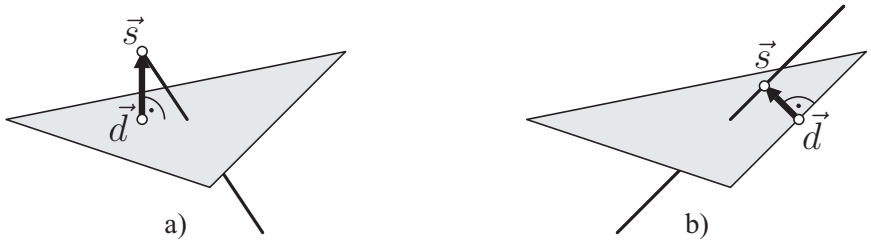


Abbildung 6.5: Im Falle einer Segment-Durchdringung muss das Dreieck zunächst aus dem Segment geschoben werden

Die Ziel-Positionen definieren für jeden Knoten des Dreiecks eine Kollisionsantwortkraft proportional zur Eindringtiefe:

$$\vec{f_a} = (\vec{a}_\mathrm{t} - \vec{a}) \cdot c \qquad \vec{f_b} = \left(\vec{b}_\mathrm{t} - \vec{b}\right) \cdot c \qquad \vec{f_c} = (\vec{c}_\mathrm{t} - \vec{c}) \cdot c$$

Hierbei ist c eine global festgelegte Konstante, welche die Stärke der Kollisionsantwortkräfte skaliert. Da die Kräfte senkrecht zur Oberfläche des Zylinders stehen und in jedem Integrationsschritt neu berechnet werden, entsteht eine vollkommen reibungsfreie Interaktion.

6.5 Modellierung von Reibung

Zusätzlich zu den reibungsfreien Kollisionsantwortkräften werden nun Kräfte für Gleitreibung und Kräfte für Haftreibung definiert. Für den Übergang zwischen Gleit- und Haftreibung wird die aktuelle Kollisionstiefe d mit einem Schwellwert $d_\mathrm{r} > 0$ verglichen. Gleitreibung wirkt nur, falls $0 < d \leq d_\mathrm{r}$, für $d > d_\mathrm{r}$ beginnt Haftreibung zu wirken.

6.5.1 Gleitreibung

Die momentane Eindringtiefe $d > 0$ zwischen einem Dreieck und einem Zylinder sei kleiner oder gleich dem vordefinierten Schwellwert d_r. In Anlehnung an die in Abschnitt 6.3.3 genannten Arbeiten wird eine Gleitreibung modelliert, die auf dem Coulomb'schen Gesetz beruht.

Das Gesetz von Coulomb beschreibt die Reibung zwischen zwei Körpern k1 und k2, die sich entlang ihrer Oberflächen gegeneinander bewegen. Die Relativgeschwindigkeit $\vec{v}_\mathrm{rel} = \vec{v}_\mathrm{k1} - \vec{v}_\mathrm{k2}$ am Kontaktpunkt entspricht der relativen Tangentialgeschwindigkeit $\vec{v}_\mathrm{t}$ (da sich die betrachteten Körper parallel zueinander bewegen) und sei ungleich 0. Nach Coulomb gilt für die auf k1 und k2 wirkenden Reibungskräfte $\vec{r}_\mathrm{k1}$ und $\vec{r}_\mathrm{k2}$:

$$\vec{r}_\mathrm{k1} \;=\; -\vec{r}_\mathrm{k2} \;=\; -\mu|\vec{f}_\mathrm{n}|\frac{\vec{v}_\mathrm{t}}{|\vec{v}_\mathrm{t}|}$$

$\vec{f}_\mathrm{n}$ entspricht der am Kontaktpunkt wirkenden Normalkraft und μ ist der sogenannte Reibungskoeffizient, dessen Wert von den aneinander reibenden Materialien abhängt.

Abb. 6.6(a) illustriert die Kollision zwischen einem Zylinder und einem Dreieck mit Eckpunkten $\vec{a}$, $\vec{b}$ und $\vec{c}$. Da sich Dreieck (Körper k1) und Zylinder (Körper

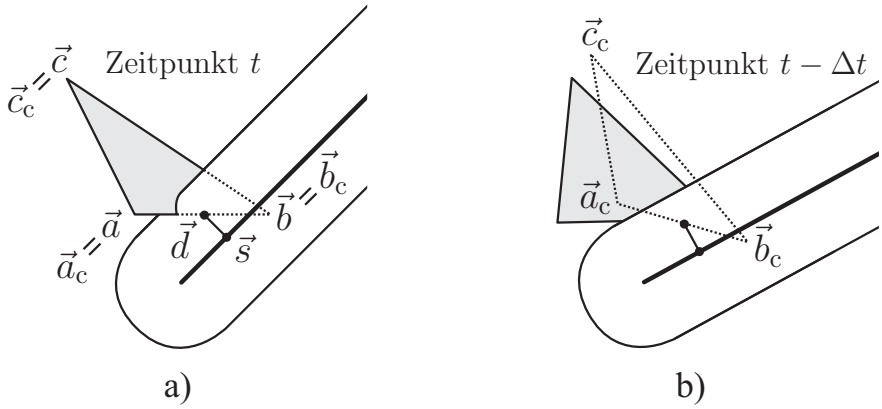


Abbildung 6.6: Bewegung der Dreiecksknoten $\vec{a}$, $\vec{b}$, $\vec{c}$ und Bewegung der am Zylinder fixierten Positionen $\vec{a}_c$, $\vec{b}_c$, $\vec{c}_c$ (der Index „c“ steht für „cylinder“)

k2) zum betrachteten Zeitpunkt t bereits gegenseitig durchdringen, existiert kein Kontaktpunkt im eigentlichen Sinn. Die Punkte $\vec{d}$ und $\vec{s}$ in Abb. 6.6(a) zeigen den Abstand zwischen Dreieck und Zylinder-Segment an. Daher dient $\vec{d}$ als Kontaktpunkt und die an $\vec{d}$ wirkende Normalkraft $\vec{f}_n$ wird proportional zur Eindringtiefe bestimmt:

$$\vec{f}_n = \frac{\vec{s} - \vec{d}}{|\vec{s} - \vec{d}|} \cdot (r - |\vec{s} - \vec{d}|) \cdot c$$

$$\left(\vec{f}_n = \frac{\vec{d} - \vec{s}}{|\vec{d} - \vec{s}|} \cdot (r + |\vec{d} - \vec{s}|) \cdot c \quad \begin{array}{l} \text{im Falle einer Durchdringung} \\ \text{zwischen Segment und Dreieck} \end{array} \right)$$

c ist die Skalierungskonstante der reibungsfreien Interaktion und r der Radius des Zylinders. $\vec{f}_n$ entspricht somit der negativen Kollisionsantwortkraft für reibungsfreie Interaktion.
Nach Coulomb fehlt noch die relative Tangentialgeschwindigkeit $\vec{v}_t$ am Kontaktpunkt. Diese kann bestimmt werden, indem man die Positionen von Dreieck und Zylinder zum Zeitpunkt t und zum Zeitpunkt $t - \Delta t$ miteinander vergleicht. Das endgültige Resultat dieser Berechnungen wäre eine (dreiecksbasierte) Reibungskraft $\vec{r}_{k1}$, die für den nächsten Integrationsschritt auf die Knoten des Dreiecks verteilt werden müsste. Allerdings kann man nicht davon ausgehen, dass die Geschwindigkeit des Kontaktpunkts repräsentativ ist für die Geschwindigkeiten der einzelnen Dreiecksknoten. Um der Tangentialbewegung des Dreiecks entgegen zu wirken, werden die drei Knoten-Kräfte $\vec{r}_a$, $\vec{r}_b$ und $\vec{r}_c$ daher separat berechnet: Die Positionen $\vec{a}$, $\vec{b}$ und $\vec{c}$ werden in Abb. 6.6(a) „am Zylinder fixiert“ (angedeutet durch Indizes „c“ für „cylinder“). In Abb. 6.6(b) ist zu sehen, wo sich $\vec{a}_c$, $\vec{b}_c$ und

$\vec{c}_{\mathrm{c}}$ zum Zeitpunkt $t - \Delta t$ befunden haben. Für die Relativgeschwindigkeit $\vec{v}_{\mathrm{rel}}$ an einem Eckpunkt $\vec{p} \in \{\vec{a}, \vec{b}, \vec{c}\}$ gilt:

$$\vec{v}_{\mathrm{rel}}(t) \;=\; \vec{v}_{\mathrm{k1}}(t) - \vec{v}_{\mathrm{k2}}(t)$$

$$= \frac{\vec{p}(t) - \vec{p}(t - \Delta t)}{\Delta t} - \frac{\vec{p}_{\mathrm{c}}(t) - \vec{p}_{\mathrm{c}}(t - \Delta t)}{\Delta t} \;=\; \frac{\vec{p}_{\mathrm{c}}(t - \Delta t) - \vec{p}(t - \Delta t)}{\Delta t}$$

Die gesuchte Geschwindigkeit $\vec{v}_{\mathrm{t}}(t)$ entspricht der Komponente von $\vec{v}_{\mathrm{rel}}(t)$ tangential zur Oberfläche des Zylinders. Als Oberflächennormale dient die Richtung von $\vec{f}_{\mathrm{n}}$. Sind die Tangentialgeschwindigkeiten $\vec{v}_{\mathrm{t}}(t)$ für alle drei Knoten bestimmt, können die Reibungskräfte $\vec{r}_a$, $\vec{r}_b$ und $\vec{r}_c$ nach dem Gesetz von Coulomb berechnet werden.
Zu beachten ist, dass eine hohe Eindringtiefe zu einer unrealistisch hohen Reibungskraft $\vec{r}$ führen kann, welche die Tangentialgeschwindigkeit $\vec{v}_{\mathrm{t}}$ eines Knotens nicht nur dämpft sondern sogar invertiert. Betragsmäßig wird eine Knoten-Kraft $\vec{r}$ daher wie folgt nach oben beschränkt, wobei m der Masse des Mesh-Knotens entspricht:

$$|\,\vec{r}\,| \leq m \cdot \frac{|\vec{v}_{\mathrm{t}}|}{\Delta t}$$

Die endgültig bestimmten Gleitreibungskräfte addieren sich auf die bereits berechneten reibungsfreien Kollisionsantwortkräfte:

$$\vec{f}_a := \vec{f}_a + \vec{r}_a \qquad \vec{f}_b := \vec{f}_b + \vec{r}_b \qquad \vec{f}_c := \vec{f}_c + \vec{r}_c$$

Bemerkung Die Geschwindigkeitsveränderung $\Delta\vec{v}$, die eine Kraft $\vec{f}$ im Verlauf eines Integrationsschritts verursacht, hängt vom verwendeten Integrationsverfahren ab. Eine Knoten-Kraft, die eine Komponente der Knoten-Geschwindigkeit exakt auslöschen soll, muss an die jeweiligen Integrationsgleichungen angepasst werden. Um diese Abhängigkeit zu vermeiden, wird mit dem hier vorgeschlagenen Modell in Kauf genommen, dass auch eine maximale Gleitreibung im Allgemeinen nicht in verschwindenden Tangentialgeschwindigkeiten resultiert.

6.5.2 Haftreibung

Für den Fall, dass die Kollisionstiefe d eines Dreiecks den vorgegebenen Grenzwert d_{r} überschreitet, soll Haftreibung wirken. Die Idee für das im Folgenden definierte Modell zeigt Gemeinsamkeiten mit den in Abschnitt 6.3.2 vorgestellten Arbeiten und mit der in Abschnitt 6.3.4 beschriebenen EYESi-Interaktion.

Die bisherige EYESi-Interaktion ist nicht aus der Motivation heraus entstanden, eine Haftreibung zu implementieren. Trotzdem zeigt sie einen solchen Effekt: Im

Anschluss an die Gewebe-Deformation werden Knoten kollidierender Dreiecke auf ihre alten Positionen relativ zum Instrument zurückgesetzt. Die gespeicherte Relativposition eines Knotens ändert erst dann ihren Wert, wenn kein Dreieck mehr kollidiert, das diesen Knoten enthält. Solange diese Bedingung nicht erfüllt ist, fixiert die Interaktion den Knoten an der Oberfläche des Instruments. In der hier vorgeschlagenen Lösung dienen die von der reibungsfreien Interaktion berechneten Ziel-Positionen $\vec{a}_\mathrm{t}$, $\vec{b}_\mathrm{t}$ und $\vec{c}_\mathrm{t}$ in Form von Relativpositionen als Basis für ein Modell, das Dreiecke am Zylinder anhaften lässt: Ähnlich wie in den Arbeiten aus Abschnitt 6.3.2 wird einem Dreieck entweder der Zustand *slip* oder der Zustand *stick* zugeordnet. Solange sich ein Dreieck im Zustand *slip* befindet, wirken reibungsfreie Kollisionsantwortkräfte und Gleitreibungskräfte. Sobald ein Dreieck eine Eindringtiefe $d > d_\mathrm{r}$ erreicht, wechselt sein Zustand von *slip* zu *stick*. In diesem Moment werden die aktuellen Ziel-Positionen $\vec{a}_\mathrm{t}$, $\vec{b}_\mathrm{t}$ und $\vec{c}_\mathrm{t}$ der reibungsfreien Interaktion in das lokale Koordinatensystem des Zylinders umgerechnet und als Relativpositionen $\vec{a}_\mathrm{c}$, $\vec{b}_\mathrm{c}$ und $\vec{c}_\mathrm{c}$ gespeichert. Während sich ein Dreieck im Zustand *stick* befindet, wirken ausschließlich Haftreibungskräfte. Diese beschleunigen die Dreiecksknoten in Richtung der abgespeicherten Zylinder-Relativpositionen:

$$\vec{f_a} = [\mathrm{transformToMeshReferenceSystem}(\vec{a}_\mathrm{c}) - \vec{a}\,] \cdot c$$

$$\vec{f_b} = [\mathrm{transformToMeshReferenceSystem}(\vec{b}_\mathrm{c}) - \vec{b}\,] \cdot c$$

$$\vec{f_c} = [\mathrm{transformToMeshReferenceSystem}(\vec{c}_\mathrm{c}) - \vec{c}\,] \cdot c$$

Die Vektoren $\vec{a}$, $\vec{b}$ und $\vec{c}$ entsprechen den aktuellen Knoten-Positionen, c ist die Skalierungskonstante der reibungsfreien Interaktion. Zu beachten ist, dass für jede Berechnung der Haftreibungskräfte eine Transformation der Relativpositionen aus dem lokalen Koordinatensystem des Zylinders in das lokale Koordinatensystem des Simulationsgitters vorgenommen werden muss. Sobald das betroffene Dreieck nicht mehr kollidiert, wechselt sein Zustand wieder von *stick* zu *slip*. Darüber hinaus verlässt ein Dreieck den Zustand *stick*, wenn es von der Oberfläche des Zylinders „abreißt“: Alle Federkräfte, die auf das Dreieck wirken, werden aufsummiert. Verfügt die resultierende Kraft über eine Komponente tangential zur Oberfläche des Zylinders, deren Betrag einen vorgegebenen Grenzwert überschreitet, löst sich das Dreieck. Möglicherweise geht dieses Dreieck in der nächsten Iteration sofort wieder in den Zustand *stick* über, dann allerdings auf Basis aktualisierter Relativpositionen.

Algorithmus 4 beschreibt die Implementation der Haftreibung im Rahmen der vollständigen Zylinder-Mesh-Interaktion:
Zeilen 1 bis 4: Theoretisch ist es möglich (wenn auch äußerst unwahrscheinlich), dass der letzte Integrationsschritt eines Frames ein Dreieck im Zustand *stick* so

Algorithm 4 Implementation of cylinder-mesh-interaction

// Triangles far away must not stick
for all triangles $t \notin$ broadPhase **do**
  state(t) $\leftarrow$ *slip*
end for
// Narrow Phase Collision Detection and Collision Response
for all triangles $t \in$ broadPhase **do**
  // Calculate nearest points and collision depth
  $(\vec{d}, \vec{s})$, interpenetration $\in \{-1, +1\} \leftarrow$ narrowPhaseCollisionDetection()
  collDepth $\leftarrow$ cylinderRadius + interpenetration $\cdot\, |\vec{d} - \vec{s}|$
  // Check if triangle state must change to *slip*
  if state(t) = *stick* AND [collDepth ≤ 0 OR forcesTearTriangleOff()] **then**
    state(t) $\leftarrow$ *slip*
  end if
  // No collision detected?
  if collDepth ≤ 0 **then**
    continue with next iteration
  end if
  // Collision response with static friction?
  if state(t) = *stick* **then**
    $\vec{f_a} \leftarrow \vec{f_a} + [\text{transformToMeshReferenceSystem}(t.\vec{a}_{\text{cyl}}) - \vec{a}\,] \cdot c$
    $\vec{f_b} \leftarrow \vec{f_b} + [\text{transformToMeshReferenceSystem}(t.\vec{b}_{\text{cyl}}) - \vec{b}\,] \cdot c$
    $\vec{f_c} \leftarrow \vec{f_c} + [\text{transformToMeshReferenceSystem}(t.\vec{c}_{\text{cyl}}) - \vec{c}\,] \cdot c$
    continue with next iteration
  end if
  // Calculate target positions (within mesh reference system)
  $(\vec{a}_{\text{tar}}, \vec{b}_{\text{tar}}, \vec{c}_{\text{tar}}) \leftarrow$ frictionlessCollisionResponse()
  // Collision response with dynamic friction
  $\vec{f_a} \leftarrow \vec{f_a} + (\vec{a}_{\text{tar}} - \vec{a}) \cdot c + \text{dynamicFrictionForce}(\vec{a})$
  $\vec{f_b} \leftarrow \vec{f_b} + (\vec{b}_{\text{tar}} - \vec{b}\,) \cdot c + \text{dynamicFrictionForce}(\vec{b})$
  $\vec{f_c} \leftarrow \vec{f_c} + (\vec{c}_{\text{tar}} - \vec{c}\,) \cdot c + \text{dynamicFrictionForce}(\vec{c})$
  // Check if triangle state must change to *stick*
  if state(t) = *slip* AND collDepth > threshold **then**
    state(t) $\leftarrow$ *stick*
    $t.\vec{a}_{\text{cyl}} \leftarrow \text{transformToCylinderReferenceSystem}(\vec{a}_{\text{tar}})$
    $t.\vec{b}_{\text{cyl}} \leftarrow \text{transformToCylinderReferenceSystem}(\vec{b}_{\text{tar}})$
    $t.\vec{c}_{\text{cyl}} \leftarrow \text{transformToCylinderReferenceSystem}(\vec{c}_{\text{tar}})$
  end if
end for

weit vom Zylinder entfernt, dass die Broad Phase des folgenden Frames dieses Dreieck aussortiert. Wie durch die ersten vier Zeilen angedeutet, muss daher sichergestellt werden, dass solche Dreiecke den Zustand *slip* erhalten.
Zeilen 7 bis 9: Für Dreieck t werden Dreieckspunkt $\vec{d}$ und Segmentpunkt $\vec{s}$ mit minimalem Abstand (> 0) berechnet. Falls sich Segment und Dreieck gegenseitig durchdringen, erhält die Variable „interpenetration“ den Wert $+1$, andernfalls den Wert -1. Mit Hilfe dieser Variable kann die Kollisionstiefe „collDepth“ korrekt bestimmt werden.
Zeilen 10 bis 13: Ein Dreieck wechselt seinen Zustand von *stick* zu *slip*, falls es nicht mehr kollidiert oder von der Zylinder-Oberfläche „abreißt“.
Zeilen 14 bis 17: Falls Dreieck t nicht kollidiert, kann der aktuelle Schleifendurchlauf abgebrochen werden.
Zeilen 18 bis 24: Falls sich Dreieck t im Zustand *stick* befindet, werden ausschließlich Haftreibungskräfte berechnet und der aktuelle Schleifendurchlauf wird anschließend abgebrochen. Die Schreibweise $t.\vec{a}_{\text{cyl}}$, $t.\vec{b}_{\text{cyl}}$ bzw. $t.\vec{c}_{\text{cyl}}$ deutet an, dass die Relativpositionen, auf denen die Haftreibung basiert, im Objekt t abgespeichert werden.
Zeilen 25 bis 26: Falls der Schleifendurchlauf diese Zeilen erreicht, müssen Ziel-Positionen für eine reibungsfreie Kollisionsantwort ermittelt werden.
Zeilen 27 bis 30: Jeder Dreiecksknoten wird in Richtung seiner Ziel-Position und entgegen seiner momentanen Tangentialgeschwindigkeit beschleunigt.
Zeilen 31 bis 37: Ein Dreieck wechselt seinen Zustand von *slip* zu *stick*, wenn seine Kollisionstiefe den Grenzwert für Haftreibung überschreitet. In diesem Fall werden die aktuellen Ziel-Positionen der reibungsfreien Kollisionsantwort in das lokale Koordinatensystem des Zylinders umgerechnet und im Objekt t abgespeichert. Somit ist sichergestellt, dass die Relativpositionen gültige Werte haben, wenn sie in den Zeilen 20 bis 22 verwendet werden.

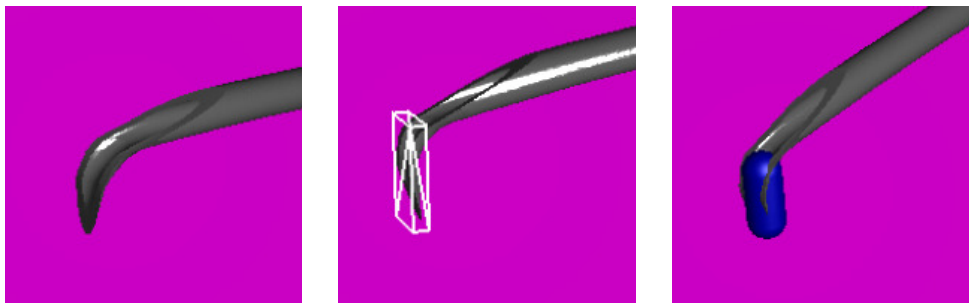
Bemerkung Wenn sich ein Dreieck im Zustand *stick* befindet, wird erzwungen, dass dieses den Zylinder über eine bestimmte Relativposition verlässt, sofern es nicht vorher von der Oberfläche abreißt. Wenn sich der am Instrument hängende Zylinder in Richtung der Relativposition und damit tendenziell in Richtung der aktuellen Position des Dreiecks bewegt, resultiert dies in einer fortdauernden Kollision und das Dreieck wird vor dem Instrument hergeschoben. Im Allgemeinen muss der Benutzer das Instrument aus der Membran herausbewegen, um die Haftreibung aufzulösen. Eine explizite Auswertung der Instrument-Bewegung für die Kollisionsbehandlung, so wie in der Arbeit von Garcia-Perez et al. [50], ist daher nicht erforderlich.

6.6 Ergebnisse

Die Instrument-Interaktion ist nun vollständig definiert. Ihre Kollisionserkennung ist dreiecksbasiert, ihre Kollisionsantwort kraftbasiert. Gleit- und Haftreibung lassen sich über Parameter steuern und können wahlweise an- und abgeschaltet werden. Die in Abschnitt 6.2 formulierten Anforderungen 2, 3 und 5 an die Implementation sind daher erfüllt. In den folgenden Abschnitten wird untersucht, inwiefern die Instrument-Interaktion den übrigen Anforderungen genügt. Hierbei wird sie mit der bisherigen EYESi-Interaktion verglichen.
Für die Test-Anwendungen wird ein Zystotom eingesetzt, wie es in Abb. 6.7(a) zu sehen ist. Die bisherige EYESi-Interaktion approximiert die Spitze des Instruments durch eine Bounding Box (Abb. 6.7(b)), die neu vorgeschlagene Interaktion durch einen abgerundeten Zylinder (Abb. 6.7(c)). Der Zylinder hat einen Radius von 0.2mm und ist inklusive Kappen 0.98mm lang.

Bei der Test-Umgebung handelt es sich um ein entkerntes EYESi-Trainingsmodul, das prinzipiell folgendermaßen konfiguriert bzw. parametrisiert ist:

- Das Zystotom wird nicht per Hand bewegt. Stattdessen wird seine Modelview in jedem Frame explizit gesetzt, so dass es sich auf einer vorgegebenen Bahn durch das Mesh bewegt.
- Für die Mesh-Simulation gelten folgende Angaben:
  - Jeder Mesh-Knoten verfügt über die Masse $m = 0.01\text{kg}$.
  - Jede Feder hat die Federkonstante $k_\text{F} = 6500\text{N/m}$.
  - Jeder Simulationsschritt zerfällt in 25 Integrationsschritte.
- Für die neue Interaktion gelten folgende Angaben:



(a) (b) (c)

Abbildung 6.7: Geometrische Approximation der Spitze eines Zystotoms

Abbildung 6.8: Knoten-spezifische Messgrößen wie z. B. Knoten-Spannungen werden auf Wellenlängen zwischen 380nm und 780nm abgebildet. Somit entsteht im Mesh ein Farbverlauf von Violett (niedrig) über Grün (mittel) zu Rot (hoch).

  - Für die Konstante k_I, welche die reibungsfreie Kollisionsantwortkraft skaliert, gilt $k_\mathrm{I} = k_\mathrm{F}$.
  - Gleitreibung wird maximal bei einer Kollisionstiefe $d = 0.04$mm. Falls ein Dreieck tiefer in den Zylinder eindringt, wirkt Haftreibung.
  - Für den Reibungskoeffizienten der Gleitreibung gilt $\mu = 1.2$.
  - Ein Dreieck, auf das Haftreibung wirkt, geht vom Zustand *stick* in den Zustand *slip* über, falls an ihm eine tangentiale Kraft zieht, die größer als $h = 3.0$N ist.

- Sollen Knoten-spezifische Messgrößen sichtbar gemacht werden, wird das Mesh in Falschfarben dargestellt. Abb. 6.8 zeigt die verwendete Farbskala.

Falls die Test-Umgebung im Folgenden von diesen Angaben abweicht, wird an entsprechender Stelle darauf hingewiesen.

6.6.1 Laufzeit

Die Messungen wurden auf einem Rechner mit Intel(R) Core(TM)2 Quad CPU Q9300 und 2.5GHz unter Windows XP durchgeführt. Um Störungen zu minimieren, wurden alle unnötigen Dienste angehalten und das Programm für die Laufzeit-Messung als einzige Anwendung gestartet.

Sowohl bisherige als auch neue Interaktion definieren eine Broad Phase, die einmal pro Frame vor Beginn der Mesh-Deformation durchgeführt wird. Die Narrow Phase der bisherigen Interaktion wird ebenfalls einmal pro Frame und zwar im Anschluss an die Deformation berechnet, die Narrow Phase der neuen Interaktion dagegen vor jedem einzelnen Integrationsschritt. In den Test-Anwendungen wird bei beiden Interaktionen auf die Durchführung einer Broad Phase verzichtet:
Die bisherige EYESi-Interaktion approximiert die Geometrie eines Instruments normalerweise durch mehrere Narrow Boxen, die von einer größeren Broad Box umschlossen sind. Da hier, wie in Abb. 6.7(b) zu sehen, nur eine einzige Narrow Box verwendet wird, erübrigt sich die Definition einer Broad Box und damit die

Berechnung einer Broad Phase.
Wieviel Laufzeit die neue Interaktion pro Frame beansprucht, hängt wesentlich davon ab, wieviele Dreiecke von der vorsortierenden Broad Phase in die rechenintensive Narrow Phase gelangen. Dies wiederum hängt davon ab, wieviele Dreiecke sich zu Beginn des Frames in direkter Umgebung der Instrument-Spitze befinden. Um das Zeitverhalten des „Flaschenhalses“ der Interaktion zu untersuchen, werden hier grundsätzlich *sämtliche* Dreiecke des Gitters an die Narrow Phase des Zylinders übergeben.

Um Messungen durchzuführen, wurde eine Test-Umgebung implementiert, wie sie in Abb. 6.11 zu sehen ist: Das Zystotom wird in einer automatisierten Bewegung im Verlauf von 100 Frames von links nach rechts durch ein fein aufgelöstes Mesh gezogen. Da der endgültige Rechenaufwand für ein einzelnes Dreieck davon abhängt, ob dieses tatsächlich kollidiert oder nicht, wurde das Mesh in Form eines schmalen Streifens generiert. Auf diese Weise wird erreicht, dass am Ende der Instrument-Bewegung nahezu alle Dreiecke mit der Instrument-Spitze kollidieren und die Berechnung einer Kollisionsantwort erfordern.

Zunächst sollen bisherige und neue Interaktion (inklusive Gleit- und Haftreibung) exemplarisch miteinander verglichen werden. Wie in Abb. 6.11 zu sehen, wurde ein Mesh bestehend aus 342 Dreiecken gewählt. Pro Frame wird die Mesh-Deformation in 25 Integrationsschritten berechnet, auch die Narrow Phase der neuen Interaktion wird daher 25 mal pro Frame durchgeführt.
In Abb. 6.9 sind die Rechenzeiten pro Frame der beiden Interaktionen gegeneinander aufgetragen. Beide Kurven steigen im Verlauf der 100 Frames an, da im Verlauf der Instrument-Bewegung immer mehr Dreiecke kollidieren. Während

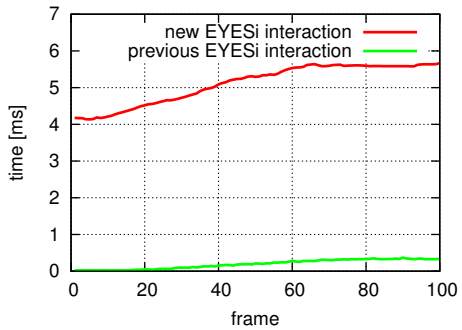


Abbildung 6.9: Die bisherige und die neue Interaktion im Vergleich bei einer Mesh-Größe von 342 Dreiecken

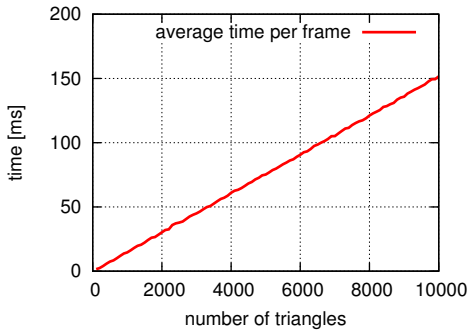
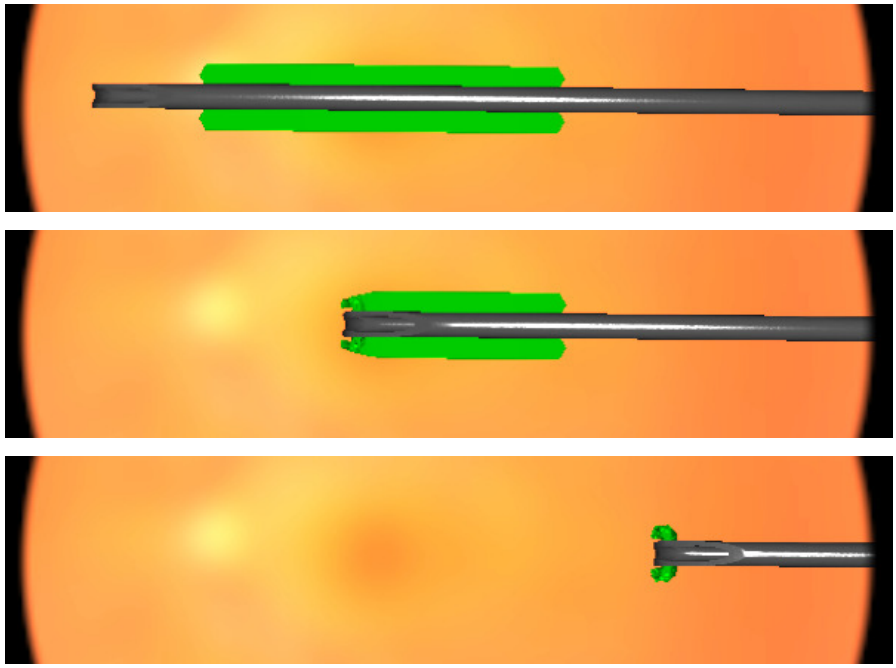
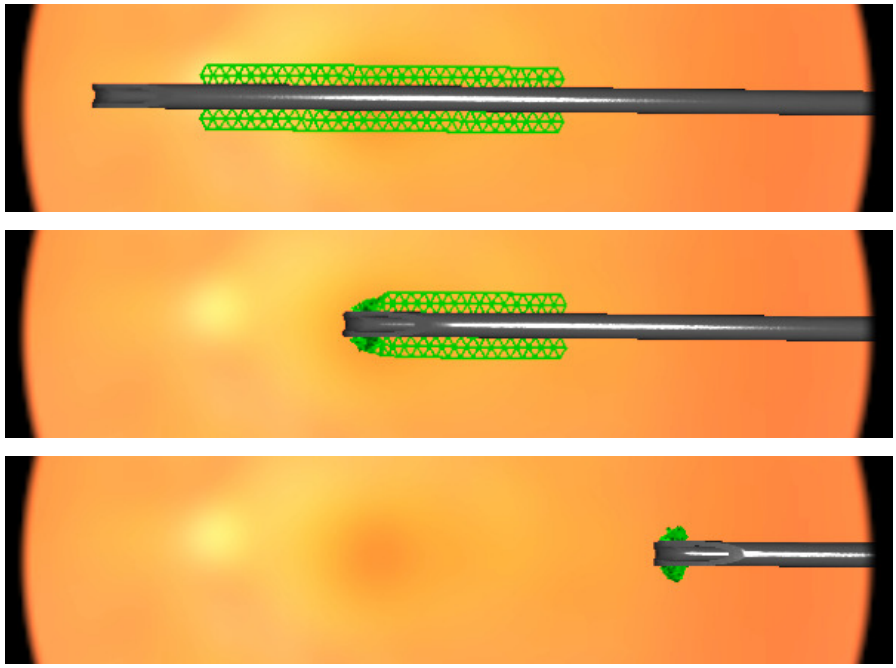


Abbildung 6.10: Die Rechenzeit für die (neue) Interaktion skaliert linear mit der Mesh-Größe



(a) bisherige Interaktion



(b) neue Interaktion

Abbildung 6.11: Test-Umgebung für die Laufzeit-Messung, hier mit einem Mesh aus 342 Dreiecken

die Rechenzeit pro Frame der bisherigen Interaktion deutlich unter 1ms bleibt, bewegt sich die Kurve der neuen Interaktion zwischen 4ms und 6ms. Dieser Unterschied ist verständlich, wenn wann man bedenkt, dass die neue Interaktion pro Frame $25 \cdot 342 = 8550$ mal nächstgelegene Punkte zwischen einem Dreieck und einem Zylinder bestimmt.

Es liegt nahe anzunehmen, dass die Laufzeit für die neue Interaktion proportional mit der Anzahl an involvierten Dreiecken anwächst. Um dies zu belegen, wurde der in Abb. 6.11 dargestellte Versuch mit 100 verschiedenen Mesh-Auflösungen durchgeführt: Die Anzahl an Dreiecken im Mesh startet mit 100 und wird in 100er-Schritten bis auf 10000 erhöht. In jedem der 100 Versuche wurde die durchschnittliche Rechenzeit pro Frame als Messwert ermittelt. Diese Werte sind in Abb. 6.10 gegen die Mesh-Größe aufgetragen und bilden eine Gerade, die eine Steigung von ca. 0.015 hat. Pro Dreieck beansprucht die neue Interaktion demnach durchschnittlich 0.015ms Rechenzeit pro Frame.

Die Messungen ergeben, dass die neue Interaktion prinzipiell für den Einsatz in einer Echtzeitanwendung geeignet ist. Allerdings muss eine angemessene Mesh-Auflösung gewählt werden und auf den Einsatz einer Broad Phase sollte nicht verzichtet werden. Denn der in der Test-Umgebung künstlich konstruierte Worst Case, in dem sich sämtliche Mesh-Dreiecke in der Narrow Phase befinden, kommt normalerweise nicht vor. Im Allgemeinen interagiert nur ein Bruchteil aller Mesh-Dreiecke mit der Instrument-Spitze.

6.6.2 Die Interaktionen im Vergleich

Im Folgenden wird die Funktionalität der neuen Interaktion anhand verschiedener Parametrisierungen getestet und mit derjenigen der bisherigen EYESi-Interaktion verglichen. In der hierfür verwendeten Test-Umgebung bewegt sich das Zystotom auf einer Kreisbahn, die senkrecht zum Mesh steht, so dass die Instrument-Spitze über die Mesh-Oberfläche hinweg streift. Das Gitter besteht aus 367 Knoten, 1026 Federn und 660 Dreiecken. Die Federn haben eine durchschnittliche Ruhelänge von 0.5mm. Um das Mesh schwach zu fixieren, verfügt jeder Randknoten über eine Homing-Position, die mit seiner Startposition übereinstimmt und über eine Homing-Konstante von 50N/m. Diese Homing-Kräfte wirken permanent, unabhängig davon, wie weit sich der Randknoten von seiner Homing-Position entfernt. Um die Effekte der verschiedenen Reibungsarten voneinander abzugrenzen, werden Gleit- und Haftreibung nicht gleichzeitig aktiviert. D. h. entweder ist $d = 0.0$ oder $d = +\infty$, wobei der Parameter d bestimmt, ab welcher Kollisionstiefe Gleitreibung in Haftreibung übergeht. Für den Reibungskoeffizient der Gleitreibung wird neben $\mu_{\text{high}} = 1.2$ auch $\mu_{\text{low}} = 0.2$ eingesetzt. Die Haftreibung

wird in insgesamt drei Abstufungen betrachtet: $h_{\text{low}} = 0.3\text{N}$, $h_{\text{medium}} = 3\text{N}$ und $h_{\text{high}} = 30\text{N}$. Das Mesh wird in Falschfarben dargestellt, um Spannungen in den Knoten sichtbar zu machen.

In Abb. 6.14(a) ist der Versuch mit der bisherigen EYESi-Interaktion zu sehen: Das Instrument bewegt sich von links nach rechts durch das Mesh, wobei die verschiebungsbasierte Kollisionsantwort der Interaktion einen unnatürlich aussehenden Graben im Mesh entstehen lässt. Die Verschiebungen, die erst nach Abschluss der Mesh-Deformation berechnet werden, resultieren in sehr hohen Spannungen (rot) und diese wiederum können zu Stabilitätsproblemen führen. (Um die Test-Anwendung mit der bisherigen EYESi-Interaktion durchführen zu können, wurde die Anzahl an Integrationsschritten auf 100 erhöht. Mit 25 Integrationsschritten war die Simulation instabil.) In Abb. 6.12 ist die durchschnittliche Knoten-Spannung im Mesh gegen die Zeit aufgetragen. Die Spannungsspitze markiert den Zeitpunkt, zu dem sich das Gitter vom Instrument losgerissen hat.
Derselbe Versuch wurde mit neuer Interaktion und Haftreibung h_{high} wiederholt (Abb. 6.14(b)): Wie in Abb. 6.14(a) verhakt sich das Zystotom im Mesh und zieht es hinter sich her. Die Falten, die um die Instrument-Spitze herum entstehen, wirken allerdings natürlicher. Hier zeigt sich, dass es mit Hilfe der Haftreibung möglich ist, hohe Zug-Kräfte auf das Mesh auszuüben ohne von der Oberfläche abzurutschen. Wie in Abb. 6.12 zu sehen, fallen die Spannungen, die im Mesh erzeugt werden, deutlich niedriger aus, obwohl die Haftreibung so hoch gewählt wurde, dass das Mesh bis zum Stopp der Instrument-Bewegung mitgezogen wurde. (Dies ist daran zu erkennen, dass die Kurve zum rechten Ende der x-Achse hin ansteigt und dort ihre Spannungsspitze erreicht.)
Wird die Haftreibung auf h_{medium} erniedrigt, ergibt sich die in Abb. 6.14(c) dargestellte Mesh-Deformation: Besonders am linken Rand des Gitters sind die

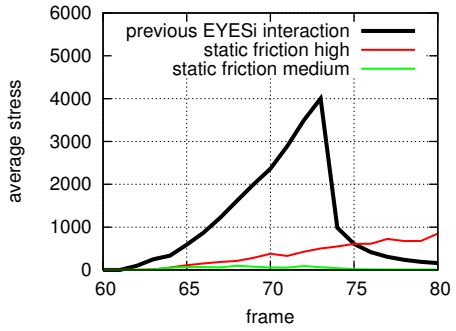


Abbildung 6.12: Bisherige und neue Interaktion im Vergleich

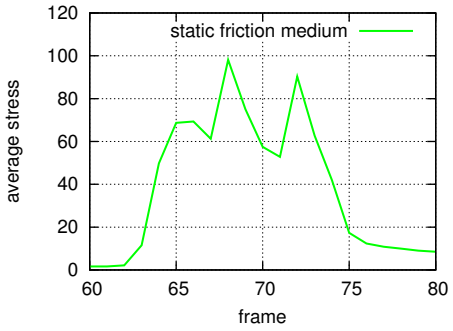
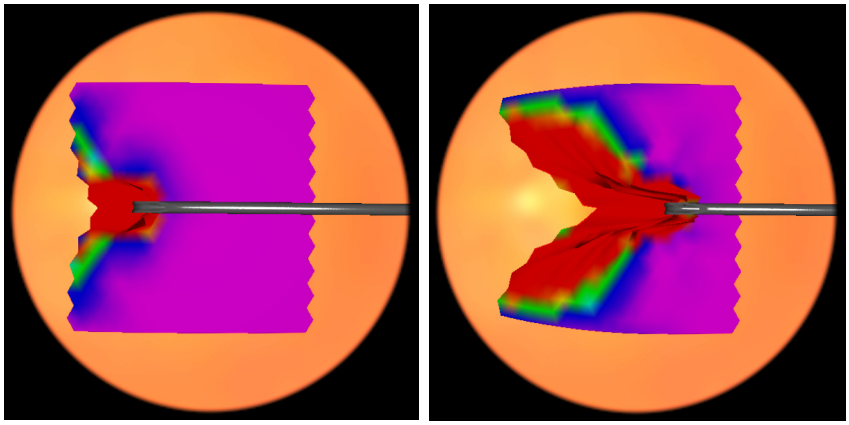
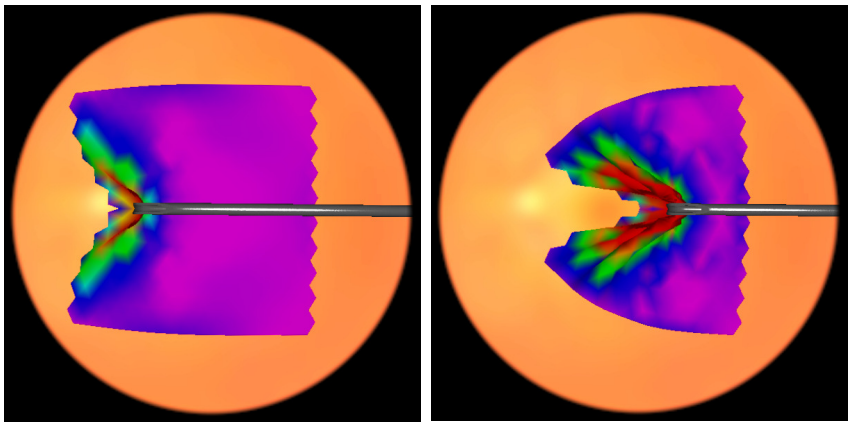


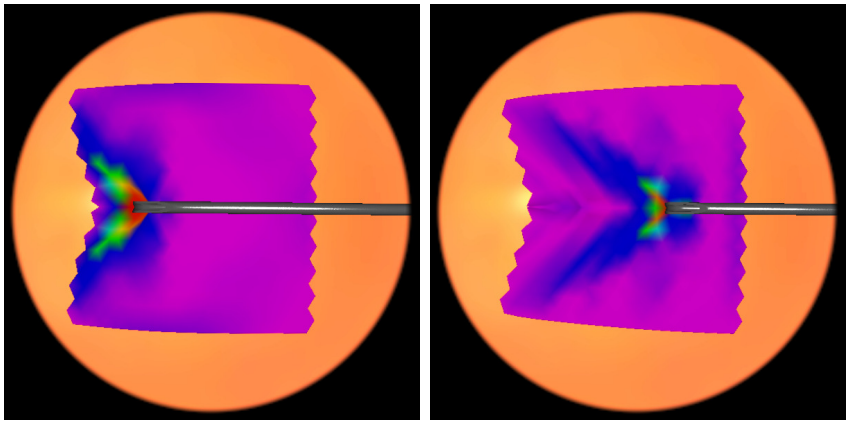
Abbildung 6.13: Verlauf für mittlere Haftreibung aus Abb. 6.12



(a) bisherige Interaktion



(b) neue Interaktion ohne Gleitreibung und mit Haftreibung h_{high}



(c) neue Interaktion ohne Gleitreibung und mit Haftreibung h_{medium}

Abbildung 6.14: Bisherige und neue Interaktion im Vergleich

nun geringeren Auswirkungen der Interaktion deutlich sichtbar. Dem Graph aus Abb. 6.12 sind allerdings kaum Informationen zu entnehmen, da sich der Spannungsverlauf für Haftreibung h_{medium} am unteren Ende der Werte-Skala bewegt. Daher stellt Abb. 6.13 dieselbe Kurve noch einmal in höherer Auflösung dar. Diese Kurve zeigt den typischen, zackigen Verlauf einer *stick-slip*-Interaktion: Jedes mal wenn sich das Zystotom verhakt, werden die kollidierenden Dreiecke solange mitgeführt, bis die internen, rückstellenden Kräfte zu stark werden und das Mesh losreißen.
Das am Graph in Abb. 6.13 zu sehende, zackige Muster stellt sich ebenfalls ein, wenn die Haftreibung auf h_{low} reduziert wird: Die Durchführung des Versuchs ist in Abb. 6.17(a) dargestellt. Eine Deformation des Gitters ist kaum wahrnehmbar, nur anhand der (geeignet skalierten) Falschfarben ist zu erkennen, dass das Instrument ein Spannungsfeld hinter sich herzieht. In Abb. 6.15 ist zu sehen, dass sich die auftretenden, durchschnittlichen Knoten-Spannungen (im Vergleich zu Abb. 6.13) um ca. eine Größenordnung verringert haben. Der *stick-slip*-Verlauf jedoch ist auch hier klar zu erkennen.
Insgesamt wurde Haftreibung nun in drei verschiedenen Abstufungen betrachtet. Abb. 6.16 zeigt die zugehörigen Spannungskurven im direkten Vergleich. Wie zu erwarten, liegt der Verlauf für h_{low} vollständig unter dem für h_{medium} und der Verlauf für h_{medium} vollständig unter dem für h_{high}.
Gleitreibung wurde bisher vernachlässigt. Um das in Abb. 6.15 betrachtete Spannungsprofil für niedrige Haftreibung h_{low} mit einem Gleitreibungsprofil zu vergleichen, wurde Gleitreibung mit μ_{low} aktiviert. Die zugehörige Mesh-Instrument-Interaktion ist in Abb. 6.17(b) zu sehen. Wie in Abb. 6.17(a) baut sich hinter dem Instrument ein wellenförmiges Spannungsfeld auf. Abb. 6.15 zeigt, dass sich die

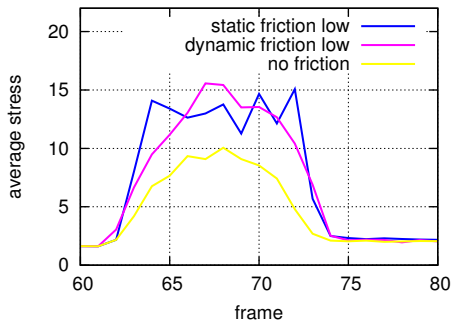


Abbildung 6.15: Haftreibung, Gleitreibung und reibungsfreie Interaktion im Vergleich

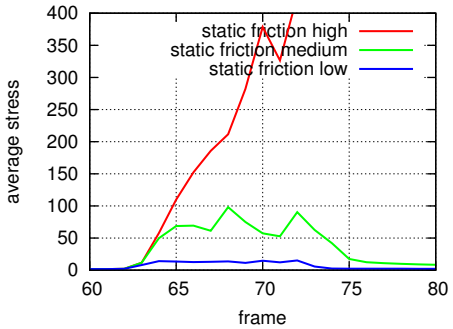
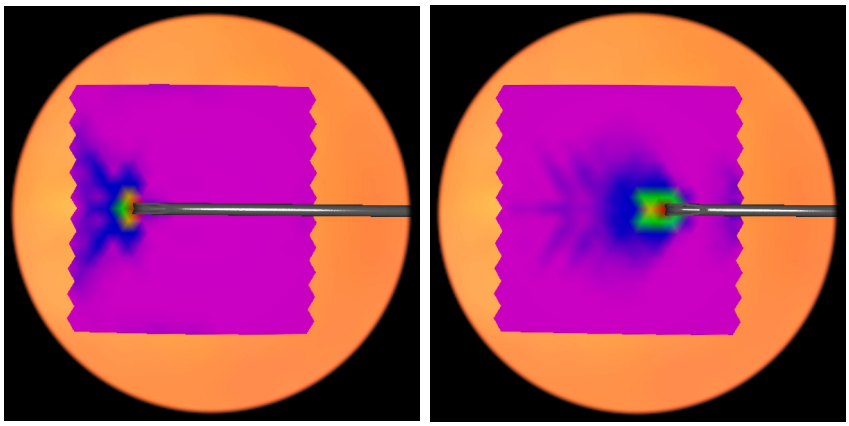
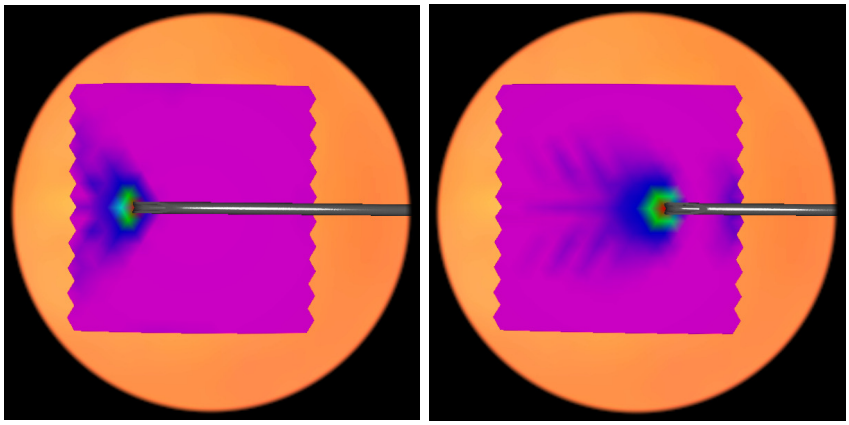


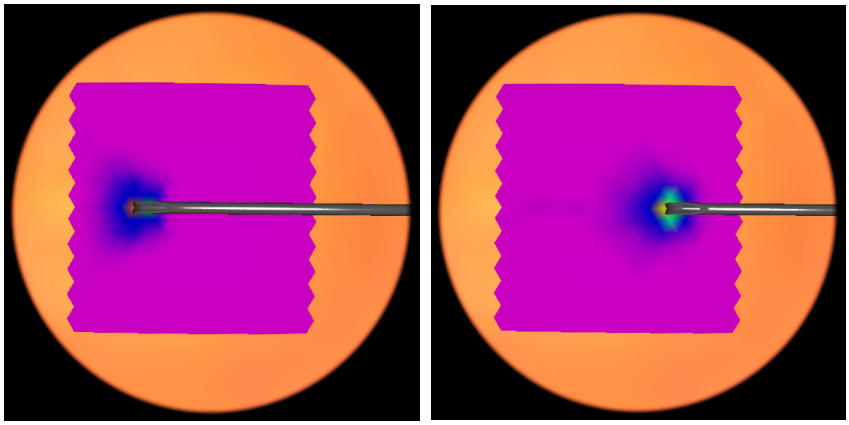
Abbildung 6.16: Niedrige, mittlere und hohe Haftreibung im direkten Vergleich



(a) geringe Haftreibung (ohne Gleitreibung)



(b) geringe Gleitreibung (ohne Haftreibung)



(c) reibungsfreie Interaktion

Abbildung 6.17: Haftreibung, Gleitreibung und reibungsfreie Interaktion im Vergleich

mittleren Knoten-Spannungen mit Gleitreibung μ_{low} im gleichen Wertebereich bewegen, wie diejenigen mit Haftreibung h_{low}. Die beiden Spannungsprofile grenzen sich jedoch dadurch voneinander ab, dass die Kurve für Gleitreibung einen eher runden, glockenförmigen Verlauf zeigt.
Auch reibungsfreie Interaktion wurde bis jetzt noch nicht betrachtet. Daher wurde Reibung vollständig ausgeschaltet, um die in Abb. 6.17(c) zu sehende Interaktion durchzuführen. Hier fällt auf, dass sich das (relativ kleine) Spannungsfeld gleichmäßig um das Zystotom herum aufbaut und mit dessen Bewegung mitläuft. Auch die wellenartigen Muster aus Abb. 6.17(a) und Abb. 6.17(b) fehlen. Das Spannungsprofil in Abb. 6.15 hat einen glockenförmigen Verlauf und nimmt sein Maximum an, wenn das Instrument seinen tiefsten Punkt erreicht. Wie zu erwarten, liegt die Kurve unter denen für niedrige Haft- und Gleitreibung.

Gleitreibung wurde bisher nur mit dem Parameter μ_{low} eingesetzt. In den zugehörigen Screenshots aus Abb. 6.17(b) ist eine Deformation des Gitters optisch nicht wahrnehmbar. Auch wenn man den Reibungskoeffizienten μ erhöht, entsteht keine so deutliche Verformung wie beispielsweise in Abb. 6.14(c). Schließlich ist das Mesh durch Homing-Kräfte an den Rändern fixiert und mit Gleitreibung kann sich das Mesh nicht am Zystotom verhaken. Um den Effekt von Gleitreibung besser sichtbar zu machen, wird das Mesh auf einen schmalen Streifen bestehend aus 132 Dreiecken zurechtgeschnitten. Darüber hinaus werden die Homing-Kräfte der Randknoten ausgeschaltet. In Abb. 6.19 und Abb. 6.20 ist zu sehen, wie das Zystotom einmal ohne Reibung und einmal mit Gleitreibung μ_{high} über das Gitter hinwegstreift. Diesmal stellen die Falschfarben keine Spannungen dar, sondern Knoten-Verschiebungen in Richtung x-Achse. Im Gegensatz zur reibungsfreien Interaktion ist mit Gleitreibung eine Verschiebung des Gitters nach rechts deutlich zu erkennen. Für beide Versuche sind die durchschnittlichen Knoten-Verschiebungen in Abb. 6.18 gegen die Zeit aufgetragen. Ohne Reibung erreicht die monoton steigende Kurve einen Wert von 0.058mm. Mit Gleitreibung verschiebt sich das insgesamt ca. 8mm lange Gitter um immerhin 0.55mm. Die Auswirkungen der beiden Interaktionen auf das Mesh unterscheiden sich demnach um eine Größenordnung.

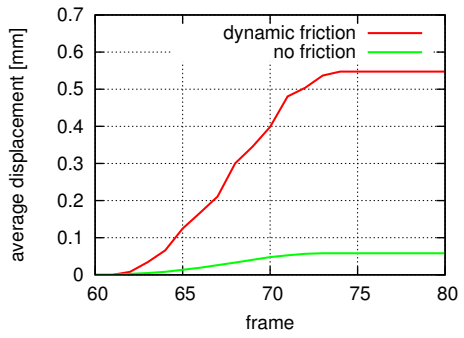


Abbildung 6.18: Verschiebung des Gitters in Richtung x-Achse

Abbildung 6.19: Verschiebung des Gitters ohne Reibung

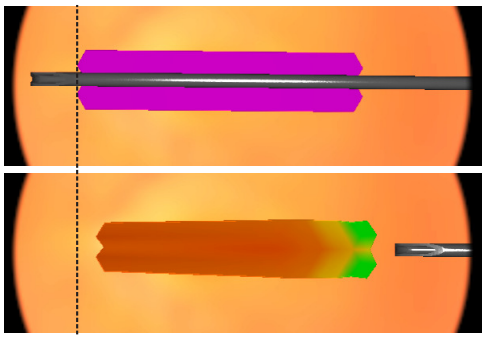


Abbildung 6.20: Verschiebung des Gitters mit Gleitreibung

6.6.3 Das Zusammenspiel mit den Reiß-Algorithmen

Abschließend soll die neue Interaktion im Zusammenspiel mit den in Kapitel 5 definierten Reiß-Algorithmen demonstriert werden. Zu diesem Zweck wird ein rechteckiges Gitter aus 660 Dreiecken generiert, dessen Rand durch starke Homing-Kräfte fixiert ist. In der Mitte des unteren Rands wird ein kleiner Einschnitt eingefügt, so dass sich zu Beginn der Simulation ein Risskeim im Mesh befindet. Die Methode, welche den Reiß-Algorithmus ausführt, wird einmal pro Frame und zwar im Anschluss an die Mesh-Deformation aufgerufen. Das Zystotom wird mit der Hand bewegt und ausgehend vom Risskeim zum oberen Rand des Gitters geführt. In den Abbildungen 6.21 und 6.22 ist dieser Versuch zu sehen, einmal mit dem pseudo-kontinuierlichen Ansatz aus Abschnitt 5.5.2 und einmal mit dem „Reißen durch progressives Schneiden“ aus Abschnitt 5.5.3.
Prinzipiell ist die dargestellte Interaktion möglich, unabhängig davon, welche Parameter für Gleit- und Haftreibung gewählt werden. Ohne Haftreibung allerdings rutscht das Zystotom leicht von der Risskante ab, was den Eingriff erschwert. Für die in den Abbildungen 6.21 und 6.22 dargestellten Versuche wurde die Haftreibung daher auf h_{high} gesetzt. Dass in Abb. 6.22 eine glattere Risskante entsteht als in Abb. 6.21, liegt an den unterschiedlichen Eigenschaften der Reiß-Algorithmen (siehe Abschnitt 5.6).
Insgesamt ergibt sich hier, dass Interaktion und Reiß-Algorithmen miteinander kompatibel sind und im Zusammenspiel das gewünschte Ergebnis zeigen: Das Zystotom „schiebt“ den Risskeim vor sich her und trennt das Mesh entlang seiner Bewegung auf. Mit der bisherigen EYESi-Interaktion dagegen ist dies nicht möglich: Ihre verschiebungsbasierte Kollisionsantwort führt zu physikalisch unplausiblen Verzerrungen, so dass sich der Riss chaotisch und vollkommen unkontrolliert ausbreitet (weshalb an dieser Stelle auf Screenshots verzichtet wurde).

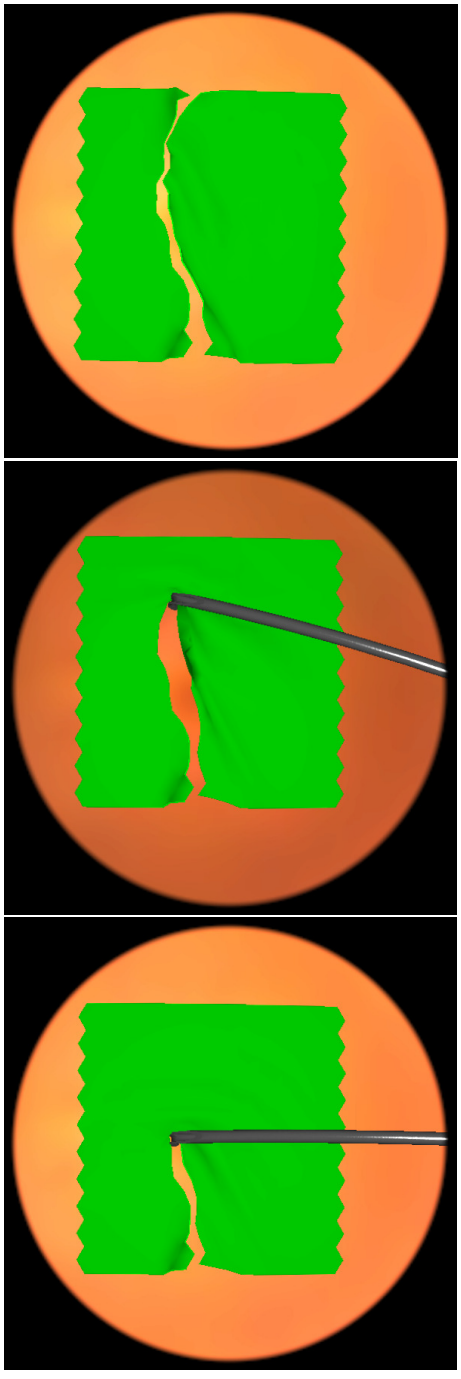


Abbildung 6.21: Die neue Interaktion im Zusammenspiel mit dem Reiß-Algorithmus aus Abschnitt 5.5.2

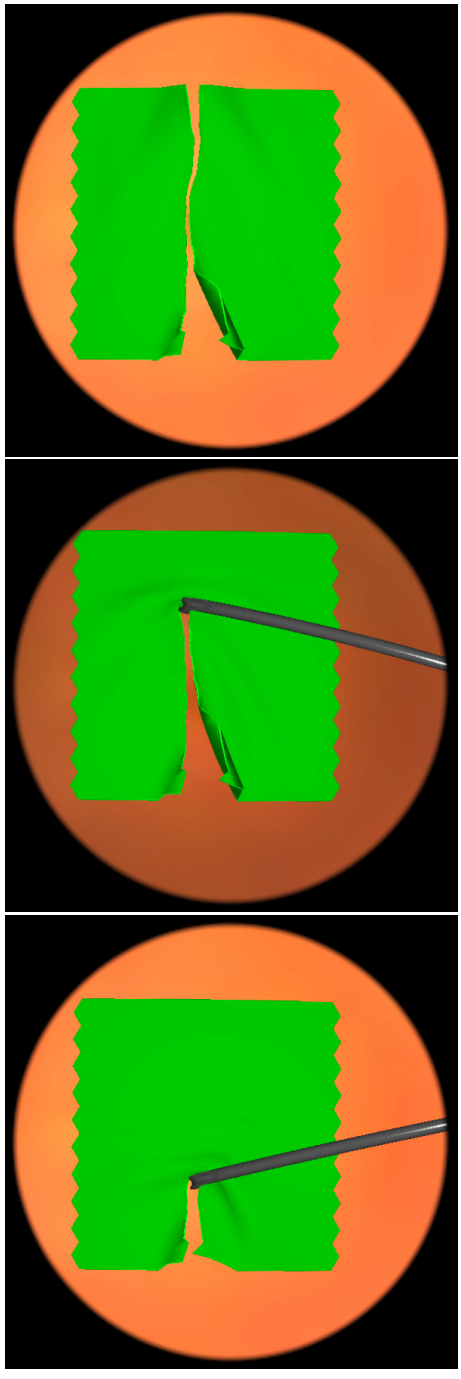


Abbildung 6.22: Die neue Interaktion im Zusammenspiel mit dem Reiß-Algorithmus aus Abschnitt 5.5.3

6.7 Zusammenfassung

In diesem Kapitel wurde die Interaktion zwischen einer deformierbaren Oberfläche und einem chirurgischen Nadel-Instrument definiert. Die Instrument-Spitze wird geometrisch durch einen starren, abgerundeten Zylinder approximiert. Um alle Kontakte zwischen Instrument und Mesh ausreichend genau erfassen zu können, wurde für den Zylinder eine dreiecksbasierte Kollisionserkennung implementiert. Die Kollisionsantwort des Zylinders berechnet Kräfte, welche auf die Knoten kollidierender Dreiecke verteilt werden. Diese Kollisionsantwortkräfte definieren entweder reibungsfreie Kräfte, reibungsfreie Kräfte plus Gleitreibungskräfte oder aber reine Haftreibungskräfte. Die Reibungsmodelle setzen auf dem Modell für reibungsfreie Kontakte auf und können über Parameter konfiguriert werden.
Laufzeit-Messungen haben ergeben, dass die neue Interaktion für den Einsatz in einer Echtzeitanwendung geeignet ist. Allerdings beansprucht sie deutlich mehr Rechenzeit als die bisherige EYESi-Interaktion, da sie Kollisionsantwortkräfte für jeden einzelnen Integrationsschritt berechnet. Die Kollisionsantwort der bisherigen EYESi-Interaktion wird nur einmal pro Frame durchgeführt. Genau dieser Unterschied jedoch macht die neue Interaktion robuster gegenüber numerischer Instabilität.
Die Modelle für Gleit- und Haftreibung sorgen für den geforderten Realitätsgrad der virtuellen Instrument-Interaktionen. Dank der Haftreibung ist es möglich, mit dem Instrument Zug-Kräfte auf die Membran auszuüben, ohne ständig von dieser abzurutschen. Da sich die Membran von der Instrument-Spitze losreißt, wenn die rückstellenden Kräfte zu hoch werden, führt die Haftreibung, im Gegensatz zur bisherigen EYESi-Interaktion, zu keinem unnatürlichen „Festkleben“ der Membran. Für vorliegende Arbeit ist dieses Ergebnis von besonderer Bedeutung, da ein Chirurg in der Lage sein muss, die simulierte Membran kontrolliert zu manipulieren.
Im Zusammenspiel mit den Reiß-Algorithmen aus vorherigem Kapitel zeigt die neue Interaktion das gewünschte Verhalten: Wenn die Instrument-Spitze mit einem Risskeim kollidiert, schiebt die Interaktion diesen voran und das Mesh wird entlang der Instrument-Bewegung aufgetrennt. Mit der bisherigen EYESi-Interaktion ist dies nicht möglich, da deren verschiebungsbasierte Kollisionsantwort eine chaotische, unkontrollierbare Rissausbreitung auslöst.

Kapitel 7

Anwendungen

Die in den vorherigen Kapiteln definierten Algorithmen sind in drei verschiedene Trainingsmodule des ophtalmochirurgischen Simulators EYESi eingeflossen. Diese drei Anwendungen werden im Folgenden inklusive ihres medizinischen Hintergrunds beschrieben.

Bemerkung Jedes der Module verwendet eine Vielzahl an Komponenten der EYESi-Software, die nicht im Rahmen vorliegender Arbeit entstanden sind. Die Modul-Beschreibungen lassen all diejenigen EYESi-Komponenten aus, die keinerlei Einfluss auf die Einbindung der neuen Algorithmen hatten. Hierunter fällt z. B. die Simulation der Iris während eines Eingriffs in der Vorderkammer.

7.1 Die Kapsulorhexis

Zu Beginn dieser Arbeit existierte bereits der Prototyp eines EYESi-Moduls für die Kapsulorhexis. Dieser wurde komplett überarbeitet und es entstand das hier beschriebene Trainingsmodul.

7.1.1 Medizinischer Hintergrund

Die Kapsulorhexis ist Teil einer *Katarakt*-Operation. Katarakt, auch *Grauer Star* genannt, bezeichnet die Trübung der Augenlinse. Im Verlauf der Operation wird die getrübte Linse entfernt und durch ein Implantat ersetzt.
Die Augenlinse ist von einer Membran, dem sogenannten *Kapselsack*, umschlossen. Dieser muss im Verlauf des Eingriffs erhalten bleiben, damit er als Behälter für die Kunstlinse dienen kann. Andererseits lässt sich die getrübte Linse nicht austauschen, ohne den Kapselsack zu beschädigen. Der Name „Kapsulorhexis“ bezeichnet die Prozedur, mit welcher der Kapselsack geöffnet wird: Der Chirurg

benutzt eine Pinzette oder ein Zystotom, um ein kleines Stück aus der Membran heraus zu reißen. Hierbei ist es besonders wichtig, dass die entstehende Öffnung kreisrund ist, mittig auf der Linse sitzt und einen bestimmten Radius weder unter- noch überschreitet. Gelingt dies nicht, kann es zu post-operativen Komplikationen kommen.
Entwickelt und erstmalig veröffentlicht wurde die Kapsulorhexis-Technik von H. V. Gimbel und T. Neuhann [52].

7.1.2 Anforderungen an die Simulation

Um eine realistische Simulation zu gewährleisten, muss das Trainingsmodul folgende Anforderungen erfüllen:

Linsen-System: Die Linse inklusive Kapselsack ist an ihrer Peripherie über elastische Fasern, die sogenannten *Zonular*-Fasern, mit dem umliegenden Gewebe verwachsen. Wenn der Arzt mit einem Instrument auf die Linse drückt, dellt sich zunächst deren Oberfläche ein. Erst mit steigendem Druck bewegt sich die gesamte Linse und kann im Extremfall aus den Zonular-Fasern reißen. Auch die EYESi-Linse soll sich eindellen lassen und an virtuellen Zonular-Fasern aufgehängt sein.

Kapselsack-Modell: Der Kapselsack muss als deformierbare Oberfläche simuliert werden.

Anhaften und Ablösen: Der Kapselsack haftet an der Oberfläche der in den Zonular-Fasern beweglichen Linse. Das Anhaften und Ablösen der Membran muss modelliert werden und die noch anhaftende Membran muss sich der Transformation der Linse und der Deformation ihrer Oberfläche anpassen.

Spannung im Kapselsack: Das Druckverhältnis zwischen Hinter- und Vorderkammer beeinflusst die Lage der Linse. Wenn Flüssigkeit durch die Instrument-Zugänge entweicht, sinkt der Vorderkammerdruck und die Linse hebt sich. Dadurch spannen sich die Zonular-Fasern und ziehen an der Oberseite des Kapselsacks. Daher soll auch der virtuelle Kapselsack in Abhängigkeit simulierter Druckschwankungen unter Spannung stehen.

Instrument-Interaktionen: Im Allgemeinen wird die Kapsulorhexis entweder mit einer Pinzette oder mit einem Zystotom durchgeführt. Beide Instrumente soll das Modul zur Verfügung stellen. Insbesondere muss der Arzt mit einer Kanüle Flüssigkeit, das sogenannte *Viskoelastikum*, in die Vorderkammer einspritzen können. Dadurch lässt sich der Kammerdruck stabilisieren und die Spannung im Kapselsack senken.

Reißen des Kapselsacks: Der Kapsulorhexis-Riss muss sich auf einer Kreisbahn bewegen. Daher muss ein Reiß-Algorithmus eingesetzt werden, der es erlaubt, den Riss auf einer beliebigen Strecke durch die Membran zu führen.

Abdriften des Kapsulorhexis-Risses: Die Kapsulorhexis gilt als eine der kritischsten Phasen im Verlauf einer Katarakt-Operation. Denn die Öffnung im Kapselsack muss zentriert und kreisrund sein, um dessen Stabilität zu bewahren. Erschwerend kommt eine besondere Eigenschaft des Kapsulorhexis-Risses hinzu: Chirurgen beschreiben, dass der Riss häufig die Tendenz zeigt, die Kreisbahn zu verlassen und zur Peripherie hin „abzudriften“. Die Ursache für dieses spontane Herauslaufen ist nicht eindeutig geklärt, das Problem scheint sich jedoch mit zunehmender Spannung im Kapselsack zu verstärken. Aus den Erfahrungen der Ärzte haben sich einige Regeln herauskristallisiert, mit deren Hilfe sich die Rissausbreitung leichter kontrollieren lässt. So sollte der Chirurg z. B. den Abstand zwischen Rissende und Instrument nicht zu groß werden lassen, damit sich die beim Ziehen aufbauende Spannung nicht diffus über die Membran verteilt.
Im Trainingsmodul für die Kapsulorhexis muss das besondere Verhalten des Risses berücksichtigt werden.

Bewertungssystem: Das EYESi-Trainingsmodul muss über ein Bewertungssystem verfügen. Erst mit Hilfe einer Bewertung durch Punkte ist der Benutzer in der Lage, seinen virtuellen Eingriff zu beurteilen und seine Fähigkeiten im Laufe mehrerer Iterationen zu verbessern.

Schwierigkeitsgrad: In der Realität hängt die Schwierigkeit einer Kapsulorhexis von der individuellen Beschaffenheit des Patientenauges ab. Daher soll das Modul über mehrere Levels mit steigendem Schwierigkeitsgrad verfügen.

7.1.3 Vergleichbare Arbeiten

Die Behandlung eines Katarakts ist die weltweit am häufigsten durchgeführte Augenoperation. *„The capsulorhexis procedure is universally acknowledged as one of the fundamental elements of modern phaco surgical techniques“* (Seibel [107]). Daher arbeiten mehrere Forschungsgruppen an Simulatoren für diesen Eingriff. Die zugehörigen Veröffentlichungen allerdings deuten nur an, welche Algorithmen für die Kapsulorhexis eingesetzt werden:
Am *Center for Advanced Studies, Research and Development* auf Sardinien lief das (inzwischen beendete) EYESIM-Projekt, in dessen Rahmen an der Simulation für eine komplette Katarakt-Operation entwickelt wurde (Agus et al. [2, 1]). Als Modell für den Kapselsack dient ein Feder-Masse-Gitter, dessen Knoten durch Homing-Federn an der Linse fixiert sind. Reißen wird durch das Aufbrechen

überdehnter Federn realisiert, jedoch wird nicht beschrieben, wie ein Riss topologisch umgesetzt wird.
Auch das INRIA Forscher-Team ALCOVE arbeitet an einem Simulator für die Katarakt-Chirurgie. Einem *Activity Report* ([101]) ist zu entnehmen, dass ein Feder-Masse-Gitter den Kapselsack modelliert und sowohl gerissen als auch geschnitten werden kann.
Webster et al. [130] haben eine Simulation für die Kapsulorhexis entwickelt, in der ebenfalls ein Feder-Masse-Modell als Kapselsack dient. Über eine von VRmagic zur Verfügung gestellte Schnittstelle haben sie ihren Ansatz auf EYESi portiert (Webster et al. [129]). Die Berechnung der Rissausbreitung ist deskriptiv und basiert auf einer nicht näher beschriebenen Auswertung der im Mesh wirkenden Kräfte.
Zu Beginn dieser Arbeit existierte ein EYESi-Prototyp für ein Kapsulorhexis-Modul, der im Rahmen einer Doktorarbeit (Grimm [56]) bei VRmagic entstanden ist: Der Kapselsack entspricht einem Feder-Masse-Gitter, dessen Knoten durch Homing-Kräfte an der Linse haften. Die Oberfläche der Linse wird durch eine starre, unbewegliche Kugelschale approximiert. Als Instrument kann eine Pinzette verwendet werden und für das Reißen wird der kantenbasierte Ansatz aus Grimm [57] eingesetzt.
Im Verlauf der vorliegenden Arbeit sind, ausgehend vom Prototyp und über mehrere Iterationen hinweg, verschiedene Versionen des Kapsulorhexis-Moduls entstanden. Eine dieser Zwischen-Versionen wurde in Weber et al. [128] veröffentlicht: Eine Adaption des progressiven Schneide-Algorithmus von Nienhuys und van der Stappen [91] sorgt für die topologische Umsetzung eines Risses, der sich auf einer beliebigen Bahn durch das Feder-Masse-Gitter bewegen kann. Die Bestimmung der Rissrichtung ist rein deskriptiv. Diese deskriptive Komponente erlaubt es, das besondere Verhalten des Kapsulorhexis-Risses an die Beschreibungen der Ärzte anzupassen. Allerdings ist es nicht gelungen, ein deskriptives Modell zu finden, dass für alle denkbaren Interaktionen zwischen Instrument und Membran eine physikalisch plausible Rissausbreitung erzeugt. Aus dieser Erfahrung heraus ist das nachfolgend beschriebene, aktuelle EYESi-Modul entstanden, in dem die Berechnung der Rissausbreitung prinzipiell physikalisch ist und nur in Sonderfällen von einer deskriptiven Komponente übernommen wird.

7.1.4 Das EYESi-Trainingsmodul

Im Folgenden wird beschrieben, wie das Kapsulorhexis-Modul aufgebaut ist, um die gestellten Anforderungen zu erfüllen:

Linsen-System (nicht von der Autorin entwickelt)**:** Eine geschlossene, triangulierte Oberfläche definiert den Umriss des Linsen-Objekts, das als Festkörper si-

muliert wird und an linear-elastischen Federn aufgehängt ist. Wenn die Bounding Box eines Instruments die Linse berührt, wird eine Eindellung auf ihrer Oberflächen-Repräsentation generiert. Erst wenn das Instrument tiefer eindringt, werden Kräfte berechnet, welche die Linse ausweichen oder sogar aus der Aufhängung reißen lassen.

Kapselsack-Modell: Der Arzt interagiert nur mit dem Teil des Kapselsacks, der sich auf der Oberseite der Linse befindet und nicht von der Iris verdeckt wird. Der virtuelle Kapselsack umschließt das Linsen-Objekt daher nicht vollständig: Die als Feder-Masse-Modell simulierte Membran wird als kreisrunde, trianguliert Oberfläche erzeugt, deren Rand sich hinter der Iris verbirgt.

Anhaften und Ablösen: Das Anhaften/Ablösen der Membran basiert auf dem Modell aus Abschnitt 4.4.2. Zu Modul-Start haften sämtliche Knoten des Feder-Masse-Gitters an der undeformierten Linse. In jedem Frame werden die Homing-Positionen noch nicht abgelöster Knoten auf die Linsenoberfläche projiziert. Somit ist gewährleistet, dass sich die Membran der Deformation der Linse anpasst. Um zu verhindern, dass sich die Membran als Ganzes ablöst, werden die hinter der Iris liegenden Randknoten von der Simulation ausgeschlossen. Ihre Positionen sind im Verlauf der gesamten Simulation auf der Oberfläche der Linse fixiert.

Spannung im Kapselsack: Das Modul verfügt über eine Variable *pressure*, die den Vorderkammerdruck als normalisierten Wert repräsentiert. Mit jeder Wertänderung von *pressure* wird die Linse entlang der z-Achse versetzt und die Spannung im Kapselsack variiert. Letzteres wird durch eine Adaption der Knoten-Ruhepositionen realisiert, die relativ zum Schwerpunkt des Linsen-Objekts definiert sind. (Zur Erinnerung: Die Ruhepositionen beschreiben den Kapselsack in unverzerrtem Zustand.) Während der Modul-Initialisierung werden die Ruhepositionen auf der Oberfläche der undeformierten Linse generiert. Um die Membran unter Spannung zu setzen, werden diese mit einem Wert < 1 multipliziert und somit ins Innere der Linse versetzt. Da die Homing-Positionen stets auf der Linsenoberfläche liegen, müssen die Homing-Kräfte die Membran dehnen, um sie über die Linse zu spannen.

Instrument-Interaktionen: Im Verlauf der Simulation fällt der *pressure*-Wert langsam aber kontinuierlich ab, da Flüssigkeit über die Instrument-Zugänge entweichen kann. Daher stellt das Modul eine Kanüle bereit, mit der Viskoelastikum in die Vorderkammer gespritzt werden kann. (Die Simulation des Viskoelastikums wurde im Rahmen einer studentischen Arbeit[1] entwickelt, die von der Autorin

[1] Joachim Schlipper, *Berechnung von Fluid-Fluid Interaktionen für den Augenoperationssimulator EYESI*, Bachelorarbeit, 2006

betreut wurde.) Somit kann der Arzt den Kammerdruck und über diesen die Spannung im Kapselsack kontrollieren.
Neben der Kanüle kann der Arzt eine Pinzette oder ein Zystotom verwenden. An die Spitze des Zystotoms ist eine Interaktion gekoppelt, wie sie in Kapitel 6 beschrieben ist. Das Greifen mit der Pinzette ist mit Hilfe von Homing-Kräften realisiert: Sobald sich die Pinzette schließt, werden für Knoten kollidierender Dreiecke Homing-Positionen relativ zum Instrument definiert. Die Homing-Kräfte werden gelöscht, sobald sich die Pinzette wieder öffnet. (Die bisherige Pinzetten-Interaktion von EYESi basiert nicht auf Kräften sondern auf Verschiebungen. Knoten kollidierender Dreiecke werden von der Simulation ausgeschlossen, so dass gegriffene Membran-Teile beim Schließen der Pinzette plötzlich „erstarren“.)

Reißen des Kapselsacks: Als Reiß-Algorithmus wird der kontinuierliche Ansatz aus Abschnitt 5.5.3 verwendet. Dieser erlaubt dem Arzt, den Riss exakt auf der gewünschten Kreisbahn zu führen. Pro Frame wird der Reiß-Algorithmus insgesamt dreimal aufgerufen und zwar im Wechsel mit der Feder-Masse-Simulation, die in $4 \cdot 10$ Integrationsschritte zerfällt.

Abdriften des Kapsulorhexis-Risses: In den Anforderungen wurde beschrieben, dass der Kapsulorhexis-Riss ein sehr eigentümliches Verhalten zeigt. Nach Aussagen der Ärzte hängt das spontane Herauslaufen des Risses mit der Spannung im Kapselsack zusammen. Daher wird auch diese im Trainingsmodul modelliert. Allerdings ist es nicht gelungen, das besondere Verhalten des Risses allein durch eine Variation der Membran-Spannung abzubilden. Daher wurde die physikalische Rissausbreitung speziell für die Kapsulorhexis um eine deskriptive erweitert: Jedes mal wenn der Risskeim den Spannungs-Schwellwert $minSproutStress$ für Reißen um x überschreitet, wird überprüft, ob x kleiner ist als als ein zweiter Schwellwert $trigger$. Nur wenn dies zutrifft, wird die Rissausbreitung physikalisch berechnet, andernfalls wird die deskriptive Rissausbreitung ausgelöst. Hierbei wird zunächst ein normalisierter Vektor ermittelt, der die aktuelle Rissausbreitung repräsentiert. In Abhängigkeit der Höhe von x wird dieser Vektor Richtung Peripherie rotiert und anschließend als Rissrichtung an den Reiß-Algorithmus übergeben. Dadurch driftet der Riss nach außen ab. (Die am Risskeim entstehende Spannung lässt sich besonders gut kontrollieren, wenn die Membran in der Nähe des Rissendes gegriffen wird. Somit wird der Arzt gezwungen, häufig nachzugreifen und insgesamt mit dem nötigen Feingefühl vorzugehen.)

Bewertungssystem: Um die Kapsulorhexis abschließend zu bewerten, wird ihre Geometrie überprüft. Hierfür werden alle Knoten aus dem Mesh extrahiert, welche die Kapsulorhexis-Kante beschreiben. Die zugehörigen Knoten-Positionen liefern ein gemitteltes Kreis-Zentrum und einen durchschnittlichen Kreis-Radius. An-

schließend wird berechnet, wie stark die Knoten-Positionen im Durchschnitt von der approximierten Kreisbahn abweichen. Somit lässt sich in Form von Punkten bewerten, wie gut zentriert die Kapsulorhexis ist, welcher Radius erreicht wurde und inwiefern eine gleichmäßig runde Öffnung entstanden ist.

Schwierigkeitsgrad: Zur Zeit existieren fünf verschiedene Schwierigkeitsgrade für die Kapsulorhexis. Startet das Modul im ersten Schwierigkeitsgrad, findet der Benutzer eine bereits zu ca. einem Viertel durchgeführte Kapsulorhexis vor. Somit kann sich ein Anfänger darauf konzentrieren, den bereits auf der Kreisbahn liegenden Riss zu beenden. Ähnlich sieht das Modul im letzten Schwierigkeitsgrad aus, hier allerdings zeigt das aktuelle Rissende nach außen zur Peripherie. Es ist Aufgabe des Benutzers, die Kapsulorhexis zu „retten“ und den Riss zurück auf die Kreisbahn zu führen. In den restlichen Schwierigkeitsgraden startet das Modul mit einem noch unversehrten Kapselsack. Alle fünf Schwierigkeitsgrade unterscheiden sich in der Handhabung des Schwellwerts *trigger*, der die deskriptive Rissausbreitung auslöst. Um so höher der Schwierigkeitsgrad, um so niedriger der Startwert von *trigger*. Durch das Einspritzen von Viskoelastikum lässt sich *trigger* bis auf einen Maximalwert bringen, dessen Höhe ebenfalls vom Schwierigkeitsgrad abhängt. Aus den fünf Schwierigkeitsgraden wurden insgesamt zehn EYESi-Levels generiert. In den Levels mit ungerader Nummerierung ist auf der Membran eine schwarze Kreislinie eingezeichnet, an der sich der Benutzer orientieren kann.

Abb. 7.1 zeigt Screenshots einer mit dem EYESi-Modul durchgeführten Kapsulorhexis. Da der transparente Kapselsack schlecht zu erkennen ist, wurden einige seiner Konturen nachgezeichnet: Die weiße Linie markiert die Kapsulorhexis-Kante, die weiß-gestrichelte die Ablösefront und die schwarze den Umriss der abgelösten Membran-Lasche.
In Bild (a) wurde die Iris ausgeblendet, um das komplette Feder-Masse-Gitter sichtbar zu machen. Wie bei einem echten Eingriff üblich, beginnt die virtuelle Kapsulorhexis damit, das Vorderkammer-Wasser durch Viskoelastikum zu ersetzen. Da beide Flüssigkeiten transparent sind, ist nur die Grenzfläche zwischen ihnen am rechten Rand von Bild (b) erkennbar. In Bild (c) wird der Kapselsack mit einem Zystotom angestochen, wobei durch die Eindellung der Linse ein ringförmiger Lichtreflex entsteht. Das Instrument wird Richtung Peripherie geführt, wodurch ein schmaler, gerader Riss entsteht. In Bild (d) ist zu sehen, wie die anfänglich rein radiale Zystotom-Bewegung in eine Bewegung senkrecht zum Riss übergeht. Dadurch wird die schwarz markierte Risskante vor dem Instrument hergeschoben und am Rissende entsteht ein Zug, der die Rissausbreitung auf eine Bahn tangential zur Iris umlenkt. Durch die schiebende Bewegung stellt sich vor dem Instrument ein Stück Membran auf, das – wie in Bild (e) zu sehen – umgelegt

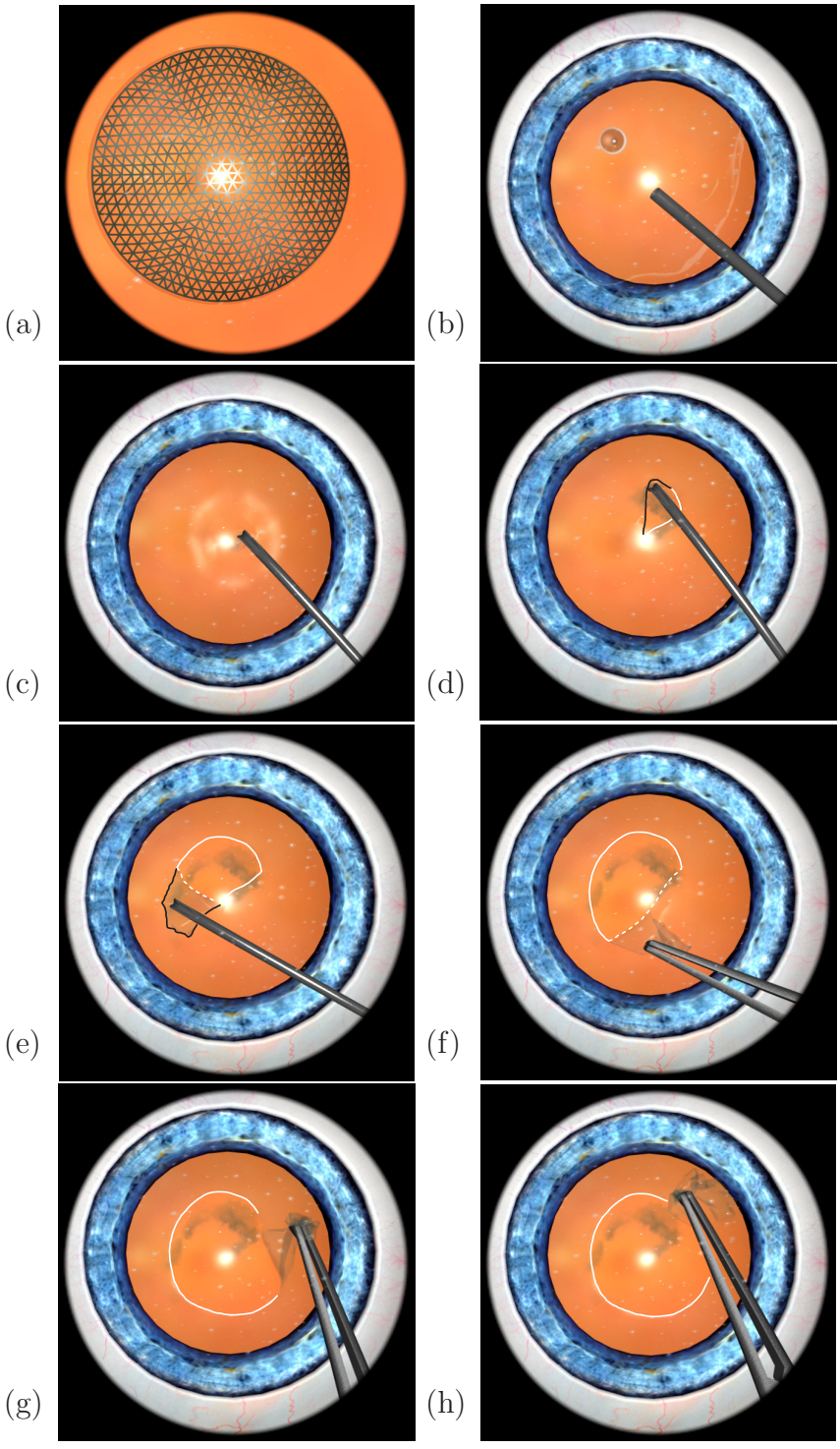


Abbildung 7.1: Durchführung einer Kapsulorhexis

und auf die Linse gedrückt wird. Der Druck ist (sowohl in der Realität als auch in der Simulation) notwendig, um das umgeklappte Stück Membran im weiteren Verlauf der Kapsulorhexis über die Linse schieben zu können ohne abzurutschen. Um hier auch den Einsatz einer Pinzette zu demonstrieren, wurde das Instrument an dieser Stelle gewechselt und die Kapsulorhexis fortgesetzt, wie in den Bildern (f) bis (h) zu sehen.

7.1.5 Ergebnisse und Diskussion

Das Kapsulorhexis-Modul hat bereits seit längerem die Produktreife erreicht und wird zusammen mit den anderen EYESi-Modulen von VRmagic verkauft und für die Ausbildung in der Augenchirurgie eingesetzt. Die Simulation erfüllt die gestellten Anforderungen und wird von erfahrenen Ärzten als überzeugend bezeichnet. Der Einsatz eines Zystotoms wurde erst durch die im Rahmen vorliegender Arbeit entwickelte Instrument-Interaktion möglich. Besonders der kontinuierliche Reiß-Algorithmus hat den Realitätsgrad des Moduls merklich erhöht.
Die deskriptive Komponente der Rissausbreitung ist allerdings kritisch zu beurteilen. Denn wenn die Rissrichtung deskriptiv ermittelt wird, besteht prinzipiell die Gefahr, dass der Benutzer die Rissausbreitung als physikalisch unplausibel wahrnimmt. Für den Ausbildungssimulator EYESi steht jedoch im Vordergrund, mit Hilfe eines Moduls bestimmte Lernziele und Trainingseffekte zu erreichen. Mit dem Kapsulorhexis-Modul soll der Benutzer lernen, dass er den Riss nicht kontrollieren kann, wenn er mit mangelndem Feingefühl vorgeht. Daher wird die deskriptive Komponente in Kauf genommen, bis sich ein physikalisches Modell findet, welches das gewünschte Verhalten abbildet.
Auch wenn die variable Spannung im virtuellen Kapselsack nicht die erhoffte Wirkung auf die physikalische Rissausbreitung hat, so ist sie dennoch ein wichtiger Teil der Simulation: Wenn der Arzt mit der Membran interagiert, kann er spüren, ob diese unter hoher Spannung steht. Denn eine hohe Vorspannung führt beispielsweise dazu, dass die Membran bereits bei geringer Zug-Belastung zu reißen beginnt. Sobald der Arzt die Spannung wahrnimmt, ist dies ein Indikator für ihn, Viskoelastikum in die Vorderkammer einzuspritzen. Die „Belohnung“ für die richtige Reaktion des Arztes besteht darin, dass das Viskoelastikum den Kammerdruck erhöht und dadurch die deskriptive Rissausbreitung positiv beeinflusst.

7.2 Peeling der Inneren Grenzmembran

Zu Beginn vorliegender Arbeit existierte bereits ein EYESi-Modul für das Peeling der Inneren Grenzmembran, das von VRmagic verkauft wird. Um dieses Modul zu verbessern, wurden einige seiner Komponenten ausgetauscht bzw. überarbeitet.

7.2.1 Medizinischer Hintergrund

Die Innere Grenzmembran, abgekürzt *ILM* für *Internal Limiting Membrane*, ist die oberste Schicht der Retina. In manchen chirurgischen Eingriffen ist es notwendig, einen Teil der ILM zu entfernen, um darunter liegende Schichten der Retina freizulegen.
Zu Beginn des Peelings wird die transparente ILM häufig mit Hilfe eines Farbstoffs sichtbar gemacht. Anschließend streicht der Arzt mit einem spitzen Instrument über die ILM, um einen ersten Einriss zu erzeugen. Mit einer Pinzette greift er die so entstandene Risskante und versucht, den zu entfernenden Teil der ILM in einem möglichst zusammenhängenden Stück von der Retina abzulösen. Denn umso stärker die ILM fragmentiert, umso häufiger muss der Chirurg nachgreifen und umso höher ist die Gefahr, die Retina mit der Pinzette zu verletzen. Abschließend werden die abgelösten Membran-Teile mit einem Saug-Schneide-Instrument, dem *Vitrektor*, aus dem Auge entfernt.

7.2.2 Anforderungen an die Simulation

Um eine realistische Simulation zu gewährleisten, muss das Trainingsmodul folgende Anforderungen erfüllen:

ILM-Modell: Die ILM muss als deformierbare Oberfläche simuliert werden. Da sie über eine für den Chirurgen spürbare Biege-Steifigkeit verfügt, muss auch diese Eigenschaft der Membran modelliert werden.

Anhaften und Ablösen: Das Anhaften der ILM an der Retina muss modelliert werden.

Reißen der ILM: Für das Reißen der ILM muss ein geeigneter Algorithmus verwendet werden.

Instrument-Interaktionen: Das Trainingsmodul muss einen Schaber (zum Anreißen der Membran), eine Pinzette und einen Vitrektor zu Verfügung stellen.

Bewertungssystem: Das EYESi-Trainingsmodul muss über ein Bewertungssystem verfügen. Erst mit Hilfe einer Bewertung durch Punkte ist der Benutzer in der Lage, seinen virtuellen Eingriff zu beurteilen und seine Fähigkeiten im Laufe mehrerer Iterationen zu verbessern.

Schwierigkeitsgrad: In der Realität schwankt die Schwierigkeit eines Peelings von Patient zu Patient. Daher soll das Modul über mehrere Levels mit steigendem Schwierigkeitsgrad verfügen.

7.2.3 Vergleichbare Arbeiten

Soweit der Autorin bekannt, ist EYESi der einzige Simulator, der ein virtuelles ILM-Peeling umsetzt. Eine der ersten produktreifen Versionen des ILM-Moduls ist in Grimm et al. [58] und Grimm [56] beschrieben. Bevor das Modul im Rahmen vorliegender Arbeit weiterentwickelt wurde, zeichnete es sich im Wesentlichen durch folgende Komponenten aus:
Als ILM dient ein Feder-Masse-Gitter, das dank spezieller Struktur-Federn eine Biege-Steifigkeit simuliert. Das Anhaften der Membran an der Retina ist ähnlich wie in Abschnitt 4.4.2 durch Homing-Kräfte realisiert. Als Ablöse-Kriterium allerdings dient die am jeweiligen Knoten gemessene Spannung. Um die Membran zu reißen, wird der in Grimm [57] beschriebene, kantenbasierte Reiß-Algorithmus eingesetzt. Für das Greifen mit einer Pinzette und das Absaugen mit einem Vitrektor wird auf in EYESi bereits vorhandene Instrument-Interaktionen zurückgegriffen. Um die Membran mit einem Schaber anreißen zu können, ist für dessen Spitze eine kraftbasierte, reibungsfreie Kollisionsantwort implementiert. Die Kollision mit einem Dreieck wird allerdings nur dann erkannt, wenn ein Knoten des Dreiecks, der Schwerpunkt des Dreiecks oder ein Mittelpunkt der Dreieckskanten mit den Hüllkörpern der Spitze kollidiert.
Das Modul verfügt über ein Bewertungssystem und sechs verschiedene Schwierigkeitsgrade.

7.2.4 Das EYESi-Trainingsmodul

Im Folgenden wird die überarbeitete, aktuelle Version des ILM-Trainingsmoduls beschrieben:

ILM-Modell: Die reale ILM deckt die gesamte Oberfläche der Retina ab. Ein Peeling allerdings beschränkt sich auf die Umgebung der *Makula*, der Stelle des schärfsten Sehens. Der Teil der Retina, in dem diese Stelle liegt, ist von den Gefäßbögen umrandet, die vom Sehnerv ausgehen. Wie bereits in der vorherigen Modul-Version, simuliert ein Feder-Masse-Gitter die ILM und deckt nur den Bereich innerhalb der Gefäßbögen ab. Biege-Elemente sind wie in Abschnitt 4.4.1 definiert.

Anhaften und Ablösen: Das Anhaften und Ablösen der Membran basiert auf dem Modell aus Abschnitt 4.4.2. Allerdings muss verhindert werden, dass sich das Feder-Masse-Gitter ausgehend von seinen initialen Randknoten (beim Reißen entstehen neue Randknoten) vom Untergrund löst. Andernfalls könnte der Benutzer merken, dass nur ein Teilstück der ILM simuliert wird. Daher werden initiale Randknoten durch besonders starke Homing-Kräfte an der Retina fixiert.

Somit ist sowohl das Ablösen als auch das Reißen dieser Knoten gegenüber allen anderen erschwert.

Reißen der ILM: Als Reiß-Algorithmus wird der kontinuierliche Ansatz aus Abschnitt 5.5.3 eingesetzt. Jede topologische Operation dieses Verfahrens musste um eine Komponente erweitert werden, welche die topologische Konsistenz von Biege-Elementen gewährleistet.
Ärzte beschreiben, dass die ILM, verglichen mit dem Kapselsack der Linse, eher spröde als duktil ist und eher „bricht“ als „reißt“. Um dies zu berücksichtigen, liegen die Schwellwerte *minInnerStress*, *minEdgeStress* und *minSproutStress* näher beinander als im Kapsulorhexis-Modul und sind insgesamt niedriger angesetzt.
Wenn in einem echten Eingriff zu ruckartig an einem Stück ILM gezogen wird, reißt dieses vom Rest der Membran ab. Entsprechend muss der Reiß-Algorithmus in diesem Fall in der Lage sein, dass gegriffene Stück ILM möglichst schnell vollständig vom übrigen Feder-Masse-Gitter abzutrennen. Daher wird der Reiß-Algorithmus pro Frame in einer separaten Schleife so häufig direkt hintereinander aufgerufen, bis am jeweiligen Risskeim eine Spannung gemessen wird, die unter *minSproutStress* liegt.

Instrument-Interaktionen: Sowohl für das Greifen mit einer Pinzette als auch für das Absaugen mit einem Vitrektor wird weiterhin auf die bisherigen EYESi-Interaktionen zurückgegriffen. Für den Einsatz eines Schabers ist momentan noch eine Vorgänger-Version der in Abschnitt 6 beschriebenen Interaktion aktiv. Da vorgesehen ist, diese veraltete Interaktion durch die aktuelle zu ersetzten, wird an dieser Stelle darauf verzichtet, die Unterschiede zwischen den beiden Versionen zu beschreiben.

Bewertungssystem: Für die Bewertung eines Peelings wird ein kreisrunder Bereich um die Makula definiert. Anschließend wird ermittelt, wieviel Prozent dieses Bereichs freigelegt wurde. Außerdem fließt in die Bewertung ein, wieviel Prozent der abgelösten Membran nicht abgesaugt wurde und noch im Auge schwimmt. In der vorherigen Modul-Version wurden diese Prozent-Angaben bestimmt, indem abgelöste und aus dem Auge entfernte Dreiecke gezählt wurden. Da der kontinuierliche Reiß-Algorithmus sowohl die Anzahl als auch die Geometrie der Dreiecke verändert, wurde das Bewertungssystem auf die Berechnung von Dreiecks-Flächeninhalten umgestellt.

Schwierigkeitsgrad: Die Realisierung verschiedener Schwierigkeitsgrade wurde im Wesentlichen von der vorherigen Modul-Version übernommen: Mit steigendem Schwierigkeitsgrad ist die ILM weniger gut sichtbar und haftet durch

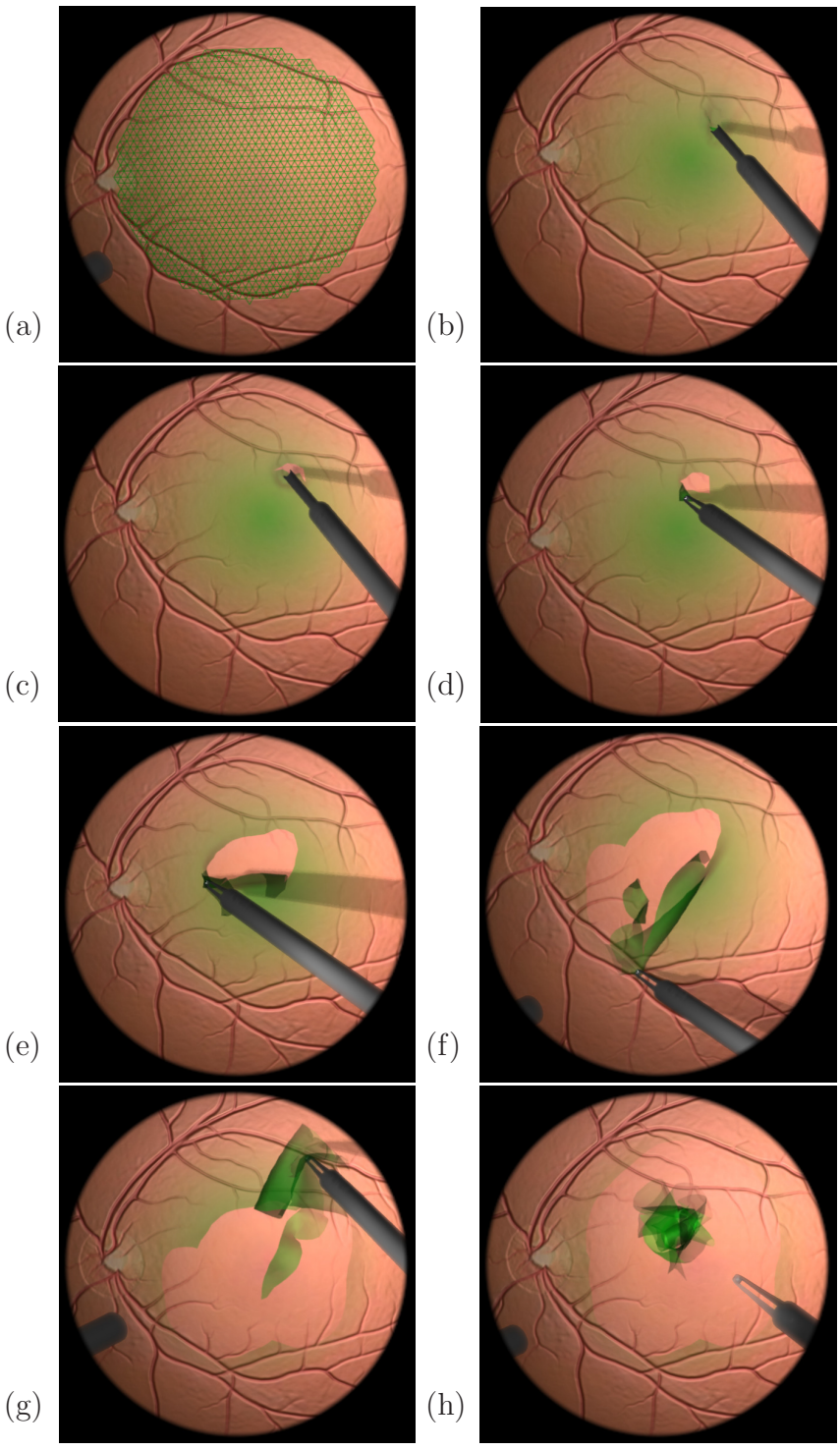


Abbildung 7.2: Peeling der Inneren Grenzmembran

erhöhte Homing-Konstanten stärker an der Retina. Letzteres führt dazu, dass ein Membran-Stück, an dem mit der Pinzette gezogen wird, leichter vom Rest der ILM abreißt.

Abb. 7.2 zeigt das Peeling einer eingefärbten ILM mit dem aktuellen EYESi-Modul. In Bild (a) ist zu sehen, wie das Feder-Masse-Gitter den Bereich innerhalb der Gefäßbögen abdeckt. Die Bilder (b) und (c) illustrieren, wie der Benutzer mit einem Schaber wiederholt über die ILM streicht und sich schließlich ein Riss senkrecht zur Instrument-Bewegung öffnet. Die so entstandene Membran-Lasche wird in Bild (d) mit einer Pinzette gegriffen und Schritt für Schritt vergrößert. In den Bildern (e) bis (h) ist zu beachten, dass dank der glatten Risskante nicht zu erkennen ist, dass der simulierten ILM eine triangulierte Oberfläche zugrunde liegt.

7.2.5 Ergebnisse und Diskussion

Das überarbeitete EYESi-Modul erfüllt die an die Simulation gestellten Anforderungen, ersetzt dessen vorherige Version und wird von VRmagic verkauft. Die wesentlichen Neuerungen betreffen die Schaber-Interaktion und das Reißen der Membran:
Die vorherige, reibungsfreie Schaber-Interaktion ließ beim Streichen über die ILM einen Riss mit zufälliger Ausrichtung entstehen. In der Realität jedoch öffnet sich der Riss hinter dem Instrument und zwar senkrecht zur Bewegungsrichtung. Vor dem Instrument stellt sich ein kleines Stück Membran auf, das anschließend mit der Pinzette gegriffen werden kann. Erst die neue Schaber-Interaktion inklusive Haftreibung bildet diesen Effekt ab.
Im Gegensatz zur Kapsulorhexis ist es während eines ILM-Peelings nicht von besonderer Bedeutung, auf welcher exakten Bahn sich der Riss fortpflanzt. Daher waren Chirurgen, die mit dem vorherigen ILM-Modul gearbeitet haben, mit der Kontrolle über die Rissausbreitung zufrieden, die ihnen der kantenbasierte Reiß-Ansatz gewährte. Die an der Retina zurückbleibende, eckige Risskante der eingefärbten ILM jedoch verriet die dreiecksbasierte Diskretisierung der simulierten Membran. Der Wechsel zum kontinuierlichen Reiß-Algorithmus hat diesen Mangel behoben und lässt die Riss-Simulation insgesamt flüssiger wirken.

7.3 Peeling einer Epiretinalen Membran

Das allererste von VRmagic für EYESi implementierte Trainingsmodul, das eine deformierbare Membran simuliert, ist ein Modul für das Peeling einer Epiretinalen Membran. In den letzten Jahren wurde dieses Modul kontinuierlich gewartet

jedoch kaum weiterentwickelt. Daher erfüllt es die inzwischen gestiegenen Anforderungen nicht mehr, die an die Virtuelle Realität von EYESi gestellt werden. Aus diesem Grund hat VRmagic beschlossen, das Modul einer umfassenden Überarbeitung zu unterziehen. Dieses Re-Design wurde im Rahmen vorliegender Arbeit begonnen und im Folgenden wird sein aktueller Stand beschrieben.

7.3.1 Medizinischer Hintergrund

Im Gegensatz zur ILM stellt eine Epiretinale Membran (abgekürzt *ERM*) eine pathologische Membran dar. Sie ist ein dünnes Häutchen, das lokal auf der Retina wuchert und die Sehkraft beeinträchtigt. Um die ERM abzulösen, benutzt der Chirurg einen Schaber oder eine Pinzette. Anschließend wird die ERM mit Hilfe eines Vitrektors abgesaugt.
Da eine ERM stellenweise sehr fest mit dem Untergrund verwachsen sein kann, besteht während des Eingriffs eine besonders hohe Gefahr für Verletzungen der Retina: Wenn der Chirurg an einem verwachsenen Stück ERM zieht, können Löcher in der Retina darunter aufreißen. Daher muss er beim Ziehen an der ERM darauf achten, wie stark sich die darunter liegende Retina verformt. An kritischen Stellen setzt er sein Instrument wiederholt neu an, bis sich die Membran schließlich löst. Darüber hinaus besteht das Risiko, die Retina durch direkten Kontakt mit der Instrument-Spitze zu beschädigen.

7.3.2 Anforderungen an die Simulation

Um die speziellen Aspekte eines ERM-Peelings abzubilden, muss die Simulation folgende Anforderungen erfüllen:

ERM- und Retina-Modell: Sowohl ERM als auch Retina müssen als deformierbare Oberflächen simuliert werden.

Instrument-Interaktionen: Das Modul muss mindestens einen Schaber, eine Pinzette und einen Vitrektor zu Verfügung stellen.

Gewebe-Reaktionen der Retina: Wenn die Instrument-Spitze ein Stück (freiliegende) Retina berührt, muss sich diese eindellen bevor das Instrument eindringt. In letzterem Fall muss ein Retinaloch entstehen. Wenn sich die Retina beim Ziehen an der ERM zu stark deformiert (siehe „Anhaften und Ablösen“), muss beim Ablösen der ERM ebenfalls ein Retinaloch entstehen.

Anhaften und Ablösen: Das Anhaften der ERM an der Retina muss modelliert werden. Hierbei besonders wichtig: Wenn ein Instrument Kraft auf ein Stück anhaftende ERM ausübt, muss sich die darunter liegende Retina zusammen mit

der ERM deformieren. Sobald sich die ERM ablöst, muss sich die Retina wieder entspannen. An ausgezeichneten Stellen soll die ERM besonders stark anhaften.

Reißen der ERM: Die simulierte ERM muss reißen können.

Bewertungssystem: Das Modul muss – wie jedes EYESi-Trainingsmodul – über ein Bewertungssystem verfügen.

Schwierigkeitsgrad: In der Realität schwankt die Schwierigkeit eines Peelings von Patient zu Patient. Daher soll das Modul über mehrere Levels mit steigendem Schwierigkeitsgrad verfügen.

7.3.3 Vergleichbare Arbeiten

Das bisherige EYESi-Modul ist die einzige der Autorin bekannte Simulation für das Peeling einer Epiretinalen Membran und ist folgendermaßen realisiert:
Die Retina entspricht einer nicht simulierten, triangulierten Oberfläche, deren Knoten auf einer Kugel, der „Augen-Kugel“, angeordnet sind. Als ERM dient ein Feder-Masse-Gitter, wobei die Ruhepositionen der ERM-Knoten ebenfalls auf der Augen-Kugel liegen.
Für die Instrumente werden die auf Knoten-Verschiebungen basierenden EYESi-Interaktionen eingesetzt.
Erst ein abgelöster ERM-Knoten darf von der Feder-Masse-Simulation bewegt werden. Solange ein Knoten anhaftet, wird er nicht simuliert und nur der Kontakt mit einem Instrument kann seine Position ändern. Ein Knoten löst sich, sobald die Länge einer seiner Federn einen Grenzwert überschreitet.
Alle ERM-Knoten verfügen über ein Flag *connected*, das jedoch nur für wenige, im Voraus ausgewählte Knoten auf *true* gesetzt ist. Wenn eine Feder eines *connected*-Knotens beim Ablösen zu stark gedehnt ist, entsteht auf der Retina darunter ein Loch. Beim Ziehen an einem *connected*-Knoten werden umliegende Retina- und ERM-Knoten leicht ins Glaskörper-Innere verschoben, um die Gefahr für ein Loch visuell anzukündigen.
Berührt ein Instrument die Retina, werden Retina-Knoten leicht nach Außen verschoben. Dringt das Instrument tiefer ein, entsteht wiederum ein Retinaloch.
Damit die ERM reißen kann, wird der kantenbasierte Algorithmus von Grimm [57] eingesetzt.
Um ein Peeling zu bewerten, wird ermittelt, wieviel Prozent der ERM gepeelt und aus dem Auge entfernt wurde. Retinalöcher führen zu Punktabzügen.
Das Modul verfügt über sechs Levels mit steigendem Schwierigkeitsgrad. Um so höher der Level, um so höher die Anzahl an *connected*-Knoten und um so geringer die Feder-Dehnung, die für die Entstehung eines Retinalochs notwendig ist.

7.3.4 Das EYESi-Trainingsmodul

Viele Komponenten des bisherigen EYESi-Trainingsmoduls basieren auf Knoten-Verschiebungen, worunter der Gesamteindruck der Simulation leidet. Daher soll die Funktionalität des neuen ERM-Moduls möglichst ausschließlich auf der Berechnung von Knoten-Kräften beruhen, die ein Feder-Masse-Modell zu einer physikalisch plausiblen Simulation verarbeitet. Der aktuelle Entwicklungsstand des neuen ERM-Moduls wird im Folgenden beschrieben:

ERM- und Retina-Modell: Der Teil der Retina, mit dem der Benutzer (voraussichtlich) interagiert, entspricht einem Feder-Masse-Gitter. Die Ruhepositionen der Retina-Knoten liegen auf der Augen-Kugel und definieren gleichzeitig die initialen Homing-Positionen der Retina-Knoten. Als Feder-Masse-Modell für die ERM wird zunächst eine exakte Kopie des Retina-Modells angelegt (inklusive Ruhepositionen und Homing-Positionen). Zwischen den Mesh-Elementen der Retina und denjenigen der ERM ist dadurch eine eindeutige Korrespondenz gegeben. Anschließend werden alle ERM-Dreiecke gelöscht, deren korrespondierendes Retina-Dreieck nicht von der Wucherung betroffen sein soll. Somit zerfallen die Retina-Knoten in zwei disjunkte Mengen: Ein *belegter* Retina-Knoten verfügt über einen korrespondierenden ERM-Knoten, ein *freier* Retina-Knoten dagegen nicht.

Instrument-Interaktionen und Gewebe-Reaktionen der Retina: Bis jetzt ist nur die Interaktion mit einem Schaber in das neue ERM-Modul integriert. Sie ist definiert, wie in Abschnitt 6 beschrieben und berechnet sowohl Gleit- als auch Haftreibungskräfte für den Kontakt zwischen Instrument und ERM. Die Reibungskräfte ermöglichen, die anhaftende ERM von der Retina zu peelen. Die Interaktion zwischen Instrument und Retina dagegen ist reibungsfrei und berechnet Kollisionsantwortkräfte ausschließlich für freie Retina-Knoten. Diese reibungsfreie Kollisionsantwort bewirkt, dass sich die Retina eindellt, bevor die Instrument-Spitze eindringen kann. Wird eine Kollision zwischen Instrument und Retina erkannt, deren Tiefe einen vorgegebenen Grenzwert überschreitet, wird ein Retinaloch auf der Retina-Textur dargestellt. Wie ein Retinaloch durch Zug an der ERM entsteht, wird im Zusammenhang mit dem Modell für Anhaften/Ablösen erläutert.

Anhaften und Ablösen: Die Homing-Positionen der ERM-Knoten liegen auf der Augen-Kugel und werden im Verlauf der Simulation nicht verändert. Das Ablösen der ERM von der Augen-Kugel ist so realisiert, wie in Abschnitt 4.4.2 beschrieben, wobei jeder ERM-Knoten zwei Grenzwerte definiert: $distanceDetach$ und $distanceHole$ mit $distanceDetach < distanceHole$. Überschreitet der Abstand zwischen aktueller Knoten-Position und Homing-Position $distanceDetach$,

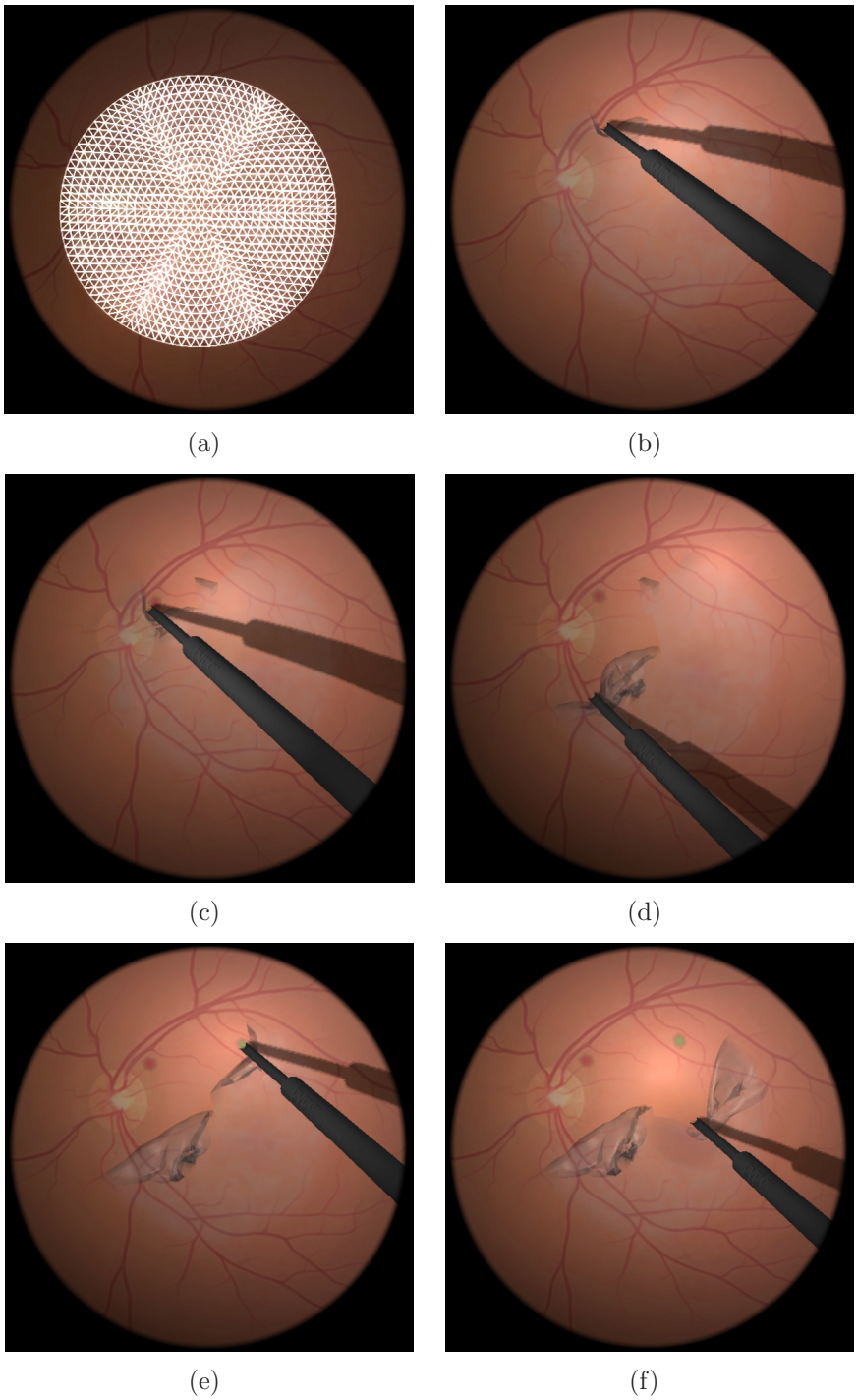
löst sich der ERM-Knoten. Überschreitet der Abstand gleichzeitig $distanceHole$, entsteht ein Loch am korrespondierenden Retina-Knoten.
Solange ein ERM-Knoten anhaftet, kontrolliert er die Homing-Position seines korrespondierenden Retina-Knotens: Vor jedem Integrationsschritt passt der ERM-Knoten die Homing-Position des Retina-Knotens seiner eigenen, aktuellen Position an. (Somit entsteht für den Benutzer der Eindruck, dass die ERM an der Retina haftet, obwohl die Homing-Positionen der ERM-Knoten auf der abstrakten Augen-Kugel liegen.) Sobald sich der ERM-Knoten löst, wird die Homing-Position des Retina-Knotens auf dessen Ruheposition zurückgesetzt.

Reißen der ERM: Um die ERM reißen zu lassen, ist (zur Zeit noch) der kantenbasierte Algorithmus von Grimm [57] aktiviert.

Bewertungssystem: Ein Bewertungssystem existiert noch nicht, kann jedoch direkt aus dem bisherigen EYESi-Modul übernommen werden.

Schwierigkeitsgrad: Die Knoten-Parameter $distanceDetach$ und $distanceHole$ steuern den Schwierigkeitsgrad eines Peelings. Hohe Werte für $distanceDetach$ sind für den Benutzer ein Hinweis für starkes Anhaften. Denn um so höher $distanceDetach$, um so stärker deformiert sich die Retina bevor sich der jeweilige ERM-Knoten ablöst. Und um so näher $distanceHole$ an $distanceDetach$ liegt, um so schwieriger wird es für den Benutzer, ein Retinaloch zu vermeiden.

Abb. 7.3 zeigt das Peeling einer Epiretinalen Membran. Das Feder-Masse-Gitter der Retina deckt den gesamten sichtbaren Bereich ab. In Abb. 7.3(a) ist die Ausdehnung des ERM-Gitters zu sehen. Da eine ERM stark transparent ist, ist sie in den folgenden Bildern nur als weißlicher Schimmer zu erkennen. In Abb. 7.3(b) verziehen sich die Adern der Retina Richtung Instrument, während dieses eine Ablösekante der ERM voranschiebt. Die Retina-Deformation weist den Benutzer darauf hin, an dieser Stelle mit besonderer Vorsicht fortzufahren. Hier jedoch entsteht ein Retinaloch (Abb. 7.3(c)), das durch einen rot eingefärbten Retina-Knoten symbolisiert wird. (Im bisherigen EYESi-Modul sehen Retinalöcher optisch ansprechender aus, bestehen allerdings auch nur aus einer Textur, welche die der Retina überlagert.) Das Peeling wird fortgesetzt (Abb. 7.3(d)) und da das Instrument zu tief in die Retina eindringt, wird ein weiteres Retinaloch am grün eingefärbten Knoten in Abb. 7.3(e) verursacht. (Die reibungsfreie Retina-Interaktion dellt das Retina-Gitter leicht ein, bevor ein Loch entsteht. Im Verlauf mehrerer Frames kann der Benutzer diese Deformation gut wahrnehmen, in einem einzelnen Screenshot jedoch ist sie kaum zu erkennen.) Abb. 7.3(f) deutet den weiteren Verlauf des Peelings an, das die ERM in zwei großen Stücken von der Retina entfernt.



(a) (b) (c) (d) (e) (f)

Abbildung 7.3: Peeling einer Epiretinalen Membran

7.3.5 Ergebnisse und Diskussion

Da sich das Modul zur Zeit noch in der Entwicklungsphase befindet, wurde es bisher von keinem Mediziner evaluiert. Nach Einschätzung der Autorin sind die vorläufigen Ergebnisse jedoch vielversprechend:
Im Gegensatz zum bisherigen EYESi-Modul wird neben der ERM auch die Retina als Feder-Masse-Modell simuliert. Da sämtliche Knoten-Bewegungen ausschließlich auf Knoten-Kräften basieren, wirkt die Simulation insgesamt „organischer“ als im bisherigen EYESi-Modul.
Dank der Konnektivität zwischen ERM- und Retina-Mesh führt das Ziehen an der ERM zur gewünschten Verzerrung der darunter liegenden Retina. Die Parameter des Ablöse-Modells steuern, wie sehr sich die Retina deformiert, bevor sich ein bestimmtes Stück ERM ablöst bzw. bevor ein Loch in der Retina entsteht. Fest verwachsene ERM-Teile verursachen stärkere Deformationen der Retina als leicht verwachsene. Für das Trainingsmodul ist die Simulation dieses Effekts von besonderer Bedeutung, da ein Chirurg lernen muss, von der Stärke einer Retina-Deformation auf die Gefahr für eine Netzhautverletzung zu schließen.
Ein weiterer, wesentlicher Beitrag zum neuen Modul ist die in Kapitel 6 beschriebene Interaktion, mit deren Hilfe alle Gewebe-Reaktionen abgebildet werden, die von einer Instrument-Retina-Interaktion und einer Instrument-ERM-Interaktion gefordert wurden. Die für die ERM-Interaktion berechneten Reibungskräfte führen zu einem realistischen Wechsel zwischen „Ablösen der ERM“ und „Abrutschen von der ERM“.
Um die ERM zu reißen, wurde der kantenbasierte Algorithmus von Grimm [57] zunächst beibehalten, da dieser problemlos mit der Mesh-Mesh-Konnektivität zwischen ERM und Retina vereinbar ist. Jeder andere Reiß-Ansatz würde/wird es wahrscheinlich erforderlich machen, topologische Veränderungen an der ERM parallel auf die Topologie des Retina-Gitters zu übertragen. Um auch im ERM-Modul eine glatte Risskante zu erhalten, bietet sich möglicherweise der Ansatz aus Abschnitt 5.5.2 an: Wenn sich der Riss pro Reiß-Schritt über die komplette, in Rissrichtung gedrehte Kante ausbreitet, würden sich die topologischen Anpassungen am Retina-Mesh auf eine Knoten-Ruheposition und einige Federnulllängen beschränken.

Kapitel 8

Diskussion und Ausblick

Die Zielsetzung für vorliegende Arbeit bestand darin, Simulations-Algorithmen zu definieren, mit denen sich virtuelle Operationsszenarien für Kapsulorhexis, ILM-Peeling und ERM-Peeling realisieren lassen. Sämtliche Entwicklungsarbeiten stehen in Verbindung mit der Simulation der Membrane, die im Verlauf dieser Eingriffe aus dem Auge entfernt werden. Die Simulations-Algorithmen haben die Aufgabe, physikalisch plausible Gewebe-Reaktionen in Echtzeit zu berechnen, wenn der Benutzer mit einer Membran interagiert.

Alle Membran-Simulationen basieren auf einem Feder-Masse-System, das eine triangulierte Oberfläche als Membran-Grundmodell definiert, in dem jede Dreieckskante eine linear-elastische Feder repräsentiert. Sämtliche Anforderungen an das Deformationsverhalten der Membran werden durch geeignete Kräfte realisiert, die – zusätzlich zu den Federkräften des Grundmodells – in der bzw. auf die Membran wirken: Kollisionsantwortkräfte lösen Objekt-Durchdringungen auf, Biege-Kräfte werden durch spezielle Struktur-Federn erzeugt, Homing-Kräfte lassen die Membran am Untergrund haften und die temporäre Adaption von Federkonstanten unterstützt das Ausbilden von Falzkanten. Alle wirkenden Kräfte summieren sich in den Knoten des Simulationsgitters und werden von einem numerischen Integrationsverfahren zu einer physikalisch plausiblen Deformation verrechnet. Auf nachträgliche, verschiebungsbasierte Anpassungen des Feder-Masse-Gitters konnte vollständig verzichtet werden. Dadurch wirken die Membran-Simulationen rundum flüssig und natürlich.
Für die numerische Integration wurde ein explizites Verfahren gewählt, das weniger Rechenzeit beansprucht als ein implizites. An expliziten Verfahren wird üblicherweise bemängelt, dass kleine Zeitschritte notwendig sind, um numerische Stabilität zu garantieren. Dieser Nachteil wird als Vorteil verwertet, indem die kleinen Schrittweiten genutzt werden, um Objekt-Durchdringungen rechtzeitig zu detektieren.

Zusammen mit der Kollisionserkennung – die durch die Einteilung in eine Broad und eine Narrow Phase beschleunigt wird – bilden die einzelnen Simulations-Komponenten ein kompaktes und effizientes Framework, das in Bezug auf die Rechenzeit genügend Spielraum für diejenigen Algorithmen lässt, die das Reißen von Membranen und die Instrument-Membran-Interaktionen simulieren.

Für das Reißen der Membrane wurden zwei neue Algorithmen entwickelt und miteinander verglichen.
Die Bruchkriterien, auf die beide Algorithmen zurückgreifen, basieren auf Auswertungen von Verzerrungen und Spannungen, die alle Aspekte einer physikalisch plausiblen Riss-Simulation abbilden: Ein Materialversagen tritt an der Stelle maximaler Belastung auf, wobei das Wachstum bereits vorhandener Risse wahrscheinlicher ist als die Entstehung neuer Risse. Die Länge einer Rissausbreitung variiert mit der Höhe der Spannung, die sich am Rissende aufgebaut hat. Die berechnete Rissrichtung zeigt an, wohin sich der Riss ausbreiten muss, um die momentane Verspannung wieder abzubauen.
Die zwei Reiß-Algorithmen unterscheiden sich darin, wie sie die Vorgaben der Bruchkriterien in die Topologie des Simulationsgitters einfügen. In wesentlichen Punkten erreichen beide topologische Verfahren gleichwertige Ergebnisse: Sie erzeugen einen Haarriss im Feder-Masse-Gitter und garantieren – im Gegensatz zu anderen Reiß-Algorithmen –, dass die Fläche der Membran erhalten bleibt. Die „on the fly“ durchgeführten Gitter-Adaptionen können einen beliebigen Rissverlauf approximieren und die entstehenden, glatten Risskanten verbergen die dreiecksbasierte Diskretisierung der Membran.
Der erste, sogenannte *pseudo-kontinuierliche* Reiß-Algorithmus gewährt dem Benutzer jedoch nicht dieselbe Kontrolle über die Rissausbreitung wie der zweite, *kontinuierliche* Ansatz: Der pseudo-kontinuierliche Algorithmus aktualisiert die Rissrichtung erst dann, wenn der Riss um eine bestimmte Länge in die zuletzt berechnete Richtung gewachsen ist. Der kontinuierliche Ansatz dagegen passt die Rissrichtung vor jedem einzelnen Rissfortschritt an die aktuelle Gitter-Deformation an und bietet dem Benutzer somit die Möglichkeit, eine vorgegebene Bahn mit dem Rissende exakt „nachzuzeichnen“. Der Preis für diese uneingeschränkte Kontrolle ist eine wachsende Anzahl an Dreiecken im Simulationsgitter. Der pseudo-kontinuierliche Algorithmus dagegen generiert neue Dreiecke nur temporär und erzielt somit einen Vorteil in Bezug auf die Rechenzeitkosten der Feder-Masse-Simulation.
Beide neuen Algorithmen erzeugen realistischere Riss-Simulationen als der bisher in EYESi verwendete, *kantenbasierte* Reiß-Algorithmus. Dieser beschränkt sich darauf, das Simulationsgitter entlang der Dreieckskanten aufzutrennen, die im Rahmen der Gitter-Initialisierung festgelegt werden.

Dank der neu implementierten Instrument-Membran-Interaktion kann der Chirurg eine Membran mit einer Nadel so bearbeiten, wie er es aus einer echten Operation gewohnt ist.
Die Kollisionserkennung der Nadelspitze detektiert Kontakte zwischen Instrument und Membran mit einer Genauigkeit, die sicherstellt, dass alle vom Benutzer erwarteten Gewebe-Reaktionen – die anschließend von der Kollisionsantwort berechnet werden – rechtzeitig eintreten. Die Kollisionsantwort löst Durchdringungen nicht nur auf, sondern lässt zusätzlich Gleit- und Haftreibungskräfte auf die Membran wirken, die der Gewebe-Reaktion den erwünschten Realitätsgrad verleihen. Die Haftreibung simuliert jede Form des Verhakens an der Nadelspitze und ermöglicht dem Benutzer, Zug-Kräfte auszuüben, mit denen sich eine anhaftende Membran kontrolliert und gezielt manipulieren lässt. Sobald diese Kräfte zu hoch werden, reißt sich die Membran von der Nadel los. Somit führt die Haftreibung zu einem realistischen Wechsel zwischen „Ablösen der Membran“ und „Abrutschen von der Membran“.
Die Modelle für Gleit- und Haftreibung verfügen über Parameter, welche die Stärke der Reibung steuern und mit denen sich die Interaktion an die jeweiligen Anforderungen anpassen lässt. Gleit- und Haftreibung sind Erweiterungen eines Grundmodells, das eingesetzt werden kann, falls ausschließlich reibungsfreie Kollisionsantwortkräfte berechnet werden sollen.
Auch im Zusammenspiel mit den Reiß-Algorithmen erzeugt die Nadel-Interaktion eine realistische Gewebe-Reaktion: Wenn der Benutzer die Nadelspitze zwischen die beiden Kanten eines Membran-Risses schiebt und das Instrument anschließend gegen das Rissende drückt, reißt die Membran entlang der Instrument-Bewegung auf. Mit der bisher in EYESi verwendeten Standard-Interaktion für Nadel-Instrumente war dies nicht möglich, da deren Kollisionsantwort physikalisch unplausible Spannungen verursacht, die zu einer chaotischen Rissausbreitung führen.
Verglichen mit der alten EYESi-Interaktion beansprucht die neu entwickelte deutlich mehr Rechenzeit. Dennoch erfüllt sie die Echtzeitbedingung und ist insbesondere numerisch stabiler als ihr Vorgänger.
Insgesamt wirkt die neue Nadel-Interaktion wesentlich realistischer als die bisherige, die keine Reibungsmodelle implementiert und Kollisionen nicht durch Kräfte sondern durch Verschiebungen von Gitter-Knoten auflöst.

Für die Simulation der Kapsulorhexis wurde – neben den Reiß- und Interaktions-Algorithmen – eine Vielzahl an Teil-Komponenten entwickelt, die sich zu einem komplexen Trainingsmodul zusammenfügen, das alle Aspekte des chirurgischen Eingriffs realistisch abbildet. Als vollwertiges EYESi-Modul verfügt es insbesondere über ein Bewertungssystem und über verschiedene Levels mit steigendem Schwierigkeitsgrad.

Der Einsatz eines Zystotoms wurde erst durch die neu entwickelte Instrument-Interaktion möglich. Dieses Modul-Feature ist besonders wichtig für die Ausbildung in Ländern (z. B. Indien), in denen aus Kostengründen auf Pinzetten verzichtet wird.
Da die Kapsulorhexis ein äußerst wichtiger und schwieriger Teil einer Katarakt-Operation ist, wird dieses Modul von den Ärzten mit besonderer Aufmerksamkeit begutachtet. Um so erfreulicher ist, dass erfahrene Chirurgen die Kapsulorhexis-Simulation als überzeugend bezeichnen und ihr einen hohen Wert für die Ausbildung zusprechen.
Von besonderer Bedeutung für die Akzeptanz des Moduls ist der eingesetzte, kontinuierliche Reiß-Algorithmus: In einigen der Kapsulorhexis-Levels ist eine schwarze Kreislinie auf dem Kapselsack eingezeichnet, an der sich der Benutzer orientieren kann. Erfahrenen Chirurgen gelingt es, diese Soll-Linie mittig zu spalten und – wie in einem echten Eingriff – eine vollkommen gleichmäßige und zentrierte Öffnung zu erzeugen. Dies zeigt, dass der Reiß-Algorithmus dieselbe Kontrolle über die Rissausbreitung ermöglicht, die der Chirurg aus der Realität gewohnt ist. Unerfahrene Chirurgen dagegen schneiden deutlich schlechter ab. Einige Anfänger benötigen äußerst viele Wiederholungen, bis ihre Kapsulorhexis die Kreislinie zufriedenstellend approximiert. Demnach stellt die virtuelle Kapsulorhexis, so wie die reale, eine Herausforderung dar, die nur mit viel Übung zu bewältigen ist.
Dass ein Arzt durch das Training am Simulator tatsächlich mehr lernt, als nur höhere Punktzahlen im EYESi-Bewertungssystem zu erreichen, belegt die wissenschaftliche Studie von Feudner et al. [38]: 62 angehende Augenchirurgen wurden in eine Trainingsgruppe und eine Kontrollgruppe aufgeteilt. Alle 62 Testpersonen nahmen an zwei Kapsulorhexis-Wetlabs teil, die im Abstand von drei Wochen stattfanden. Im Verlauf dieser drei Wochen hat die Trainingsgruppe – im Gegensatz zur Kontrollgruppe – zwei EYESi-Drylabs absolviert. Jede Testperson der Trainingsgruppe musste insgesamt achtzehn EYESi-Aufgaben bearbeiten. Zehn dieser Aufgaben stammten aus abstrakten EYESi-Modulen (*forceps training* und *anti-tremor training*), die restlichen acht Aufgaben entsprachen verschiedenen Levels des Kapsulorhexis-Moduls. Das Ergebnis der Studie ist, dass die Trainingsgruppe im zweiten Wetlab signifikant bessere Kapsulorhexis-Resultate erzielte als im ersten Wetlab. In der Kontrollgruppe dagegen konnten keine bzw. nur geringfügige Leistungssteigerungen vom ersten zum zweiten Wetlab festgestellt werden.

Das bereits zu Beginn der Arbeit existierende ILM-Modul wurde überarbeitet, um dessen Simulationen zu verbessern.
Das bisherige ILM-Modul basierte auf dem kantenbasierten Reiß-Algorithmus, welcher die dreiecksbasierte Topologie des Membran-Gitters deutlich erkennen

lässt. Im ILM-Modul ist dies besonders störend, da die ILM grün eingefärbt ist, wodurch sich die unnatürlich aussehende Risskante klar von der darunter liegenden Netzhaut abhebt. Durch den kontinuierlichen, Topologie-unabhängigen Reiß-Algorithmus wurde dieses Problem vollständig behoben.
Ein weiterer, wesentlicher Aspekt der Modul-Überarbeitung bestand darin, die Interaktionen für Nadel-Instrumente zu verbessern. Nadeln werden häufig zu Beginn eines ILM-Peelings eingesetzt, wenn noch keine Membran-Lasche vorhanden ist, die mit einer Pinzette gegriffen werden kann. Die bisher im ILM-Modul eingesetzte Interaktion, die keine Reibung modelliert, führte zu einem zufälligen „Aufplatzen“ der ILM, wenn der Benutzer versuchte, einen ersten Einriss in der anhaftenden Membran zu erzeugen. Die neue Interaktion dagegen simuliert die vom Chirurgen erwartete Gewebe-Reaktion: Wenn der Benutzer mit der Nadelspitze über die ILM streicht, sorgen die Reibungskräfte der Interaktion dafür, dass sich die Membran in Richtung der Instrument-Bewegung verzerrt und anschließend senkrecht zur Bewegungsrichtung aufreißt.
Der Austausch der Reiß- und Interaktions-Algorithmen zog die Anpassung anderer Modul-Komponenten – wie z. B. des Ablöse-Modells und des Bewertungssystems – nach sich. Das vollständig überarbeitete ILM-Modul wurde wieder in die VRmagic-Produktpalette aufgenommen und bietet einen höheren Realitätsgrad als sein Vorgänger.

Für das Peeling einer ERM wurde der Prototyp eines Moduls entwickelt, welches das bisherige EYESi-Modul ersetzen soll. Zu den wichtigsten Neuerungen gehören die Modelle für die Nadel-Interaktionen und für das Anhaften der ERM an der Netzhaut. (Für das Membran-Reißen wurde der kantenbasierte Algorithmus aus dem bisherigen ERM-Modul vorläufig beibehalten.)
Vollkommen neu ist, dass neben der ERM auch die Netzhaut als Feder-Masse-System simuliert wird und dass beide Membran-Gitter durch das Modell für Anhaften/Ablösen aneinander gekoppelt sind. Somit führt jede Interaktion mit der einen Membran zu einer realistischen Gewebe-Reaktion der anderen. Bevor sich ein Stück ERM ablöst, deformiert sich die darunter liegende Netzhaut. So wie in der Realität lässt sich aus der Verformung der Netzhaut ableiten, wie stark die ERM an einer bestimmten Stelle verwachsen ist. Dieser Aspekt ist von besonderer Bedeutung, da eine starke Netzhaut-Deformation die Gefahr für ein Netzhautloch ankündigt. Um die Schwierigkeit eines ERM-Peelings variieren zu können, definiert das Ablöse-Modell Parameter, über die sich steuern lässt, wie leicht ein Netzhautloch entsteht.
Die Reibungsmodelle der neu entwickelten Nadel-Interaktion ermöglichen dem Benutzer, die Membrane auf realistische Art zu manipulieren. Einerseits erlauben sie, die anhaftende ERM abzupellen, andererseits kontrollieren die Modell-Parameter, wie leicht die Membran von der Instrument-Spitze abrutscht. Somit

kann der Schwierigkeitsgrad des Peelings auch über die Interaktions-Parameter angepasst werden.
Sämtliche Simulationen des neuen Moduls basieren auf Kraftberechnungen. Daher wirkt das neue Modul insgesamt realistischer und „organischer“ als das alte, das viele Aspekte der Membran-Deformation durch explizite Verschiebungen von Gitter-Knoten abzubilden versucht.

Zukünftige Arbeiten werden sich auf die Fertigstellung des neuen ERM-Moduls konzentrieren.
Der Prototyp benutzt noch – so wie das bisherige ERM-Modul – den kantenbasierten Reiß-Algorithmus, der das Simulationsgitter entlang von Dreiecksseiten auftrennt. Dass der Wechsel zu einem besseren Reiß-Algorithmus geringere Priorität hatte als die Entwicklung der Interaktions- und Ablöse-Modelle, hat folgenden Grund: Die Kontrolle über die Rissausbreitung, die der kantenbasierte Ansatz gewährt, ist für ein Membran-Peeling prinzipiell ausreichend, da der Riss keiner exakt vorgegebenen Bahn folgen muss. Darüber hinaus ist eine ERM stark transparent, so dass die unnatürliche Geometrie der Risskante weniger auffällt als in der grün eingefärbten ILM. Allerdings bedeutet „weniger auffallend“ keineswegs „nicht auffallend“. Daher soll der kantenbasierte Ansatz durch den pseudokontinuierlichen oder den kontinuierlichen Reiß-Algorithmus ausgetauscht werden. Um eine Wahl zwischen beiden Algorithmen zu treffen, muss überprüft werden, wie gut kompatibel diese mit der gekoppelten ERM-Netzhaut-Simulation sind.
Sonstige Weiterentwicklungen des ERM-Moduls werden beispielsweise den Einsatz von Scheren betreffen. In einem realen ERM-Peeling werden Scheren benutzt, um stark verwachsene ERM-Teile von der Netzhaut loszuschneiden. Die bisherige Scheren-Interaktion von EYESi beschränkt sich darauf, mit den Klingen kollidierende Dreiecke zu löschen, sobald sich die Schere schließt. Daher soll eine neue Interaktion entwickelt werden, die ein flächenerhaltendes Schneiden erlaubt.

Literaturverzeichnis

[1] M. Agus, E. Gobbetti, G. Pintore, G. Zanetti, and A. Zorcolo. Real-time cataract surgery simulation for training. In *Proceedings of the Eurographics Italian Chapter Conference*. Eurographics Association, 2006.

[2] M. Agus, E. Gobbetti, G. Pintore, G. Zanetti, and A. Zorcolo. Real time simulation of phaco-emulsification for cataract surgery training. In *Proceedings of the 3rd Workshop in Virtual Reality Interactions and Physical Simulations*. Eurographics Association, 2006.

[3] R. Alterovitz, K. Goldberg, and A. Okamura. Planning for steerable bevel-tip needle insertion through 2d soft tissue with obstacles. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1640–1645, 2005.

[4] P. M. A. Areias and T. Belytschko. Analysis of three-dimensional crack initiation and propagation using the extended finite element method. *International Journal for Numerical Methods in Engineering*, 63(5):760–788, 2005.

[5] Z. Bao, J.-M. Hong, J. Teran, and R. Fedkiw. Fracturing rigid materials. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):370–378, 2007.

[6] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 43–54. ACM, 1998.

[7] A. H. Barr. Global and local deformations of solid primitives. *SIGGRAPH Computer Graphics*, 18(3):21–30, 1984.

[8] K.-J. Bathe. *Finite-Elemente-Methoden*. Springer-Verlag, second edition, 2002.

[9] D. Bielser, P. Glardon, M. Teschner, and M. Gross. A state machine for real-time cutting of tetrahedral meshes. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 377. IEEE Computer Society, 2003.

[10] D. Bielser, P. Glardon, M. Teschner, and M. Gross. A state machine for real-time cutting of tetrahedral meshes. *Graphical Models*, 66(6):398–417, 2004.

[11] D. Bielser and M. H. Gross. Interactive simulation of surgical cuts. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, pages 116–125. IEEE Computer Society, 2000.

[12] D. Bielser, V. A. Maiwald, and M. H. Gross. Interactive cuts through 3-dimensional soft tissue. *Computer Graphics Forum*, 18(3):31–38, 1999.

[13] J. Bonet and R. D. Wood. *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press, first edition, 1997.

[14] D. E. Breen, D. H. House, and M. J. Wozny. Predicting the drape of woven cloth using interacting particles. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 365–372. ACM, 1994.

[15] D. E. Breen, S. Mauch, R. T. Whitaker, and J. Mao. 3d metamorphosis between different types of geometric models. *Computer Graphics Forum*, 20(3), 2001.

[16] S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer-Verlag, second edition, 2002.

[17] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 594–603. ACM, 2002.

[18] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 28–36. Eurographics Association, 2003.

[19] M. Bro-Nielsen. Finite element modeling in medical vr. *Journal of the IEEE*, 86(3):490–503, 1998.

[20] J. Brown, K. Montgomery, J.-C. Latombe, and M. Stephanides. A microsurgery simulation system. In *Proceedings of the 4th International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 137–144. Springer-Verlag, 2001.

[21] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popović. Interactive skeleton-driven dynamic deformations. *ACM Transactions on Graphics*, 21(3):586–593, 2002.

[22] J. T. Chang, J. Jin, and Y. Yu. A practical model for hair mutual interactions. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 73–80. ACM, 2002.

[23] K.-J. Choi and H.-S. Ko. Stable but responsive cloth. *ACM Transactions on Graphics*, 21(3):604–611, 2002.

[24] T. J. Chung. *Applied Continuum Mechanics*. Cambridge University Press, first edition, 1996.

[25] S. Cotin, H. Delingette, and N. Ayache. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):62–73, 1999.

[26] S. Cotin, H. Delingette, and N. Ayache. A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation. *The Visual Computer*, 16(8):437–452, 2000.

[27] C. Daux, N. Moes, J. Dolbow, N. Sukumar, and T. Belytschko. Arbitrary branched and intersecting cracks with the extended finite element method. *International Journal for Numerical Methods in Engineering*, 48(12):1741–1760, 2000.

[28] F. B. de Casson and C. Laugier. Simulating 2d tearing phenomena for interactive medical surgery simulators. In *Proceedings of the Computer Animation Conference*, page 9. IEEE Computer Society, 2000.

[29] G. Debunne, M.-P. Cani, M. Desbrun, and A. Barr. Adaptive simulation of soft bodies in real-time. In *Proceedings of the Computer Animation Conference*, page 15. IEEE Computer Society, 2000.

[30] G. Debunne, M. Desbrun, A. Barr, and M.-P. Cani. Interactive multiresolution animation of deformable models. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation*, pages 133–144. Springer-Verlag, 1999.

[31] G. Debunne, M. Desbrun, M.-P. Cani, and A. H. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 31–36. ACM, 2001.

[32] H. Delingette, S. Cotin, and N. Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In *Proceedings of the Computer Animation Conference*, page 70. IEEE Computer Society, 1999.

[33] S. P. DiMaio and S. E. Salcudean. Simulated interactive needle insertion. In *Proceedings of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, page 344. IEEE Computer Society, 2002.

[34] B. Eberhardt, A. Weber, and W. Strasser. A fast, flexible, particle-system model for cloth draping. *IEEE Computer Graphics and Applications*, 16(5):52–59, 1996.

[35] E. English and R. Bridson. Animating developable surfaces using nonconforming elements. *ACM Transactions on Graphics*, 27(3):1–5, 2008.

[36] C. Ericson. *Real-Time Collision Detection*. Elsevier, first edition, 2005.

[37] O. Etzmuß, M. Keckeisen, and W. Straßer. A fast finite element solution for cloth modelling. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 244. IEEE Computer Society, 2003.

[38] E. M. Feudner, C. Engel, I. M. Neuhann, K. Petermeier, K.-U. Bartz-Schmidt, and P. Szurman. Virtual reality training improves wet-lab performance of capsulorhexis: results of a randomized, controlled study. *Graefe's Archive for Clinical and Experimental Ophthalmology*, 247(7):955–963, 2009.

[39] C. Forest, H. Delingette, and N. Ayache. Surface contact and reaction force models for laparoscopic simulation. In *Proceedings of the International Symposium on Medical Simulation*, pages 168–176. Springer-Verlag, 2004.

[40] N. Foster and D. Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

[41] N. Foster and D. Metaxas. Controlling fluid animation. In *Proceedings of the 1997 Conference on Computer Graphics International*, page 178. IEEE Computer Society, 1997.

[42] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 181–188. ACM Press/Addison-Wesley Publishing Co., 1997.

[43] H. Freyler. *Augenheilkunde für Studium, Praktikum und Praxis*. Springer-Verlag, 1985.

[44] T.-P. Fries and H.-G. Matthies. Classification and overview of meshfree methods. Technical Report 2003-3, Technical University Braunschweig, Institute of Scientific Computing, 2004.

[45] A. Fuhrmann. *Interaktive Animation textiler Materialien*. PhD thesis, Technische Universität Darmstadt, Fachbereich Informatik, 2006.

[46] A. Fuhrmann, C. Groß, and V. Luckas. Interactive animation of cloth including self collision detection. In *Proceedings of the 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2003.

[47] F. Ganovelli, P. Cignoni, C. Montani, and R. Scopigno. A multiresolution model for soft objects supporting interactive cuts and lacerations. *Computer Graphics Forum*, 19(3):271–282, 2000.

[48] F. Ganovelli, J. Dingliana, and C. O'Sullivan. Buckettree: Improving collision detection between deformable objects. In *Proceedings of the 16th Spring Conference on Computer Graphics*, 2000.

[49] F. Ganovelli and C. O'Sullivan. Animating cuts with on-the-fly re-meshing. Eurographics Short Presentation, 2001.

[50] V. Garcia-Perez, E. Munoz-Moreno, R. de Luis-Garcia, and C. Alberola-Lopez. A 3d collision handling algorithm for surgery simulation based on feedback fuzzy logic. In *Proceedings of the IEEE EMBS International Conference on Information Technology Applications in Biomedicine*, 2006.

[51] S. F. F. Gibson and B. Mirtich. A survey of deformable modeling in computer graphics. Technical Report TR-97-19, Mitsubishi Electric Research Laboratories, 1997.

[52] H. V. Gimbel and T. Neuhann. Development, advantages, and methods of continuous circular capsulorhexis technique. *Journal of Cataract and Refractive Surgery*, 16(1), 1990.

[53] Y. Gingold, A. Secord, J. Y. Han, E. Grinspun, and D. Zorin. A discrete model for inelastic deformation of thin shells. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004.

[54] O. Goksel, S. E. Salcudean, S. P. DiMaio, R. Rohling, and J. Morris. 3d needle-tissue interaction simulation for prostate brachytherapy. In *Proceedings of the 8th International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 827–834. Springer-Verlag, 2005.

[55] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 171–180. ACM, 1996.

[56] J. Grimm. *Interaktive Echtzeitmodellierung von biologischem Gewebe für Virtuelle Realitäten in der medizinischen Ausbildung*. PhD thesis, Universität Mannheim, Fakultät für Mathematik und Informatik, 2005.

[57] J. Grimm. Tearing of membranes for interactive real-time surgical training. In *Proceedings of the Annual Medicine Meets Virtual Reality Conference*, pages 153–159. IOS Press, 2005.

[58] J. Grimm, C. Wagner, and R. Männer. Interactive real-time simulation of the internal limiting membrane. In *Proceedings of the International Symposium on Medical Simulation*, pages 153–160. Springer-Verlag, 2004.

[59] D. Harmon, E. Vouga, R. Tamstorf, and E. Grinspun. Robust treatment of simultaneous collisions. *ACM Transactions on Graphics*, 27(3):1–4, 2008.

[60] M. Hauth. Numerical techniques for cloth simulation. SIGGRAPH Course Notes, 2003.

[61] M. Hauth, O. Etzmuss, B. Eberhardt, R. Klein, R. Sarlette, M. Sattler, K. Daubert, and J. Kautz. Cloth animation and rendering. Eurographics Tutorial Notes, 2002.

[62] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[63] D. L. James and D. K. Pai. Artdefo: Accurate real time deformable objects. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 65–72. ACM Press/Addison-Wesley Publishing Co., 1999.

[64] D. L. James and D. K. Pai. Real time simulation of multizone elastokinematic models. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.

[65] D. L. James and D. K. Pai. Multiresolution green's function methods for interactive simulation of large-scale elastostatic objects. *ACM Transactions on Graphics*, 22(1):47–82, 2003.

[66] Z. Kačić-Alesić, M. Nordenstam, and D. Bullock. A practical dynamics system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 7–16. Eurographics Association, 2003.

[67] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[68] U. G. Kühnapfel, H. K. Çakmak, and H. Maaß. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers & Graphics*, 24(5):671–682, 2000.

[69] J. Lasseter. Principles of traditional animation applied to 3d computer animation. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 35–44. ACM, 1987.

[70] M. LeDuc, S. Payandeh, and J. Dill. Toward modeling of a suturing task. In *Proceedings of the Graphics Interface Conference*, pages 273–279. A K Peters, 2003.

[71] Y.-J. Liu, K. Tang, and A. Joneja. Modeling dynamic developable meshes by the hamilton principle. *Computer-Aided Design*, 39(9):719–731, 2007.

[72] F. Losasso, G. Irving, and E. Guendelman. Melting and burning solids into liquids and gases. *IEEE Transactions on Visualization and Computer Graphics*, 12(3):343–352, 2006.

[73] O. Mazarak, C. Martins, and J. Amanatides. Animating exploding objects. In *Proceedings of the Graphics Interface Conference*, pages 211–218. Morgan Kaufmann Publishers Inc., 1999.

[74] S. Melax. Dynamic plane shifting bsp traversal. In *Proceedings of the Graphics Interface Conference*, pages 213–220. Canadian Human-Computer Communications Society, 2000.

[75] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler. Stable real-time deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 49–54. ACM, 2002.

[76] M. Müller and M. Gross. Interactive virtual materials. In *Proceedings of the Graphics Interface Conference*, pages 239–246. Canadian Human-Computer Communications Society, 2004.

[77] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. *ACM Transactions on Graphics*, 24(3):471–478, 2005.

[78] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 141–151. Eurographics Association, 2004.

[79] M. Müller, L. McMillan, J. Dorsey, and R. Jagnow. Real-time simulation of deformation and fracture of stiff materials. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation*, pages 113–124. Springer-Verlag, 2001.

[80] M. Müller, M. Teschner, and M. Gross. Physically-based simulation of objects represented by surface meshes. In *Proceedings of the 2004 Conference on Computer Graphics International*, pages 26–33. IEEE Computer Society, 2004.

[81] N. Molino, Z. Bao, and R. Fedkiw. A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics*, 23(3):385–392, 2004.

[82] N. Molino, Z. Bao, and R. Fedkiw. A virtual node algorithm for changing mesh topology during simulation. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques: ACM SIGGRAPH 2005 Courses*, page 4. ACM, 2005.

[83] W. Mollemans, F. Schutyser, J. V. Cleynenbreugel, and P. Suetens. Fast soft tissue deformation with tetrahedral mass spring model for maxillofacial surgery planning systems. In *Proceedings of the 7th International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 371–379. Springer-Verlag, 2004.

[84] A. Mor and T. Kanade. Modifying soft tissue models: Progressive cutting with minimal new element creation. In *Proceedings of the 3rd International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 598–607. Springer-Verlag, 2000.

[85] A. B. Mor. *Progressive Cutting with Minimal New Element Creation of Soft Tissue Models for Interactive Surgical Simulation*. PhD thesis, Carnegie Mellon University Pittsburgh, The Robotics Institute, 2001.

[86] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.

[87] M. Neff. A visual model for blast waves and fracture. Master's thesis, University of Toronto, Department of Computer Science, 1998.

[88] M. Neff and E. Fiume. A visual model for blast waves and fracture. In *Proceedings of the Graphics Interface Conference*, pages 193–202. Morgan Kaufmann Publishers Inc., 1999.

[89] H.-W. Nienhuys. *Cutting in deformable objects*. PhD thesis, Universiteit Utrecht, Faculteit Wiskunde en Informatica, 2003.

[90] H.-W. Nienhuys and A. F. van der Stappen. A delaunay approach to interactive cutting in triangulated surfaces. Technical Report UU-CS-2002-044, Utrecht University, Institute for Information and Computing Sciences, 2002.

[91] H.-W. Nienhuys and A. F. van der Stappen. A delaunay approach to interactive cutting in triangulated surfaces. In *Proceedings of the 5th International Workshop on Algorithmic Foundations of Robotics*, pages 113–130. Springer-Verlag, 2003.

[92] H.-W. Nienhuys and A. F. van der Stappen. Interactive needle insertions in 3d nonlinear material. Technical Report UU-CS-2003-019, Utrecht University, Institute for Information and Computing Sciences, 2003.

[93] A. Norton, G. Turk, B. Bacon, J. Gerth, and P. Sweeney. Animation of fracture by physical modeling. *The Visual Computer*, 7(4):210–219, 1991.

[94] J. F. O'Brien, A. W. Bargteil, and J. K. Hodgins. Graphical modeling and animation of ductile fracture. *ACM Transactions on Graphics*, 21(3):291–294, 2002.

[95] J. F. O'Brien and J. K. Hodgins. Graphical modeling and animation of brittle fracture. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 137–146. ACM, 1999.

[96] M. Pauly, R. Keiser, B. Adams, P. Dutre, M. Gross, and L. J. Guibas. Meshless animation of fracturing solids. *ACM Transactions on Graphics*, 24(3):957–964, 2005.

[97] G. Picinbono, H. Delingette, and N. Ayache. Real-time large displacement elasticity for surgery simulation: Non-linear tensor-mass model. In *Proceedings of the 3rd International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 643–652. Springer-Verlag, 2000.

[98] G. Picinbono, H. Delingette, and N. Ayache. Nonlinear and anisotropic elastic soft tissue models for medical simulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1370–1375. IEEE Computer Society, 2001.

[99] G. Picinbono, J.-C. Lombardo, H. Delingette, and N. Ayache. Improving realism of a surgery simulator: Linear anisotropic elasticity, complex interactions and force extrapolation. *Journal of Visualization and Computer Animation*, 13(3):147–167, 2002.

[100] N. Pietroni, F. Ganovelli, P. Cignoni, and R. Scopigno. Splitting cubes: A fast and robust technique for virtual cutting. *The Visual Computer*, 25(3):227–239, 2009.

[101] Project-Team ALCOVE. Interacting with complex objects in collaborative virtual environments. Activity Report INRIA, 2004.

[102] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Proceedings of the Graphics Interface Conference*, 1995.

[103] W. J. M. Rankine. *Manual Of Applied Mechanics (1868)*. Kessinger Publishing, 2007.

[104] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, 21(3):279–288, 2002.

[105] W. Schnell, D. Gross, and W. Hauger. *Technische Mechanik 2: Elastostatik*. Springer-Verlag, fifth edition, 1995.

[106] T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Computer Graphics*, 20(4):151–160, 1986.

[107] B. S. Seibel. *Phacodynamics - Mastering the Tools and Techniques of Phacoemulsification Surgery*. SLACK Incorporated, fourth edition, 2005.

[108] D. Serby, M. Harders, and G. Székely. A new approach to cutting into finite element models. In *Proceedings of the 4th International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 425–433. Springer-Verlag, 2001.

[109] E. Sifakis, K. G. Der, and R. Fedkiw. Arbitrary cutting of deformable tetrahedralized objects. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 73–80. Eurographics Association, 2007.

[110] J. Smith, A. Witkin, and D. Baraff. Fast and controllable simulation of the shattering of brittle objects. In *Proceedings of the Graphics Interface Conference*, pages 27–34, 2000.

[111] J. Smith, A. Witkin, and D. Baraff. Fast and controllable simulation of the shattering of brittle objects. *Computer Graphics Forum*, 20(2):81–90, 2001.

[112] K. Steele, D. Cline, P. K. Egbert, and J. Dinerstein. Modeling and rendering viscous liquids: Research articles. *Computer Animation and Virtual Worlds*, 15(3-4):183–192, 2004.

[113] D. Steinemann, M. Harders, M. Gross, and G. Szekely. Hybrid cutting of deformable solids. In *Proceedings of the IEEE Conference on Virtual Reality*, pages 35–42. IEEE Computer Society, 2006.

[114] A. Tabiei and J. Wu. Development of the dyna3d simulation code with automated fracture procedure for brick elements. *International Journal for Numerical Methods in Engineering*, 57(14):1979–2006, 2003.

[115] J. Teran, S. Blemker, V. N. T. Hing, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 68–74. Eurographics Association, 2003.

[116] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, 1988.

[117] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, pages 269–278. ACM, 1988.

[118] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 205–214. ACM, 1987.

[119] M. Teschner, B. Heidelberger, M. Müller, and M. Gross. A versatile and robust model for geometrically complex deformable solids. In *Proceedings of the 2004 Conference on Computer Graphics International*, pages 312–319. IEEE Computer Society, 2004.

[120] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1):61–81, 2005.

[121] B. Thomaszewski and M. Wacker. Bending models for thin flexible objects. In *Proceedings of the 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2006.

[122] D. L. Tonnesen. *Dynamically Coupled Particle Systems for Geometric Modeling, Reconstruction, and Animation*. PhD thesis, University of Toronto, Department of Computer Science, 1998.

[123] G. Turk. Interactive collision detection for molecular graphics. Technical Report TR90-014, University of North Carolina at Chapel Hill, Department of Computer Science, 1990.

[124] G. van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, 2(4):1–13, 1997.

[125] L. Verlet. Computer experiments on classical fluids, thermodynamical properties of lennard-jones molecules. *Physical Review*, 159(1):98–103, 1967.

[126] P. Volino and N. Magnenat-Thalmann. Comparing efficiency of integration methods for cloth simulation. In *Proceedings of the 2001 Conference on Computer Graphics International*, page 265. IEEE Computer Society, 2001.

[127] P. Volino and N. Magnenat-Thalmann. Accurate garment prototyping and simulation. *Computer-Aided Design & Applications*, 2(5):645–654, 2005.

[128] K. Weber, C. Wagner, and R. Männer. Simulation of the continuous curvilinear capsulorhexis procedure. In *Proceedings of the 3rd International Symposium on Biomedical Simulation*, page 113–121. Springer-Verlag, 2006.

[129] R. Webster, J. Sassani, R. Haluck, R. Shenk, M. Harris, J. Blumenstock, J. Gerber, C. Billman, and A. Benson. Simulating the continuous curvilinear capsulorhexis procedure during cataract surgery on the EYESI^{TM} system. In *Proceedings of the Annual Medicine Meets Virtual Reality Conference*, pages 592–595. IOS Press, 2005.

[130] R. Webster, J. Sassani, R. Shenk, and G. Zoppetti. Simulating the continuous curvilinear capsulorhexis procedure during cataract surgery. In *Proceedings of the 15th International Conference on Modelling and Simulation*, pages 262–265. ACTA Press, 2004.

[131] M. Wicke, M. Botsch, and M. Gross. A finite element method on convex polyhedra. *Computer Graphics Forum*, 26(3):355–364, 2007.

[132] M. Wicke, D. Steinemann, and M. Gross. Efficient animation of point-sampled thin shells. *Computer Graphics Forum*, 24(3):667–676, 2005.

[133] A. Witkin and D. Baraff. Physically based modeling. SIGGRAPH Course Notes, 2001.

[134] World Health Organization. Prevention of avoidable blindness and visual impairment. Report by the Secretariat for the Sixty-Second World Health Assembly, 2009.

[135] G. D. Yngve, J. F. O'Brien, and J. K. Hodgins. Animating explosions. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 29–36. ACM Press/Addison-Wesley Publishing Co., 2000.

[136] G. Zachmann and E. Langetepe. *Geometric Data Structures for Computer Graphics*. A K Peters, Ltd., first edition, 2006.

[137] D. Zhang and M. M. F. Yuen. Collision detection for clothed human animation. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, page 328. IEEE Computer Society, 2000.

[138] J. Zhang, S. Payandeh, and J. Dill. Haptic subdivision: An approach to defining level-of-detail in haptic rendering. In *Proceedings of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, page 201. IEEE Computer Society, 2002.

Abkürzungsverzeichnis

2D	zweidimensional
3D	dreidimensional
AWP	Anfangswertproblem
BVH	Bounding Volume Hierarchy
DGL	Differentialgleichung
DOG	Deutsche Ophthalmologische Gesellschaft
ERM	Epiretinal Membrane
EYESi	Eye Surgery Simulator
FDM	Finite Difference Method
FEM	Finite Element Method
FFD	Free-Form Deformation
FPGA	Field Programmable Gate Array
FVM	Finite Volumen Methode
GUI	Graphical User Interface
ILM	Internal Limiting Membrane
MV	Modelview
OLED	Organic Light Emitting Diode
PDGL	Partielle Differentialgleichung
REM	Randelementmethode
VR	Virtual Reality
XFEM	Extended Finite Element Method