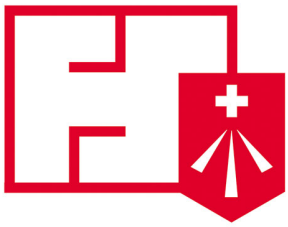


fachhochschule stralsund

university of applied sciences

fachbereich elektrotechnik + informatik

school of electrical engineering + computer science

Physikalische Simulation und Visualisierung der Korrektur von Zahnfehlstellungen mittels Spangen

Bachelorarbeit

zur Erlangung des Grades Bachelor of Science Informatik
der Fakultät für Elektrotechnik und Informatik
der Fachhochschule Stralsund

vorgelegt von

Name: Kemnitz Vorname: Stefan

Geb. am: 17.03.1988 in: Schwedt/Oder

Erstprüfer: Prof. Dr. sc. hum. Hans-Heino Ehricke

Danksagung

Der Autor möchte sich bei dem “Zentrum für angewandte Informatik, flexibles Lernen und Telemedizin”, und Professor Kordaß für die Bereitstellung der Daten bedanken, die in diesem Projekt benutzt wurden. Die Unterstützung durch das Zentrum für Zahn-, Mund- und Kieferheilkunde, die Poliklinik für Kieferorthoädie und des physikalischen Institut der Ernst-Moritz-Arndt Universität Greifswald war überaus hilfreich. Besonders möchte sich der Autor bei, Dr. Alexander Spassov, Professor Bernd Kordaß und Professor Ralf Schneider für die Zusammenarbeit bedanken.

Zusammenfassung

Das Ziel der Kieferorthopädie (KFO) ist, die Verbesserungen des dentofazialen Erscheinungsbilds durch gezielte Bewegung von Zähnen oder Teilen des Kiefers. Ein weiteres Anliegen der KFO ist die Minimierung der, die Behandlungszeit um die Schmerzen der Patienten zu minimieren. Für eine optimale und patientenorientierte KFO-Behandlungsplanung, stellt die Simulation von Zahnbewegungen einen Weg dar, potentielle Probleme frühestmöglich festzustellen, um sie zu Vermeiden. Neue Behandlungsansätze können im Voraus getestet werden, um Schäden zu reduzieren und langwierige Behandlungen zu verkürzen.

In dieser Arbeit werden zuerst die Physik der starren Körper, die theoretischen Grundlagen für die Bewegung eines Zahns durch den Zahnhalteapparat aus zahnmedizinischer Sicht und die Grundlagen der Visualisierung beleuchtet. Ein großer Teil der Arbeit beschäftigt sich mit der Implementation der grafischen Darstellung durch einen Szenengraphen. Hierbei wird ein Szenengraph verwendet, der den Zugriff auf die Zustandsmaschine von OpenGL durch einen Szenengraphen abstrahiert. Desweiteren wird auf die Umsetzung und Automatisierung von Tests, die die Funktionstüchtigkeit der Software gewährleisten sollen, eingegangen.

Basierend auf diesen Grundlagen wird ein Programm vorgestellt, welches es erlaubt, die auf Zähne und Zahnhalteapparat einwirkenden Kräfte zu simulieren und zu visualisieren. Besonders wichtig sind die nutzerangepasste Bedienbarkeit der Software und die Visualisierung. Die Kräfte und die dynamische Systemantwort des Zahnhalteapparats sind dabei vom Nutzer einstellbar oder über Messdaten zuführbar. Eine Validierung mit bereits aufgezeichneten Daten soll die Anwendbarkeit qualifizieren.

Inhaltsverzeichnis

Einleitung

1.1 Motivation

Im Rahmen einer Zusammenarbeit zwischen dem Zentrum für Zahn-, Mund- und Kieferheilkunde, der Polilinkik für Kieferorthopädie und dem physikalischen Institut der Ernst-Moritz-Arndt Universität Greifswald werden die Krafteinwirkungen von Zahnspangen auf Zähne und den Zahnhalteapparat untersucht. Hierbei besteht die Hauptaufgabe darin, herauszufinden weshalb sich die Zähne in bestimmter Art und Weise bewegen und wie diese Bewegungen vorhersagbar gemacht werden können. Grundlage der Bewegung von Zähnen ist die Theorie der Bewegung starrer Körper, wobei die entscheidenden Größen die wirkenden Kräfte und Drehmomente sind. Deshalb ist eine Visualisierung der wirkenden Kräfte und der Drehmomente für ein umfassendes Verständnis unerlässlich. Allerdings muss man sich bewusst sein, dass die komplexen biologischen Vorgänge mit Knochenab- und aufbau nicht im Detail beschrieben werden, sondern nur empirisch parametrisiert als Gegenkraft des Gewebes berücksichtigt werden.

1.2 Relevanz

Durch die Vorhersage von Zahnbewegungen kann die Behandlungszeit eines Patienten wesentlich verkürzt werden. Es ist möglich Komplikationen frühzeitig zu erkennen und Zahnbewegungen können für den Patienten verständlicher gemacht werden.

Die Zahnbewegung ist ein komplizierter und komplexer biologischer Prozess, bei welchem sich der Zahn im Durchschnitt um ca. 1mm in 4-6 Wochen bewegt. Deshalb ist eine präzise Therapiesetzung und Durchführung notwendig, um langandauernde Behandlungszeiten zu reduzieren, denn sie bedeuten mehr Kosten und ein erhöhtes Risiko für Gewebsschäden. Zudem scheuen viele Patienten eine KFO-Behandlung aufgrund der langen Behandlungszeiten (ca. 2-3 Jahre). Eine vorhersagbare, ohne Umwege zielgerichtete Zahnbewegung kann die Behandlungszeit deutlich reduzieren.

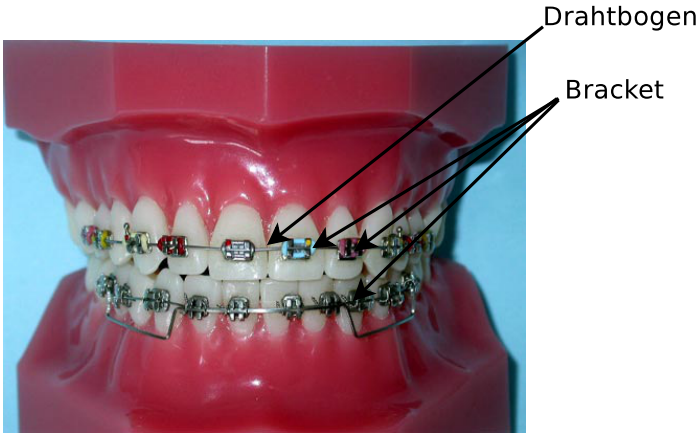


Abbildung 1.1: Modell eines Gebisses [11]

Eine Veränderung der Zahnposition kann durch Ganzbögen oder durch Teilbögen erreicht werden. Eine festsitzende Spange besteht aus einem Drahtbogen und den Brackets. Die Brackets werden mittels eines Klebers auf der Krone vestibulär oder lingual aufgebracht und stellen den Aufhängepunkt für den Drahtbogen dar. Soll ein Zahn bewegt werden, so wird der Draht gebogen (aktiviert) und in den Brackets eingegliedert, so dass eine Spannung entsteht. Dabei entsteht eine Kraft, welche am Bracket angreift und diese auf den Zahn überträgt. Je nach Zielsetzung und Therapieplan werden Drähte mit unterschiedlichen Eigenschaften (Material, Dicke, Form oder Flexibilität) verwendet. Die Beschreibung der Zahnbewegung über ein numerisches Modell ist daher sehr kompliziert, da man sowohl physikalische Gesetzmäßigkeiten als auch medizinische Kenntnisse für eine realistische Beschreibung verbinden muss.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist es, die Bewegung von Zähnen durch die Physik der starren Körper zu simulieren. Diese Beschreibung der komplexen biologischen und mechanischen Vorgänge bei Zahnbewegungen und der Wechselwirkung von Zahn, Zahnspange, Weichgewebe und Knochen ist im Limit der Physik starrer Körper sicher nur eine Näherung der Wirklichkeit, leistet aber über parametrische Anpassung an die Realität sehr gute Dienste für den Zahnarzt und in der Lehre zur Planung von Behandlungen und für den Patienten als Aufklärungshilfe. Hiermit lassen sich die Bewegungen ausreichend genau beschreiben und sind im Gegensatz zu Finite-Elemente-Rechnungen auf normaler Hardware umsetzbar. Zur Validierung des Modells und zum einfachen Verständnis der Ergebnisse, wird viel Wert auf eine Visualisierung des Ergebnisses gelegt. Als Grundlage für die Berechnung werden idealisierte Gitternetzdaten benutzt. Eine Anpassung an spezifische Daten von Patienten wird nicht Teil dieser Arbeit sein, da dies auch mit erschwerten datenrechtlichen Auflagen verbunden wäre.

Die beschriebene Anwendung “VisTeethDyn” (Bild 1.2) wird aus drei Hauptkomponenten bestehen, nämlich der Dynamik starrer Körper, den medizinischen Vorgaben und der Visualisierung. Die Dynamik starrer Körper ist über die physikalischen Grundgleichungen vorgegeben. Die medizinischen Vorgaben definieren zum Beispiel Drehpunkt oder Drehachse der zu bewegenden Zähne oder Zahngruppen, etwa durch die Art der Befestigung der Zahn-

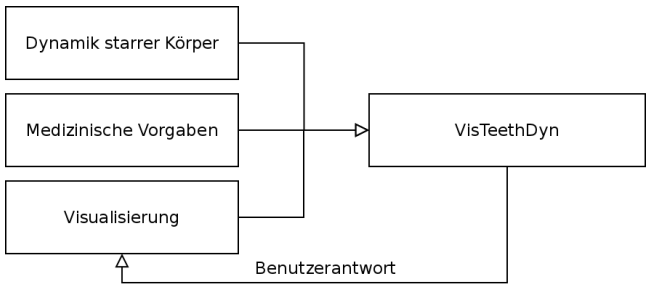


Abbildung 1.2: Aufbau der Software

spangen. Weiterhin bestimmt das biologische System und seine Reaktion auf die Zahnspange den effektiven Drehpunkt und mögliche Gegenkräfte durch den Zahnhalteapparat. Die Visualisierung ist die Grundlage der Interaktion mit dem Anwender, nämlich dem Zahnarzt oder Kieferorthopäden, den Studenten der Zahnmedizin, oder aber auch mit dem Patienten. Insofern hat dieses Element eine besondere Bedeutung für das Projekt und bestimmt zum großen Teil die mögliche Anwendbarkeit in der Praxis. Die drei Hauptkomponenten bedingen sich gegenseitig: Medizinische Vorgaben beeinflussen die Dynamik, stellen aber auch Ansprüche an die Visualisierung.

1.4 Aufbau der Arbeit

Im Kapitel 'Grundlagen' werden die Physik der starren Körper, sowie die Grundlagen für die Bewegung von Zähnen aus zahnmedizinischer Sicht beleuchtet. Das Kapitel 'Vorarbeiten' wird die bereits existierenden Arbeiten und die Notwendigkeit der Entwicklung eines neuen Programms im Rahmen dieser Arbeit diskutiert. Die Anforderungen an dieses Projekt werden in der Anforderungsdefinition behandelt, welche die "Use cases" und nichtfunktionalen Anforderungen an die Software beschreiben werden. Der grundlegende Aufbau des Programms wird im Kapitel 'Entwurf' behandelt und konzeptionelle Entscheidungen werden erklärt. Im Kapitel 'Implementation' wird das entwickelte Programm vorgestellt und auf die Implementation der physikalischen und zahnmedizinischen Grundlagen eingegangen. Danach werden die Ergebnisse präsentiert, eine Leistungsbewertung vorgenommen und ein Ausblick auf zukünftige Entwicklungen gegeben.

Grundlagen

2.1 Zahnmedizinische Grundlagen

In diesem Abschnitt werden die zahnmedizinischen Grundlagen erklärt, die für die Umsetzung der Simulation notwendig sind. Die Hauptkomponenten des menschlichen Kiefers, die Zähne, das Zahnfleisch, die Wurzelhaut, der Zahnhalteapparat, sowie die Verbindung dieser Komponenten werden hierbei näher beleuchtet. Außerdem soll erklärt werden welche Mittel und Methoden verwendet werden um eine Zahnstellung mittels Zahnspangen zu verändern. Hierfür kommen Brackets und Drahtbögen zum Einsatz. Nur durch ein genaues Verständnis der Verknüpfung dieser Grundelemente kann eine exakte Simulation erfolgen.

2.1.1 Aufbau des Kiefers

Der menschliche Kiefer besteht aus dem Kieferknochen, in welchen sich Zahnfächer befinden. In diesen Fächern, oder auch Alveolus dentalis ist das Zahnzement (Cementum) durch die Wurzelhaut mit dem Fach verbunden. “Die Wurzelhaut füllt den 0,1-0,3 mm großen Periodontalspalt zwischen Alveolarknochen und Zahnzement” [6].

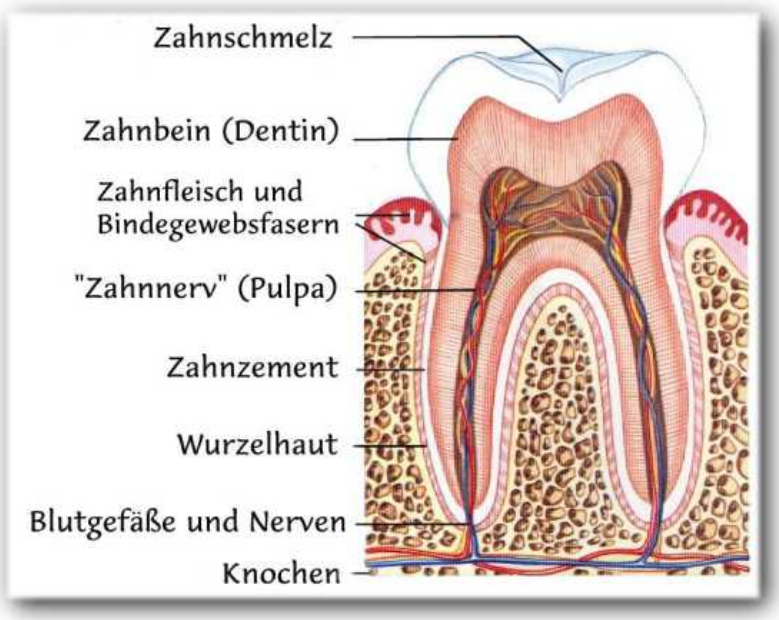


Abbildung 2.1: Aufbau des Zahnhalteapparats [9]

Der Zahn ist mittels Fasern (Sharpey-Faser) im Alveolarknochen aufgehangen. Dadurch ist es dem Zahn möglich, minimale Bewegungen auszuführen, um Kräfte die während des Kauens und Schluckens entstehen abzufedern oder dafür zu sorgen, dass größere Kräfteinwirkungen nicht zu Schädigungen im Kieferknochen führen.

2.1.2 Zahnstellung

Die Zahnstellung unterliegt einer enormen Variationsbreite. Bestimmte Variationen der Zahnstellung können das psychosoziale Wohlbefinden eines Menschen stören. In diesem Fall ist eine Korrektur der Zahnstellung indiziert, um das Wohlbefinden des Patienten zu Verbessern. Ein geringer Teil der Variationen der Zahnstellung stört die Kau- und Sprechfunktion. In diesen Fällen kann eine gezielte Zahnstellungsänderung zur Verbesserung der Kau- und Sprechfunktion führen.

2.1.3 Zahnbewegung

Um die Position der Zähne im Kiefer zu verändern, gibt es einerseits die Möglichkeit eines chirurgischen Eingriffs, wenn extreme Variationen der Kieferlagen vorliegen, die das Wohlbefinden des Patienen stark beeinträchtigen. Andererseits besteht die Möglichkeit durch Einwirkung von Kräften auf einen Zahn, eine Ausrichtung zu erzeugen. Man versucht den chirurgischen Eingriff zu umgehen, da dieser Risiken mit sich bringt. Da extreme Variationen der Kieferlagen sehr selten sind, wird im Großteil der Fälle durch kieferorthopädische Apparaturen Druck auf einen Zahn angewandt. Genau betracht kommt es bei der Anwendung einer Kraft (siehe Bild 2.2 A) auf einen Zahn über einen längeren Zeitraum zu einem Knochenabbau- und Knochenaufbauprozess im Alveolus dentalis.

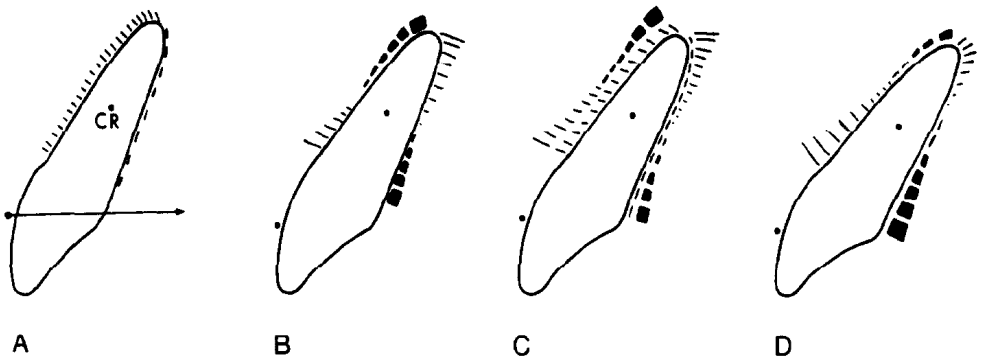


Abbildung 2.2:
Knochenauf- und Knochenabbauprozess
Breite Striche Knochenabbauzone
Dünne Striche Knochenaufbauzone
CR - Center of Resistance [?]

"Durch den kieferorthopädischen Druck wird der Zahn in seinem Knochenfach ("Alveole") ausgelenkt. Auf der Seite, von der der Druck kommt werden die Fasern gespannt, sie ziehen am Knochen. Auf der anderen Seite werden die Fasern zusammengedrückt." [12]

Durch das Zusammendrücken wird der Blutfluss in den Gefäßen vermindert, was zur Folge hat, dass ein Knochenabbauprozess in Gang gesetzt wird. Auf der anderen Seite setzt zugleich ein Aufbauprozess an, der den Raum zwischen Zahnzement und Alveolus dentalis auffüllt.

Zahnbewegung durch Zahnspangen

Zur Verschiebung von Zähnen hat sich heute die Bewegung durch festsitzende Geräte, sogenannte Multibandapparaturen durchgesetzt. Eine Zahnspange besteht aus Brackets und Drahtbögen. Brackets sind kleine Befestigungselemente aus Kunststoff, Keramik oder Metall, die dazu benutzt werden den Draht am Zahn zu fixieren. Sie werden mit besonderen Klebern auf dem Zahn befestigt und übertragen die Kraft, die durch den Drahtbogen erzeugt wird, auf den Zahn. Im Allgemeinen gibt es zwei Vorgehensweisen. Es gibt die Möglichkeit einen elastischen Drahtbogen in die Form eines idealen Zahnbogens zu bringen, oder einen Drahtbogen so vorzubiegen, so dass eine geziehlte Bewegung entsteht. Der Drahtbogen wird dann in ein Bracket eingespannt und mit einem anderen Draht befestigt. Dieser strebt, durch seine Elastizität dazu, in die Ausgangslage zurückzukommen. Dabei überträgt er die Kraft auf die Brackets, welche wiederum die Kraft auf die Zähne übertragen und damit die Zähne in die Form des Drahtbogens bringen.

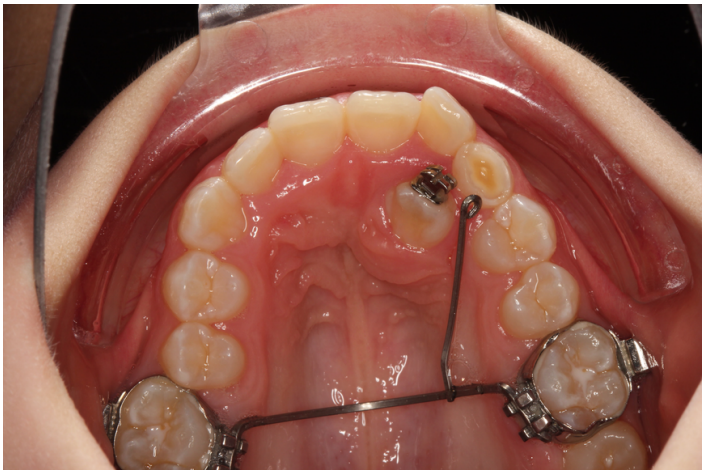


Abbildung 2.3: Vorgespannter Drahtbogen

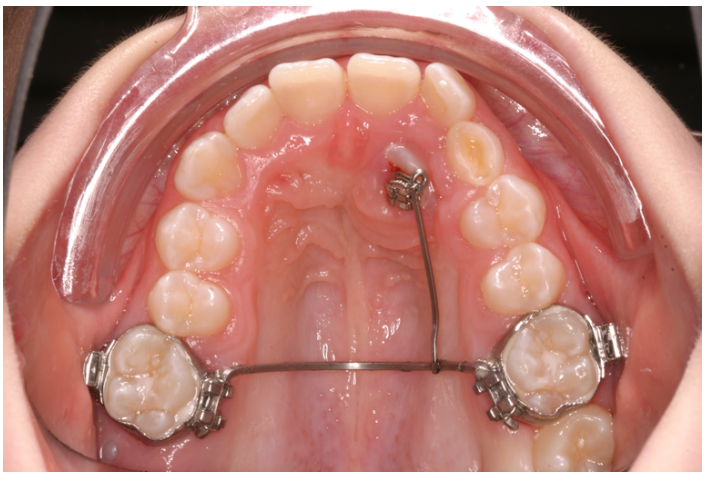


Abbildung 2.4: Drahtbogen, der versucht in seine Ruhelage zurück zu kehren

Die verwendeten Drahtbögen haben aufgrund unterschiedlicher Materialien, unterschiedliche Materialeigenschaften. Diese Materialeigenschaften sorgen für eine unterschiedliche Erzeugung von Kräften. Drähte können sich auch durch ihre Form unterscheiden. So gibt es zum Beispiel runde und vierkantige Drähte. Ein vierkantiger Draht erlaubt, aufgrund seiner unfähigkeit sich im Brackt zu Drehen, eine bessere Kontrolle der Zahnbewegung.

Typodont

In der KFO werden Typodonts aus Kunststoff und Wachs benutzt um Zahnbewegungen zu simulieren. Beispielzähne können im Typodont eingesetzt und Multibandapparate angebracht werden. Durch Erhitzen des Typodonten wird das Wachs weich und der Widerstand des Typodonten sinkt. Durch die Abnahme des Widerstands ist es der Apparatur möglich, Zahnattrappen, die im Wachs stecken, in Bewegung zu setzen.

Die durch den Typodonten simulierte Bewegung stellt eine außreichend genaue Näherung dar, um Erfahrungen zu gewinnen. Jedoch ist der Nachteil des Verfahrens, dass die Typodonten immer wieder in ihre Ausgangsform zurück gebracht werden müssen, wodurch ein wiederholtes Ausprobieren sehr zeitaufwändig ist. Außerdem sind die Kosten mit bis zu Eintausend Euro relativ hoch.

2.2 Physikalische Grundlagen

Möchte man die Bewegung der Zähne, wie durch die medizinischen Vorgaben beschrieben, simulieren, so bietet sich die Physik der starren Körper (engl. rigid body dynamics) an. Durch diese Methode ist es möglich die Bewegung der Zähne durch den Alveolar zu berechnen, da es sich um sehr langsame Bewegungen handelt.

Eine andere Möglichkeit der Simulation wäre die Finite Elemente Methode (FEM). Durch diese Methode, bei der alle Körper in endliche kleine Teile aufgeteilt werden, ist es möglich, zu einem genaueren Ergebnis zu kommen. Der Einsatz dieser Methode wird einerseits dadurch erschwert, dass sich solch komplexe biologische Prozesse nicht einfach über Materialkonstanten beschreiben lassen, da es nur wenige Informationen, über die Auf- und Abbaugeschwin-

digkeiten von Geweben, gibt. Andererseits ist die FEM sehr rechen- und speicherintensiv und würde einen Einsatz auf einem Desktop-PC nahezu unmöglich machen (Anforderung 4.2.4)

2.2.1 Starrer Körper

Ein starrer Körper ist dadurch definiert, dass sich zu jedem Zeitpunkt alle Punkte des Körpers in einem zeitlich konstanten Abstand zueinander befinden. Durch diese Eigenschaft ist es möglich, den Körper auf den Massenschwerpunkt englisch "center of mass" (COM), eine Orientierungsmatrix und den Trägheitstensor herunterzubrechen (siehe Seite 10). Alle Aufwendungen von Kraft auf den starren Körper werden dann nur noch auf den COM angewandt und mittels Orientierungsmatrix und Trägheitstensor lässt sich eine neue Orientierungsmatrix und Position des COM errechnen.

Massenschwerpunkt - center of mass

Der Massenschwerpunkt ist definiert durch:

$$\vec{r}_s = \frac{1}{m_g} \int_K \vec{r} \cdot d\,m \tag{2.1}$$

m_g = Gesamtmasse
$\vec{r}_s$ = Massenschwerpunkt
$\vec{r}$ = Ortsvektor des Massenelements $d\,m$ des Körpers

Für Punktmassen gilt

$$\vec{r}_s = \frac{\sum_{n=1}^{N} \vec{r}_n \cdot m(\vec{r}_n)}{m_g} \tag{2.2}$$

$\vec{r}_s$ = Massenschwerpunkt
N = Anzahl der Punkte des Körpers
$\vec{r}_n$ = Koordinate eines Punkts im Körper
$m(\vec{r}_n)$ = Masse von $\vec{r}_n$

Dabei wird eine durch die Masse $m(\vec{r}_n)$ gewichtete Summation des Vektors $\vec{r}_n$ vorgenommen, aus deren Mittel sich der COM $\vec{r}_s$ ergibt. Soll das Objekt nur verschoben werden, genügt dieser Punkt um zu beschreiben, an welcher Position sich das Objekt nach der Verschiebung befindet. Bei Rotationen ist dieser Punkt der Punkt um den sich der Körper dreht. Um sich besser vorstellen zu können an welcher Stelle sich der Massenschwerpunkt befindet, kann man einen Stab auf einem Finger balancieren. Der Massenschwerpunkt befindet sich bei einem ausbalancierten Stab genau über dem Finger.

Trägheitstensor

Das Trägheitsmoment gibt "den Widerstand eines starren Körpers gegenüber einer Änderung seiner Rotationsbewegung an" [22] Um diese für einen Körper im dreidimensionalen Raum

zu beschreiben, wird ein sogenannter Trägheitstensor benutzt. Dieser ist durch

$$I = \iiint_V \rho(x,y,z) \left((x^2+y^2+z^2)\mathbf{E}_3 - \begin{bmatrix} xx & xy & xz \\ yx & yy & yz \\ zx & zy & zz \end{bmatrix} \right) dx\,dy\,dz \tag{2.3}$$

I	=	Trägheitstensor	x, y, z	=	Raumdimensionen
$\mathbf{E}_3$	=	dreidimensionale Einheitsmatrix			

definiert.

Der Trägheitstensor kann auch anders interpretiert werden, da er es ermöglicht das Trägheitsmoment L zu jeder möglichen Rotationsachse $\vec{\omega}$ zu berechnen.

$$L = \frac{1}{\vec{\omega}^2} \sum_{i=1}^{3} \sum_{j=1}^{3} I_{ij} \vec{\omega}_i \vec{\omega}_j \tag{2.4}$$

L	=	Trägheitsmoment	$\vec{\omega}$	=	Drehachse
I	=	Trägheitstensor	i, j	=	1-3 entsprechen den Raumrichtungen x,y,z

Der Tensor bezieht sich dabei immer auf den COM und kann zum Beispiel bei erzwungenen Drehungen durch den Steinerschen-Satz angepasst werden.

$$I_{ij} = I_{ij} + m(\sum a_k^2 \delta_{ij} - a_i\, a_j) \tag{2.5}$$

Dies ist, wenn die Drehachse für ein Objekt vorgegeben werden soll wichtig.

a	=	Verschiebungsvektor	δ_{ij}	=	Kronecker-Delta

Orientierung

Um eine Rotation darzustellen, sind unterschiedliche Ansätze möglich. Der gebräuchlichste ist die Rotation um eine der Koordinatensystemsachsen (x- y- z-Achse). Wird diese Art von Darstellung einer beliebigen Drehung verwendet, muss eine Rotation aus mehreren, hintereinander ausgeführten Rotationen bestehen, um jede mögliche Ausrichtung zu erreichen. Einfacher ist es jedoch eine Matrix zu verwenden, welche die Summe der Einzeldrehungen zusammenfasst.

2.2.2 Anwendung von Kraft auf einen starren Körper

Um einen starren Körper zu bewegen, werden Punkte $\vec{r}_n$ im Raum und Kräfte $\vec{f}_n$ benötigt. Die Punkte $\vec{r}_n$ geben dabei an, an welchen Stellen die Kräfte auf den Körper appliziert werden. Da die aufgewendeten Kräfte $\vec{f}_n$ nicht nur in Translation resultieren, sondern auch in Rotation, wird aus $\vec{f}_n$ durch

$$\vec{f}_b = \sum_{n=1}^{N} \vec{f}_n \tag{2.6}$$

$\vec{f}_b$ = Akkumulierte Kraft auf den Körper $\vec{f}_n$ = Teilkraft auf den Körper

und

$$\vec{\tau}_b = \sum_{n=1}^{N} \vec{r}_n \times \vec{f}_n \tag{2.7}$$

$\vec{\tau}_b$ = Drehmoment $\vec{r}_n$ = Punkt bzgl. Körper

die Gesamtkraft $\vec{f}_b$ und der Torque $\vec{\tau}_b$ berechnet. In dem in dieser Arbeit vorgestellten Programm, soll es möglich sein die Drehbeweung eines Objektes einzuschränken. Hierfür wird die Rotationsachse als Referenz herangezogen.

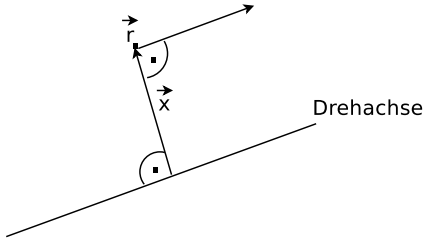


Abbildung 2.5: Dreieck in Beweungsrichtung

wenn

$$\tau_{\vec{\omega}} = \sum_{n=0}^{N} \vec{x} \times \vec{\omega} \tag{2.8}$$

$$\vec{x} =\perp \vec{\omega} \tag{2.9}$$

$\tau_{\vec{\omega}}$ = Drehmoment bzgl. einer Drehachse

Lineare Bewegung Der Torque oder zu deutsch Drehmoment ist das Äquivalent zur Kraft bei linearen Bewegungen und wird für die Errechnung des Drehimpulses und die Winkelgeschwindigkeit benötigt. Um die neue Position eines Körpers nach dem Aufwenden einer Kraft zu bestimmen, wird für kleine Zeitschritte t durch

$$\vec{r}_s{}' = \vec{r}_s + \vec{v}\,t \tag{2.10}$$

t = Zeit $\vec{v}$ = Lineare Geschwindigkeit

der neue COM und die lineare Geschwindigkeit $\vec{v}$ durch

$$\vec{v}' = \frac{\vec{f}}{m}\,t + \vec{v} \tag{2.11}$$

errechnet.

Drehbewegung Um die Ausrichtung des Objektes zu errechnen, sind ein wenig kompliziertere Methoden notwendig. Dazu werden der Drehimpuls sowie die Winkelgeschwindigkeit bestimmt.

Der Drehimpuls $\vec{L}$ ergibt sich aus

$$\vec{L}' = \vec{\tau}_b\,t + \vec{L} \tag{2.12}$$

und die Winkelgeschwindigkeit aus

$$\vec{\omega} = I^{-1}\vec{L} \tag{2.13}$$

$\vec{L}$	=	Drehimpuls	$\vec{\tau}_b$	=	Drehmoment
$\vec{\omega}$	=	Winkelgeschwindigkeit	I^{-1}	=	inverser Trägheitstensor

Die neue Orientierung errechnet sich dann durch:

$$\mathbf{A}' = \mathbf{A} + t\,\vec{\omega}\mathbf{A} \tag{2.14}$$

$\mathbf{A}$	=	Orientierung	t	=	Zeit

.

2.3 Grundlagen der Visualisierung

In den folgenden Abschnitten werden die Grundlagen, der für dieses Problem wichtigen Visualisierungsmethoden erklärt. Als Erstes wird die Visualisierung durch OpenGL erleutert, anschließend die verwendete Methode der Visualisierung durch einen Szenengraph näher erläutert.

2.3.1 OpenGL

Open Graphics Library (OpenGL) ist eine ursprünglich von "Silicon Graphics" entwickelte Bibliothek zur Erstellung von dreidimensionalen Grafiken und Animationen. Sie wird in vielen wissenschaftlichen Programmen benutzt und ist im Bereich der Computer Aided Design (CAD)-Programme quasi Standard. Der heutige Entwickler "Khronos Group" bezeichnet die OpenGL-Spezifikation als "The Industry's Foundation for High Performance Graphics" [23].

OpenGL wurde als Zustandsmaschine entwickelt. Das heißt, dass zum Beispiel die Farbe eines Objektes immer durch einen globalen Zustand definiert wird. Wird die Farbe Rot als aktuelle Zeichenfarbe definiert, sind alle Objekte, die daraufhin dargestellt werden Rot, erst durch die Änderung des Zustandes zum Beispiel zu Grün werden die folgenden Objekte in Grün dargestellt (siehe Abbildung 2.6).

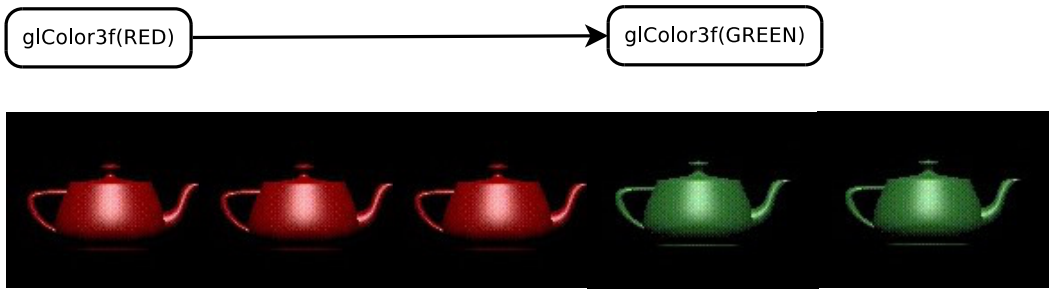


Abbildung 2.6: OpenGL Zustandsmaschine

Genau nach diesem Prinzip funktioniert auch die Berechnung von Koordinaten und Texturen. Ist ein Zustand, zum Beispiel für eine Matrixtransformation angegeben, so wird dieser auf alle folgenden Objekte angewandt, bis ein neuer Matixtransformationszustand erstellt wurde.

Treten Fehler in der Benutzung der Zustandsmaschine auf, können diese leicht Einfluss auf die ganze Darstellung nehmen. Um dies zu umgehen, wurden Konzepte entwickelt den Zugriff auf die Zustandsmaschine von OpenGL zu vereinfachen. Eines dieser Konzepte wird im nächsten Abschnitt 2.3.2 beschrieben.

2.3.2 Visualisierung durch einen Szenengraphen

Selbstgeschriebene Anwendungen, die eine eigene Abstraktionsschicht auf den Basisgrafikkommandos aufbringen, sind meist nicht portabel und erlauben keine direkte Interaktion mit den grafischen Objekten. [29] Grafische Toolkits wie OpenInventor sind in der Lage, grafische Daten als änderbare Objekte darzustellen. Das heißt, dass es dem Bibliotheksbenutzer ermöglicht wird "to specify what it is" [29] und sich nicht darum zu kümmern "how to draw it" [29]. Desweiteren soll der Nutzer fähig sein, durch eine integrierte Steuerung in der Szene mit den grafischen Objekten zu interagieren und nicht durch Tastenkürzel oder externe Widgets Einfluss auf die Szene nehmen zu müssen. Dabei kommt bei vielen Toolkits ein Szenengraph (SG) zum Einsatz. Dieser stellt eine Datenbank dar, die alle grafischen Objekte und Interaktionselemente aufbewahrt. In den folgenden Abschnitten werden einige dieser Basiselemente beschrieben und darauf eingegangen, welchen Zweck sie im Szenengraphen haben.

Szenengraph Wird versucht mit der Zustandsmaschine wie OpenGL, Objekte, deren Position im Raum voneinander abhängen, darzustellen, kann es schnell dazu kommen, dass die für die Darstellung wichtigen Speicher- und Ladeoperationen vergessen werden. Dies soll an einem Beispiel verdeutlicht werden.

Soll ein Unterobjekt wie zum Beispiel ein Rad an einem Auto gezeichnet werden, werden als Erstes die Transformationen für das Auto durchgeführt. Es erfolgen beispielsweise Transformationen zu der Position, an der sich das Auto befindet und die Karosserie wird gezeichnet. Danach wird der aktuelle Zustand der Transformationsmatix gespeichert und die Transformationen für ein Rad erfolgen. Dazu wird eine Transformation zur Position des zu zeichnenden Rades vorgenommen, worauf auch noch Drehungen durch eine Rotationsmatix erfolgen können. An dieser Stelle werden die Punkte, aus dem die Gitternetzdaten des Rades bestehen, mit der aktuellen Transformationsmatrix multipliziert und eine Darstellung des Rades wird erzeugt. Wird, nachdem die Darstellung abgeschlossen ist, vergessen, die ursprüngliche Matrix des Autos wiederherzustellen, werden alle folgenden Räder in Relation zum bereits gezeichneten Rad dargestellt.

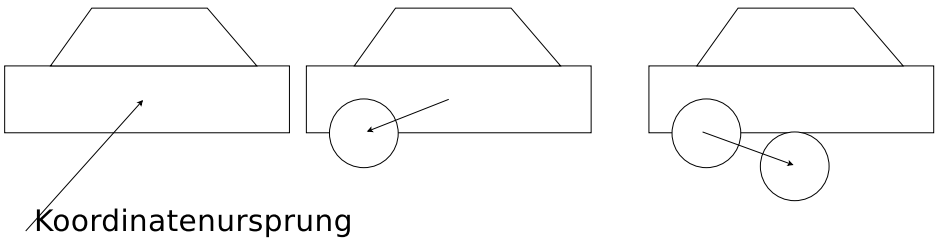


Abbildung 2.7: Zustandsmaschine

Dieser Fehler kann nicht nur bei Transformationen, sondern auch bei allen anderen Statusinformation wie zum Beispiel Farben und Texturen auftreten.

Um dies zu verhindern, wird ein Szenengraph eingesetzt. Szenengraphen werden von vielen modernen Grafikbibliotheken angeboten. Als prominente Vertreter sind hier OpenSceneGraph, OpenSG, Java3D, Open Inventor (OI) und Coin3D zu nennen.

Ein SG ist ein Directed Acyclic Graph (DAG) in dem sich hierarchisch geordnet Objekte befinden, die bei ihrer Traversierung Einfluss auf die darunterliegende Zustandsmaschine nehmen.

Sind diese Objekte in den SG eingehangen, so werden alle Speicherungs- und Ladeoperationen durch Separatoren vollzogen. Das Verhalten von Szenengraphen ist nicht standardisiert, doch im Allgemeinen funktioniert der Szenenaufbau durch Einhängen von Objekten in den SG. Die wichtigsten Vertreter dieser Szenengraph Objekts (SGOs) sind Transformationen, Separatoren, Gitternetze, Manipulatoren und Aktionen und sollen in den folgenden Abschnitten erklärt werden.

Transformationen

Transformationen werden benutzt um Objekte zu bestimmten Positionen zu bewegen, sie im Raum zu drehen oder um sie in der Größe zu verändern. Sie werden meist durch eine 4x4 Matrix dargestellt. Diese enthält, wie in Abbildung **??** zu sehen alle Informationen die für die Veränderung eines Objektes im Raum gebraucht werden.

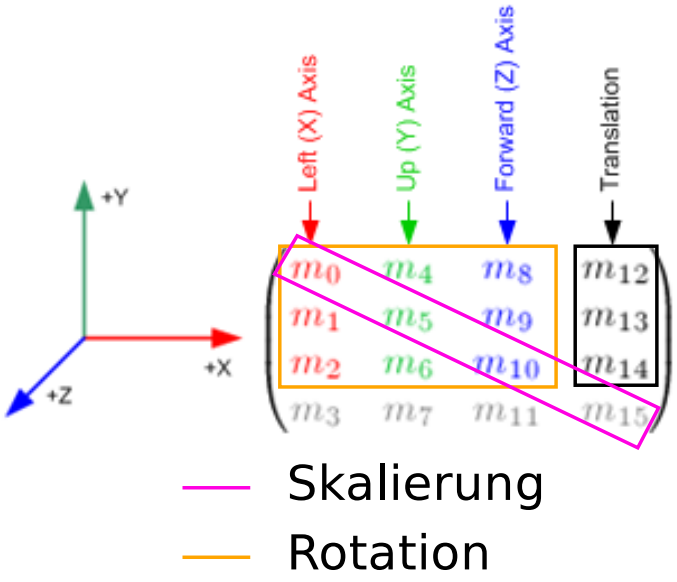


Abbildung 2.8: OpenGl Matrix

Dabei verändert der SG die Zustandsinformationen der Zustandsmaschine. Um die richtige Benutzung der Transformationen zu gewährleisten, werden Separatoren benutzt, welche im nächsten Abschnitt erklärt werden.

Separator

Ein Separator ist ein Gruppierungsknoten, der dafür sorgt, dass alle Zustandsinformation vor der Traversierung der Unterknoten gespeichert und nachdem alle Unterobjekte behandelt wurden, wieder in den ursprünglichen Zustand zurück versetzt werden. Ein Separator

kann ein Unterobjekt von einem anderen Separator sein, was zur Folge hat, dass man eine hierarchische Strukturierung aufbauen kann. Es ist aber auch möglich, dass ein Separator mehreren anderen Separatoren untergeordnet ist. Wird der Baum beim Zeichnen traversiert, so kann mit einer mehrfachen Unterordnung ein Objekt viele Male gezeichnet werden. Ein spezieller Separator ist der Wurzelknoten. Dieser beinhaltet alle Objekte in der Szene. Einzelne Objekte in der Szene sind meist durch einen Separator getrennt. Dieser beinhaltet dann die visuelle Repräsentation des Objektes als Gitternetz.

Darstellung von Gitternetzdaten

Um Objekte in einem SG darzustellen, bieten sich mehrere Methoden an. Zum einen ist es möglich Objekte durch Grundobjekte wie Quader, Kugel oder Zylindern darzustellen. Diese Art der Darstellung ist eher einfach und für komplexe Objekte nicht geeignet. Deswegen gibt es in allen SG Methoden, um Gitternetzdaten anzuzeigen. Dazu werden Objekte aus unterschiedlichen Formaten als Listen von Koordinaten und Listen von Indizes in den SG geladen. Diese Gitternetzdaten werden dann, wenn Matrixtransformationen vorhanden sind, von ihren Körperkoordinaten in Weltkoordinaten umgerechnet und von OpenGL angezeigt.

Interaktion

Um mit den Daten der Szene zu interagieren, werden zum Beispiel bei Coin3d, OpenInventor und OpenSceneGraph Manipulatoren benutzt. Diese fügen in einen spezifizierbaren Separator eine Transformation ein, welche durch eine visuelle Repräsentation steuerbar ist. Durch einen Manipulator kann die Position und Ausrichtung eines Objektes durch sogenannte “Dragger” verändert werden.

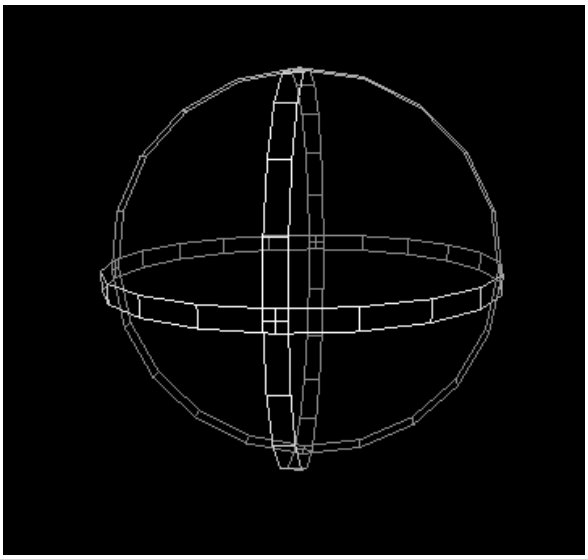


Abbildung 2.9: Visuelle Representation eines Manipulators [1]

Ein Manipulator besteht meistens aus mehreren “Draggern”. Diese können mit einem Mausklick angefasst und bewegt werden, um eine Änderung zu erzeugen.

Kollisionserkennung

Zusätzlich zu den im voraus erwähnten SGO, verfügen einige SG über die Fähigkeit Kollisionen zu erkennen. In dem hier vorgestellten Programm ist es wichtig festzustellen, ob ein Zahn mit einem anderen kollidiert ist. Denn wenn dies der Fall ist, muss die Simulation abgebrochen werden, da nicht genau beschrieben ist, was passiert, wenn Zähne kollidieren.

Vorarbeiten und Schlussfolgerung

3.1 Existierende Arbeiten

3.1.1 Zahnmedizin

Ideen zur technischen Beschreibung von Zahnbewegungen, werden bereits in "Some engineering principles of possible interest to orthodontists" [14] von 1917 beschrieben. Darin werden grundligende Prinzipien der Zahnbewegung verdeutlicht, die erst in den 1980-1990 Jahren, durch Smith, Burstone, Lindauer und Hocevar wieder aufgegriffen worden sind. Die in den neunziger Jahren verfassten Grundlagen der Zahnbewegung sind in den Publikationen "The Basics of Orthodontic Mechanics" [24], "Understanding, planning, and managing tooth movement: orthodontic force system theory" [7] und "Mechanics of tooth movement" [28] zu finden.

Es gibt nur wenige Umsetzungen von Simulationen oder Visualisierungen von Zahnbewegungen. Eine kommerziel erwerbliche Version von "Biomechanics in Orthodontics" stellt das einzige Programm da, mit dem zu Lehrzwecken Zahnbewegungen veranschaulicht werden können. Neben diesem Programm, gibt es ein Werkzeug des Herstellers Invisalign, dass, durch den Hersteller errechnete Bewegungen, anzeigen kann.

Biomechanics in Orthodontics

Seit 2008 wird von Prof. Dr. md. dds. Giorgio Fiorelli und Prof. Dr. Odont. dds. Birte Melsen eine Multimedia Software geschrieben, welche es ermöglicht Zahnbewegungen zu Lehrzwecken zu simulieren [10].

Im Rahmen eines Kurssystem, werden Bewegungen durch Videos und Animationen verdeutlicht. "21 chapters (+ 2 to be added soon), 629 pages, 4000 pictures, 250 animations/video, 500 references with link to the original source, 120 clinical cases, fully interactive." [10]

Invisalign

Der Hersteller Invisalign fertigt Zahnspangen aus Kunststoff an, die aufgrund ihrer Durchsichtigkeit nicht erkennbar sein sollen. Als Hilfestellung für Zahnärzte, ist eine Darstellung der Veränderung, die durch die Zahnspange erreicht werden soll, dem Softwarepacket beigelegt. Der Ablauf einer Zahnspangenerstellung bei Invisalign ist aber so, dass der Kieferorthopäde den Zahnabdruck an den Hersteller schickt, dieser ein dreidimensionales Zahnmodell erstellt und dieses Modell an den Kieferorthopäden schickt. Die Änderungen die dann erfolgen sollen werden schriftlich an den Hersteller übermittelt, welcher dann das Modell anpasst. Eine Version in der Interaktiv eine Bewegungssimulation vorgenommen werden kann ist nicht erhältlich.

3.1.2 Physik

Literatur

Physik Engines

Es stehen viele Möglichkeiten zur Verfügung, die Physik starrer Körper zu simulieren, jedoch ist die Anzahl von Algorithmen und Routinen, die aus den Engines gebraucht werden, gering und es wird ein Maß an Flexibilität benötigt, welches mit solchen generischen Lösungen nur schwer erreichbar ist. Der Eingriff in die Interna der physikalischen Berechnungen ist dabei so groß, dass eine Neuentwicklung mehr Sinn macht als die umständliche Benutzung einer vordefinierten Lösung. Durch eine Eigenentwicklung werden außerdem die Abhängigkeiten zu anderen Paketen gering gehalten, so dass schneller auf Nutzeransprüche eingegangen werden kann. Für die eigene Implementation der Physik wird auf die Publikationen “The Next Frontier”, “Angular Effects” und “The Third Dimension” [16] zurückgegriffen. In ihnen wird erklärt, wie die grundsätzlichen physikalischen Gegebenheiten in Gleichungen zusammengefasst werden können. Zusätzlich ist “Klassische Mechanik” von Herbert Goldstein [15] als Nachschlagewerk für Grundgleichungen zu erwähnen. Aus diesem Buch sind, aufgrund der allgemeinen bezeichnung als Standardlektüre, die Variablenbezeichnungen übernommen.

Tensorbestimmung

Um den Tensor eines beliebigen Objekts, welches als Gitternetz vorliegt, bestimmen zu können, wurde von Brian Mirtich ein Algorithmus zur einfachen Bestimmung des Tensors beschrieben. Unter der Voraussetzung, dass der Körper eine gleichverteilte Dichte hat, ist es möglich durch Volumenintegrale und einer vorgegebenen Dichte den Tensor zu bestimmen. Wie dieser Algorithmus funktioniert, kann in “Fast and accurate computation of polyhedral mass properties” [26] nachgelesen werden.

3.1.3 Programme

3.1.4 Szenengraphenlösungen

Als SG können unterschiedliche Implementationen verwendet werden. Da die Kompatibilität zu OI aber eine erstrebenswerte Eigenschaft ist, fiel die Entscheidung Coin3D zu benutzen, da diese kompatibel zu OpenInventor-Daten ist und aktiv weiterentwickelt wird. Der quelloffene Teil von OpenInventor wird nach Angaben von Wikipedia nicht mehr oder nur sporadisch

weiterentwickelt (“The open source version from SGI is not currently maintained and SGI has not shown any commitment to do further development of the library” [13]). Open Inventor Daten sind aber auch heute noch ein weitverbreiteter Standard.

Andere SG-Implementationen wie OpenSceneGraph, OpenSG oder OGRE stellen aber durchaus brauchbare Alternativen dar. Da als Programmiersprache C++ verwendet werden soll, eigenen sich Frameworks wie Java3D und jMonkeyEngine natürlich nicht.

3.2 Notwendigkeit einer Neuentwicklung

Eine Neuentwicklung ist aus mehreren Gründen wichtig. Zum einen sollen Kosten gespart werden, die bei der Nutzung von der in 3.1.3 aufgeführten Software aufkommen würden und zum anderen soll eine genauere Abdeckung der Anforderungen erreicht werden, da die Anwendbarkeit in der Lehre und in Zahnarztpraxen nicht durch die bestehende Software gewährleistet ist. Diese ist notwendig um ein besseres Verständnis für die Bewegungen zu gewinnen und um eine einfachere Anwendbarkeit zu erreichen. Ob das Verständnis gewonnen ist oder nicht, kann dann von Prüfern anhand von vordefinierten Szenarien getestet werden. Dabei wird einem Studenten eine zu lösende Situation vorgegeben, die vom Prüfer auf Effektivität und Richtigkeit geprüft wird. Im Gegensatz zur Arbeit an Typodonten können somit mehrere Lösungen einfacher und schneller durchgespielt werden.

Anforderungsdefinition

In diesem Kapitel wird erläutert, welche Ansprüche an die Software gestellt werden. Dazu werden die funktionalen Anforderungen durch "Use cases" und die nichtfunktionalen Anforderungen durch ISO/IEC 9126 beschrieben.

4.1 Funktionale Anforderungen

In diesem Abschnitt soll die geplante Verwendung der Software erklärt werden.

Eine Softwarelösung für das Problem der Zahnbewegung soll hauptsächlich für zwei Dinge erfolgen. Zum einen soll die Software für die Lehre benutzt werden und zum anderen um Voraussagen für konkrete Behandlungen zu treffen.

Die "Use cases" für diese Softwareentwicklung sind im Appendix Abschnitt A auf Seite 63 zu finden, werden hier aber aus Übersichtlichkeitsgründen kurz erklärt.

4.1.1 Datensatz laden

Der Benutzer der Software soll in der Lage sein, präparierte Inventor-Daten zu laden. Hierbei soll es dem Nutzer möglich sein, die Datei durch einen Dateiauswahldialog auszuwählen, um sie dann in das Programm zu laden. Sind die Daten geladen, zeigt sich dem Nutzer eine visuelle Repräsentation der Zähne.

4.1.2 Bracket hinzufügen

Sind die Daten geladen soll die Möglichkeit bestehen, Brackets zu den Zähnen hinzuzufügen. Hierzu wird das "Bracket hinzufügen"-Tool angewählt, welches es erlaubt eine Bracket auf der Oberfläche zu platzieren. Durch Anklicken der Oberfläche des Zahns wird eine visuelle Repräsentation eines Brackets hinzugefügt. Das Bracket ist im Nachhinein durch Klicken anwählbar.

4.1.3 Bracket ausrichten

Nachdem ein Bracket hinzugefügt ist, soll es möglich sein das Bracket auszurichten. Durch Anklicken eines Brackets erscheint eine Drehungssteuerung, die es erlaubt das Bracket aus-

zurichten.

4.1.4 Draht hinzufügen

Durch Anwählen des "Draht hinzufügen"-Tools ist es möglich, einem Bracket einen Draht hinzuzufügen. Dafür wird das Tool ausgewählt und auf das Bracket geklickt in dem sich der Draht befinden soll. Durch die Funktion Bracket mit Draht verbinden wird einem Bracket mitgeteilt das es in den Drahtbogen eingeglieder ist. Dies erzeugt eine visuelle Repräsentation eines Drahtes.

4.1.5 Parametrisierten Draht hinzufügen

Es soll möglich sein einen Parametrisierten Draht hinzuzufügen. Dabei wird aus einer Datei eine Drahtbeschreibung geladen, die eine Matrix eines Kraftfeldes enthält. Diese Drahtbeschreibung funktioniert dann wie ein normaler Draht.

4.1.6 Drehachse festlegen

Ist ein Bracket an einem Zahn angebracht, ist es durch Auswahl des "Drehachse festlegen" Tools möglich, ein Bracket auf die Drehung um eine festgelegte Achse zu beschränken. Dazu wird das Bracket mit dem ausgewählten Tool angewählt. Ein Stab, der die Drehachse verdeutlicht, wird nun angezeigt.

4.1.7 Bewegung simulieren

Die Simulation wird durch "Simulation starten" begonnen und kann jederzeit mit "Simulation pausieren" gestoppt werden.

4.1.8 Systemantwort des Alveolars simulieren

Ist die Simulation im Gange, so wird die Systemantwort des Zahnfachs auf die Kraft die durch einen Zahn aufgebracht wird, simuliert. Eine visuelle Darstellung der Gegenkräfte gibt an, wie groß diese sind, damit sie mit den Kräften der Spange verglichen werden können.

4.1.9 Kollision von Zähnen erkennen

Da nicht vorhersagbar ist, was passiert, wenn Zähne kollidieren, muss eine Kollisionserkennung festellen ob Kollisionen zwischen Zähnen aufgetreten sind. Wird eine Kollision erkannt, so wird der Benutzter informiert.

4.1.10 Zähne verblocken

Es soll möglich sein einen Zahn an der momentanen Position "festzufrieren" und wieder "aufzutauen". Dies ist nötig da man bei einigen Zähen davon ausgehen kann, dass sie sich nicht bewegen oder vom Kieferorthopäden so fixiert werden, dass sie es nicht tun.

4.2 Nichtfunktionale Anforderungen

Den ISO-Standard kann man bei der ISO [18] käuflich erwerben oder im Internet nachlesen [4]. Er gliedert die nicht-funktionalen Anforderungen in die Kategorien "Übertragbarkeit", "Wartbarkeit", "Effizienz", "Funktionalität", "Zuverlässigkeit" und "Benutzbarkeit und Gebrauchstauglichkeit".

Abbildung 4.1: ISO/IEC 9126 [19]

4.2.1 Funktionalität

Die Software muss dazu in der Lage sein, das Qualitätsmerkmal der "Richtigkeit" in dem Sinne zu erfüllen, dass die Anwendung vergleichbare Ergebnisse liefert, wie es Behandlungen an Menschen erbracht haben (Toleranz von 1 mm Abweichung). Weitreichendere Anforderungen an Interoperabilität, Sicherheit und Ordnungsmäßigkeit müssen nicht erfüllt werden.

4.2.2 Zuverlässigkeit

Die Software soll über eine gewisse "Reife" verfügen. Im Speziellen heißt das, dass es nur in einen von zehn Szenarien zu Ausfällen kommt oder sich das System von Fehlern erholen kann. Die Software sollte nur bedingt fehlertolerant sein. Treten Fehler bei der Handhabung auf, sollte sich das System nicht ausschalten, sondern Fehlermeldungen ausgeben. Treten hingegen Fehler bei der Simulation auf, ist die Ausgabe einer Fehlermeldung und der Programmabbruch erwünscht.

4.2.3 Benutzbarkeit und Gebrauchstauglichkeit

Die Software muss intuitiv verständlich sein ("Verständlichkeit") und sich einfach erlernen lassen ("Erlernbarkeit"). Eine einfache Benutzung steht dabei im Vordergrund ("Benutzbarkeit").

Hingegen soll "Attraktivität" nur soweit gewährleistet sein, dass sie die "Verständlichkeit" positiv beeinflusst.

4.2.4 Effizienz

Das "Verbrauchsverhalten" der Softwarelösung sollte so sein, dass das fertige Programm auf einem durchschnittlichen Computer (CPU: 2,6 GHz RAM: 2Gb) lauffähig ist. Dabei sollte die Antwortzeit des Systems bei einer durchschnittlichen Maschine nicht länger als 0.5 Sekunden betragen.

4.2.5 Wartbarkeit

Es werden keine besonderen Anforderungen an die Wartbarkeit gestellt.

4.2.6 Übertragbarkeit

Die Qualitätsanforderung "Übertragbarkeit", soll in dem Sinne eingegangen werden, dass die Software auf Linux und Windows lauffähig ist ("Anpassbarkeit") und eine Portierung nach Mac OS X auch einfach vollzogen werden kann.

4.3 Priorisierung

Für diese Entwicklung sind die Qualiätsanforderungen der "Erlernbarkeit", "Verständlichkeit","Übertragbarkeit" und "Richtigkeit" am wichtigsten. Die Zielgruppe dieses Programmes sind Studenten in der Ausbildung und Zahnärzte. Wichtig ist dabei, dass das fertige Programm auf den Maschinen der Studenten beziehungsweise Zahnärzte lauffähig ist. Ist dieses Ziel erreicht stellt die Richtigkeit das nächstwichtige Ziel da.

Andere Aspekte wie die "Zuverlässigkeit" sind wichtig, aber im Rahmen einer Bachelorarbeit nur zu einem gewissen Maße erreichbar.

Analyse

5.1 Visualisierung

Um die Anforderung nach “Erlernbarkeit” und “Verständlichkeit” zu erreichen, soll die Interaktion mit dem Programm möglichst gering gehalten werden. Durch wenige, einfache Funktionen ist schneller eine bessere Verständlichkeit erreicht als durch viele komplizierte. Desweiteren sollen keine Effekte wie Überblendungen oder ähnliche nicht zum Verständnis beitragende Zusätze benutzt werden. Durch eine minimalistische Gestaltung erhöht sich auch die “Robustheit” und die “Wartbarkeit” der Software, da keine unnötigen Eigenschaften im Code vorhanden sind. Das Programm verfolgt damit den keep it short and simple (KISS) Ansatz.

5.2 Interagierende Komponenten

Im zu entwickelnden Programm werden hauptsächlich nur zwei Komponenten miteinander interagieren. Zum einen die physikalischen Komponenten, die in Simulationsklassen gefasst werden, zum anderen die Visualisierungskomponenten, die als Visualisierungsklassen bezeichnet werden sollen. Aufgrund des Prinzips der Einbettung der interaktiven Komponenten in den SG werden diese nicht als gesondertes Modul betrachtet, sondern als Visualisierungskomponente.

5.3 Qualitätssichernde Maßnahmen

Um die Anforderung nach “Reife” (Abschnitt 4.2.2) zu erreichen, werden Unittests vorgenommen. Diese werden die Software in Klassen- und Modultests testen. Hierbei wird erst die Funktionsfähigkeit einer Klasse und im Nachhinein auch die Zusammenarbeit von Klassen getestet.

Die Verifikation der Anforderung nach “Richtigkeit” (siehe Abschnitt 4.2.1) wird durch einen Vergleich mit Patientendaten erfolgen. Dazu werden einzelne Bilder der Simulation

gegen die Bilder eines Patienten verglichen. Eine ausführliche Verifikation, mit vielen Datensätzen soll nicht Teil dieser Arbeit sein. Hierzu wird es nötig sein Tests zu schreiben, die überprüfen, ob die richtigen Bewegungsrichtungen von Zähnen von der Software erzeugt werden könnnen und ob die Systemantwort des Alveolars auf die aufgebrachten Kräfte plausibel ist. Zusätzlich werden Bauvorgänge automatisch auf unterschiedlichen Architekturen durchgeführt. Zielarchitekturen sind aufgrund der vorhandenen Maschinen: Gentoo Linux (x86), Gentoo Linux (ARM), Windows 7 (VirtualBox x86) und Windows XP (VirtualBox x86). Um festzustellen, ob alle Bibliotheksabhängigkeiten vorhanden sind, wird ein Installationstest für beiden Windowsversionen vollzogen werden. Diese Tests werden den Systemtest darstellen.

Entwurf

In diesem Kapitel sollen die grundsätzlichen Entwurfsentscheidungen erklärt werden. Es wird beschrieben, welchen Aufbau die Software haben wird und wie sichergestellt werden kann, dass die Simulation den Qualitätsansprüchen genügt. Dabei wird erklärt, welchen Einfluss die "Use cases" aus Abschnitt 4.1 auf den Entwurf der Software haben.

6.1 Zahnmedizinische Approximationen

6.1.1 Zähne

Datenerhebung

Für dieses Programm wurden idealisierte Daten von Zähen verwendet, die dem Autor vom "Zentrum für angewandte Informatik, flexibles Lernen und Telemedizin", Prof. Dr. Bernd Kordaß zu Forschungszwecken überlassen wurde [17]. Die Zähne in diesem Modell sind durchschnittliche Zähne und stellen ein komplettes Gebiss, von den Schneidezähnen zu den Mahlzähnen, dar.

Die Konvertierung von Daten aus Computed Tomograph (CT) oder Magnetresonanztomographie (MRT)-Daten, die ein Gebiss beschreiben, in das OI-Format ist eine andere Aufgabe und wurde in dieser Arbeit nicht umgesetzt. Solche Daten können durch eine Digital-Volumen-Tomographie (DVT) gewonnen werden und müssen in Gitternetzdaten umgewandelt werden. Dabei ist wichtig, dass die Daten die Wurzel des Zahns beinhalten, um genauere Resultate zu erzeugen. Es gibt intraorale bildgebende Verfahren, die in der Lage sind auch ohne Strahlung oder aufwendige Prozedur CT-Bilder von den Kronen der Zähne zu erzeugen. Der Nachteil besteht darin, dass die Wurzeln der Zähne so nicht erfasst werden können. Sind die Wurzeln nicht vorhanden, können diese aber durch idealisierte Wurzeln nachgebildet werden.

6.1.2 Alveolarknochen

Der Alveolarknochen verursacht die Gegenkraft, die den Zahn festhält.

Wie bei den Typodonten ist die Zahnbewegung in diesem Programm nur eine Approximation. Als Approximation wird das Zahnfleisch im Programm auf eine bestimmte Höhe eingegrenzt, da keine Daten für dieses vorliegen. Diese Höhe wird so eingestellt, dass die Zähne des Ober- und Unterkiefers bis zu einer natürlichen Höhe mit Zahnfleisch bedeckt sind (siehe Abbildung 6.1). Es wäre möglich die Beschreibung des Alveolarknochen für jeden einzelnen Zahn zu implementieren jedoch reicht die Genauigkeit für diese Zwecke aus.

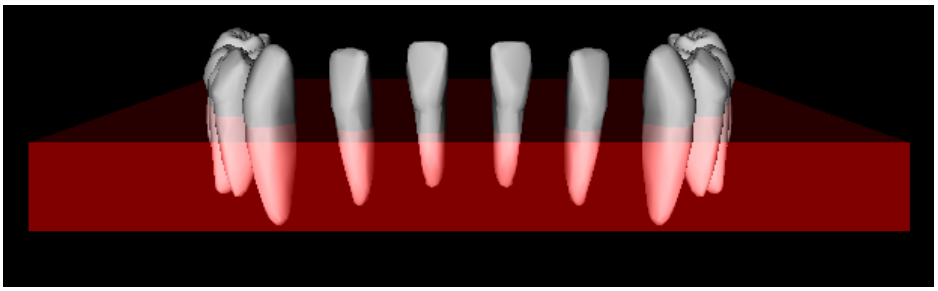


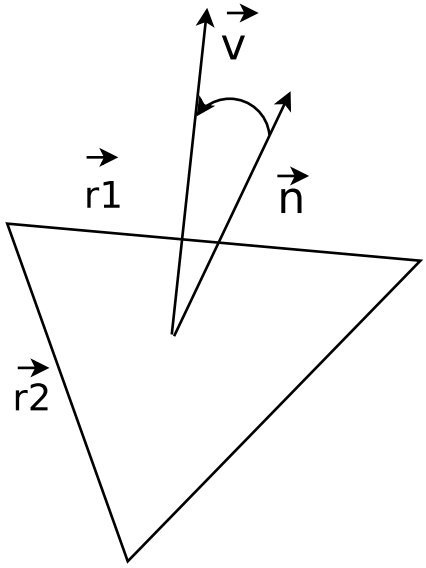
Abbildung 6.1: Bedeckung mit Zahnfleisch

Dieses Vorgehen ist auch in dem Sinne vorteilhaft da sich in der späteren Anwendung die Flächen der Zähne, die unter dem Zahnfleisch liegen einfacher detektieren lassen. Zum anderen können die Materialeigenschaften des Alveolarknochens nicht genau vorhergesagt werden. Deshalb wird dieser als einheitlich angenommen. An allen Stellen wird auf eine Bewegung mit der gleichen Gegenkraft geantwortet. Durch genauere Daten, die auch die Dichte des Knochens beinhalten wäre eine genauere Beschreibung der Gegenkraft möglich. Da aber solche Bilder nur durch aufwändige Scanverfahren erstellbar sind, wird hier davon abgesehen eine dichteabhänige Response einzubauen.

6.2 Physikalische Überlegungen

Die Bewegung des Zahns durch den Alveolar, wird als Bewegung eines starren Körpers approximiert. Hierbei wird der Zahn durch die in Abschnitt 2.2 erläuterten Gleichungen bewegt. Interessant wird es allerdings erst, wenn die Gegenkräfte des Alveolars mit in die Bewegung einbezogen werden.

Als einfachste Beschreibung der biologischen Reaktion des Gewebes auf Zahnverschiebungen, wird zur Berechnung einer empirisch skalierten Gegenkraft nur die senkrechte Projektion der Fläche der Zahnelemente in Bewegungsrichtung benutzt.



$$\vec{f}_{anti} = -\frac{(\vec{r}_1 \times \vec{r}_2) \cdot \vec{v}}{2} \vec{f} \qquad (6.1)$$

$\vec{f}_{anti}$ = Gegenkraft
$\vec{r}_n$ = Richtungsvektoren des Dreiecks
$\vec{v}$ = Bewegungsrichtung

Abbildung 6.2: Dreieck in Beweungsrichtung

Die entstehende Gegenkraft $\vec{f}_{anti}$ wird dann auf den Mittelpunkt des Dreiecks angewandt und führt somit zu einer Gegenbewegung und Gegenrotation.

Eine andere Methode die Gegenkraftddarzustellen ist es, die Geschwindigkeiten im System nach jedem Zeitschritt auf Null zurück zu setzten. Aufgrunddessen, dass die Geschwindigkeiten in einem System wie dem menschlichen Kiefer sehr gering sind, kann dieses Verfahren als gute Näherung eingesetzt werden.

6.3 Programmumsetzung

In diesem Abschnitt sollen, als erstes die verwendeten Bibliotheken erklärt und im Nachhinein der Aufbau der Software dargelegt werden.

6.3.1 Verwendete Bibliotheken

Hauptsächlich wurde versucht die Anzahl der verwendeten Bibliotheken gering zu halten, damit sich die Softwarelösung einfacher auf andere Betriebssysteme portieren lässt. Es werden drei große Komponenten verwendet. Als Visualisierungskomponente “Coin3D” mit “Qt” und zum Test der Software die Bibliothekensammlung “Boost”. In den folgenden Abschnitten wird auf Coin3D und Boost näher eingegangen.

Coin3D

Als Grafikbibliothek kommt die von “Kongsberg Oil & Gas Technologies” entwickelte Bibliothek Coin3D oder kurz Coin zum Einsatz, da diese quelloffen und frei lizenzierbar ist. Die Library ist eine freie Variante von Open Inventor, einer 3D-Grafikbibliothek für C++ und wurde wie Open Inventor dazu geschrieben OpenGL auf einem höheren Niveau anzusprechen. Diese Bibliothek stellt einen Szenengraph zur Verfügung. Dieser erlaubt es dreidimensionale Objekte in Gruppen zusammenzufassen und die Zustandsinformationen der Objekte wie

die Ausrichtung im Raum oder die Farbe, zu verwalten. Durch die Bereitstellung unterschiedlicher Viewer kann der User einfach mit der Visualisierung interagieren. Diese sind für unterschiedliche Plattformen implementiert, wie zum Beispiel die für die Bibliotheken Qt und Gtk. Zusätzlich gibt es spezifische Implementationen für die Mac- und Windows Anzeigesysteme. Coin3D bietet Klassen für den Szenengraph, lineare Algebra, eine Speicherverwaltung und Implementationen von Viewern für unterschiedliche Betriebssysteme.

Laut Herstellerangaben wird die Coin3D-Bibliothek in vielen Bereichen benutzt. Anwendung findet sie unter anderem in der Medizin, in CAD Anwendungen und in der Visualisierung von Forschungsdaten. Da OI nicht mehr weiterentwickelt wird, aber die API ein weitverbreiteter Standard ist, wurde mit Coin3D der komplette Code neu geschrieben, die API von OI aber beibehalten um Kompatibilität zu gewährleisten. In den folgenden Abschnitten wird auf einige spezifische SGO eingegangen, die im Programm Verwendung finden werden.

Aktionen Möchte man Informationen über den aktuellen Zustand eines Objektes im SG erhalten, so werden Aktionen eingesetzt. Diese Aktionen traversieren den SG und akkumulieren dabei die Statusinformationen. Im Folgenden wird dies am Beispiel des Raypickings verdeutlicht.

Wenn der Benutzer im Interface auf die Projektion klickt, wird ein Strahl generiert, der aus dem Augpunkt in Richtung der Szene ausgesandt wird. Dieser Strahl wird nun nacheinander, von den sich im SG befindlichen Matrixtransformationen, umgewandelt. Trifft die Traversierung auf ein Objekt, so wird versucht den Schnittpunkt festzustellen. Auf diese Art und Weise kann die genaue Position auf der Oberfläche des Objekts festgestellt werden. Im Speziellen stellt Coin3D hierfür die “SoRayPickAction” zur Verfügung. Damit ist der ganze Ablauf der Strahltransformation abstrahiert und Positionen, an denen zum Beispiel die Brackets angebracht werden, können berechnet werden.

Manipulatoren Ein weiterer Knoten im Graphen kann ein Manipulator sein. Dieser kommt zum Einsatz, wenn der Nutzer mit dem SG interagieren möchte. Es gibt Trackball-Manipulatoren, Box-Manipulatoren oder Transformer-Manipulatoren. Ein Manipulator besteht im Allgemeinen aus einem oder mehreren grafischen Objekten, die durch Anklicken und Ziehen veränderbar sind. Ein Dragger kann vom Nutzer benutzt werden, um ein mit dem Dragger assoziiertes Objekt zu verändern.

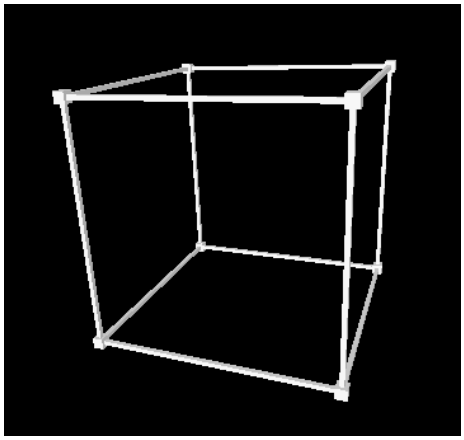


Abbildung 6.3: Box Maniplator

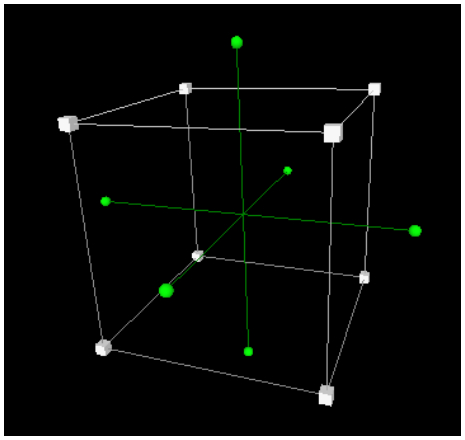


Abbildung 6.4: Transformer Manipulator

Speicherverwaltung

Damit keine Information im SG verloren gehen, ist in Coin3D eine Speicherverwaltung integriert. Diese verhindert, dass Speicherlöcher entstehen können. Wird zum Beispiel eine Transformation in den SG unter einem Separator eingefügt, so kann es passieren, dass der Separator gelöscht wird und die Transformation im Speicher verbleibt. Verhindert wird dies durch einen integrierten Referenzzähler, der alle Objekte, die von keinem anderen Objekt referenziert sind, automatisch löscht. Für C++ Programmierer kann dies anfänglich sehr verwirrend sein, da es nicht möglich ist ein Objekt auf dem normalen Weg (Delete-Operator) zu löschen. Um ein explizites Löschen zu verhindern, wurden die Destruktoren der Klassen als Protected deklariert. Will man explizit Objekte aus dem SG löschen, so sind hierfür die Funktionen "unref()" und "unrefNoDelete()" vorgesehen. Mit letzterer ist es möglich das Objekt zu entfernen, um es danach gleich an einer anderen Stelle einzuhängen. [20] Das Speichermanagement kann mit der Benutzung eines "shared_ptr" verglichen werden, wie er zum Beispiel in der "Boost Smart Pointers" Bibliothek vorkommt.

Lineare Algebra

Häufig verwendete Operationen, die im Zusammenhang mit grafischen Anwendungen gebraucht werden, sind in einigen Grunddatentypen zusammengefasst. Matrizen, Vektoren, Listen von unterschiedlichen Basisdatentypen sind als Szenengraph-Basisobjekte verfügbar und bieten die gleichen Speicherverwaltungs-Features wie die Szenengraphobjekte.

Listing 6.1: Kapselung in Szenengraph Objekte

```
1
2 SoMatrixTransformation* m = new SoMatrixTransformation;
3
4 SbRotation r;
5 SbVec3f axis( 1.0,1.0,1.0 );
6 float angle = M_PI / 2;
7 r.setValue( axis, angle );
8 m->matrix.getValue().setRotate(r);
```

Im Listing 6.3.1 kann man sehen, dass die Basistypen SbRotation, SbVec3f und SbMatrix dazu genutzt werden eine Rotationsmatrix im SoMatrixTransformation Objekt m zu erstellen. Dabei werden eine Rotationsachse und ein Winkel definiert aus der wiederum eine Rotation (SbRotation) entsteht. Diese wird verwendet, um dann die Rotationsmatrix im Objekt m zu verändern.

Qt - Toolkit

Als Implementation für den Viewer wurde Qt verwendet. Qt ist für Windows, Mac und Linux verfügbar. Somit muss keine andere Implementation des Viewers vorgenommen werden, wenn die Anwendung auf ein anderes System portiert werden muss. Implementationen von Viewern, wie zum Beispiel SoWin und SoXt sind nicht für alle Betriebssysteme verfügbar. Gtk wäre eine Alternative zu Qt, wurde vom Autor aber als nicht portabel genug erachtet. Ein Vergleich der beiden Frameworks steht unter [2] zur Verfügung.

Boost

Um größtmögliche Portabilität zu gewährleisten, wurde die Boost Bibliothekssammlung benutzt. Boost verfügt über eine sehr große Sammlung von unterschiedlichen Subbibliotheken, die alle auf Portabilität und Performance optimiert wurden. Unter anderem bietet sie die Möglichkeit Tests durchzuführen und eine abschaltbare Assertion in den Code einzubauen.

Assertion Assertionen sind Tests auf Erfüllung einer bestimmten Bedingung. In der Mathematik werden Definitions- und Wertebereich einer Funktion definiert. Ähnlich ist es mit Assertionen diese werden verwendetet um gewisse Zustände zu überprüfen. Zum Beispiel kann bei einer Funktion, welche die Wurzel einer Zahl x errechnen soll, getestet werden ob x größer Null ist. Wenn dies nicht der Fall sein sollte, so wird ein Fehler ausgegeben und das Programm wird beendet.

Boost implementiert dies durch das Macro "BOOST_ASSERT" und "BOOST_ASSERT_MSG".

Listing 6.2: Assertion

```
1 double mySqrt( double x ) {
2     BOOST_ASSERT_MSG( x > 0, "x is not bigger than 0" );
3     return sqrt(x);
4 }
```

Der Vorteil der Boost-Implementation gegenüber dem Standard "assert" von C++ ist, dass alle Asserts wahlweise ausgeschaltet oder in "throw"-Anweisungen umgewandelt werden können. Dies hat direkten Einfluss auf die Unittests, da diese abbrechen würden wenn ein Fehler auftritt.

Boost Test Zur Testgestaltung bietet sich für C++ neben den Standard Programmen CppUnit und CxxTest auch Boost an. Die Menge an Code, die für einen Test geschrieben werden muss, ist aber bei Boost sehr klein [27]. Was den Vorteil mit sich bringt, dass Fehler, die durch die Tests entstehen, gering gehalten werden können. Andere Testumgebungen sind hingegen nur für sehr große Projekte ausgelegt und skalieren nicht so gut für kleine Aufgaben. Ein Boost Testfall ist in der minimalen Ausführung nicht größer als das folgende Beispiel.

Listing 6.3: Boost Testfall

```
1 #include <boost/test/unit_test.hpp>
2 BOOST_AUTO_TEST_CASE( TestName )
3 {
4
5 }
```

6.3.2 Grundaufbau

Die Visualisierung wird hierarchisch aufgebaut. Damit ist Coin3D in der Lage, die unterschiedlichen Visualisierungsobjekte in der gleichen Hierarchie zu verwalten, wie sie im Code vorliegen werden. Der zusammenfassende Knoten ist die Klasse Dentition. Sie beinhaltet die Zähne, die Drähte, die Brackets und die Darstellung des Alveolars und ist wie alle anderen Klassen auch in Visualisierung und Simulation aufgespalten (siehe nächster Abschnitt 6.3.2). Alle Interaktionen mit der Darstellung werden durch die Klasse Dentition vorgenommen.

Wird zum Beispiel ein Bracket an einem Zahn angebracht so wird es der Dentition hinzugefügt. Diese fügt dem Bracket eine visuelle Darstellung hinzu und ordnet diese an der richtigen Stelle im Szenengraphen ein.

Dateiformat

Als Dateiformat wurde das OI-Dateiformat gewählt und führte auch zur Wahl von Coin3D als SG-Library. "Es gibt zwei Arten für die Speicherung von Inventor Dateien: ASCII und Binary, die wechselweise ineinander überführt werden können. Der Vorteil von Binarydateien ist, dass sie schneller in den Szenegraph geladen werden. Inventordateien lassen Verweise auf weitere Dateien zu (Include Konzept wie in C Headerdateien). Das erlaubt, die Geometriebeschreibungen mit vielen Dreiecksinformationen von der Beschreibung der Struktur zu trennen." [25]

Für dieses Projekt wurde die ASCII-Variante gewählt, da diese einfacher zu bearbeiten ist und somit Änderungen einfacher vorzunehmen sind. Die OI-Dateien sind dabei in der Art aufgebaut, wie auch der Szenengraph, der aus ihnen geladen wird. Es gibt einen Wurzelknoten, dem andere Knoten untergeordnet sind.

Damit die physikalischen Quantitäten COM und Trägheitstensor berechenbar sind, wurden Object File Format (OFF)-Dateien verwendet. Um jedem Objekt in der Inventor-Datei die richtige OFF-Datei zu zuweisen, wurde der Info-Knoten mit einer Zeichenkette beschrieben, die den Dateinamen angibt. Es sollte möglich sein, eine Überführungsalgorithmus zu schreiben, der die Dreiecksdaten aus dem OI-Format in ein geeignetes Format übersetzt. Jedoch bietet sich durch die Benutzung von beiden Dateiarten eine weniger fehleranfällige Methode an die Daten zu kommen.

Aufteilung in Simulation und Visualisierung

Um mögliche Austauschbarkeit der Visualisierung zu gewährleisten und den Einfluss der Visualisierung auf das Projekt gering zu halten, ist die Simulation von der Visualisierung getrennt. Dies hat den Vorteil, dass die Simulation mit wenigen Änderungen auch ohne die Visualisierung durchspielbar sein kann. Eingriffe der Visualisierung in die Simulation werden sich aus Effektivitätsgründen nicht vermeiden lassen, sind damit aber auf ein Minimum reduziert.

Wie in Abbildung 6.5 dargestellt werden die Klassen in Simulations-Klassen (Sim) und Visualisierungs-Klassen (Vis) aufgeteilt und von den Dentition-Klassen verwaltet.

Programmablauf

Der Programmablauf soll so gestaltet sein, dass der Benutzer das Programm startet und aufgefordert wird eine Datei zu laden. Ist die Datei geladen, können Brackets und Drähte, wie in den Use cases 4.1 beschrieben, hinzugefügt werden. Sind alle Einstellungen vorgenommen worden, wird die Simulation gestartet. Sie kann im Ablauf unterbrochen und wieder weitergeführt werden.

Testgestaltung

Die Tests werden in Klassen-, Modul- und Systemtests eingeteilt. Die Klassen- und Modultest werden dafür im einfachsten Fall dafür sorgen die Konstruktoren und Destuktoren der

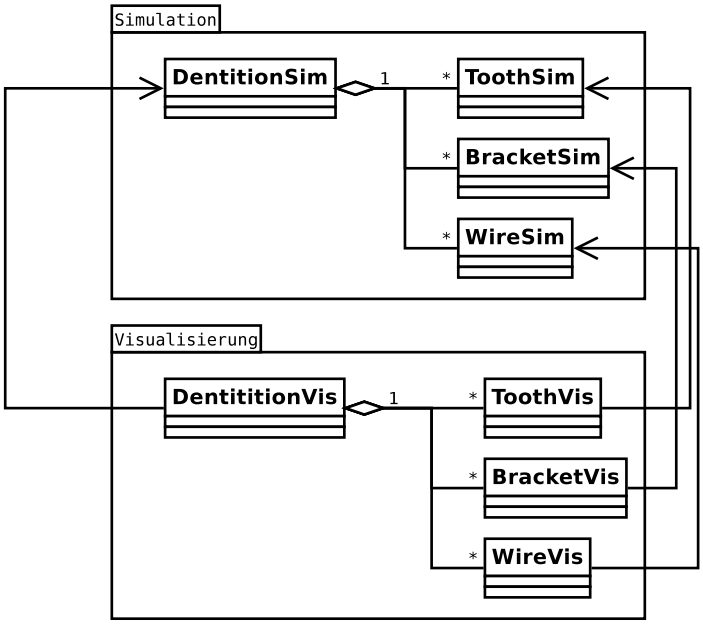


Abbildung 6.5: Trennung von Simulation und Visualisierung

Klassen und die Zusammenarbeit einfacher Objekte miteinander zu testen. Damit soll die Anforderung der Zuverlässigkeit ausgebaut werden. Automatisierte Systemtests werden dann die Übereinstimmung der Simulation mit realen Patientendaten überprüfen.

Implementation

7.1 Umsetzung der medizinischen Voraussetzungen

Um die medizinischen Voraussetzungen zu implementieren wurden Restriktionen eingeführt. Eine Restriktion wird auf ein Objekt angewandt, welches dieser Restriktion dann gehorchen muss. Hierbei werden zwei Restriktionen unterschieden. Zum einen die Rotationsrestriktion (rotation restriction) und die Bewegungsrestriktion (freeze restriction). Objekte die in ihrer Funktionalität einschränkbar sind, können eine Restriktion aufnehmen und verhalten sich dann entsprechend.

Die beiden Restriktionen leiten sich von einer abstrakten Basisklasse "Restriction" ab. Somit können beliebig viele Arten von Restriktionen hierarchisch gegliedert implementiert werden.

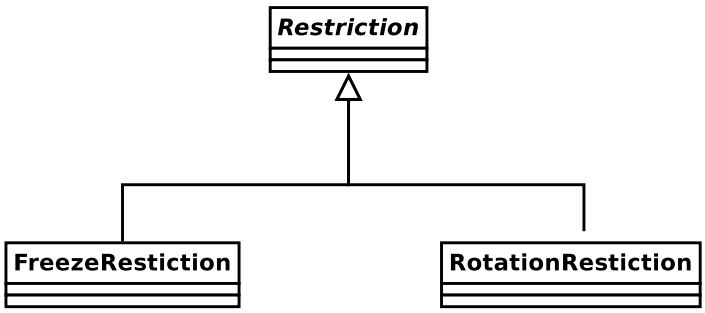


Abbildung 7.1: Resticktionen

Eine "freeze restriction" erlaubt es, ein Objekt an der jetzigen Position "einzufrieren". Damit ist es nicht mehr in der Lage sich zu bewegen und kann somit als verblockt angesehen werden.

Die "rotation restriction" wird bei allen Brackets angewandt, durch welche ein eckiger Draht führt.

7.2 Umsetzung der physikalischen Voraussetzungen

Die Hauptkomponente der Simulation ist die Rigidbody-Klasse. Diese Klasse setzt die Bewegung von Körpern im Raum durch und erzeugt die Positionsverschiebung sowie die Rotation.

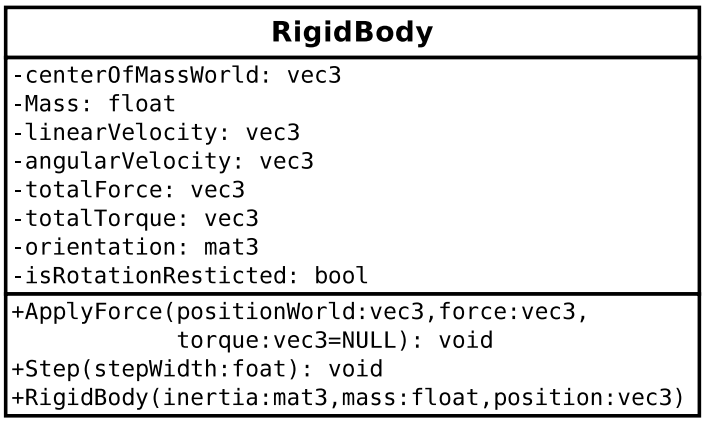


Abbildung 7.2: RigidBody Klasse

Durch Eingabe von Kräften mit der Methode "ApplyForce" wird für einen Zeitschritt eine Kraft an einer bestimmten Position in Weltkoordinaten angebracht. Es können beliebig viele Kräfte appliziert werden, die dann zu den totalen Kräften und Drehmomenten addiert werden. Nach einem Aufruf der Methode "Step" werden die Variablen "orientation" und "centerOfMassWorld" aktualisiert. Am Ende der Step-Methode werden die Variablen totalForce und totalTorque wieder auf Null zurückgesetzt.

7.2.1 Trägheitstensor und Massenberechnung

Um ein Objekt zu instanziieren wird der Trägheitstensor, die Masse des Objekts sowie der COM benötigt. Um diese Quantitäten zu errechnen, wird der Code von Brian Mirtich [26] benutzt, welcher auf seiner Internetseite als Beispiel zur Verfügung steht. Dieser ist mit ein paar Änderungen in der Lage OFF-Dateien zu lesen.

OFF-Dateien bestehen nur aus einem Header in dem die Anzahl von Vektoren, der Flächen und der Kanten steht, sowie einem "Datenkörper" aus Koordinaten und Indizes.

Listing 7.1: Object File Format [6]

```
1 OFF numVertices numFaces numEdges
2 x y z
3 x y z
4 ... numVertices like above
5 NVertices v1 v2 v3 ... vN
6 MVertices v1 v2 v3 ... vM
7 ... numFaces like above
```

Sie sind sehr einfach und man kann sie zum Beispiel einfach in Blender aus Open Inventor-Dateien erzeugen. Im für dieses Programm benutzten Dateiformat befindet sich ein Knoten an jedem Gitternetz, welcher den dazugehörigen OFF-Dateinamen angibt.

7.2.2 ApplyForce-Methode

Durch die ApplyForce-Methode wird eine Kraft auf den starren Körper aufgebracht. Mittels der Parameter "positionWorld", "force" und den optionalen Parameter "torque" ist es möglich, eine Kraft an einem bestimmten Punkt anzubringen und optional ein vordefiniertes Drehmoment anzugeben. Dies ist nötig, wenn genauere Beschreibungen von Zahnspangen in die Simulation eingehen sollen, da diese einen gewissen Torque erzeugen können. Dabei ist darauf zu achten, dass sich dieser nicht zwingend innerhalb oder auf der Oberfläche des Körpers, den man versucht zu bewegen, befinden muss.

Die RigidBody-Klasse verfügt über die Fähigkeit die Rotationsachse des Körpers einzuschränken. Je nachdem, ob die Achse eingeschränkt wurde, wird die Kraftanwendung unterschiedlich behandelt. Gemäß dem Fall, dass die Achse nicht eingeschränkt ist, wird die Kraft gemäß Formel 2.6 und 2.7, in "totalForce" ($\vec{f}_b$) und "totalTorque" ($\vec{\tau}_b$) umgewandelt. Für den Fall, dass die Rotationsachse eingeschränkt wurde, wird Formel 2.8 benutzt, um "totalTorque" zu errechnen.

7.2.3 Step-Methode

In der Step-Methode wird die eigentliche Arbeit verrichtet. In ihr werden die Formeln 2.10, 2.11, 2.12 und 2.13 umgesetzt. Der Ablauf ist dabei folgender:

1. Bestimmung des neuen Massenschwerpunkts $\vec{r}_s$ durch Formel 2.10
2. Bestimmung der neuen linearen Geschwindigkeit $\vec{v}$ durch Formel 2.11
3. Bestimmung der neuen Winkelgeschwindigkeit $\vec{\omega}$ durch Formel 2.13
4. Bestimmung der neuen Orientierung $\mathbf{A}$ durch Formel 2.14
5. Reorthogonalisierung der Orientierung

7.2.4 Reorthogonalisierung

Aufgrund dessen, dass die Orientierung wie in 2.14 beschrieben, akkumuliert wird, addiert sich über die Zeit, wie bei allen anderen Additionen in der Simulation, ein Fehler auf. Um diesen minimal zu halten, wird die Matrix $\mathbf{A}$ von einer Reorthogonalisierungsfunktion durch

Listing 7.2: Reorthogonalisierung

```
1
2 void Orthogonalize( double* m ){
3
4   double x[3] = { m[0] , m[1] , m[2] };
5   double y[3] = { m[3] , m[4] , m[5] };
6   double z[3];
7
8   vect3f::Normalize(x);
9   vect3f::Normalize(y);
10  cross3f(x,y,z);
11  vect3f::Normalize(z);
12  cross3f(z,x,y);
13
14  m[0] = x[0] ; m[3] = y[0] ; m[6] = z[0];
```

```
15    m[1] = x[1] ; m[4] = y[1] ; m[7] = z[1];
16    m[2] = x[2] ; m[5] = y[2] ; m[8] = z[2];
17 }
```

korrigiert.

7.3 Umsetzung der Visualisierung

Damit sich alle Komponenten der Visualisierung (Zähne, Brackets und Drähte) an den richtigen Stellen befinden, sind diese im Szenengraphen wie folgt "aufgehangen".

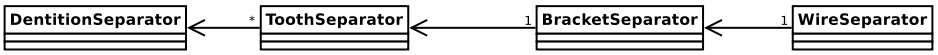


Abbildung 7.3: Hirarchie

Damit ist gewährleistet, dass alle Objekte in ihrem eigenen Koordinatensystem arbeiten können. Die Gitternetzdaten der Zähne und der Brackets werden aus OI-Daten entnommen. Die Visualisierung der Drähte wird im Programm von Linienprimitiven übernommen.

7.4 Umsetzung des Programmablaufes

Wird das Programm gestartet, werden die Daten aus einer vorbereiteten OI-Datei geladen. Diese Datei wurde um Felder für die korrekte OFF-Datei erweitert. Die Funktion "fillDentition" wird dazu genutzt, um die OFF-Dateien in den Info-Knoten der OI-Datei zu finden, um sie dann mithilfe der in Abschnitt 7.2.1 beschriebenen Funktionen in Trägheitstensor und Massenschwerpunkt umzurechnen. Aus ihnen wird dann ein starrer Körper generiert und ein ToothSim-Objekt wird erzeugt. Ist dieses erstellt, wird dieses an das globale DentitionVis-Objekt gegeben, welches das dazugehörige ToothVis-Objekt (siehe 7.4.3) erstellt und es in den Szenengraphen einhängt.

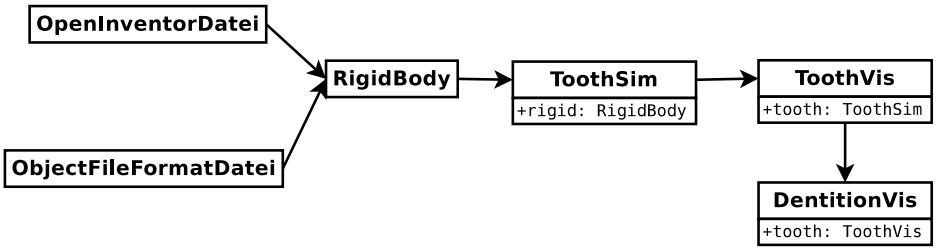


Abbildung 7.4: Zahnobjekterstellung

Sind alle Zähne geladen, so können Einstellungen am momentanen Zustand vorgenommen werden. Durch Anklicken von Buttons (Implementiert durch QPushButtons) an der Seite des Viewers ist es möglich, ein spezielles Tool auszuwählen. Zur Auswahl stehen das Bracket hinzufügen-, Draht hinzufügen-, Restriktion hinzufügen- und Bracket mit Draht verbinden-Tool. Diese setzen die "Use cases" Abschnitte "Daten laden", "Bracket hinzufügen", "Draht

hinzufügen" und "Drehachse festlegen" um. Die Interaktion mit der Szene erfolgt dabei durch einen Selection-Callback und durch Raypicking (siehe Entwurfsabschnitt 6.3.1), welches im nächsten Abschnitt näher erläutert wird.

7.4.1 Interaktion

Für die "grobe" Interaktion mit der Szene bietet Coin3D, Selection und Deselection Callback-Funktionen an. "Grob" heißt in diesem Sinne, dass nur ganze sichtbare Objekte angewählt werden können, nicht hingegen Punkte auf diesen Objekten. Dazu wird dem Wurzelknoten durch

Listing 7.3: Interaktion

```
1
2 void made_selection( void* userdata, SoPath* path ){
3     // behandle selektion
4 }
5
6
7 {
8     root = new SoSelection; // Spezialisierung von SoSeparator
9     ...
10    root->addSelectionCallback( made_selection, (void*) 1L );
11    root->addDeselectionCallback( made_selection, (void*) 0L );
12    ...
13 }
```

"addSelectionCallback" mitgeteilt, dass er, wenn ein Objekt angewählt wird, die Funktion "made_selection" rufen soll. Diese wertet dann mithilfe der "Dentition-Klasse" aus, zu welchem Objekt (ToothVis, BracketVis oder WireVis) das ausgewählte Objekt gehört.

Damit Punkte in der Szene ausgewählt werden können, wird die von Coin3D angebotene Klasse "SoRayPickAction" benutzt. Dazu wird als Erstes eine Callback-Funktion "event_cb" eingerichtet. Diese wird gerufen, wenn der Nutzer auf die Bildschirmebene klickt. Wird festgestellt, dass die Maustaste mit der der Nutzer geklickt hat, die rechte Maustaste war, so wird eine "SoRayPickAction" erzeugt die den gesamten SG traversiert. Es ist wichtig einzelne Punkte anwählen zu können, da es nur so möglich ist, ein Bracket an spezifischen Stellen auf der Oberfläche eines Zahnes zu platzieren.

7.4.2 Simulationsschritte

Sind alle Einstellungen an der Simulation vorgenommen, kann durch "Bewegung simulieren" (Use case Abschnitt 4.1.7) die Simulation in Gang gesetzt werden. Hierbei wird ein "Sensor" benutzt. Der "SoIdleSensor" aus dem Dentition-Objekt, ruft sobald CPU-Zeit zur Verfügung steht die Methode "IdleFunction" des Dentition-Objekts. Diese bewirkt dann wiederum den Aufruf der Step-Methoden der von dem Dentition-Objekt verwalteten Unterobjekte. Hierbei werden erst alle Positionen, die sich durch die letzte Bewegung verändert haben, neu berechnet, um daraufhin die neuen Bewegungen zu errechnen.

7.4.3 Visualisierungs-Klassen

Aus Austauschbarkeits- und Übersichtsgründen wurde der Visualisierungs-Code vom Simulations-Code der Anwendung getrennt. Hierzu werden Visualisierungs (VIS)-Klassen benutzt, die auf die Daten der Simulation (SIM)-Klassen zugreifen.

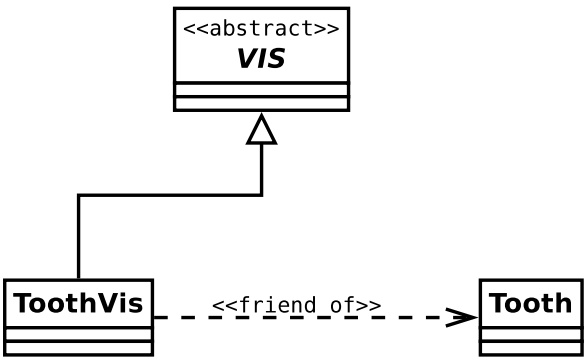


Abbildung 7.5: Aufteilung in VIS und SIM

Eine Visualisierungs-Klasse besitzt immer eine Referenz auf die dazugehörige Simulations-Klasse (zum Beispiel BracketVIS auf SlotSIM). Ein weiterer Vorteil, der durch die Trennung entsteht, ist eine leichtere Testbarkeit. Eine Simulationsklasse ist auch ohne die Visualisieungsklasse lauffähig und kann deshalb darauf getestet werden, ob Fehler möglicherweise erst durch die Darstellung aufgetreten sind.

7.4.4 Fähigkeitenvererbung

"Implementing everything under the umbrella of a do-it-all interface is not a good solution, for several reasons. Some important negative consequences are intellectual overhead, sheer size, and inefficiency. Mammoth classes are unsuccessful because they incur a big learning overhead, tend to be unnecessarily large, and lead to code that's much slower than the equivalent handcrafted version." [8]

Angelehnt an dieses Zitat aus dem Buch "Modern C++ Design" [8] wurde vermieden einzelne, nicht zusammenhängende Funktionen in einzelne Interfaces zu packen. Stattdessen sind die Fähigkeiten einzelner Visualisierungsobjekte in einzelne Klassen getrennt worden. Auf diese Art und Weise ist es einfacher kleine Klassen zu verwalten, die dann von den Objekten, die spezielle Funktionalitäten brauchen, geerbt werden könnnen. Eine neue Funktion anzulegen verändert dann nur die Funktion der Objekte, die von diesen Funktionsklassen ableiten. Im Gegensatz zu diesem Vorgehen könnten einfach alle anzeigbaren Objekte neue Funktionen erben, was aber zu unnötiger Größe der Klassen sowie möglichem Fehlerpotenzial führen würde.

Die VIS-Klassen erben durch eine Mixinfunktionalität Fähigkeiten. Mixins funktionieren durch Mehrfachvererbung und sind in Sprachen wie Ruby, D, oder C# implementiert. Mehrfachvererbung wird häufig als gefährlich angesehen, kann aber bei richtiger Anwendung ein sehr mächtiges Werkzeug sein. Eine VIS-Klasse kann die Fähigkeiten Auswählbarkeit, Suchbarkeit und Positionsbestimmbarkeit ableiten.

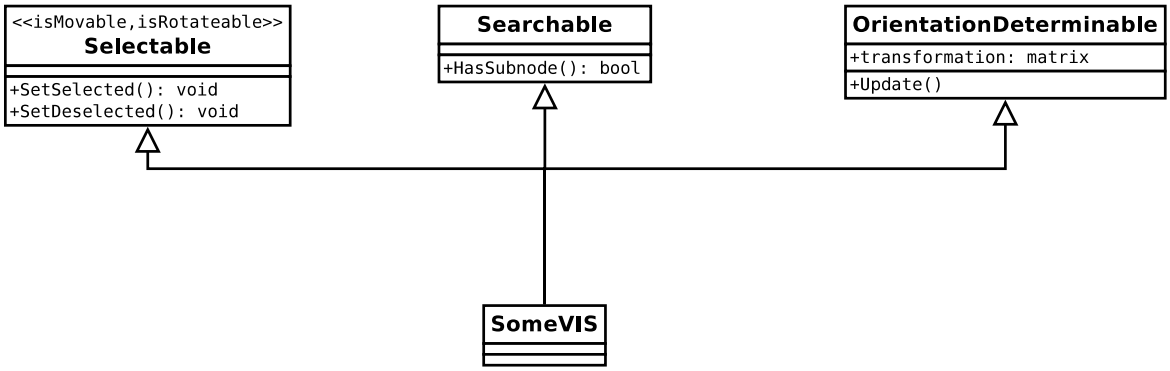


Abbildung 7.6: Funktionsvererbung

Es gibt gewisse Nachteile, die bei der Benutzung von Mixins die aufgrund der Mehrfachvererbung, auftreten. Wird eine Variable in mehreren Mixinklassen gleich genannt, kann es zu Namenskonflikten kommen. Werden diese aber sauber in den "private"-Bereichen der Klasse geführt, ist ein versehentlicher Zugriff unmöglich. Bei Mehrfachvererbung wird häufig das "Diamond-Problem" angeführt [3], bei dem im Zusammenhang mit virtuellen Methoden Probleme auftreten können. Dieses stellt aber kein Problem dar, weil bei Mixins keine virtuellen Methoden eingesetzt werden.

Die Auswählbarkeits-Basisklasse verfügt über die Eigenschaft, Fähigkeiten wie die Drehbarkeit und die Verschiebbarkeit hinzuzufügen. Diese sind als Templateargumente vorhanden um die Klasse zu verkleinern.

Listing 7.4: Klasse selectable

```
1 template < bool tIsRotateable = false , bool tIsMoveable = false >
2 class Selectable{
3     ...
4 };
```

Dabei ist durch jeweils ein Templateargument "isMovable" und "isRotateable" eine automatische Erstellung von Manipulatoren möglich. Werden die Argumente nicht ausdrücklich mit "true" spezifiziert so kommt es nicht zur Integration der Eigenschaft in die Klasse.

7.5 Umsetzung der Tests

7.5.1 Funktionstest

Damit verifiziert werden kann, ob alle Bewegungen der Zähne sinnvoll sind, werden Testfälle angelegt. Ein Testfall prüft, ob bei einem bekannten Input ein bekannter Output erreicht wird. Damit kann sichergestellt werden, dass die Simulation die richtigen Werte produziert und die Qualitätsanforderung der "Zuverlässigkeit" und "Fehlertoleranz" (Abschnitt 4.2.2) erfüllt. In dieser Simulation werden die einzelnen Klassen in Klassentests und die Zusammenarbeit der Klassen in Modultests untersucht. Somit kann sichergestellt werden, dass Fehler die durch die Progammumsetzung auftreten minimiert werden. Die Korrektheit der Simulation wird hingegen in einem Systemtest geprüft.

Beginnt man ein System zu prüfen, in dem man die Zusammenarbeit aller Klassen miteinander testet, so können Fehler nur schwer gefunden werden. Zum einen kann man sich nicht sicher sein, wo der Fehler auftrat, zum anderen weiß man nicht, welcher Teil des Codes für das Auftreten verantwortlich ist. Mit einem Klassentest wird sichergestellt, dass Fehler nur innerhalb einer Domäne auftreten. Hierbei gibt es ein einheitliches Vorgehen. Als Erstes wird festgestellt, ob eine Klasse sich erzeugen und zerstören lässt. Denn sind die Konstruktoren beziehungsweise Destruktoren falsch, so resultiert dies in einem unbrauchbaren Objekt. Erst dann wird getestet, ob alle Methoden die richtigen Ergebnisse liefern.

Sind alle Fehler in einer Klasse im Klassentest behoben worden, so können im Unittest nur noch Fehler in der Zusammenarbeit auftreten.

Im Speziellen werden bei Boost "BOOST_AUTO_TEST_CASE" Makros benutzt, um einen Testfall anzulegen.

Listing 7.5: Testcase

```
1
2 struct Initializer {
3     Initializer()
4       : t()
5     {
6         BOOST_TEST_MESSAGE("constructing");
7     }
8     ~Initializer()
9     {
10        BOOST_TEST_MESSAGE("destructing");
11    }
12    Obj t; // das zu testende objekt
13 };
14
15
16 BOOST_FIXTURE_TEST_SUITE( ObjTest , Initializer )
17
18 BOOST_AUTO_TEST_CASE( ObjTest )
19 {
20    // hier folgen bedingungen die testen ob
21    // t die richtigen daten aus einer funktion zurueckgibt
22
23    double a = t.Sqrt(1);
24    BOOST_CHECK( !std::isinf(a) );
25    BOOST_CHECK( !std::isnan(a) );
26    double b = t.Sqrt(-1);
27    BOOST_CHECK( std::isinf(b) );
28    BOOST_CHECK( std::isnan(b) );
29 }
30
31 BOOST_AUTO_TEST_SUITE_END()
```

Testablauf

Die Tests werden automatisch von einem Build-Server (Jenkins) durchgeführt, wenn eine neue Codeversion in das Repositorium des Servers geladen wurde. Der Server stößt eine Kompilation des Codes auf unterschiedlichen Systemen an. Dabei ist der Server selbst ein Klient des Bauvorgangs (Linux ARM-Architektur). Zwei virtuelle Maschinen sind gleichzeitig

damit beschäftigt eine Windows Portierung vorzunehmen. Ist der Code übersetzt, werden die Tests ausgeführt. Sollten Fehler auftreten, ist klar, dass ein Fehler durch die Portierung aufgetreten ist. Nach einem erfolgreichen Übersetztungsvorgang werden die Dateien, die für eine Ausführung unter Windows notwendig sind, gepackt und auf einen FTP-Server geladen. Eine virtuelle Maschine mit einer Windows XP Version, auf der keine Bibliotheken installiert sind, läd das Archiv vom FTP-Server, entpackt es und führt die Anwendung aus. Dieser Schritt stellt einen einfachen Installationstest dar.

7.6 Portierung

Die vorgenommene Portierung auf Windows wurde mit Hilfe des MinGW-Compilers vorgenommen. Dieser Compiler stellt eine Portierung des Gnu Compiler Collection (GCC) dar und ist deshalb einfach an das hauptsächlich unter Linux entwickelte Programm anpassbar. Eine Portierung von "Gnu Make" wird benutzt, um den Kompilationsprozess zu steuern. Die Makedateien sind mit den Linuxdateien nicht kompatibel, aber durch Anpassung an die Windows Umgebung war es möglich eine gesonderte Makedatei herzustellen. Eine Umsetzung durch "Microsoft Visual Studio" wurde nicht in Betracht gezogen, da nicht jeder Zugang zu diesem teuren System hat und eine kommandozeilenbasierte Übersetzung des Codes umständlich ist. Diese ist wichtig, da der Bauvorgang des Progammes automatisch vom Server an Klientensysteme verteilt wird und dazu nur ein Kommandozeilenzugang zur Verfügung steht.

Ergebnisse und Einordnung

In diesem Kapitel soll die Übereinstimmung der Simulation mit Patientendaten vorgestellt werden. Als Erstes wird dabei ein realer Fall erläutert, woraufhin mit der Simulation versucht wird, den gleichen Behandlungsverlauf nachzuvollziehen.

8.1 Beispielbehandlung

Die als Vergleich herangezogene Behandlung besteht aus zwei Phasen, in denen als Erstes ein Zahn extrudiert wird und anschließend an die Position eines gezogenen Milchzahns verschoben wird. Ziel der ersten Phase der Behandlung ist es, einen nicht durchgebrochenen Zahn in die Zahnreihe zu bewegen. Dazu wurden zwei Molaren, durch einen $0.32" \times 0.32"$ Stahlbogen (Transpalatinalbogen in Abbildung 8.1 rot dargestellt) passiv (ohne Aktivierung) verbunden. An diesem Transpalatinalbogen wurde ein $0.16" \times 0.16"$ Titan Molybden Alloy (TMA) Bogen angebracht (blau dargestellt) und so aktiviert, dass er den chirurgisch freigelegten Zahn aus seinem Knochenfach extrudiert.

In der zweiten Phase der Behandlung wird der Eckzahn (Dens caninus - gelb) gezogen, um Platz für den extrudierten Zahn zu machen. Durch eine Druckfeder (in Abbildung 8.4 grün dargestellt) wird verhindert, dass sich der seitliche Schneidezahn (orange) auf den Prämolar (blau) zubewegt. Die Zugfeder (rot) erbringt eine Kraft, die den Zahn an die freigewordene Stelle bewegt. Leider existiert, aufgrund der noch andauernden Behandlung kein Bild der

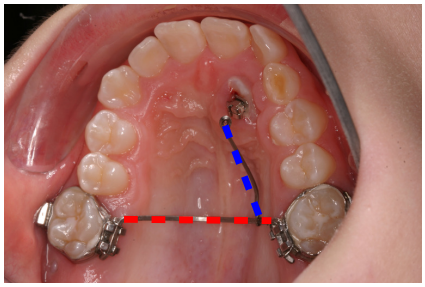


Abbildung 8.1: Vorgespannter Drahtbogen

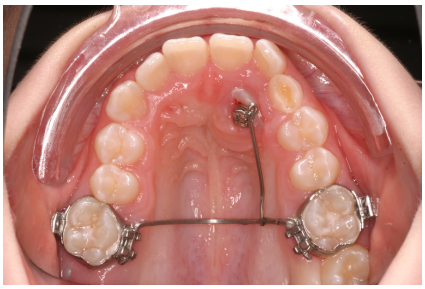


Abbildung 8.2: Drahtbogen ist in das Bracket eingegliedert

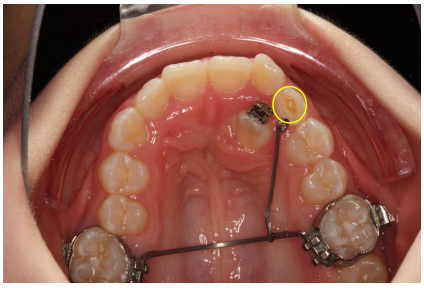


Abbildung 8.3: Deutliche Bewegung zum Betrachter

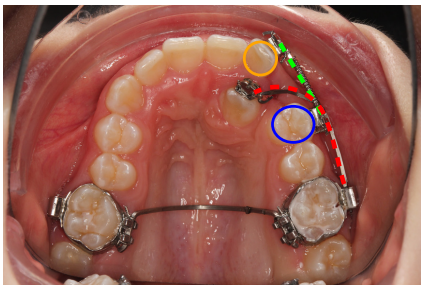


Abbildung 8.4: Nach Ziehen des Eckzahns

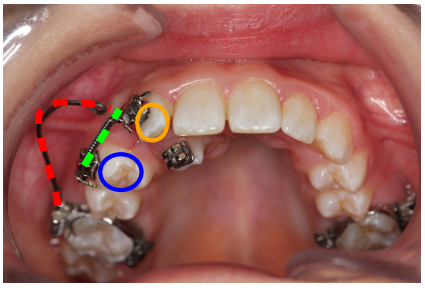


Abbildung 8.5: Vorderansicht

Endlage des Zahnes. Jedoch ist davon auszugehen, dass der Zahn in Richtung der Lücke kippen wird.

8.2 Ergebnisse

Um die Behandlung aus dem letzten Abschnitt umzusetzen, wurden folgende Schritte vollzogen. Als Erstes wurden aus Übersichtlichkeitsgründen die Zähne des Oberkiefers sowie die Siebener gelöscht und der zu simulierende Zahn an die richtige Stelle bewegt. Um die Kraft durch den Drahtbogen nach oben zu simulieren, wurde ein Bracket am Sechser befestigt und ein Drahtbogen in diesem eingesetzt. Dieser Drahtbogen wurde dann, wie in Abbildung 8.6 und 8.7, ausgerichtet.

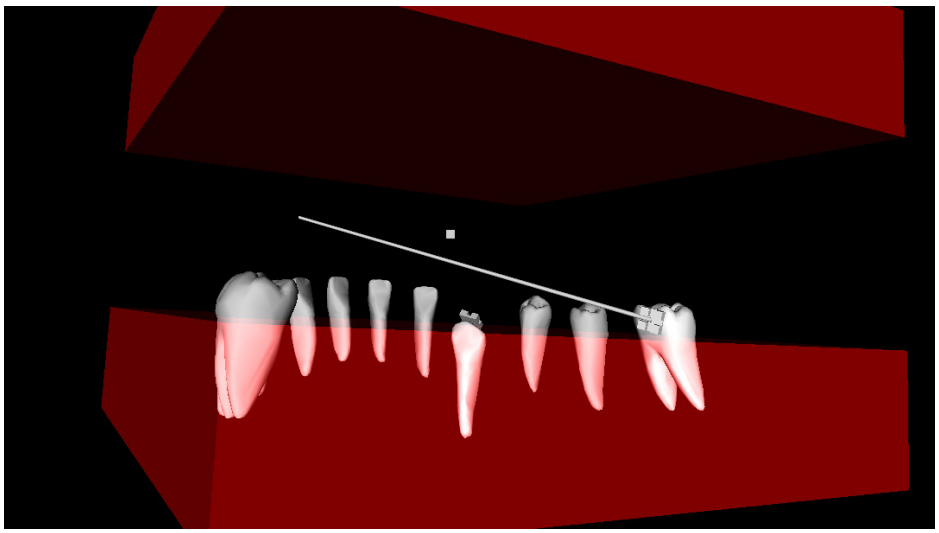


Abbildung 8.6: Zahn in Ausglanslage. Drahtbogen ist auf eine Position über dem Zahn aktiviert

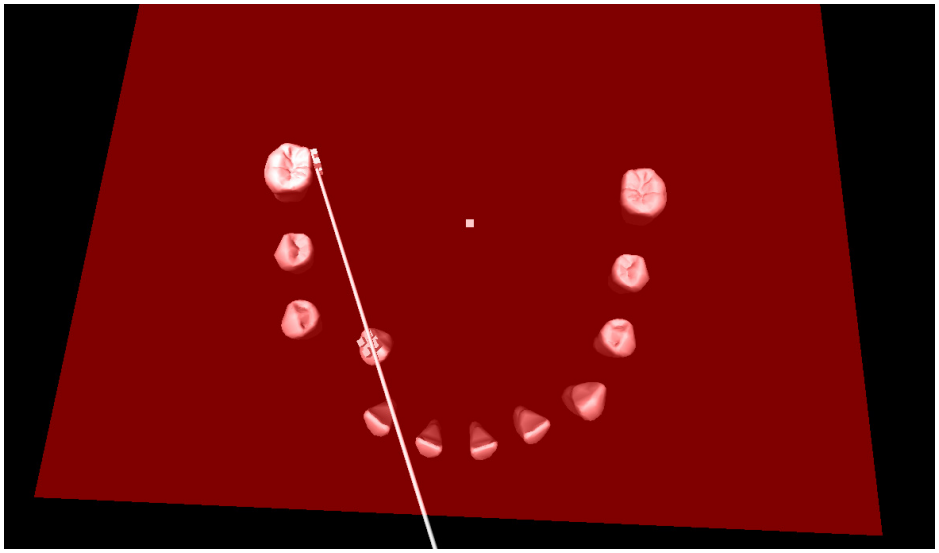


Abbildung 8.7: Ansicht von oben auf die Ausgangslage

Im nächsten Schritt wurde das Bracket auf dem Eckzahn mit dem Draht verbunden. Eine graue Linie wird eingeblendet, um die momentane Lage des Drahtes zu verdeutlichen. Nach dem Start der Simulation werden Pfeile angegeben, welche die Kraft (Pfeil nach oben) und das Drehmoment (sehr kleiner Pfeil zur Seite) verdeutlichen.

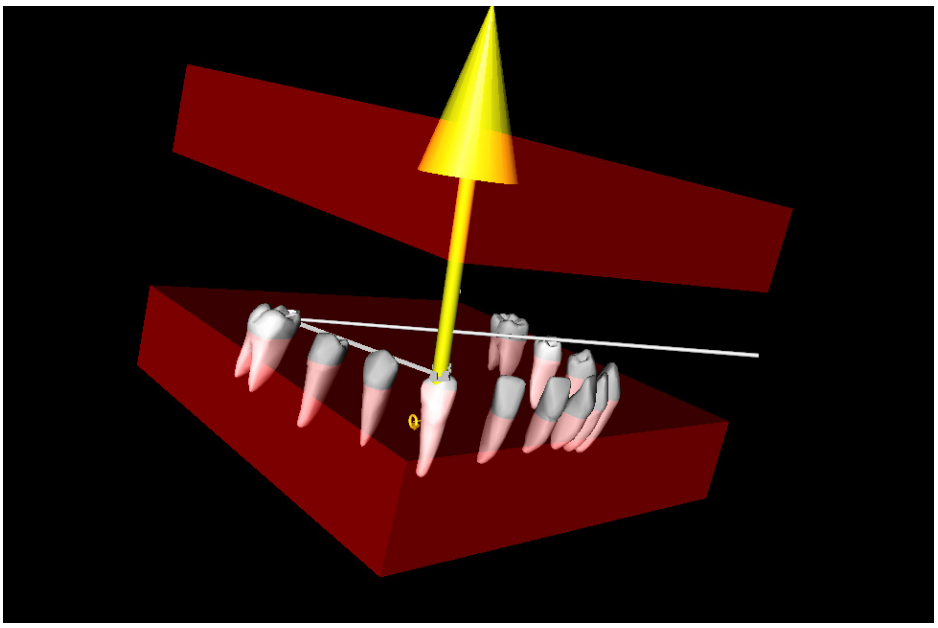


Abbildung 8.8: Kraft die nach oben auf den Eckzahn wirkt

Im zweiten Teil der Behandlung wurde das Bracket am Sechser samt Draht gelöscht und ein neues, auf der anderen Seite angebracht (Abbildung 8.9). Das Bracket des Eckzahns wurde entfernt und ein Neues auf der vestibulär befestigt. Dieses Bracket wurde dann mit dem Drahtbogen des am Sechser angebrachten Brackets verbunden (Abbildung 8.11).

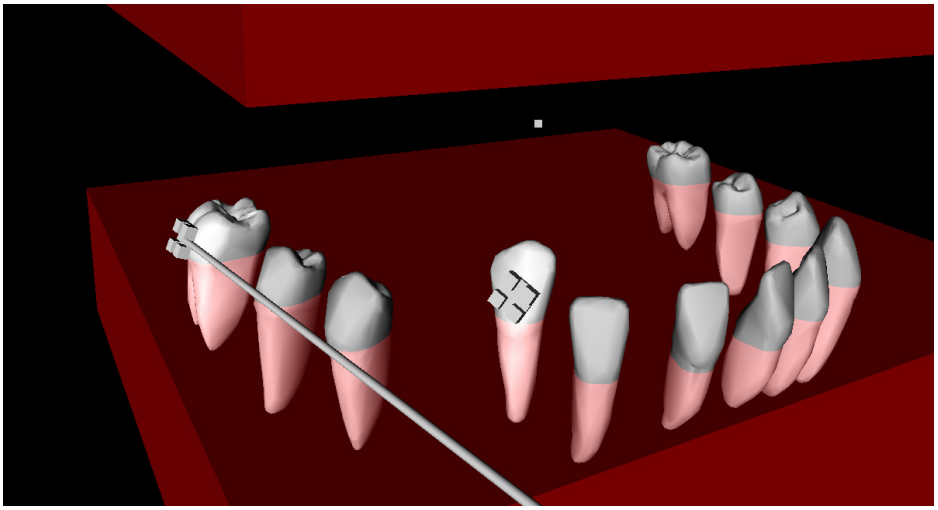


Abbildung 8.9: Extrudierter Zahn und am Sechser befestigetes Bracket mit Drahtbogen

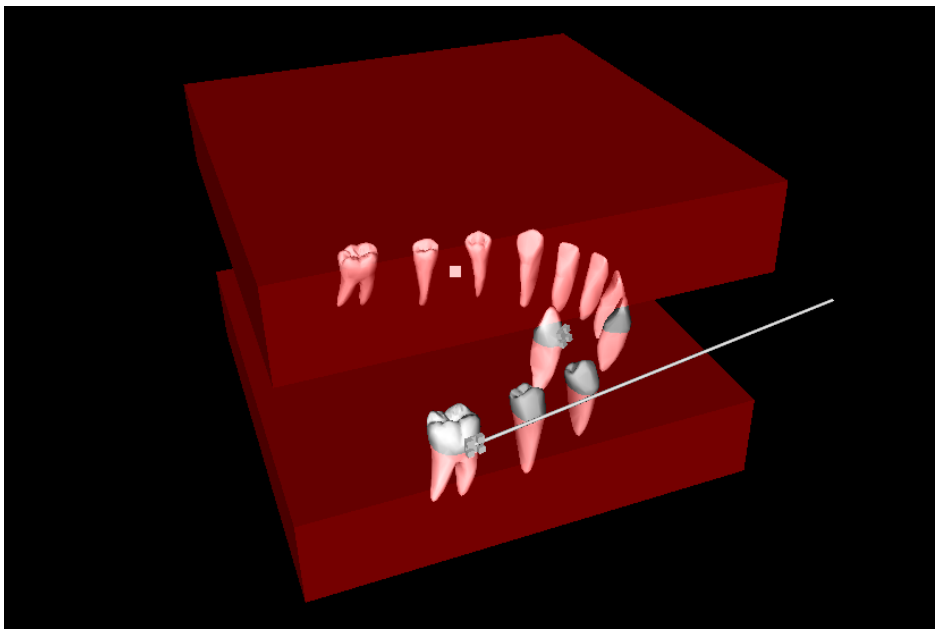


Abbildung 8.10: Sichtwinkel von hinten rechts

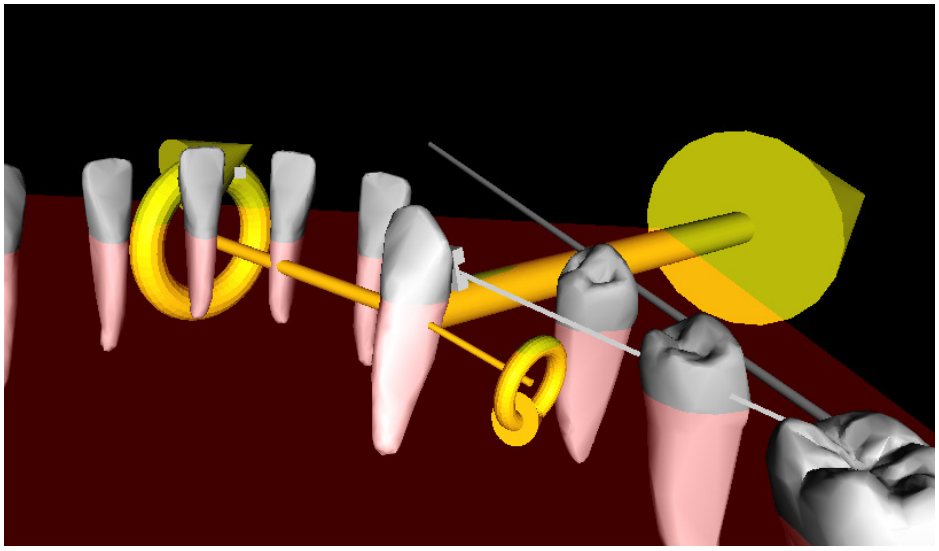


Abbildung 8.11: Kraft auf den Zahn in Richtung Zahnlücke mit zustätzlicher Drehbewegung und Gegendrehung durch das Zahnfleisch

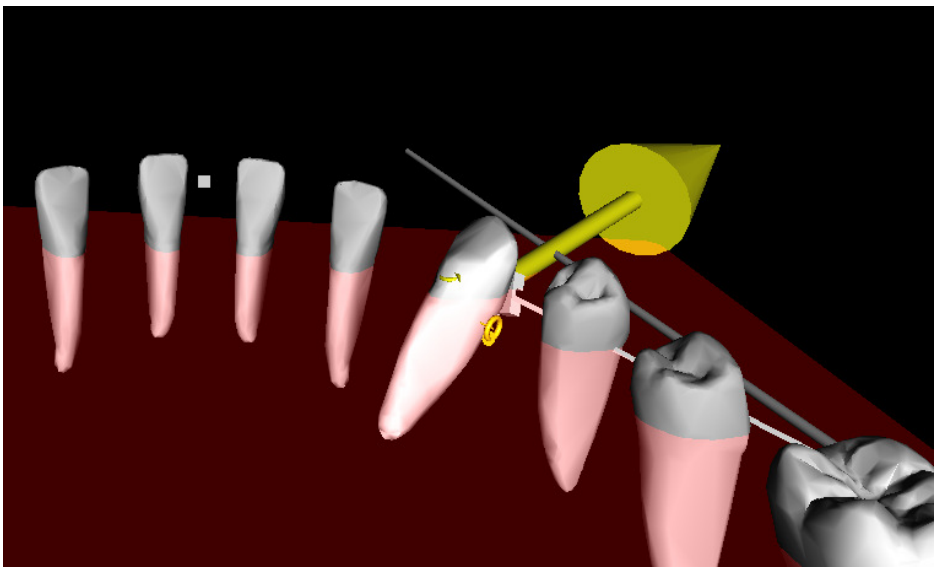


Abbildung 8.12: Deutliche Kippung des Zahnes

8.3 Ergebnisdiskussion

Im Vergleich zur realen Behandlung liegt die erste Phase der Simulation dicht an der Realität. Es ist sichtbar, dass die Bewegung, aufgrund des sinkenden Widerstands, schneller wird, desto mehr der Zahn aus dem Alveolar herausgekommen ist.

In der zweiten Phase lässt sich eine Kippung des Zahnes in Richtung der Lücke erkennen, wie sie zumindest vom Kieferorthopäden vorhergesagt ist. Ob diese Kippung realistisch stark ist, kann nur durch einen Vergleich mit der zukünftigen Behandlung erfolgen. Ist diese nicht wie erwartet, muss das Gegenkraftmodell anders spezifiziert werden. Im vorherigen Beispiel wurde eine Gegenkraft von 50% angenommen. Durch eine Korrektur auf zum Beispiel 90% kann, wie in Abbildungen 8.13 und 8.14, ein anderes Ergebnis erreicht werden.

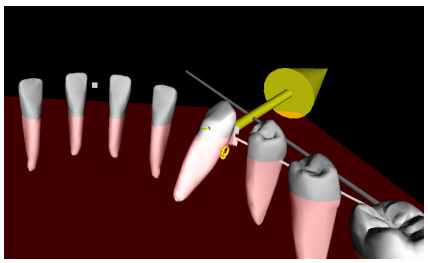


Abbildung 8.13: Kippung bei 50% Gegenkraft

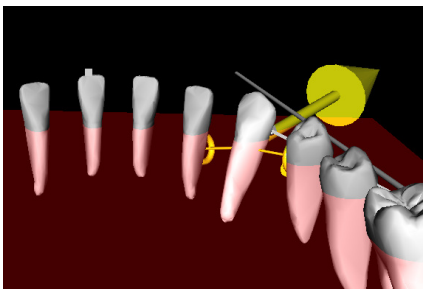


Abbildung 8.14: Kippung bei 90% Gegenkraft

Die Spezifizierug des Gegenkraftanteils ist nun die Aufgabe der Kieferorthopäden.

Zusammenfassung und Ausblick

9.1 Zusammenfassung

Die Hauptaufgabe dieser Arbeit besteht darin, die Einwirkung von Kraft durch Zahnspangen auf Zähne zu untersuchen. Hierbei besteht das Interesse darin herauszufinden, weshalb sich die Zähne in bestimmter Art und Weise bewegen. Um dies zu erreichen, ist wie folgt vorgegangen worden.

Als Erstes wurden die Grundlagen der Zahnbewegung und das notwendige Wissen über den Zahnhalteapparat vermittelt. Dabei wurde erklärt, wie Zahnstellungen durch kieferorthopädische Apparaturen verändert werden können. Anschließend ist die, für die Umsetzung des in dieser Arbeit verwendeten Programms, wichtige Physik erläutert worden. Es wurden die Formeln beschrieben, die in der Implementation des Programmes umgesetzt werden mussten, um die Bewegung der Zähne zu simulieren. Danach sind die Grundlagen der Visualisierung durch einen Szenengraphen geschildert worden. Zu beschreiben welchen Zusammenhang es zwischen Zustandsmaschine und Szenengraph gibt, stellte dabei die Hauptaufgabe da.

Das Kapitel der Vorarbeiten beschäftigte sich damit die bereits geleisteten Vorarbeiten auf dem Feld der Zahnmedizin, der Physik und der Informatik zu beleuchten.

Auf Basis der Grundlagen und der Vorarbeiten wurden die Anforderungen an das zu entwickelnde Programm spezifiziert. Hierfür wurden die Anforderungen nach funktionalen Anforderungen auf Basis von Use cases und die nicht-funktionalen Anforderungen auf Basis des ISO9126 Standards beschrieben. Es wurde herausgestellt, dass die Anforderung nach Erlernbarkeit und Verständlichkeit sowie Richtigkeit und Übertragbarkeit die Hauptaspekte darstellen, nach denen entwickelt werden muss.

In der Analyse sind die Anforderungen untersucht und qualitätssichernden Maßnahmen beschrieben worden. Als qualitätssichernden Maßnahmen wurden Klassen-, Modul- und Systemtests erdacht, die in den späteren Phasen näher spezifiziert wurden.

Im Kapitel Entwurf ist auf Überlegungen zur Datenbeschaffung und die Zusammenarbeit einzelner Komponenten der Software eingegangen worden. Eine Methode, nach der die Gegenkraft berechnet werden kann, wurde beschrieben, um die Systemantwort des Alveolars beschreiben zu können. Eine nähere Beschreibung der zur Visualisierung eingesetzten Bibliothek Coin3D sowie der Testumgebung Boost.Test erfolgte, um die Implementation vorzubereiten.

Anhand der Beschreibung der Umsetzung, der medizinischen und physikalischen Voraussetzungen wurde eine Erklärung des Programmes vorgenommen, welche durch die Besonderheiten der Implementation in C++ durch Mixinklassen vervollständigt wurde.

Abschließend ist eine Ergebnispräsentation, durch den Vergleich einer realen Behandlung, mit einer simulierten vorgenommen worden und festgestellt worden, dass sich durchaus ähnliche Bewegungen erzeugen lassen, das Gegenkraftmodell aber noch genauer spezifiziert werden

muss.

9.2 Ausblick

Es gibt einige Dinge, die in einer zukünftigen Implementation umgesetzt werden sollten. Zum einen müssen bessere Daten erhoben und zum anderen sollen die Einsatzmöglichkeiten der Anwendung erweitert werden. Nur mit besseren Daten kann die Anwendung eine genauere Gegenkraftberechnung erstellen. Zusätzlich zu diesen besseren Daten kann auch ein anderes Gegenkraftmodell verwendet werden. Es gibt die Möglichkeit die Systemantwort durch eine Flüssigkeitsreibung proportional zur Geschwindigkeit zu errechnen.

Zur Verbesserung der Datenlage wurden bereits Daten aus einem CT-Gerät von den Zähnen des Autors erstellt. Eine Weiterverarbeitung in Daten, für die in dieser Arbeit beschriebenen Anwendung, wird in nächster Zeit vollzogen werden. Doch nicht nur die Daten für die Zähne sind von besonderem Interesse, auch die Daten der Drähte und ihr Einfluss auf die Simulation ist ein wichtiger Faktor. Hierzu soll im Rahmen einer Doktorarbeit eines Zahnmediziners eine Datenbank aufgebaut werden. In ihr sollen Kraft-Auslenkungs-Daten für unterschiedliche Drahtbögen mit unterschiedlichen Materialeigenschaften gespeichert werden. Mit den neuen Daten kann die Anwendung dann soweit erweitert werden, dass eine realistischere Berechnung von bestimmten Behandlungsszenarien erfolgen kann.

Wenn durch Tests mit Nutzern und den darauffolgenden Verbesserungen eine hohe Stabilität der Anwendung gewährleistet werden kann, ist es möglich, die Anwendung zur Prüfung von Zahnmedizinern und Kieferorthopäden zu verwenden. Durch Vorgabe eines Szenarios, das von einem zu testenden Studenten zu lösen ist, kann eine gute, nachvollziehbare Prüfungssituation geschaffen werden. Notwendig für die Vorgabe eines Szenarios ist es, dass Prüfer diese Szenarios erstellen können, um diese dann vom Studenten bearbeiten zu lassen. Hierzu muss eine Speicherfunktion für die kombinierte Struktur des Szenengraphen und der Simulationsstruktur implementiert werden.

Aufgrund dessen, dass durch diese Softwarelösung Positionsveränderungen der Zähne vorhergesagt werden können, kann auch eine Veränderung des über den Zähnen liegenden Gewebes vorhergesagt werden. Auf Basis dieser Information kann eine 3D-Ansicht eines Gesichtes verändert werden, um eine bessere Patientenaufklärung vorzunehmen. Um an Daten für ein Gesicht zu kommen, ist ein Projekt in Planung, bei dem eine X-Box Kinect Kamera verwendet werden soll, um möglichst preisgünstig Daten zu erhalten.

Auf der Seite der Programmumsetztung gibt es Verbesserungspotenzial im Programmerzeugungsprozess. Durch Benutzung von “Boost.Build” als Ersatz für “Gnu Make”, ist es möglich, eine vollständige Abstraktion vom Compiler zu erreichen. Dadurch ist eine Portierung auf ein anderes Betriebssystem einfacher und im Gegensatz zu den jetzt verwendeten Makefiles muss nur noch ein Script verwendet werden, um die Anwendung zu erstellen. In der jetzigen Implementation werden alle Teile der Anwendung für die Tests immer wieder neu übersetzt. Durch ein Linken gegen die Anwendung selbst, wäre es möglich viel Zeit zu sparen, die bei der ständigen Neuübersetztung verschwendet wird.

Abbildungsverzeichnis

Glossar

Typodont
Ein Zahnmodell für Studenten, an dem Eingriffe geübt werden können

Alveolus dentalis
deut. Zahnfach. "bezeichnet man eine Vertiefung in den Kieferknochen, in der ein Zahn mit seiner Wurzel steckt" [30]

Assertion
Test ob eine logische Aussage zutrifft

Coin3D
Ist eine frei verfügbare Version von Open Inventor

Massenschwerpunkt
Punkt in dem alle Masse eines Objekts abstrahiert ist

Computer Tomograph
Gerät zur medizinischen, dreidimensionalen Bildgebung

CppUnit
Unittest Framework - Portierung von JUnit nach C+

CxxTest
Unittest Framework - Testframework ist in Java geschrieben

Finite Element Methode
Sehr aufwändige Methode, bei der die an der Berechnung beteiligten Komponenten in endlich viele Zellen unterteilt werden

GCC
Ansammlung von unterschiedlichen Compilern, hauptsächlich für C, C+und Objective-C

Gtk
Gimp Toolkit - Bibliothek für Userinterfaces auf verschiedenen Betriebssystemen

lingual
zur Zunge hingewandt

MinGW
Implementierung des GCC für Windows

Molaren
Mahlzähne

Multibandapparatur
Apparatur aus mehreren Drähten und Brackets. ugls Zahnspange

Object File Format
Sehr einfaches Dateiformat für dreidimensionale Objekte

Open Inventor
Ist eine 3D-Grafik API von SGI. Sie bietet höhere Funktionalität für OpenGL, wie zum Beispiel einen Szenengraph

OpenGL
"eine Spezifikation für eine plattform- und programmiersprachenunabhängige Programmierschnittstelle zur Entwicklung von 2D- und 3D-Computergrafik" [21]

Qt
Plattform zur Entwicklung von Userinterfaces auf unterschiedlichen Betriebssystemen

Szenengraph-Basisobjekte
Basis Objekte im Szenengraph mit wenig funktionalität

Szenengraphobjekte
High level Objekte im Szenengraph

Szenengraph
Hierarchisch gegliederter Baum, der es erlaubt die Zustandsinformationen der zugrundeliegenden Zustandsmaschine komfortabel zu kontrollieren

Unittest
zu Deutsch Modultest. Test der Zusammenarbeit von Klassen

verblockt
Zähne werden so fixiert, dass sie sich nicht mehr bewegen können

vestibulär
zur Wange hingewandt

Zahnzement
Der Teil des Zahns an dem die Sharpey Fasern befestigt sind

Akronyme

CAD	Computer Aided Design
COM	Massenschwerpunkt englisch “center of mass”
CT	Computed Tomograph
DAG	Directed Acyclic Graph
DVT	Digital-Volumen-Tomographie
FEM	Finite Elemente Methode
GCC	Gnu Compiler Collection
KFO	Kieferorthopädie
KISS	keep it short and simple
MRT	Magnetresonanztomographie
OFF	Object File Format
OI	Open Inventor
OpenGL	Open Graphics Library
SG	Szenengraph
SGO	Szenengraph Objekt
SIM	Simulation
TMA	Titan Molybden Alloy
VIS	Visualisierungs

Symbolverzeichnis

$\vec{\omega}$	Drehachse
$\vec{L}$	Drehimpuls
m_g	Gesamtmasse
$\vec{v}$	lineare Geschwindigkeit
$\vec{f}$	Kraft
$\vec{f}_b$	Kraft bzgl. Massenschwerpunkt
δ	Kronecker-Delta
m	Masse

$\vec{r}_s$	Massenschwerpunkt
$\mathbf{A}$	Orientierung
$\vec{r}_n$	Koordinaten von Punkten im Körper
$\vec{\tau}_b$	Drehmoment
$\tau_{\vec{\omega}}$	Das Drehmoment bezüglich eine bestimmten Achse
L	Trägheitsmoment
I	Trägheitstensor
$\vec{\omega}$	Winkelgeschwindigkeit

Literaturverzeichnis

[1] Dragger. http://doc.coin3d.org/images/Coin/draggers/trackball.png.

[2] Gtk vs qt. http://www.wikivs.com/wiki/GTK_vs_Qt. 09. 02. 2012.

[3] In c++, what's the diamond problem, and how can it be avoided? http://www.programmerinterview.com/index.php/c-cplusplus/diamond-problem/.

[4] Iso 9126 software quality characteristics. http://www.sqa.net/iso9126.html. 04. 02. 2012.

[5] Object file format (.off). http://segeval.cs.princeton.edu/public/off_format.html. 04. 02. 2012.

[6] Wurzelhaut. http://flexikon.doccheck.com/Wurzelhaut. 25. 01. 2012.

[7] Richard A. and Hocevar. Understanding, planning, and managing tooth movement: Orthodontic force system theory. *American Journal of Orthodontics*, 80(5):457 – 477, 1981.

[8] Andrei Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[9] Dr. Elke Seitz und Uli Kloppmann. Gingivitis und parodontitis. http://www.dr-elke-seitz.de/Gingivitis1.htm25.01.2012. 02. 02. 2012.

[10] Dr. md. dds. Giorgio Fiorelli Prof. dds. Dr. Odont. Birte Melsen . Biomechanics in orthodontics 3.0. http://www.libra-ortho.com/en/index.php?option=com_content&task=view&id=159&Itemid=63. 25. 01. 2012.

[11] Dr. med. dent. Jasmin Kesseler. Klassische brackets. http://www.kfoteam-wesel.de/brackets_klass.htm.

[12] Dr. med. dent. Torsten Blens. Wie können zähne bewegt werden? http://www.zahnklammern.de/allg_bewegung.html. 25. 01. 2012.

[13] Egallois. Open inventor. http://en.wikipedia.org/wiki/Open_Inventor. 16. 01. 2012.

[14] Fish GD. Some engineering principles of possible interest to orthodontists. *Dent. Cosmos*, 1917.

[15] Herbert Goldstein. *Klassische Mechanik*. Addison-Wesley, 1981.

[16] Chris Hecker. Rigid body dynamics. http://chrishecker.com/Rigid_Body_Dynamics. 03. 02. 2012.

[17] Hanau Heraeus Kulzer GmbH. Demo-cd heraeus premium zähne. [CD-ROM], 2005.

[18] ISO. Homepage. http://www.iso.org/iso/home.html. 08. 02. 2012.

[19] Christian Johner. Vorsicht mit funktionalen und nicht-funktionalen anforderungen! http://www.ehealthkarriere.de/tag/iso-9126.

[20] Josie Wernecke. The inventor mentor: Programming object-oriented 3d graphics with open inventor^{TM}. http://www-evasion.imag.fr/~Francois.Faure/doc/inventorMentor/sgi_html/ch03.html. 20. 01. 2012.

[21] Jstaudemeyer. Opengl. http://de.wikipedia.org/wiki/OpenGL. 16. 01.2012.

[22] KaiMartin. Trägheitsmoment. http://de.wikipedia.org/wiki/Tr%C3%A4gheitsmoment. 20. 01. 2012.

[23] Khronos Group. Homepage. http://www.opengl.org/.

[24] Steven J Lindauer. The basics of orthodontic mechanics. *Seminars in Orthodontics*, 7(1):2–15, 2001.

[25] Michael Menner. Coin3d - 3d visualisierungs toolkit. Januar 2008.

[26] Brian Mirtich. Fast and accurate computation of polyhedral mass properties. *J. Graph. Tools*, 1:31–50, February 1996.

[27] Noel. Exploring the c++ unit testing framework jungle. http://gamesfromwithin.com/exploring-the-c-unit-testing-framework-jungle. 02. 02. 2012.

[28] Richard J. Smith and Charles J. Burstone. Mechanics of tooth movement. *American Journal of Orthodontics*, 85(4):294 – 307, 1984.

[29] Paul S. Strauss and Rikk Carey. An object-oriented 3d graphics toolkit. *SIGGRAPH Comput. Graph.*, 26:341–349, July 1992.

[30] Uwe Gille. Zahnfach. http://de.wikipedia.org/w/index.php?title=Zahnfach&action=history. 25. 01. 2012.

Selbstständigkeitserklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel sind angegeben. Die Arbeit hat mit gleichem bzw. in wesentlichen Teilen gleichem Inhalt noch keiner Prüfungsbehörde vorgelegen.

Greifswald , 22. Februar 2012

Ort, Datum, Stefan Kemnitz

Appendix

A Use cases

Use case Nummer	1
Bezeichnung	Datensatz laden
Akteure	Benutzer
Vorbedingungen	Das Programm ist gestartet
Nachbedingungen	Der Datensatz ist geladen
Standardablauf	1. Benutzer wählt Datensatz laden aus 2. System antwortet durch einen Dialog zur Dateiauswahl 3. Benutzer wählt die zu ladende Datei an 4. System lädt die Datei und zeigt sie bei Erfolg an.

Use case Nummer	2
Bezeichnung	Bracket hinzufügen
Akteure	Benutzer
Vorbedingungen	Alle Daten sind geladen
Nachbedingungen	Eine visuelle Repräsentation eines Brackets befindet sich auf der Oberfläche des Zahns
Standardablauf	1. Benutzer wählt Bracket erstellen aus 2. System zeigt einen geänderten Curser an 3. Benutzer klickt auf die Oberfläche eines Zahnes 4. System antwortet durch die Erstellung einer visuellen Repräsentation eines Brackets

Use case Nummer	3
Bezeichnung	Bracket ausrichten
Akteure	Benutzer
Vorbedingungen	Ein Bracket ist an einem Zahn angebracht
Nachbedingungen	Das Bracket ist nach den Wünschen des Nutzers ausgerichtet
Standardablauf	1. Benutzer wählt ein Bracket durch Anklicken an 2. System antwortet durch Farbliche Änderung des Brackets und zeigt einen Dragger an der das Drehen des Objekts ermöglicht 3. Benutzer dreht den Bracket in die richtige Ausrichtung

Use case Nummer	4
Bezeichnung	Draht hinzufügen
Akteure	Benutzer
Vorbedingungen	Mindestens zwei Brackets sind angebracht
Nachbedingungen	Eine visuelle Repräsentation eines Drahtes befindet sich zwischen den beiden Brackets
Standardablauf	1. Benutzer wählt das Wire-Tool aus 2. System antwortet durch ein geänderten Cursor 3. Benutzer wählt das erste Bracket an 4. System antwortet mit einer Farblichen Änderung des Brackets 5. Benutzer wählt das zweite Bracket an 6. System antwortet durch die Erstellung einer visuellen Repräsentation eines Drahtes

Use case Nummer	5
Bezeichnung	Drehachse festlegen
Akteure	Benutzer
Vorbedingungen	Ein Bracket ist an einem Zahn angebracht
Nachbedingungen	Die Drehachse des Zahnes ist festgelegt und eine visuelle Repräsentation dieser Festlegung existiert
Standardablauf	1. Benutzer wählt Drehachse festlegen aus 2. System antwortet durch Änderung des Cursors 3. Benutzer klickt ein Bracket an 4. System antwortet durch Hinzufügung der visuellen Repräsentation der Drehachsenbeschränkung und Schränkt die Drehbewegung des Zahnes auf eine Rotation um diese Achse ein.

Use case Nummer	6
Bezeichnung	Bewegung simulieren
Akteure	Benutzer
Vorbedingungen	Brackets und Drähte sind angebracht
Nachbedingungen	
Standardablauf	1. Benutzer wählt Simulation Starten an 2. System antwortet durch das Starten der Simulation

Use case Nummer	7
Bezeichnung	Bracket eingliedern
Akteure	Benutzer
Vorbedingungen	Brackets und Draht sind angebracht
Nachbedingungen	Bracket ist mit Draht verbunden
Standardablauf	1. Benutzer wählt Bracket verbinden aus 2. System antwortet durch veränderten Cursor 3. Benutzer klickt ein Bracket an 4. System markiert das Bracket farblich 5. Benutzer klickt eine Stelle auf dem Draht an 6. System stellt einen Draht dar der vom Bracket in dem der Draht eingehangen ist bis zum neu eingehangenen Bracket geht.