

Elastische Registrierung eines 3D-Oberflächenmodells anhand von aktiv gemessenen Stützdaten

Studienarbeit

vorgelegt von

Kristine Haase und Christian Bauer



Fraunhofer Institut
Angewandte
Informationstechnik

Institut für Computervisualistik
Arbeitsgruppe Computergraphik

Forschungsbereich Life Science Informatik

Betreuer: Dipl.-Math. Ron Schwarz, Fraunhofer-Institut FIT,
Forschungsbereich Life Science Informatik

Prüfer: Professor Dr.-Ing. Stefan Müller, Universität
Koblenz-Landau, Campus Koblenz

Juli 2004

Inhaltsverzeichnis

1	Problemstellung	1
1.1	Aufgabendefinition	1
1.2	Hintergrund	2
2	Grundlagen	4
2.1	Medizinische Grundlagen	4
2.1.1	Begriffe	4
2.1.2	Mechanische Beinachse	4
2.2	Das Tracking-System	6
2.3	KneeNavigator	8
2.4	Landmarken	10
2.5	Registrierungstechniken	12
2.5.1	Rigide Transformation	12
2.5.2	Affine Transformation	12
2.5.3	Nicht - lineare Transformation	13
2.6	Die Applikation ModelViewer	15
3	Lösung	19
3.1	Idee	19
3.2	Thin Plate Splines	20
3.3	Mathematischer Hintergrund	22
3.4	Implementation Elastische Registrierung mit korrespondierenden Landmarken . .	26
3.5	Implementation Elastische Registrierung ohne korrespondierende Landmarken . .	29
3.5.1	Korrespondenzfindung	30
3.5.2	Approximierender Thin Plate Spline	32
3.6	Visualisierung	34
3.7	Integration in die Software KneeNavigator	37
4	Ergebnisse	39
4.1	Freiheitsgrade	41
4.2	Anwendungsfälle	49
4.2.1	Optimale Messung	49
4.2.2	Anwendungsfehler: Pointer verrutschen	49
4.2.3	Vertauschung links-rechts	50
4.2.4	Simulation von krankem Gewebe	51
4.3	Bewertung	51
5	Zusammenfassung	53
6	Ausblick	54

Abbildungsverzeichnis

1	Schematische Darstellung eines Knies mit Erklärungen [1]	4
2	Darstellung der relevanten Achsen im Bein [2]	5
3	Wichtige Bestandteile des Trackingsystems; links: Kamerakopf, mitte: Zubehör, rechts: Marker [http://www.ndigital.com/polaris.php]	6
4	Ausschnitte aus dem KneeNavigator	9
5	Ausschnitt aus dem KneeNavigator	9
6	Landmarken am Femur demonstriert [3]	11
7	Beispiele am Objekt Lena	12
8	Beispiele am Objekt Lena	13
9	Komplettes Klassendiagramm der Applikation	15
10	Klassendiagramm der wichtigsten Klassen	16
11	Links: Nur anatomische Landmarken; Rechts: mit Flächen der Kno- dylen	29
12	Ansicht des unbearbeiteten Originalknochens im Modelviewer . .	35
13	Ansicht des Originalknochen im Modelviewer mit Landmarken . .	36
14	Originalbilder (a,c) und Deformationsbild (b), TPS - Deformation mit 10 Landmarken	39
15	TPS - Deformationen nach [4]	39
16	TPS - Deformationen nach [4]	40
17	KneeNavigator; Links: ursprüngliche Darstellung; Rechts: verbes- serte Darstellung	40
18	Seitenansicht: Threshold=10mm	41
19	Untenansicht: Threshold=10mm	41
20	Untenansicht: Threshold=5mm	42
21	Seitenansicht: Threshold=5mm	42
22	Seitenansicht: Threshold=1mm	43
23	Untenansicht: Threshold=1mm	43
24	Draufansicht: Gewicht=1	44
25	Untenansicht: Gewicht=1	44
26	Untenansicht: Gewicht=0,1	45
27	Draufansicht: Gewicht=0.005	45
28	Untenansicht: Gewicht=0.005	46
29	Seitenansicht: Gewicht=0,0	46
30	Seitenansicht: Punkte=1196	47
31	Seitenansicht: Punkte=598	47
32	Seitenansicht: Punkte=398	48
33	Untenansicht: Punkte=299	48
34	KneeNavigatorsituation 1	49
35	KneeNavigatorsituation 2	50
36	KneeNavigatorsituation 3	50
37	KneeNavigatorsituation 4	51
38	unebenes Gewebe am Knochen, simuliert mit Maske	51

Abstract

The medical software KneeNavigator which was developed for the surgery of total knee arthroplasty displays measured data as point clouds which are difficult to interpret. This situation has been renewed by building a adapted surface model depending on these point clouds. This partner thesis presents and implements a two-stage process for flexible registration. First, a three-dimensional surface model is adjusted, with the help of thin plate splines, to a small number of corresponding landmarks while maintaining the form of the surface. This leads to a registration. Second, several surface areas characterized by not corresponding measurement data are approximated to increase the accuracy of the surface model. Moreover, the results are presented and some usages with different parameters are illustrated.

1 Problemstellung

1.1 Aufgabendefinition

Diese Studienarbeit entstand in Zusammenarbeit mit dem Fraunhofer-Institut FIT, Forschungsbereich Life Science Informatik und in enger Kooperation mit der aus-gegründeten Firma LOCALITE GmbH, die das Navigationssystem *KneeNavigator* entwickelt und in die Praxis eingeführt hat. Die Software wird bei Operationen zur Implantation von Knieendoprothesen zur Unterstützung des Chirurgen eingesetzt und näher im Kapitel 2.3 beschrieben. Um die Schnittwinkel zu berechnen, die für die Passgenauigkeit der Prothese essentiell sind und dem Operateur vorge-schlagen werden, müssen während der Operation mit Hilfe eines Trackingsystems die Koordinaten der Punkte auf der Knochenoberfläche eingelesen werden. An-hand dieser Schnittwinkel werden die Instrumente ausgerichtet und ermöglichen dem Benutzer ein exaktes Vorbereiten der Knochen. Die Oberflächendaten werden zum jetzigen Zeitpunkt als Punktwolken dargestellt und ermöglichen dem Opera-teur eine visuelle Kontrolle der aufgenommenen Daten. Die wenigen Messdaten bedeuten eine Ansammlung von Punkten im Raum, aus denen kaum Informati-on gefiltert werden kann. Diese Sequenz der Anwendung soll verbessert werden, indem mit Hilfe dieser Punktwolke ein Knochenmodell, das dem Knochen des Patienten entspricht, entwickelt werden soll. Die Idee zur Lösung ist, ein allgemei-nes Oberflächenmodell eines Knochens an diese wenigen Punkte anzupassen. Die Schwierigkeit dabei besteht darin, die spezifische Form der Oberfläche zu erhal-ten. Diese elastische Registrierung soll in zwei Stufen durchgeführt werden, zum einem die Anpassung des Knochens mit korrespondierenden Landmarken (global), siehe Kapitel 3.2, zum anderen die Anpassung ohne fixe Punkte (lokal), behandelt im Kapitel 3.5. Neben der Einarbeitung in das Themengebiet und das Finden ei-nes geeigneten Algorithmus gehört zu der Durchführung zunächst die Entwicklung einer eigenständigen Applikation und im Anschluss daran die Integration und An-passung der Visualisierung der Software in den KneeNavigator.

1.2 Hintergrund

Eine Verbindung vieler Gebiete der Computergrafik zu einem einzelnen Arbeitsfeld stellt das Medical Image Processing dar.

Die Medizin hat sich in ihrer Durchführung in den letzten Jahren durch eine Vielzahl von Technisierungen und Verbesserungen erneuert. Durch die Entwicklung von bildgebenden Verfahren wie Ultraschall, Infrarot, Röntgenstrahlen, Radioaktivität und Magnetresonanz ist sie heute in der Lage jede Struktur und Funktion des menschlichen Körpers darzustellen. Weitere Anwendungsfelder sind Diagnosen, bildgestützte Therapie, computergestützte Operationen, Trainings-simulatoren, Vorbehandlungen und Planungen sowie Gültigkeitsprüfungen für klinische Hypothesen.

Für diese Problemstellungen werden individuelle Anforderungen gestellt. Da die Berechnungen zum Teil quantitativ, zum Teil qualitativ sind, müssen die Datenpakete in ein-, zwei- oder dreidimensionaler Genauigkeit schnell und effizient verarbeitet werden können.

Die Bilder müssen gegen Vorlagen, gegen Bilder ihrer Art oder gegen Bilder einer anderen Aufnahmeart angepasst werden, wobei bei allen Aspekten eine unterschiedliche Zeit und Lage zu berücksichtigen sein können. Jede Aufnahmeart hat dabei ihre individuelle Realitätsnähe, Vorteile und Grenzwerte. Die Processing-Pipeline wird beim Medical Image Processing auf Grund der auftretenden Algorithmenklassen in fünf Ebenen aufgesplittet: Enhancement, Segmentation, Quantification, Registration und Visualisation [[5],preface].

Das Enhancement beschäftigt sich mit der Verbesserung der Bildqualität und gilt damit als Vorstufe für die folgenden Arbeitsschritte.

Die Segmentation findet und markiert die Interessensbereiche im Bild und trennt sie bei Bedarf vom Hintergrund.

Die Quantification extrahiert diese gefundenen Bereiche und verarbeitet sie zu bearbeitbaren Datenpaketen wie Texturen, Oberflächen und Volumen.

Die anschließende Registration findet korrespondierende Bereiche in Bildern. Hierbei können die Bildart, Aufnahmezeitpunkt, Aufnahmeobjekt und Anzahl der Bilder variieren.

Die Visualisation schließlich stellt die berechneten Informationen dar. Hierzu gehören aber auch Animationen, Darstellung von Informationsmaßen, Interaktion und Benutzerführung sowie die Darstellung von funktionaler und temporärer Zusatzinformation.

Die schließende Gruppe ist das Arbeitsgebiet, in dem wir uns für diese Aufgabe befinden. Erst mit Reduktion der Rechnerkosten mit gleichzeitigem Anstieg der Grafikperformance und vor allem den physikalischen Entwicklungen an den Aufnahmetechniken konnte die Visualisierung in der Medizin derart eingebunden werden, wie sie heute Bestandteil der Processing-Pipeline ist. Der Anspruch der Medizin ist die vollständige Visualisierung jeden Schrittes der Behandlungsabfolge mit virtueller Interaktion und taktilem Feedback. Dabei können die vielen funktionalen Eigenschaften in mehreren Darstellungsweisen verarbeitet werden,

um unterschiedliche Interpretationsmöglichkeiten zu schaffen und simultan die Korrelation zwischen quantitativer und qualitativer Information. Auf Grund dieser Eigenschaften hat sich die Visualisierung zu einem autonomen Forschungsgebiet entwickelt und obwohl eine hohe Entwicklungsdichte zu verzeichnen ist, werden auf diesem Gebiet für die nächsten Jahre viele Problemstellungen erhalten bleiben. Einige Systeme sind in der Vergangenheit entwickelt worden und nicht zum Einsatz gekommen, weil die Computertechnik noch nicht weit genug entwickelt war. Dieses Leid wird trotz der heutigen hohen Rechnerqualität zunächst bleiben. Die breiten Entwicklungen erfordern auch in Zukunft schnellere und effizientere Algorithmen. Allein die Erhöhung der Kameraauflösung um den Faktor zwei bedeutet eine vierfache Vergrößerung des Bildareals und eine Verachtfachung des Datenvolumens. Kompressionstechniken und Archivgestaltungen werden dementsprechend neue Aktualität erhalten. [[5] , S.659ff und future view]

2 Grundlagen

Dieses Kapitel dient zur Erklärung der Hintergrundvariablen. Es handelt sich um Begriffe und Themen, die benutzt werden und nicht selbsterklärend sind.

2.1 Medizinische Grundlagen

2.1.1 Begriffe

Es wurde nicht direkt mit medizinischen Themen gearbeitet, trotzdem wird das medizinische Vokabular beim Erklären der Verhältnisse am Knochen verwendet.

In der Medizin wird ein eigenes Koordinatensystem verwendet, in dessen Mittelpunkt sich der menschliche Körper befindet. In diesem System sind für die Freiheitsgrade neue Begriffe eingeführt worden. So bezeichnet medial auf der links-rechts-Achse die körperzugewandte Seite, lateral körperabgewandt. Ähnlich verhält es sich mit den Bezeichnungen dorsal und ventral. Dorsal bedeutet dabei die Rückseite und ventral die Bauchseite betreffend. Distal und proximal sind analog zu den vorherigen Begriffen die Bezeichnungen für die Richtung von der Körpermitte weg und zur Körpermitte hin. Anterior und posterior sind keine Richtungen sondern eine Ortsbezeichnung für vordere und hintere Objekte. Desweiteren bezeichnet der lateinische Ausdruck Femur den Oberschenkelknochen und Tibia das Schienbein [2].

2.1.2 Mechanische Beinachse

Das Knie mit seinen angrenzenden Organen Femur und Tibia muss vorgestellt werden. Die Abbildung 1 zeigt die wichtigsten Bestandteile des menschlichen Knies und hilft, einen Überblick zu erhalten.



Abb. 1: Schematische Darstellung eines Knies mit Erklärungen [1]

Das Ziel des Kneenavigator ist die Berechnung Einhaltung der mechanischen Beinachse. Mit ihrer Hilfe können die unterstützenden Maßnahmen für den Operateur berechnet werden.

Eine korrekte Ausrichtung des Beines ist Voraussetzung für eine optimale Verteilung der Kräfte, die bei jeder Bewegung auf das Knie wirken.

Ein wichtiger Wert ist hierbei die mechanische Achse (auch Traglinie, TL). Abbildung 13 zeigt ein Bein in schematischer Form mit Angabe der Achse TL und den wichtigen Winkeln.

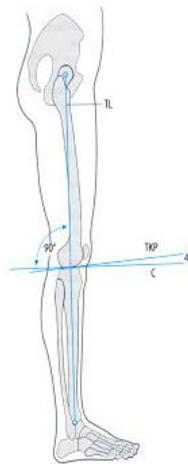


Abb. 2: Darstellung der relevanten Achsen im Bein [2]

Die mechanische Achse verbindet das Zentrum des Hüftkopfes mit dem Mittelpunkt des Sprunggelenkes.

Mit der Auflagefläche des Fußes bildet die mechanische Achse einen Winkel. Dieser Winkel spezifiziert Fehlstellungen und Erkrankungsbilder, wenn er außerhalb seiner normalen Werte von 87-90 Grad liegt.

Zusammen mit der Achse der Tibia kann ein Durchschnittswinkel errechnet werden. Dieser liegt im Bereich von 178-188 Grad. Ist er größer, wird der Zustand als X-Bein (*genu valgum*) bezeichnet, ist er kleiner als O-Bein (*genu varum*). Aus diesen Werten wird ersichtlich wie wichtig eine korrekte Ausrichtung des Beins ist. Speziell für die eingebaute Prothese müssen die Werte exakt sein, da sich dadurch in erheblichem Maße der Gesundheitszustand und die Beweglichkeit des Patienten, das Schmerzvolumen und die Dauerhaftigkeit der Prothese bestimmt.

2.2 Das Tracking-System

Für die Eingabe von Daten wurde eine Stereo-Kamera von NDI (Polaris) [6] verwendet. Sie gilt als weitentwickelteste Kamera für das optische Tracking.

Eine Stereo-Kamera nimmt analog zum menschlichen Auge zwei Bilder von einer Szene auf, um damit Tiefeninformationen zu erhalten. Die im Raum platzierten aktiven und passiven Marker erkennt sie mit einem Sichtfeld von 70×30 Grad. In Abbildung 3 ist die Kamera mit ihren Utensilien dargestellt.



Abb. 3: Wichtige Bestandteile des Trackingsystems; links: Kamerakopf, mitte: Zubehör, rechts: Marker [<http://www.ndigital.com/polaris.php>]

Die ihr zugehörigen passiven Pointer enthalten je drei in einem Dreieck zueinander platzierte Marker, wodurch die Kamera alle 6 Freiheitsgrade erkennt und die Position im Raum berechnen kann.

Am anderen Ende der Pointer befindet sich ein Metallstift, mit welchem die Oberflächen abgefahren oder einzelne Messpunkte angetastet werden können.

Dieses System zeichnet sich durch Berechnungen in Echtzeit und eine hohe Präzision aus. Weiterhin ist die Einfachheit der Anwendung von großem Vorteil.

Für die Anbindung an unser System war lediglich das Setzen der Verbindung zum Computer über die Schnittstelle COM1 und das Implementieren einer Klasse zur Steuerung der Kamera notwendig.

Zunächst muss die Geräteart abgefragt werden, um den geeigneten Treiber laden zu können. Hier ist die Anbindung weiterer Kamerasysteme möglich:

```
if (TRACKING_DEVICE == POLARIS) {
    trackingDriver = new LocPolarisTrackingDriver(COM_PORT,
        new TrackingErrorHandler(owner));
    ((LocPolarisTrackingDriver)
        trackingDriver).setRomfile("OHST-Pointer.rom", POINTER_INDEX);
} else {
    trackingDriver = new LocFOBTrackingDriver(COM_PORT, new
        TrackingErrorHandler(owner));
}
```

Danach werden die Schnittstellen initiiert:

```
try {
    trackingDriver.initConnection();
    trackingDriver.startMeasurement();
}
catch (LocTrackingException e) {
    e.printStackTrace();
}

timer = new Timer(100, this);
timer.start();

if (trackingDriver == null) return;
```

Die `actionPerformed` wird bei jedem Event aufgerufen und prüft die Sichtbarkeit der Marker. Da dies hier durch die Verwendung des Timers ein kontinuierliches Aufrufen bedeutet, wurde die Datenabfrage in eine separate Methode geschrieben, die nur bei einer Messung aufgerufen wird.

```
public LocLocatorData getLocatorData() {
    trackingDriver.update();
    return trackingDriver.getLocatorData(POINTER_INDEX);
}
```

In der `actionPerformed` wird nach der Prüfung des Kamerastatus der Messpunkt aus der zurückgegebenen Matrix ausgelesen und weitergeleitet.

```
LocLocatorData data = getLocatorData();
if(data.status==LocLocatorData.VALID) {
    Point3d reflectionPoint =new Point3d(data.locatorMatrix.get(0, 3),
    data.locatorMatrix.get(1, 3),
    data.locatorMatrix.get(2, 3));
}
```

Auf Grund dessen, dass nicht kontinuierlich mit der Kamera gearbeitet werden konnte, da weitere Mitarbeiter an Projekten mit diesem Trackingsystem arbeiteten, wurde eine Klasse erstellt, die das Vorhandensein der Kamera simuliert. Die einzulesenden Werte waren in dieser Zeitspanne Daten von vorherigen realen Aufnahmen mit der Kamera.

Dadurch verringerte sich das Umschreiben des Programmiercodes auf eine Zeile, genauer: der Aufruf der Klasse `TrackingServer` oder `TrackingServerDummi`.

2.3 KneeNavigator

Der KneeNavigator dient der Steigerung der Genauigkeit der Schnitte bei einem vollständigen Austauschen des Kniegelenks.

Die Hauptfunktion der Software ist die Berechnung der mechanischen Achse des Beines und deren Abbildung auf die Position und Orientierung der Instrumente zur Führung des Benutzers. Die Besonderheit der Vorgehensweise ist die Unabhängigkeit von schädigenden, zeitaufwendigen und teuren CT-Aufnahmen während und vor einer Operation. Das Abfahren der interessanten Regionen am Knochen während der Operation führt zu keiner Strahlenbelastung oder weiteren Eingriffen am Körper, zudem konnten mit Testläufen keine gravierenden Zeitverschiebungen zu herkömmlichen Operationen festgestellt werden. Die Durchführung einer analogen Operation ist gekennzeichnet durch präoperative Winkelberechnungen, intraoperativer Kennzeichnung der Schnitte mit Winkelschablonen und dem starken Abhängigkeitsverhältnis zur Routine und Fähigkeit des Operateurs[2].

Der KneeNavigator sorgt für eine auf 0.1 Grad präzise Ausrichtung der Instrumente, welche er durch die Berechnung der mechanischen Achse und die Positionierung anhand der eingelesenen anatomischen Landmarken und wichtigen Flächen erreicht.

Durch diese Eigenschaften entfällt die Operationsplanung vollständig. Die Ermittlung der mechanischen Achse erfolgt darüber hinaus nicht mit aufwendigen bildgebenden Verfahren sondern über eine kinematische Erfassung der individuellen Patientenanatomie während des Eingriffs. Dem Operateur wird beim Einsatz der Software keine Entscheidung abgenommen oder vorenthalten. Die Anzeigen stellen einen Vorschlag oder eine Kontrolle dar, über deren Umsetzung der Arzt frei entscheidet. Die Abbildungen zeigen Momentaufnahmen der Software. In der Abbildung 4, links ist ein Zwischenbild dargestellt. Es weist auf den kommenden Abschnitt der Operation. In den Abbildungen 4, rechts und 5 zeigen einige typische Interaktionsfenster, welche alle jeweils wichtigen Informationen und Ansichten darstellen. Die Bilder dienen einer besseren Vorstellung der Umsetzung der Software.

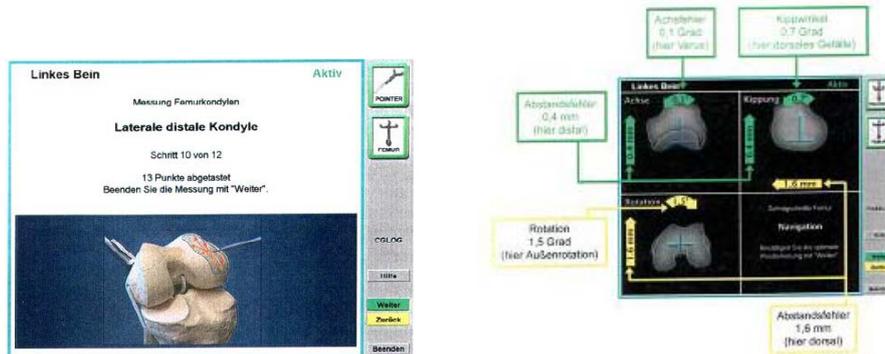


Abb. 4: Ausschnitte aus dem KneeNavigator

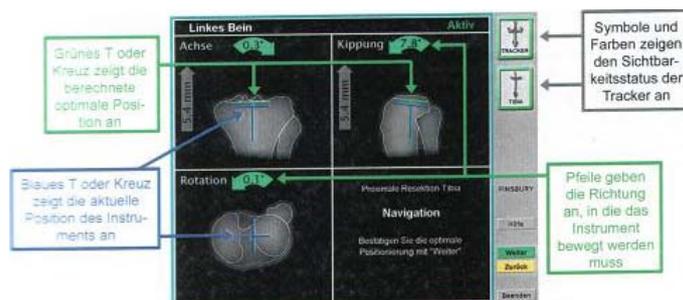


Abb. 5: Ausschnitt aus dem KneeNavigator

2.4 Landmarken

Landmarken (*fiducials*) sind markante Punkte eines Objektes, die schnell aufgefunden werden können.

Sie können automatisch, semiautomatisch oder manuell in einem Objekt markiert oder gesetzt werden. Informationen hierzu sind in [7], [8], [9] zu finden. Wenn man zur Veranschaulichung einen Würfel nimmt, so wären die naheliegenden markanten Punkte alle acht Ecken in nummerierter Form. Jedoch sind auch andere (z.B. Mittelpunkt) je nach Verwendung anwendbar.

An ihnen lässt sich die Größe und Lage des Würfels nach einer Veränderung nachvollziehen. Mit Hilfe der Landmarken werden verschiedenste Problematiken in der Computergrafik gelöst. Eine Anwendung ist das schnelle Manövrieren durch virtuelle Welten.

In unserem speziellen Fall haben die Marken eine weitere Funktion. Sie stellen nicht nur bestimmte Punkte im Objekt dar, mit denen wir in der größeren Form den Knochen transformieren wollen, sie sind zusätzlich anatomische Landmarken, was bedeutet, dass sie einen Knochen aus anatomischer Sicht charakterisieren und zudem jedem Mitarbeiter mit einer medizinischen Ausbildung, im speziellen der Operateur, geläufig sind.

Wir haben an unserem Ausgangsknochen 11 Landmarken von Herrn M. Bublat von der Firma LOCALITE bestimmen lassen, die vollständig den anatomischen, in der Medizin weitläufig bekannten Landmarken entsprechen. Er verfügt über die notwendige Erfahrung beim korrekten Positionieren der Punkte. Diese Marken sind in der Abbildung 6 dargestellt, wobei folgende Punkte zu erkennen sind:

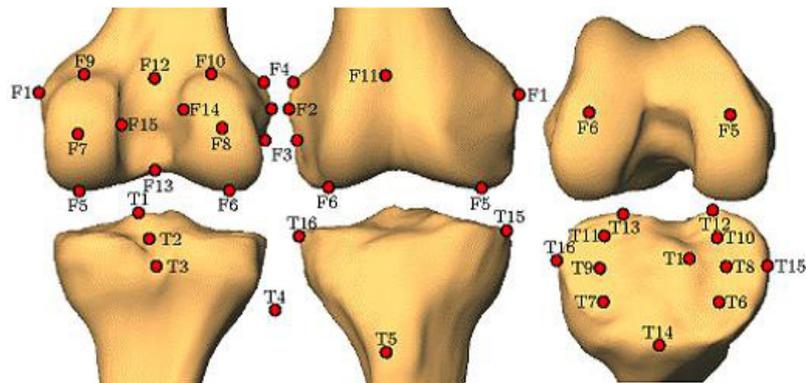


Abb. 6: Landmarken am Femur demonstriert [3]

1. Laterale Epikondyle (F1)
2. Mediale Epikondyle (F2)
3. Tiefster Punkt des Patella Gleitlagers (F13)
4. Fossa Intercondylica (F12)
5. Ansatzpunkt hinteres Kreuzband (F15)
6. Ventraler Kortikalispunkt (F11)
7. Laterale dorsale Kondyle (F7)
8. Mediale dorsale Kondyle (F8)
9. Laterale distale Kondyle (F5)
10. Mediale distale Kondyle (F6)

Die fehlende Landmarke (11) befindet sich im Femurkopf. Sie wird durch die mechanische Achse des Beins festgelegt und von der Navigations-Software berechnet.

Dieselben Punkte werden während der Operation vom Chirurgen mit einem kabellosen Pointer abgetastet, wobei die Position des Pointers kontinuierlich bestimmt wird und korrespondieren damit zu den Punkten im Ausgangsmodell.

Die Landmarken erfüllen in unserem Fall die Aufgabe der groben Registrierung von 11 Punkten des Knochenmodells auf die neuen 11 Datenwerte bei Beibehaltung der Oberflächenform. Diese Art der Registrierung wird im Kapitel 3.4 behandelt.

2.5 Registrierungstechniken

Jede Registrierungstechnik versucht eine Transformation zwischen korrespondierenden Bildpunkten unterschiedlicher Bilder zu finden. Neben der Unterscheidung der Techniken anhand der Dimensionen und deren Kombination untereinander, spielt auch die Art der Transformation und damit die Anzahl der Parameter und Freiheitsgrade eine Rolle. Hier sollen kurz die Möglichkeiten aufgezeigt werden, um die Auswahl der Lösung erklären zu können.

2.5.1 Rigide Transformation

Diese Transformationsart stellt die einfachste Anwendung dar. Zum Registrieren stehen sechs Freiheitsgrade zur Verfügung; je eine Translation und Rotation für jede Dimension. Diese Transformation ist nur für Objekte sinnvoll, die ihre Form erhalten und nur Lage und Position verändern, und das bedeutet, dass es nur bedingt für den menschlichen Körper sinnvoll ist. Knochennahe Organe werden auf diese Art registriert, jedoch versagt diese Methode bei sich verformenden Organen (Brain-Shift). Selbst mit Fixieren von Körperteilen kann der Fehler nicht behoben werden [10].

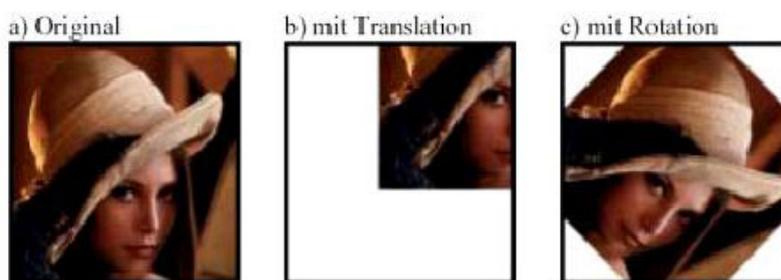


Abb. 7: Beispiele am Objekt Lena

2.5.2 Affine Transformation

Erweitert man die rigide Transformation um weitere zwei Freiheitsgrade, Skalierung (Größenänderung) und Scherung (seitliche Verzerrung), können auch Formveränderungen der Objekte registriert werden. Die Anzahl der Freiheitsgrade steigt damit auf zwölf an. Sowohl die rigide als auch die affine Transformation profitieren von nur einer Transformationsmatrix, die zur Berechnung ausreicht. Ein sich deformierender Körper kann hiermit besser behandelt werden, für die Eigenschaften des menschlichen Körpers reicht dies ebenfalls noch nicht aus [10].



Abb. 8: Beispiele am Objekt Lena

2.5.3 Nicht - lineare Transformation

Erweiterungen der affinen Transformation werden nach [10] als nicht - linear bezeichnet. Mit ihrer Hilfe wird es möglich auf Deformationen zu reagieren. Die Erweiterungen sind individuell, Beispiele sind: Dehnung, Verjüngung, Krümmung, Torsion oder kombinierte Transformationen.

Für unsere Aufgabe kommen also weder eine reine rigide noch eine reine affine Registrierung in Frage. Zur Berechnung einer nichtlinearen Transformation sind weitere Parameter zu beachten. Die Registrierung kann anhand von extrinsischen Bildmerkmalen bestimmt werden. Extrinsische Merkmale sind Bildobjekte, die nicht zum Patienten gehören und können entweder invasiv oder nicht-invasiv am Körper oder Organ befestigt werden, wobei die invasiven genauere Ergebnisse liefern. Neben den extrinsischen Merkmalen gibt es noch eine weitere Gruppe, die intrinsischen Merkmale. Sie sind auf die Daten, die der Körper liefert angewiesen. Werkzeuge können landmarken-, segmentierungs- oder voxelbasiert sein.

Landmarken benutzen anatomische, auffällige und leicht wiederfindbare Merkmale des Organs und können ohne Eingriffe am Organ im Bild markiert werden. Die häufigste Anwendung finden Landmarken bei nicht-deformierbaren Objekten, da dort mit rigidem und affinen Transformationen sehr gute Ergebnisse erzielt worden sind. Obwohl wir ein verformbares Objekt benutzen, wollen wir auf Grund der vielen positiven Eigenschaften der Landmarken für Patienten die landmarkenbasierte Transformation verwenden.

Segmentierungsbasierte Transformationen benutzen nur die Daten aus den Bildern und ihre geometrischen Informationen, um sie möglichst nah zueinander abzubilden.

Voxelbasierte Transformationen können sehr gut automatisiert werden. Sie verwenden statt Pixel oder Bildpunkte Voxel und rechnen mit deren Eigenschaften. Weiterführende Erläuterungen dazu in [10].

Die in unserer Lösung verwendete Registrierungsart ist eine Kombination aus affiner und nichtlinearer, intrinsischer Transformation. Dadurch entfällt die reine

Vorregistrierung (z.B. mit dem ICP-Algorithmus [11]) zur Angleichung der Koordinatensysteme und der groben Ausrichtung des Objektes vollständig und braucht nicht beachtet zu werden.

2.6 Die Applikation ModelViewer

Zur Durchführung der Studienarbeit wurde als Programmiersprache Java in der Version 1.4.2 und die Erweiterung Java3D verwendet. Für die verwendeten VRML-Dateien wird der VRML-Loader Vrml97 genutzt.

Ausgangspunkt für die Programmierarbeit bildete das Java-Programm Modelviewer. Dieses Tool dient dazu, VRML-Dateien zu laden und sie in einem Fenster darzustellen.

Um die Anforderungen an das Thema der Studienarbeit zu erfüllen, fand eine Erweiterung des Programmcodes um neue Methoden und Klassen statt. Dadurch entstand ein neues Tool, mit dem die zu erledigenden Aufgaben wie zum Beispiel die Darstellung von zwei Objekten oder die Aufnahme von Trackingdaten möglich sind. In der Abbildung 9 ist ein Klassendiagramm dargestellt, welches ein Überblick über das gesamte Projekt gibt. In Abbildung 10 sind die wichtigsten Klassen und ihre Methoden abgebildet.

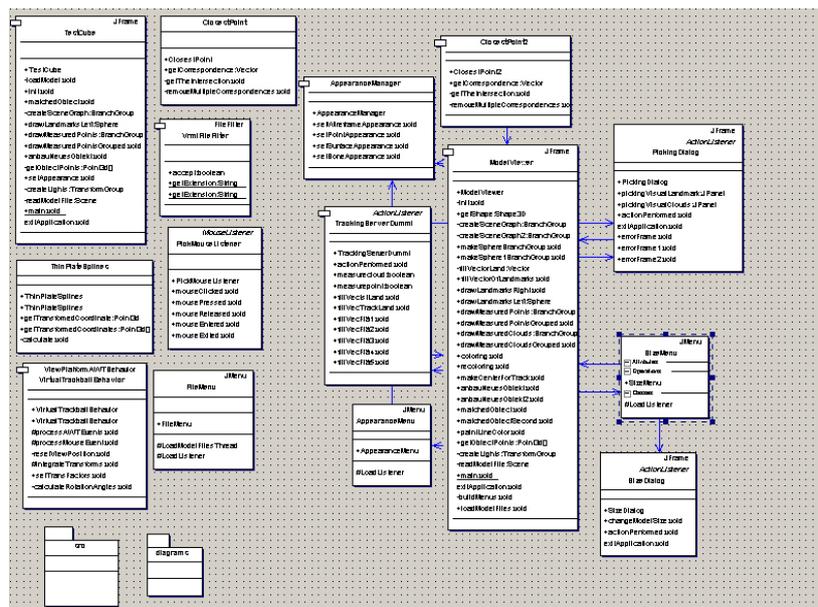


Abb. 9: Komplettes Klassendiagramm der Applikation

Als zentrale Klasse bleibt *ModelViewer* erhalten. In ihr befinden sich Methoden zum Laden der Dateien, zum Erstellen der Szenegraphen und neuer Objekte und zur Regelung der Abläufe im Programm.

Die Klasse *FileMenu* steuert den Menüeintrag „File“ und dient auch weiterhin dazu, eine Datei zu laden bzw. das Programm zu beenden.

Mit den Klassen *SizeMenu*, *SizeDialog*, *AppearanceMenu* und *AppearanceManager* wurden zwei weitere Menüpunkte in der oberen Programmzeile hinzu-

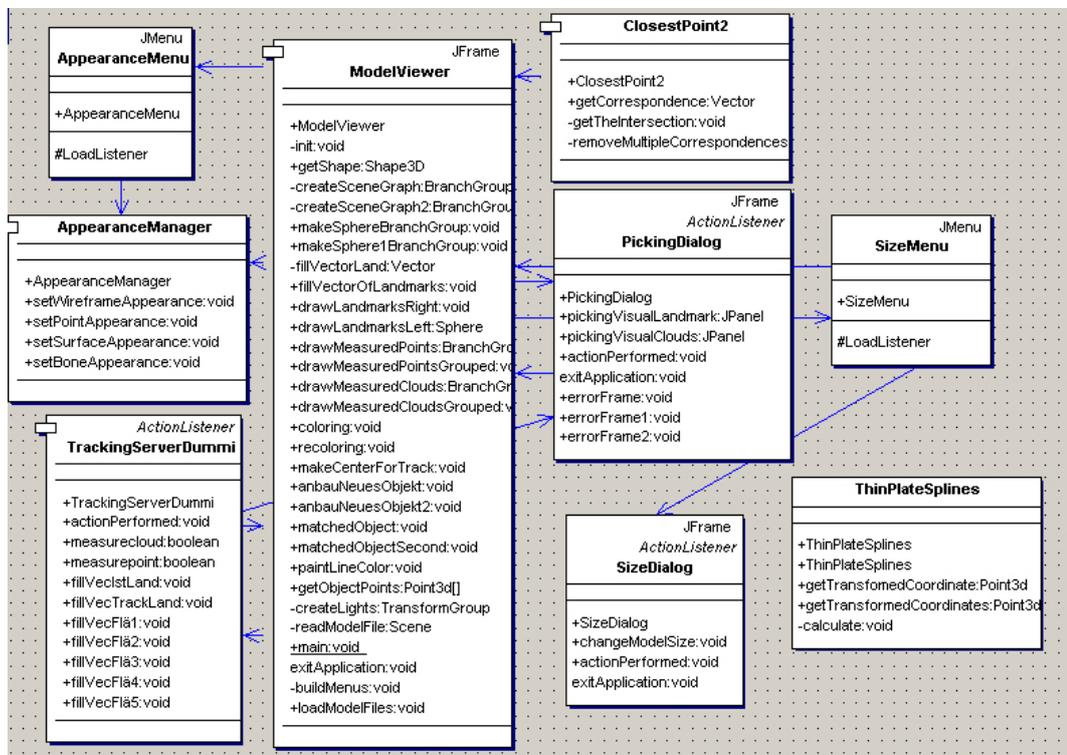


Abb. 10: Klassendiagramm der wichtigsten Klassen

gefügt. Über das Menü „Size“ wird ein neues Dialogfenster geöffnet. In diesem ist es möglich, an den 3D-Objekten der Szenerie über die drei Raumachsen einzelne oder einheitliche Skalierungen vorzunehmen. Diese erfolgen über die Änderung einer Transformationsgruppe in den jeweiligen Szenegraphen. Im Appearance-Menü kann die Darstellungsart ausgewählt werden. So können die Objekte als gefüllte Polygone, Polygongitter oder als reine Punktdarstellung betrachtet werden. In *AppearanceManager* sind außerdem die verschiedenen Appearances der möglichen Objekte und im Speziellen der mit unterschiedlichen Aufgaben versehenen Spheres zusammengefasst.

VirtualTrackBallBehavior legt die Art und Weise fest, wie sich die Objekte mit Hilfe der Maus in den Darstellungsfenstern rotieren, verschieben und zoomen lassen. Sie wird in den *ModelViewer* als Mousebehavior übergeben.

Da es Teil der Studienarbeit ist, ein Modell anhand von Landmarken zu deformieren, musste es ermöglicht werden, an einem 3D-Modell diese Landmarken festzulegen und deren Objektkoordinaten auszulesen. Dazu wurde das in Java3D vorhandene Picking verwendet. Diese Aufgabe wird von der Klasse *PickMouseListener* erfüllt. Bei einem Mouseevent auf ein Objekt im linken Darstellungsfens-

ter werden die x- und y-Koordinaten erfasst und in die entsprechende Position im Raum des Canvas umgerechnet. Von diesem Punkt aus verläuft ein Sichtstrahl in den Raum und sammelt alle Schnittpunkte. Von diesen wird der zum Startpunkt nächste ausgegeben. Außerdem wird zur Markierung eine Kugel an dieser Stelle platziert. Dazu wird eine Sphere mittels Translation um die Koordinaten des ermittelten Schnittpunktes verschoben und dem Szenegraphen hinzugefügt.

Für die Studienarbeit findet ein zuvor festgelegtes 3D-Modell eines Oberschenkelknochens Verwendung. Explizit für dieses Modell wurden dann die Positionen der anatomischen Landmarken mittels des zuvor beschriebenen Pickings ermittelt und die Koordinaten ausgelesen. Das Setzen der Landmarken wurde von einem Experten (siehe 2.4) übernommen, um eine möglichst hohe Präzision zu erhalten. Anschließend wurden die so ermittelten Daten in einem Vektor zusammengefasst.

Das Kernstück der neuen Arbeitsumgebung ist der Dialog zur Aufnahme von Trackingdaten im rechten unteren Bereich des Programmfensters. Die Klasse *Picking-dialog* bietet zwei Dialogfenster an, die mittels Reiter ausgewählt werden können. Das erste Dialogfenster dient der Aufnahme der 11 anatomischen Landmarken, das zweite der Aufnahme der Oberflächenbereiche des Knochens. Im Grunde können alle gewünschten Oberflächenbereiche eingelesen werden, es wird aber von der Vorgaben des KneeNavigator ausgegangen. Das bedeutet, es werden fünf Flächen benötigt, vier für die Kondylen und eine für den ventralen Kortikalispunkt.

Um die Werte zu erhalten wird die Klasse *TrackingServer* benötigt. Sie setzt auf den bereits vorhanden Klassen von LOCALITE auf und bildet so das Verbindungsglied zwischen diesem Programm und der NDI Polaris Stereokamera (siehe Kapitel 2.2).

Beim Initialisieren von *TrackingServer* wird ein Timer gestartet, der in bestimmten Zeitintervallen (hier: 100 ms) die Funktion *actionPerformed()* aufrufen.

In dieser wird zuerst überprüft, ob die überlieferten Daten gültig sind. Ist dies der Fall, so werden die Statusfenster in beiden Dialogen grün mit dem Text „bereit“ dargestellt und damit dem Benutzer signalisiert, dass sich der Pointer im Sichtfeld der Kamera befindet und ein Einlesen möglich ist. Ist dies nicht der Fall, wenn der Tracker zum Beispiel verdeckt sein sollte, werden die beiden Statusfenster auf die Farbe rot und mit dem Text „blockiert“ gesetzt.

Wird im Dialog zu den Landmarken ein Button betätigt, um eine Landmarke einzulesen, so wird von *TrackingServer* die aktuelle Position des Trackers ausgelesen und in einer Variablen vom Typ *Point3D* gespeichert und an einen Vektor übergeben. Zusätzlich wird der gemessene Punkt zur visuellen Kontrolle in Form einer Kugel im rechten Fenster visualisiert. Mit dem Button „Zurücksetzen“ werden die bis zum jetzigen Zeitpunkt vorgenommenen Messungen gelöscht und die Aufnahme der Landmarken kann erneut vorgenommen werden. Der Button „ok“ erfüllt mehrere Funktionen. Zum einen wird der erste Schritt der Deformation mit Hilfe von Landmarken ausgeführt, zum anderen wird die Ausführung des zweiten Deformationsschrittes freigeschaltet.

Soll im zweiten Dialog nun eine Fläche aufgenommen werden, so wird der zuvor bereits erwähnte Timer verwendet. Durch das Betätigen einer der Aufnahme-Buttons wird ein Boolean gesetzt, so dass in der *actionPerformed()* neben der Gültigkeitsprüfung auch ein Messvorgang ausgeführt wird. Hier wird wie beim einzelnen Punkt die Position des Trackers ausgelesen, in einem Vektor gespeichert und mittels Kugel im rechten Fenster dargestellt. Die Rücksetz-Buttons dienen analog zum Dialog der Landmarken zum Löschen des Vektors und der Möglichkeit die Messung zu wiederholen. In diesem Dialog wird mit dem Button „ok“ das zweite Verfahren ohne Landmarken gestartet.

Die unerwähnten Klassen *ThinPlateSplines* und *ClosestPoint2* beinhalten die beiden verwendeten Verfahren und werden in den folgenden Abschnitten behandelt.

3 Lösung

3.1 Idee

Im Allgemeinen kann bei einem operativen Eingriff zum Einbau einer Vollprothese davon ausgegangen werden, dass starke Veränderungen am Kniegelenk vorzufinden sind. Sei es durch degenerative Ursachen, Entzündungen mit Gewebeumbildungen oder traumatische Einwirkungen. Diese Veränderungen sind die Ursache für eine Abnutzung des Gelenks. Aus diesem Grund sind viele Formen des Knochens möglich, allerdings wird nur selten ein Knochen in seiner typischen Form vorzufinden sein. Dieser Aspekt ist bei der Deformation zu berücksichtigen, da er das Maß der Veränderung bestimmt. Es muss so viel Veränderung zugelassen werden, wie ein zum Beispiel nur einseitig abgenutzter Knochen benötigt, aber die Form insofern weitestgehend erhalten, dass die noch intakte Seite des Knochens exakt dargestellt wird. Dies zeigt, dass eine reine Deformation in globaler Ebene nicht ausreichen kann, genauso wenig wie eine lokale Deformation das gewünschte Ergebnis erwarten lässt.

Dementsprechend müssen beide Deformationsarten miteinander gekoppelt werden, um ein ansprechendes Ergebnis zu erhalten. Mathematisch betrachtet suchen wir eine Funktion, die geformt von Kontrollpunkten, die gesuchte Krümmung möglichst genau bestimmen. Im Allgemeinen sind Splines ein bewährtes Mittel zur Lösung dieses Problems.

Recht einfach strukturierte Splines, wie sie gewöhnlich verwendet werden, bestehen aus mehreren Funktionsteilen, die Polynome niedrigen Grades beinhalten. Damit aber eine glatte Kurve entsteht darf sie keine Sprünge haben (Stetigkeit) und nicht abknicken (Differenzierbarkeit). In diesem Fall sollen folgende Randbedingungen gestellt sein. Sie soll durch alle Stützpunkte laufen. Es handelt sich demnach um eine Interpolationsfunktion. Außerdem muss für die Übergänge Stetigkeit für die erste und zweite Ableitung gelten. Splines, die diese Bedingung erfüllen, haben allerdings die Eigenschaft, dass ihr Verlauf zwischen den Stützstellen nicht beeinflussbar ist. Die Funktionen verlaufen oft mit extremen Schwingungen. Um diese Sprunghaftigkeit zu verhindern, wird eine Zusatzbedingung in Form einer Minimierungsbedingung eingeführt. Diese lässt sich durch die Minimierung eines Integrales über die Ableitungen der Splines formulieren [12].

In der Praxis und im Speziellen in medizinischen Anwendungen, wie der Registrierung von Gehirnbereichen, hat sich hier die Thin Plate Spline-Klasse bewährt [13], bei der die Minimalbedingung näherungsweise physikalisch als Biegeenergie interpretiert werden kann. Mithilfe dieser Funktion werden die neuen Objektpunkte errechnet. Die Thin Plate Splines erschienen in der Recherche am viel versprechendsten.

3.2 Thin Plate Splines

1917 entwickelte D'Arcy Thompsen eine neue Methode, um Oberflächen aufeinander abzubilden. Er legte ein Netz um seine zu verformenden Objekte und erreichte durch die Verzerrung dieser Transformationsnetze eine Deformation. 1983 veröffentlichte Terzopoulos als erster einen Artikel, in dem er Thin Plate Splines für die Lösung von Problemen der Computergrafik verwendete. Die Thin Plate Splines in unserer Anwendung sind eine Erweiterung seiner Methode um Landmarken und wurde 1989 von Fred L. Bookstein in das Medical Image Processing eingeführt[4].

Thin Plate Splines sind ein näherungsweise physikalisch basiertes 2D-Schema zur Interpolation. Sie stellen eine Generalisierung von natürlich kubischen Splines im 1D Raum dar und sind ein Mittel, um geometrisch unabhängige Oberflächen über eine verteilte Datenmenge zu berechnen [14].

Die Namensgebung beruht auf den Eigenschaften der erzeugten Oberfläche, da sie sich ähnlich wie eine dünne Metallplatte verhält. Das heißt, die Oberfläche wird gezwungen sich an ihren Stützstellen nicht zu transformieren und mittels der Minimierung des Energieterms werden die Eigenschaften der restlichen elastischen, globalen Transformationen festgelegt.

Die Idee ist das Anwenden der Transformation einer Landmarke auf ihre Nachbarschaft. Das Anwendungsfeld umfasst alle non-rigiden Deformationen, die als Energieminimierung umgesetzt werden können.

Nach [15] muss der folgende Energieterm minimiert werden, damit die Nähe erhalten bleibt:

$$E_f = \int \int_{R^2} (f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2) dx dy \quad (1)$$

Zusammen mit den Interpolationsbedingungen läßt sich nach [14] für den Spline die folgende Funktion beschreiben:

$$f(x, y) = a_1 + a_2x + a_3y + \sum_{i=1}^n w_i f_i(P) = a_1 + a_2x + a_3y + \sum_{i=1}^n w_i U(|P_i - (x, y)|) \quad (2)$$

1. mit $U(r) = r^2 \log r^2$ als Basisfunktion,
2. $a = [a_1 a_2 a_3]^t$ als affine Transformation
3. w als nichtlineare Transformation
4. p als Landmarken

Diese Funktion besteht aus einem affinen Anteil und einer Linearkombination aus sog. radialen Basisfunktionen [16]. Diese Funktionen zentrieren sich um die Sollpunkte und haben die Form:

$$f_i(P) = \|P_i - P\|^2 * \log \|P_i - P\| \quad (3)$$

was der Funktion

$$U(r) = -r^2 \log r^2, \quad r = \sqrt{x^2 + y^2} \quad (4)$$

in der Gleichung (2) entspricht.

Für die Minimierung muss desweiteren nach [17] gelten:

$$\sum_{i=1}^p w_i = \sum_{i=1}^p w_i x_i = \sum_{i=1}^p w_i y_i = 0 \quad (5)$$

wodurch sich ein lineares Gleichungssystem aufstellen läßt. Trotz des schwierigen mathematischen Hintergrundes und Ausgangsmaterials existieren einfache, linear zu implementierende Algorithmen, die genutzt werden können und damit wird direkt zur algebraischen Berechnung im nächsten Abschnitt übergeleitet.

3.3 Mathematischer Hintergrund

Thin Plate Splines haben eine elegante Algebra, die die idealisierte Krümmungs- bzw. Biegungsenergie einer dünnen Metallplatte beschreibt. Mit Hilfe der Algebra ist die lineare Berechnung der TPS Parameter a und w , die zur Berechnung der Funktion notwendig sind, möglich. Zur Übersicht sind die folgenden Erklärungen in 5 Schritte unterteilt:

1. Algebraischer Ansatz
2. Hinführung zum linearen Gleichungssystem
3. Berechnen der Werte für das Gleichungssystem
4. Einsetzen der Parameter in TPS Gleichung
5. Erweiterung auf die dritte Dimension

Algebraischer Ansatz

Das TPS Modell ist gegeben durch:

$$f(x,y) = a_1 + a_2x + a_3y + \sum_{i=1}^n w_i U(|P_i - (x,y)|) \quad (6)$$

mit

- $U(r) = r^2 \log r^2$ als Basisfunktion,
- $a = [a_1 a_2 a_3]^t$ als affine Transformation
- w als non-lineare Transformation
- p als Landmarken

und ergibt in Matrixform folgende Funktion:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = A \begin{bmatrix} 1 & x & y \end{bmatrix} + \sum_{i=1}^n w_i U(r) \quad (7)$$

Hinführung zum linearen Gleichungssystem

Trennt man die x und y Koordinaten erhält die beiden Gleichungen:

$$f_x(x,y) = a_0 + a_1x + a_2y + \frac{1}{2} \sum_i \lambda_i r_i^2 \log r_i^2 \quad (8)$$

$$f_y(x,y) = b_0 + b_1y + b_2x + \frac{1}{2} \sum_i \mu_i r_i^2 \log r_i^2 \quad (9)$$

mit $r_i^2 = (x - x_i)^2 + (y - y_i)^2$

Die Interpolationsbedingung (2) und die Bedingung (5) führen zu dem Gleichungssystem $L * WA = Y$ mit dem die Parameter $a_0, a_1, a_2, b_0, b_1, b_2, \lambda_i$ und μ_i berechnet werden können und das wie folgt aussieht:

$$\begin{bmatrix} 0 & U(r_{12}) & \dots & U(r_{1n}) & 1 & x_1 & y_1 \\ U(r_{21}) & 0 & \dots & U(r_{2n}) & 1 & x_2 & y_2 \\ \dots & \dots & \dots & \dots & 1 & \dots & \dots \\ U(r_{n1}) & U(r_{n2}) & \dots & 0 & 1 & x_n & y_n \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 \\ x_1 & x_2 & \dots & x_n & 0 & 0 & 0 \\ y_1 & y_2 & \dots & y_n & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{\lambda_1}{2} & \frac{\mu_1}{2} \\ \frac{\lambda_2}{2} & \frac{\mu_2}{2} \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \frac{\lambda_n}{2} & \frac{\mu_n}{2} \\ a_0 & b_0 \\ a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} = \begin{bmatrix} \zeta_1^x & \zeta_1^y \\ \zeta_2^x & \zeta_2^y \\ \cdot & \cdot \\ \cdot & \cdot \\ \zeta_n^x & \zeta_n^y \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (10)$$

mit

- $r_{ij} = r_{ji}$
- $\zeta_i = x_i - x'_i$ für f_x und $\zeta_i = y_i - y'_i$ für f_y

Die letzten drei Reihen von L garantieren, dass die Koeffizienten w in der Summe null sind und ihre Kreuzprodukte mit den y und x Koordinaten von P_i null. Dabei müssen die Landmarken x_i, y_i und x'_i, y'_i sich entsprechen.

Berechnen der Werte für das Gleichungssystem

Wie die Matrizen zusammengesetzt sind, wird im folgenden beschrieben. Die große Matrix auf der linken Seite des LGS wird aus vier Matrizen in der Form

$$\begin{bmatrix} K & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} w \\ a \end{bmatrix} = \begin{bmatrix} v \\ 0 \end{bmatrix} \quad (11)$$

zusammengesetzt [4]. Dabei ist K definiert durch die Distanzen zwischen den Landmarken:

$$K = \begin{bmatrix} 0 & U(r_{12}) & \dots & U(r_{1n}) \\ U(r_{21}) & 0 & \dots & U(r_{2n}) \\ \dots & \dots & \dots & \dots \\ U(r_{n1}) & U(r_{n2}) & \dots & 0 \end{bmatrix} \quad (12)$$

mit K als $n \times n$ Matrix. P wird mit den Koordinaten der Landmarken gefüllt:

$$P = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & \dots & \dots \\ 1 & x_n & y_n \end{bmatrix} \quad (13)$$

mit P als $n \times (p + 1)$ Matrix. Zur Lösung des Gleichungssystems werden die Matrizen umgestellt:

$$L^{-1}Y = (W|a_1 a_x a_y)^T \quad (14)$$

Die zweite Matrix der Multiplikation ist gegeben durch:

$$Y = (V|000)^T \quad (15)$$

mit Y als $(n + p + 1) \times p$ Matrix.

Der Lösungsvektor WA des Gleichungssystems enthält in den letzten drei Zeilen die Parameter a, a_1, a_2 . Die restlichen Werte entsprechen den Parametern $w_1 \dots w_n$. Damit sind alle benötigten Werte errechnet worden. Die Vektoren A und w basieren also auf den Beziehungen der Landmarken beider Bilder.

Einsetzen der Parameter in TPS Gleichung

Die mit diesem Gleichungssystem errechneten Parameter a und w können direkt für die Lösung der Gleichung

$$f(x, y) = a_1 + a_2 x + a_3 y + \sum_{i=1}^n w_i U(|P_i - (x, y)|) \quad (16)$$

eingesetzt werden.

Die Splines brechen die Funktion in einen linearen Teil (affine Transformation) und einen affin-freien Teil. Der affine Anteil repräsentiert das Verhalten gegen Infinity und die inverse Operation erhöht den Aufwand von $O(n^2)$ zu $O(n^3)$. Die gesamte Berechnung ist invariant gegenüber der Translation oder Rotation von Landmarken. Sie ist gegenüber der x, y Rotation invariant, weil die Biegeenergie ein Skalar ist. Sollten aber x', y' rotieren, bewegt sich die Spline in gleichem Winkel der Rotationsachse mit.

Erweiterung auf die dritte Dimension

Die Änderungen zur Anpassung der Algebra auf die dritte Dimension sind sehr gering [4]. Die Funktion $U(r)$ lautet nun $U(r) = -|r|$ statt $U(r) = r^2 \log r^2$. Wodurch sich auch die Formel (2) wie folgt ändert:

$$f_i(P) = (|P_i - P|) \quad (17)$$

Die TPS Gleichung ändert sich zu:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = A \begin{bmatrix} 1 & x & y & z \end{bmatrix} + \sum_{i=1}^n w_i U(r) \quad (18)$$

und die Y-Matrix wird erweitert zu:

$$Y = (V|0000)^T \quad (19)$$

so dass die Lösung des Gleichungssystems folgende Form annimmt:

$$L^{-1}Y = (W|a_1 a_x a_y a_z)^T \quad (20)$$

3.4 Implementation

Elastische Registrierung mit korrespondierenden Landmarken

Die Berechnungen die explizit zu den TPS gehörten haben wir in eine separate Klasse geschrieben. Durch die detaillierten Vorgaben der Algebra konnten wir direkt beginnen. Im Initialisierungsbereich sind die globalen Variablen aufgelistet.

```
private Vector landmarks;      //static (non-varying)landmarks
private Vector trackPoints;    //tracked (new) points

private Matrix aTranspose;    // transposed a matrix
private Matrix wTranspose;    // transposed w matrix
```

Das Füllen der Matrizen haben einfache for-schleifen in der Methode *calculate* übernommen. Die Daten aus den übergebenen Vektoren werden übergeben, ggf. erweitert und in die Matrix eingesetzt.

```
private void calculate() {
    int n = landmarks.size();    //number of landmarks
                                // fills the p-matrix
    Matrix p = new Matrix(n, 4); // P-matrix contains landmark points
    for (int i = 0; i < n; i++) {
        Point3d point = (Point3d) landmarks.elementAt(i);
        p.set(i, 0, 1.0d);
        p.set(i, 1, point.x);
        p.set(i, 2, point.y);
        p.set(i, 3, point.z);
    }

                                // Y-matrix contains tracked points
                                // fills the y-matrix
    Matrix y = new Matrix((n + 4), 3); // Y-matrix contains tracked points
    for (int i = 0; i < n; i++) {
        Point3d point = (Point3d) trackPoints.elementAt(i);
        y.set(i, 0, point.x);
        y.set(i, 1, point.y);
        y.set(i, 2, point.z);
    }

    //fills the k-matrix, contains the distances between all landmarks
    Matrix k = new Matrix(n, n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            Point3d point1 = (Point3d) landmarks.elementAt(i);
```

```

        Point3d point2 = (Point3d) landmarks.elementAt(j);
        k.set(i, j, -point1.distance(point2));
    }
}

```

Dank dem eingebetteten JamaPaket (<http://math.nist.gov/javanumerics/jama/>) können algebraische Befehle durch Methodenaufrufe abgearbeitet werden. Die folgende Zeile transponiert die Matrix p :

```
ptranse = p.transpose();
```

Nachdem alle benötigten Matrizen initialisiert und gefüllt worden sind, folgt die Eingliederung in die große L-Matrix, die Bestandteil des Gleichungssystems ist.

```

l.setMatrix(0, n - 1, 0, n - 1, k);
l.setMatrix(n, n + 3, 0, n - 1, p.transpose());
l.setMatrix(0, n - 1, n, n + 3, p);
l.setMatrix(n, n + 3, n, n + 3, nuller);

```

Dieses Lösungssystem kann mit nur einem Methodenaufruf gelöst und in WA gespeichert werden.

```
WA = l.solve(y);
```

Für die abschließende, eigentliche Berechnung der neuen Koordinaten müssen die wichtigen Daten aus der WA-Ergebnismatrix ausgelesen werden. Hier werden sie in den Matrizen a und w gespeichert.

```

// in the last 4 rows of WA find aTranspose
Matrix a = WA.getMatrix(n, n + 3, 0, 2);
aTranspose = a.transpose();

// from first to n.th row of WA find wTranspose
Matrix w = WA.getMatrix(0, n - 1, 0, 2);
wTranspose = w.transpose();

```

Die Hauptrechnung ist die doppelte for-Schleife in der Methode `getTransformedCoordinates` über die Punkte des Objekts und die Landmarken. Es wird zu jeder Landmarke der euklidische Abstand zur aktuellen Position errechnet und in der C-Matrix festgehalten. Jetzt fehlt nur noch das Definieren der aktuellen Position als `Point3d`. Daraufhin sind alle Daten vorhanden, die zur Berechnung der Gleichung (16)(siehe Kapitel 3.2) benötigt werden.

```

for (int j = 0; j < size; j++) { //size is the length
of the array of object points
    for (int i = 0; i < n; i++) { //n is number of landmarks
        c.set(i, 0, -point.distance((Point3d) landmarks.elementAt(i)));
    }

    //save information in matrix, with 1 added in first row
    Matrix pkt = new Matrix(4, 1);
    pkt.set(0, 0, 1);
    pkt.set(1, 0, point.x);
    pkt.set(2, 0, point.y);
    pkt.set(3, 0, point.z);
}

```

Noch innerhalb der äußeren for-schleife wird die Berechnung durchgeführt und das Ergebnis in das Array zurückgeschrieben, damit die Position des Punktes im Verhältnis zu den restlichen Objektdaten bestehen bleibt, da wir kein `IndexTriangularArray` verwendet haben.

```

Matrix neuPkt = aTranspose.times(pkt); // ATranspose*pkt in neupkt
Matrix sum = wTranspose.times(c); // wTranspose*c in sum
neuPkt.plusEquals(sum); //addition of both sides of equation formula

```

```

Point3d transformedPoint = new Point3d(neuPkt.get(0, 0),
neuPkt.get(1, 0), neuPkt.get(2, 0));

```

```

transformedData[j] = transformedPoint;
}

```

```

return transformedData;
}

```

```

//array with transformed coordinates

```

Ist die äußere Schleife abgearbeitet, sind für jeden Punkt des Objekts eine zusätzliche Summe von Schleifen, die der Anzahl der Landmarken entspricht durchlaufen worden. Die ursprüngliche Berechnung des euklidischen Abstandes mittels `MATH.pow()` wurde durch `x*x` ersetzt, da der Performanceunterschied enorm war. Die endgültige Version verwendet die Methode `distance(Point3d)` der Klasse `Point3d` von Java.

3.5 Implementation

Elastische Registrierung ohne korrespondierende Landmarken

Nach der Registrierung unter Verwendung korrespondierender Landmarken, mit der eine erste globale Deformation durchgeführt wurde, wird jetzt im zweiten Arbeitsschritt mittels einer lokalen Deformation eine Anpassung des Modells an die Messdaten im Detail erfolgen. Diese Daten sind hier keine anatomischen Landmarken, sondern die fünf Flächen der lateral dorsalen und distalen, der medial dorsalen und distalen Kondylen und des ventralen Kortikalispunktes, die während der Operation vermessen werden. Bei der Aufnahme der Punktwolke entstehen leicht Ungenauigkeiten. Ein interpolierendes Verfahren, wie die in den vorherigen Abschnitten verwendeten Thin Plate Splines, würde hier mit hoher Wahrscheinlichkeit zu einer rauhen Oberfläche führen und damit ein nicht befriedigendes Ergebnis liefern. Um eine gleichmäßige Oberfläche zu erhalten, soll deswegen ein approximierendes Verfahren angewendet werden.

Nötig ist dieses Verfahren, da die anatomischen Landmarken nicht genügend Informationen enthalten, um die Erscheinung des vermessenen Knochens ausreichend zu beschreiben. Solche Informationen, wie die generellen Ausmaße der Kondylen, sind in den abgetasteten Flächen enthalten und ermöglichen es, das 3D-Modell noch weiter zu deformieren, damit es dem echten Knochen des Patienten möglichst ähnlich scheint. Ein Beispiel hierfür ist in der Abbildung 11 zu sehen. Im linken Bild ist ein Knochen nur anhand der anatomische Landmarken deformiert, während im rechten Bild der Knochen auch an die gemessenen Flächen angepasst wird.

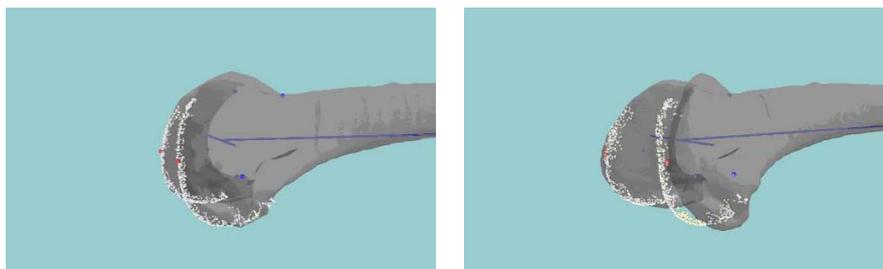


Abb. 11: Links: Nur anatomische Landmarken; Rechts: mit Flächen der Knodylen

Nach Rohr [13] ist es nun möglich, Thin Plate Splines nicht nur interpolierend zu verwenden, sondern auch mit approximierenden Eigenschaften zu versehen. Diese Möglichkeit machten wir uns zu nutze und verwendeten auch für die elastische Registrierung ohne Landmarken die Thin Plate Splines, diesmal jedoch um Punkte zu approximieren.

Da die Thin Plate Splines für ihre Berechnung auf das Vorhandensein von

Punktkorrespondenzen angewiesen sind, musste eine Lösung gefunden werden, um für die Messdaten entsprechend korrespondierende Punkte zu erhalten. Danach ist es dann möglich die lokale Deformation der Oberfläche mittels erneuem Einsatz der Thin Plate Splines durchzuführen.

3.5.1 Korrespondenzfindung

Zuerst musste geklärt werden, nach welchem Kriterium die Korrespondenz zwischen einem gemessenen Punkt und dem 3D-Modell des Knochens bestimmt wird. Entgegen anderer Möglichkeiten [18] fiel die Entscheidung auf den euklidischen Abstand. Somit ist der Punkt auf dem Modell das korrespondierende Gegenstück, der den kürzesten Abstand zum Ausgangspunkt besitzt.

Die Berechnung der Korrespondenzen zwischen den gemessenen Punktwolke und dem 3D-Modell wird von der Klasse `ClosestPoint2` vorgenommen. Dem Konstruktor wird dabei das Punkte-Array des Knochenmodells sowie eine Instanz des `ModelViewers` übergeben. Mit der Methode `getCorrespondence(Vector cloud)` wird ein Vektor aus Punktkoordinaten übergeben und die Berechnung der Korrespondenzen vorgenommen. Dabei wird für jeden Messpunkt die Methode `getTheIntersection(Point3d point)` aufgerufen:

```
Point3d merker = modell[0];    // model: array of points of the
bone
    double distance = point.distance(merker);
    double temp;
    for (int j = 1; j < modell.length; j++) {
        temp = point.distance(modell[j]);
        if (temp < distance) {
            merker = modell[j];
            distance = temp;
        }
    }
}
```

Dabei wird in einer Schleife über alle Punkte des Modells die Distanz berechnet. Die kürzeste Distanz wird in der Variablen `distance` der dazugehörige Punkt in der Variablen `merker` gespeichert.

Bevor der gefundene Punkt dem Ergebnis-Vektor `correspondences` hinzugefügt wird, erfolgt eine Überprüfung der Distanz:

```
if (distance <= threshold) { // related elements have same index
    distances[correspondences.size()] = distance;
    correspondences.addElement(merker);
}
```

```

    } else {
        correspondences.addElement(null);
    }

```

Ist der Punkt außerhalb der mit *threshold* festgelegten Entfernung, wird anstatt dem Punkt ein Null-Element dem Ergebnis-Vektor hinzugefügt. Durch diese Überprüfung kann verhindert werden, dass zu weit vom Objekt entfernte Punkte beim späteren Einsatz des Thin Plate Splines die Deformation nachteilig beeinflussen. Ausreißer werden somit eliminiert. Das Null-Element hat den Sinn, dass sich dadurch zwei korrespondierende Punkte im Ergebnis-Vektor und dem übergebenen Punkt-Vektor am selben Index befinden.

Bisher kann ein Punkt des Modells mehrmals das korrespondierende Gegenstück für verschiedene Punkte sein. Um eventuellen Problemen bei der Berechnung des Thin Plate Splines aus dem Weg zu gehen, wird eine Eliminierung von vielfachen Korrespondenzen vorgenommen:

```

private void removeMultipleCorrespondences() {
    for (int i = 0; i < correspondences.size(); i++) {
        for (int j = i + 1; j < correspondences.size(); j++) {

            Point3d p1 = (Point3d) (correspondences.elementAt(i));
            Point3d p2 = (Point3d) (correspondences.elementAt(j));

            if (p1 != null && p2 != null && p1.equals(p2)) {
                if (distances[j] >= distances[i]) {
                    instance.paintLineColor(points[j], p2, Color.green);
                    correspondences.set(j, null);
                } else {
                    instance.paintLineColor(points[i], p2, Color.green);
                    correspondences.set(i, null);
                    break;
                }
            }
        }
    }
}

```

Alle Elemente des Ergebnis-Vektors *correspondences* werden untereinander verglichen. Ist keines der Elemente null und sind sie identisch, werden die im Array *distances* gespeicherten Distanzen verglichen. Das Element mit der größeren Distanz wird durch ein Null-Element ersetzt und eine grüne Linie zwischen dem

gelöschten Punkt und seiner Korrespondenz gezeichnet. Dadurch wird bei vielfachen Punktkorrespondenzen diejenige beibehalten, welche die kürzeste Distanz aufweist.

Abschließend wird zwischen den finalen Korrespondenzen eine rote Linie gezeichnet:

```
for (int i = 0; i < points.length; i++) {
    Point3d corr = (Point3d) (correspondences.elementAt(i));

    if (corr != null) {
        instance.paintLineColor(points[i], corr, Color.red);
    }
}
```

Der Ergebnis-Vektor *correspondences* enthält jetzt für einen Punkt des Übergabe-Vektors am selben Index das korrespondierende Gegenstück, oder ein Null-Element, sofern die Distanz zwischen den Punkten zu groß war, um weiterhin beachtet zu werden.

Bevor der Ergebnis-Vektor und der Vektor der Messdaten an den approximierenden TPS übergeben werden, müssen zuvor noch die im Ergebnis-Vektor enthaltenen Null-Elemente und die dazu korrespondierenden Gegenstücke im anderen Vektor entfernt werden. Außerdem wird vor die beiden Vektoren der Vektor der getrackten Landmarken angehängt. Jedoch sollen diese Punkte nicht approximiert sondern interpoliert werden. Auf diese Weise wird verhindert, dass die Landmarken durch eine erneute Deformation ihre Position ändern.

3.5.2 Approximierender Thin Plate Spline

Wie in [13] beschrieben, können den TPS nicht nur interpolierende, sondern auch approximierende Eigenschaften verliehen werden.

Die entsprechende Literatur ([17], [19]) liefert hierzu eine Lösung, die der zuvor beschriebenen Lösung zu den Thin Plate Splines (3.2) sehr ähnlich ist.

Das dazugehörige Gleichungssystem zur Berechnung der beiden Matrizen a und w hat die folgende Form:

$$(K + \sigma)w + Pa = v \quad (21)$$

$$P^T w = 0 \quad (22)$$

σ ist hierbei eine Diagonalmatrix der Form:

$$\sigma = \begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_n \end{bmatrix} \quad (23)$$

Die Werte $\sigma \geq 0$ lassen sich als Standardabweichungen für nur näherungsweise bekannte Sollpunkte deuten. Ist der Wert 0, so wird die zugehörige Landmarke interpoliert. Umso größer der Wert von σ gewählt wird, desto schwächer wird dieser Punkt approximiert.

Um ein interpolierenden TPS in einen approximierenden TPS umzuformen, bedarf es also nur einer kleinen Änderung. Diese besteht darin, auf der Diagonalen der K-Matrix, die beim interpolierenden TPS 0 beträgt, σ_1 bis σ_n zu setzen.

```
//notzero: first index of diagonal not set to 0
for (int i = notzero; i < n; i++) { //n: size of the array
    k.set(i, i, value); //k: K-Matrix; value: sigma
}
```

Diese for-Schleife wurde in der Klasse ThinPlateSplines nach der normalen Berechnung der K-Matrix hinzugefügt und ermöglicht somit den Einsatz von approximierenden Thin Plate Splines wofür beim Methodenaufruf jetzt noch zusätzlich die Werte für notzero und value (σ) festgelegt werden können.

Von der Möglichkeit für σ verschiedene Werte zu verwenden, wurde bei dieser Anwendung bereits Gebrauch gemacht. Wie am Ende von 3.5.1 beschrieben, sollen zur Sicherheit die anatomischen Landmarken erneut interpoliert werden. Dazu werden in der K-Matrix die entsprechenden σ -Werte auf 0 gesetzt, während die anderen Werte ein Approximationsgewicht $\sigma > 0$ erhielten.

3.6 Visualisierung

Im Allgemeinen dient eine ansprechende und effektive Visualisierung dazu, dass die Daten und Informationen möglichst so aufbereitet werden, dass

1. der Benutzer möglichst viele (zusammengehörige) Daten auf einmal wahrnehmen kann, ohne die Übersicht zu verlieren
2. der Kontext der Daten erhalten bleibt, bzw. dargestellt wird
3. die Navigation „in“ den Daten möglichst einfach ist
4. eventuell neue Erkenntnisse und Zusammenhänge ersichtlich sind
5. keine lange Einarbeitungszeit in die Visualisierung notwendig ist

Nicht alle aufgeführten Aspekte sind hier relevant. Wichtig für eine möglichst realitätsnahe dreidimensionalen Darstellung sind unter anderem Verfahren zur Beleuchtung, Schattierung und Texturierung und der Einsatz von Grafikkbeschleunigung zu Realisierung in Echtzeit. Bis auf die Rechenzeit des Algorithmus war darauf aber kaum Einfluss möglich, da der VRML - Loader viele Funktionen übernommen hat. Die Rechenzeit beträgt im Moment bei Berechnung der maximalen Anzahl von Werten 60 Sekunden. Es wurde aber eine Beschränkung verwendet, die zu keinerlei optischen Beeinträchtigungen des Ergebnisses führt. Sollten mehr als 200 Punkte in den Punktwolke pro Fläche aufgenommen worden sein, wird die Anzahl der Punkte halbiert, die in die Berechnung eingehen. Die Halbierung der Messwerte bedeutet bei dem Algorithmus eine nahezu Halbierung der Laufzeit, das bedeutet, bei 150 Punkten pro Fläche ist eine Laufzeit von 30 Sekunden zu erwarten. Es werden aber Verbesserungsvorschläge im Kapitel 6 gemacht, um die Performance zu erhöhen.

Im Modelviewer ist der Knochen in hellgrau vor einem schwarzen Hintergrund dargestellt (dieser wurde für den Druck der Bilder auf hellgrün gesetzt). Durch die Beleuchtung aus zwei Richtungen zusätzlich zur Umgebungsbeleuchtung wurde Schatten erzeugt.

Das Hauptfenster ist in drei Bereiche aufgeteilt. Zwei Frames für den Original- und den Ergebnisknochen, sowie ein Reitermenü zur Eingabe der Daten. Für den Originalknochen sind die als Landmarken definierten Objektpunkte für den Benutzer relevant. Sie wurden als kleine Kugeln (rot) in das Objekt eingefügt, wodurch ihre Lage vergleichbar wird. Außerdem können dadurch bei der Eingabe der Daten diejenigen markiert werden (grün), die aktuell eingelesen werden sollen, da die Reihenfolge für die Berechnung Bedingung ist.

Für den Ergebnisknochen sind die gleichen Daten wichtig. Die aktuell eingelesenen Daten erscheinen umgehend auf dem Bildschirm. Hier kommen noch die Punktwolke hinzu, deren einzelne Punkte ebenfalls mit Kugeln (blau) dargestellt sind. Des weiteren wurden Darstellungsweisen implementiert, die über die normale Ansicht hinausgehen. Zusätzlich zur gefüllten Ansicht ist es möglich, das

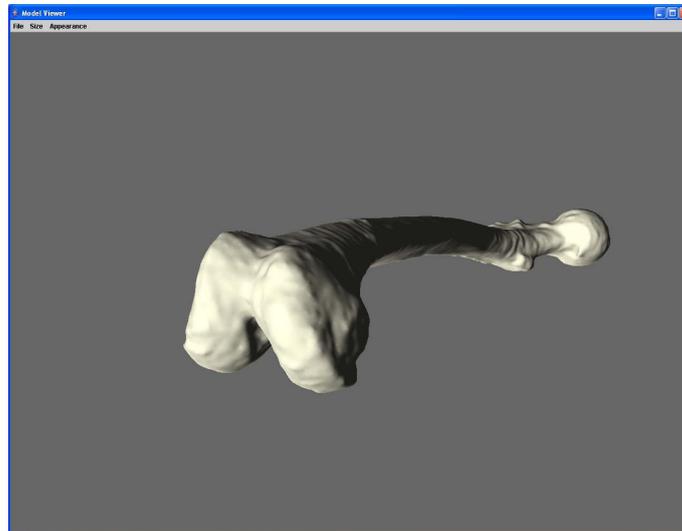


Abb. 12: Ansicht des unbearbeiteten Originalknochens im Modelviewer

Dreiecksnetz oder die reinen Objektpunkte darzustellen. Diese Funktionen werden über den Menüeintrag „Appearance“ aufgerufen. Weiterhin wurde die Appearance so gewählt, dass das Backface culling ausgeschaltet ist, so dass die Rückseiten der Dreiecke gezeichnet werden. Somit ist es möglich sich bei der späteren Approximation auch die innerhalb der Oberfläche liegenden Punkte anzuschauen. Im KneeNavigator dagegen sind einige Änderungen zu verzeichnen. Im Großen und Ganzen wurde die Visualisierung der Darstellungssituation von der Software übernommen. Die Änderungen betrafen also nur den Knochen. Dieser ist in einem dezenten grün-blau dargestellt, was an die hauptsächlich vorkommenden Farben in einem Operationssaal erinnert. Die vorhandene Animationsszene wurde beibehalten, weswegen eine Navigation in den Knochen zur Betrachtung der innenliegenden Punkte nicht zulässig war. Darum ist eine Appearance gesetzt, die transparent ist. Um die vorherige Lage der Punktwolke darstellen zu können, wurde ein blaues Achsenkreuz in die Szene eingefügt. Dabei entsprach die z-Achse der mechanischen Beinachse. Dieses Kreuz wurde ebenfalls beibehalten aber die Achse verlängert, so dass sie direkt im Femurkopfmittelpunkt endet. Dadurch ist eine genaue Interpretation des angezeigten Knochens und dessen angezeigten Daten möglich. Da der Knochen um das Achsenkreuz herumliegt, wäre es ohne die Transparenz nicht sichtbar. Damit der Knochen von allen Seiten betrachtet werden kann, ohne interagieren zu müssen, dreht sich der Knochen kontinuierlich um die einzelnen Achsen.

Die als Flächenwerte aufgenommen Punktwolke werden in einem hellen Grau dargestellt und fallen damit nicht besonders in Auge, was bei 1200 Datenwerten angenehm ist. Die Punkte dagegen, die aus den Schwellwerten herausgefiltert werden und damit nicht in die Berechnung eingehen, werden gelb dargestellt. Dadurch

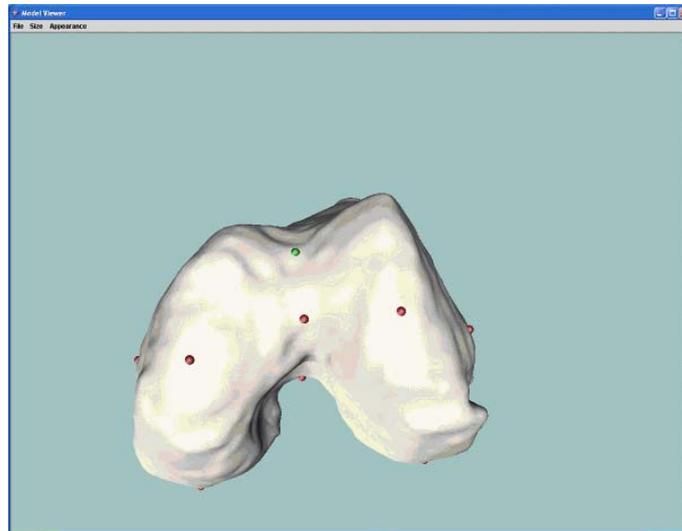


Abb. 13: Ansicht des Originalknochen im Modelviewer mit Landmarken

ist unmittelbar eine gut gemessene Wolke von einer schlechten zu unterscheiden. Ein weiteres Mittel zur Unterscheidung einer guten Messung und einer Wiederholungsbedürftigen ist die gesamte Berechnung. Wie in den Tests gezeigt, ist der auf Grund von schlechter Messung unnatürlich verformte Knochen intuitiv zu erkennen.

Damit wurden die relevantesten Daten mit den Farben rot, grün, blau und gelb dargestellt.

3.7 Integration in die Software KneeNavigator

Die nächste Aufgabe bestand darin, die Ergebnisse für den KneeNavigator von LOCALITE aufzuarbeiten und zu integrieren. Dazu soll in der Phase des Programmlaufes, in der die aufgenommenen Messdaten als Punktwolke zur Kontrolle der Messung dargestellt werden, ein mit dem von uns implementierten Verfahren deformierter Knochen hinzugefügt werden.

Dazu wurden von LOCALITE die für diesen Arbeitsschritt benötigten Klassen *LocScatterDisplayStep*, *LocScatterStep* und *LocFemurScatterDisplayStep* zur Verfügung gestellt, um benötigten Änderungen vorzunehmen. Die benutzten Klassen zur Durchführung der Integration waren *ThinPlateSplines* und *ClosestPoint2*. Zusätzlich wurde als Hauptklasse für unser Verfahren *LocFemurDeformation* hinzugefügt, in der der Ablauf der Arbeitsschritte organisiert wird. Sie wird in der Klasse *LocScatterDisplayStep* aufgerufen, welche dazu dient, die bei der Operation gemessenen Daten auf dem Bildschirm darzustellen.

Es wurde dabei den Ansatz verfolgt, ein möglichst modulares Paket zu integrieren, welches Daten erhält, mit diesen arbeitet und sein Ergebnis wieder ausgibt, ohne dass an den Klassen des KneeNavigators zu große Änderungen vorgenommen werden müssen.

Mittels mehrerer *get()*-Methoden der Klasse *LocFemurAnatomy* lassen sich die Koordinaten der gemessenen Punkte zurückgeben. Dies geschieht in *LocFemurScatterDisplayStep* und von dort werden diese Daten übergeben, um sie für die TPS nutzbar zu machen, über zwei Methoden an *LocFemurDeformation*.

Danach erfolgt der Aufruf der Methode *init()*. Hierbei wird als erstes der Knochen geladen. Um die lange Ladezeit der VRML-Datei zu umgehen wird in *LocFemurDeformation* stattdessen das Punkte-Array des 3D-Modells, welches als Java-Objekt gespeichert wurde, verwendet.

Danach wird der interpolierende TPS (3.2) mit den zum Modell gehörenden Landmarken und den soeben am Patienten gemessenen Landmarken berechnet und das Punkte-Array des Knochens aktualisiert.

Als nächstes wird, um die Geschwindigkeit zu erhöhen, bei sehr großen Punktemengen deren Anzahl reduziert, indem nur jeder zweite Punkt der gemessenen Wolken verwendet wird. Etwaige Qualitätsverluste werden im Kapitel 4 besprochen.

Mit diesen werden dann die Punktkorrespondenzen (3.5.1) für die Flächen gesucht, dabei werden zusätzlich Punkte, die auf Grund einer zu großen Distanz entfernt wurden, dem Array *notUsedCloudPoints* hinzugefügt.

Nachdem die Vektoren wie in unserer Applikation bearbeitet wurden (siehe Ende 3.5.1), wird der approximierende TPS (3.5) angewendet. Danach wird mit dem erneut deformierten Punkte-Array ein 3D-Objekt erstellt, welches einer *BranchGroup* hinzugefügt wird.

Eine weitere Änderung der KneeNavigator-Klassen bestand darin, *LocScatterDisplay*, welche für den Aufbau des Szenegraphen verantwortlich ist, um zwei Methoden zu erweitern. Der erste Methode *addObject(BranchGroup bg)* dient dazu, die BranchGroup des neu erstellten Knochens an die Szene anzuhängen. Die zweite Methode *addScatter2(LocMatrix4D[] scatter, Color color)* benutzt das Array *notUsedCloudPoints*, um somit die nicht verwendeten Punkte farblich hervorgehoben darzustellen.

Bis auf zwei neue Methoden in *LocScatterDisplay*, der Verwendung von *LocFemurDeformation* in *LocFemurScatterDisplayStep* und einiger zusätzlicher Aufrufe bereits vorhandener Methoden um die Messdaten zu erhalten, blieb der Sourcecode des KneeNavigators unverändert. Das Ziel einer möglichst modularen Integration haben wir dadurch erreicht und zeigen damit, dass eine weitere Anwendung in anderen Bereichen leicht zu realisieren ist.

4 Ergebnisse

Allgemeine Anwendungen

Bevor sich mit den Ergebnissen der vorliegenden Arbeit beschäftigt wird, soll ein kurzer Einblick in die in der Literatur vorgestellten Ergebnisse gegeben werden. In [20] werden die TPS als erfolgreich bei Lippen- und Gesichtsbewegungen nach [21][22] eingesetzt vorgestellt. Ein Bild aus [20] zeigt die Fähigkeit der TPS verschiedene Gesichter auf einander zu matchen:

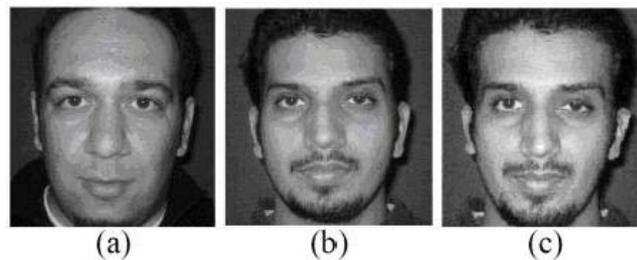


Abb. 14: Originalbilder (a,c) und Deformationsbild (b), TPS - Deformation mit 10 Landmarken

Der ausschlaggebende Artikel [4] für die Entscheidung zur Verwendung der Thin Plate Splines zeigt die Arbeitsergebnisse anhand von aufeinander registrierten Körpern. Es wurden unterschiedlich gebaute Menschen stehend und von stehend zu sitzend deformiert. Die vorgestellten Bilder sind in den Abbildungen 15 und 16 dargestellt.



Abb. 15: TPS - Deformationen nach [4]



Abb. 16: TPS - Deformationen nach [4]

Anwendung am KneeNavigator

Nach den Erläuterungen auf den vorherigen Seiten werden nun die Resultate präsentiert. Es wird im folgenden zunächst ein Vergleich zwischen der Ausgangssituation und dem Ergebnisbild gegeben. Damit kann der Unterschied direkt gesehen werden. Für weitere Interpretationsmöglichkeiten wurden Testfälle durchgeführt. Diese Situationen wurden in zwei Themengebiete aufgeteilt, zum einen das unterschiedliche Setzen der Freiheitsgrade im Algorithmus. Im zweiten Schritt sind Anwendungsfälle nachempfunden worden. Vier unterschiedliche Szenarien wurden dabei betrachtet. Die Abbildung 17 zeigt links die ursprüngliche Situation im KneeNavigator. Auf der rechten Seite wird das Resultat dargestellt.

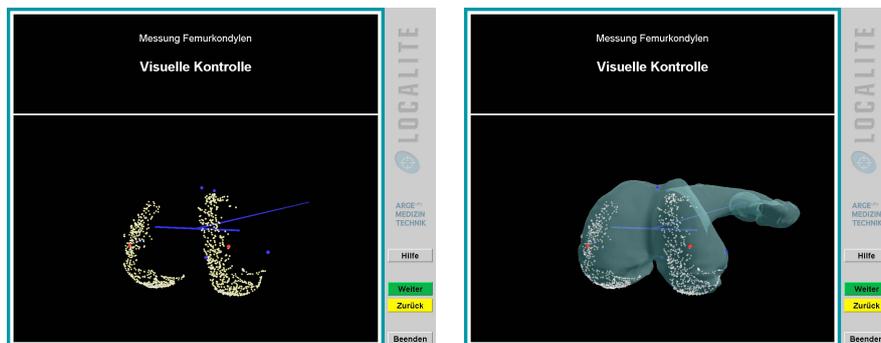


Abb. 17: KneeNavigator; Links: ursprüngliche Darstellung; Rechts: verbesserte Darstellung

4.1 Freiheitsgrade

Drei Parameter haben direkten Einfluss auf die Ergebnisse der elastischen Registrierung und der dafür benötigten Laufzeit:

- Distanz-Grenzwert bei Korrespondenzfindung
- Gewicht der Approximierung
- Anzahl der verwendeten Messdaten

In Versuchen wurden für diese drei Parameter die Werte ermittelt, die ein bestmögliches Zeit/Leistungs-Verhältnis lieferten. Während das Gewicht der Approximation nur Einfluss auf die Qualität des Ergebnisses hatte, waren die anderen beiden Parameter an der Laufzeit als auch an der Qualität beteiligt. Für diese Versuche wurde ein Testdatensatz verwendet, der zur Verfügung gestellt wurde.

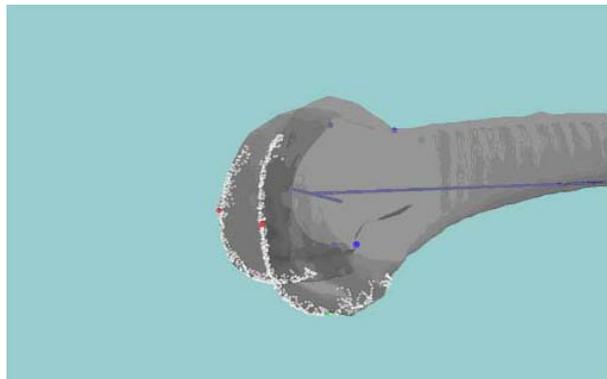


Abb. 18: Seitenansicht: Threshold=10mm

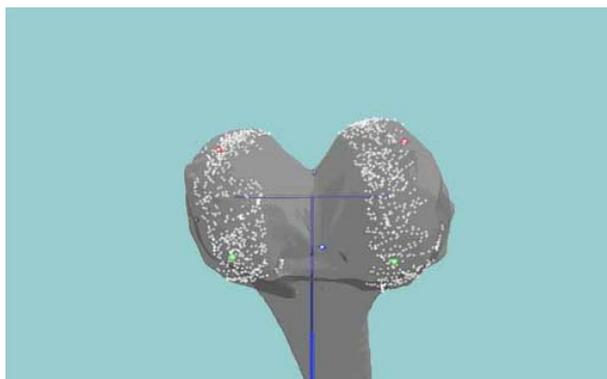


Abb. 19: Untenansicht: Threshold=10mm

Der erste betrachtete Parameter war der Distanz-Grenzwert für die Korrespondenzfindung. Dabei wurde die größtmögliche Anzahl von 1196 Punkten der Messdaten und eine gering gehaltene Approximation von 0.005 verwendet. Begonnen wurde mit großen Werten wie 100 mm, um dann den Threshold in 10er Schritten zu senken, bis 10 mm erreicht wurden. Bis dahin wurde bei der Korrespondenzfindung kein Punkt entfernt. In diesen Fällen werden sämtliche Punkte für den approximierenden TPS verwendet und ergeben somit ein komplettes Ergebnis, wie man in den Abbildungen 18 und 19 sehen kann.

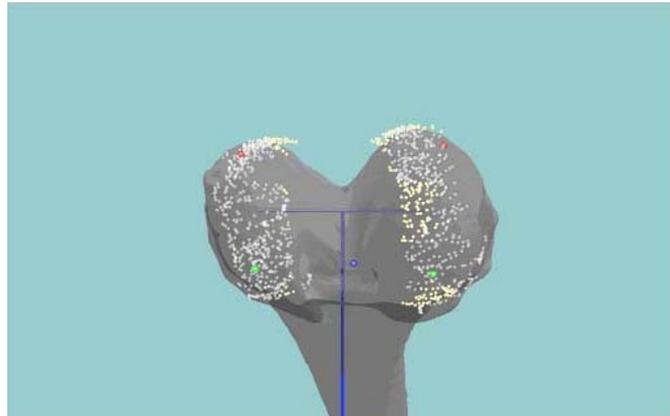


Abb. 20: Untenansicht: Threshold=5mm

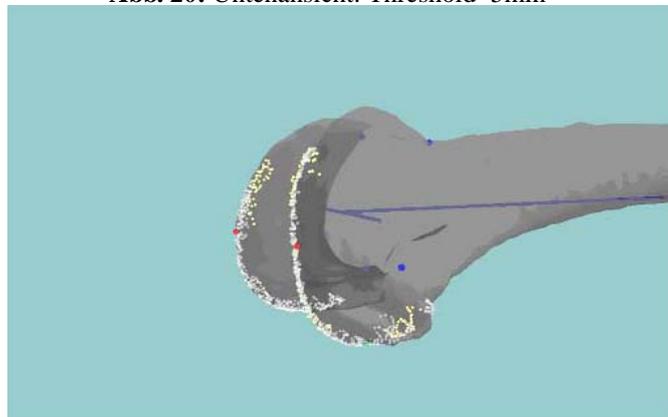


Abb. 21: Seitenansicht: Threshold=5mm

Bei einem Wert von 5 mm wurden bereits 252 Punkte entfernt, was einer Ausfallquote von über 20% entspricht. Bei einer derartigen Rate gehen schon entscheidende Oberflächenmerkmale verloren (Abbildungen 20 und 21). Senkt man diesen Wert weiter, zum Beispiel auf 1 mm werden sogar über 900 der knapp 1200 Punkte entfernt und eine sinnvolle Deformation ist praktisch kaum noch zu erkennen (Abbildungen 22 und 23). Folglich sollte, sofern ein Aussortieren an Punkten erwünscht ist, ein Wert zwischen 5 und 10 mm gewählt werden. Der Zeitgewinn beim Weglassen so vieler Punkte ist zwar nicht gering (ca 30%), jedoch kann ein derartiger Qualitätsverlust nicht in Kauf genommen werden.

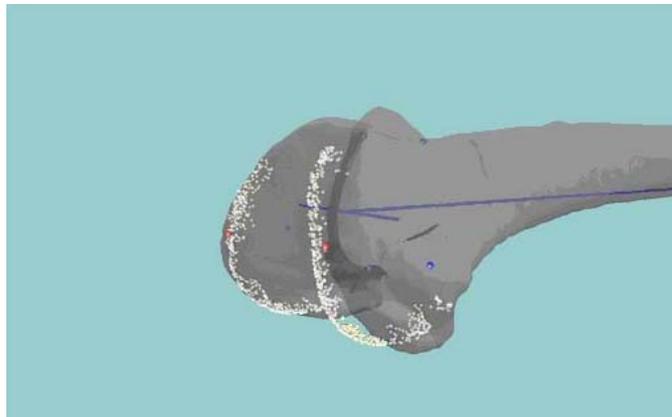


Abb. 22: Seitenansicht: Threshold=1mm

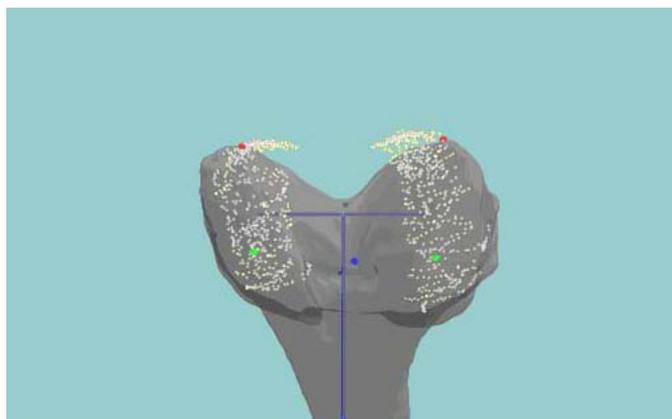


Abb. 23: Untenansicht: Threshold=1mm

Der zweite Parameter war das Gewicht für die Approximation. Aus [13] war bekannt, dass ein Wert zwischen 0 und 1 sinnvoll erschien.

Wie man in den Abbildungen 24 und 25 erkennen kann, ist die Approximation bei einem Wert von 1 sehr stark und das 3D-Modell wird nur unzureichend an die gemessenen Oberflächen angepasst.

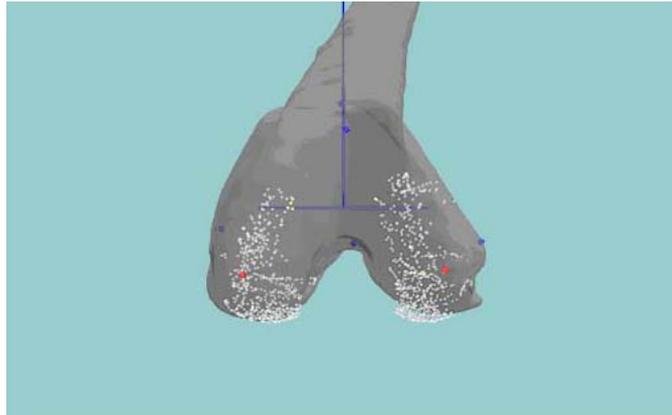


Abb. 24: Draufsicht: Gewicht=1

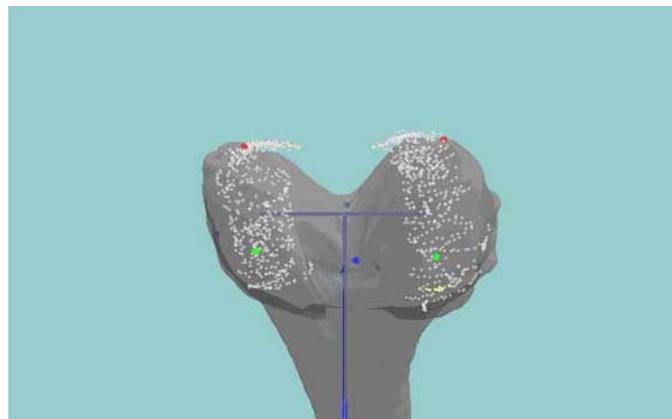


Abb. 25: Untenansicht: Gewicht=1

Die Reduktion des Gewichts auf 0,1 (siehe Abbildung 26) weist noch immer Bereiche auf, in denen die Messdaten ziemlich weit von der Oberfläche des 3D-Modells entfernt sind, was für unser Vorhaben die gemessene Oberfläche möglichst genau wiederzugeben nicht ausreichend ist. Ab einem Gewicht von 0,01 und

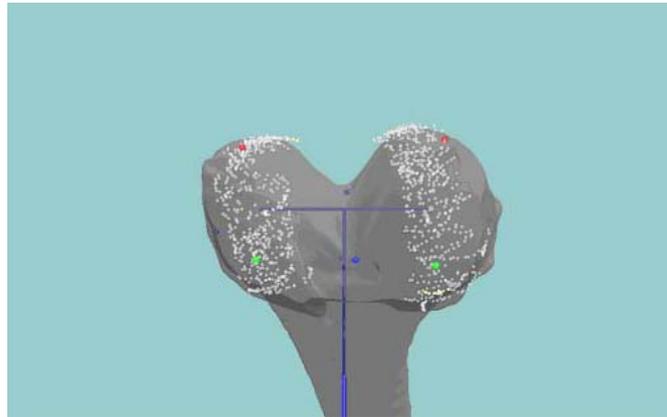


Abb. 26: Untenansicht: Gewicht=0,1

kleiner liegen Oberfläche und Punktwolke sehr nah beieinander und das Erscheinungsbild wirkt insgesamt sehr ansprechend (Abbildungen 27 und 28).

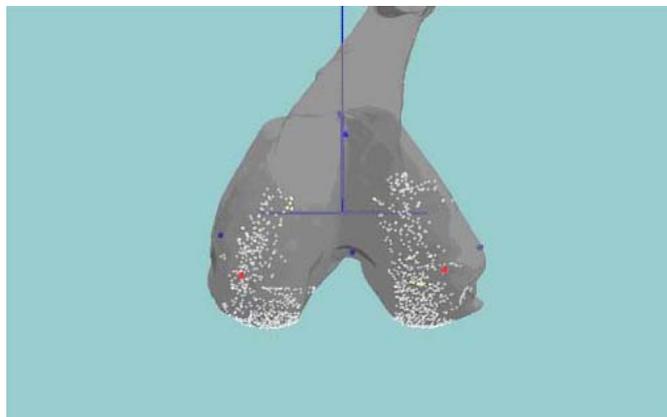


Abb. 27: Draufansicht: Gewicht=0.005

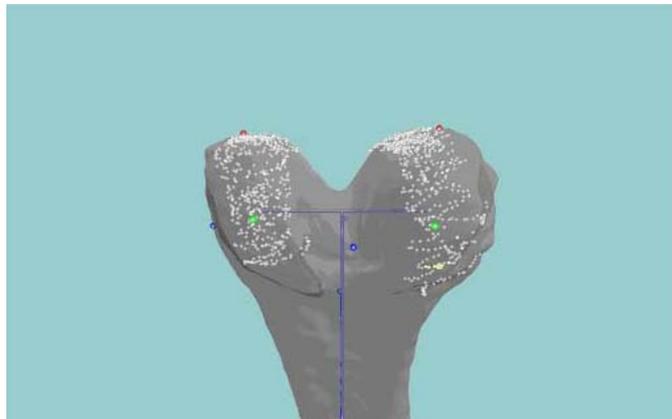


Abb. 28: Untenansicht: Gewicht=0.005

Da die Oberfläche möglichst genau dargestellt werden soll, liegt der Gedanke nah, auch die Punktwolke zu interpolieren, wovon aber abzuraten ist, wie man in Abbildung 29 gut sehen kann. Da die Messungen nie absolut genau sein können, würde eine Interpolation ebenfalls keine genaue Wiedergabe der Oberfläche erreichen.

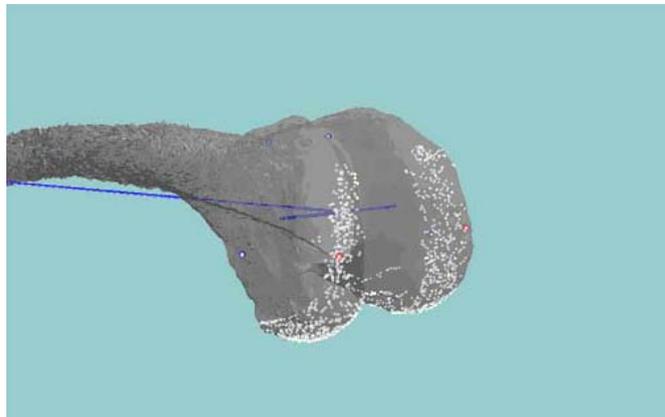


Abb. 29: Seitenansicht: Gewicht=0,0

Womit man zu dem Schluss kommt, dass ein Wert zwischen 0,01 und 0,0 ein optimales Ergebnis liefert.

Die Anzahl der verwendeten Punkte war der dritte Parameter. Dieser hat den direktesten Einfluss auf die Laufzeit, und die Frage, ab wievielen Punkten die Qualität merkbar sinkt, musste geklärt werden. Für diese Versuche wurden die als Threshold 7,5mm und als Gewicht 0,05 verwendet, welche sich als sehr gute Werte herausstellten.

Bei der vollen Anzahl von 1196 Punkten betrug die Laufzeit 61435 ms, auf einem Rechner mit Pentium4 2,4GHz und 1024MB RAM, was hier als Maximalwert gelten soll. Bei einer Reduktion auf 598 Punkte, indem nur jeder zweite gemessene Wert verwendet wurde, verkürzte sich die Laufzeit auf 40235ms, ohne in sichtbare Qualitätsverluste zu resultieren. Bei nur einem Drittel der Punkte (398) beträgt die Laufzeit 35407 ms bei ähnlich gutem Ergebnis. In den Abbildungen 30, 31 und 32 können die drei Varianten verglichen werden.

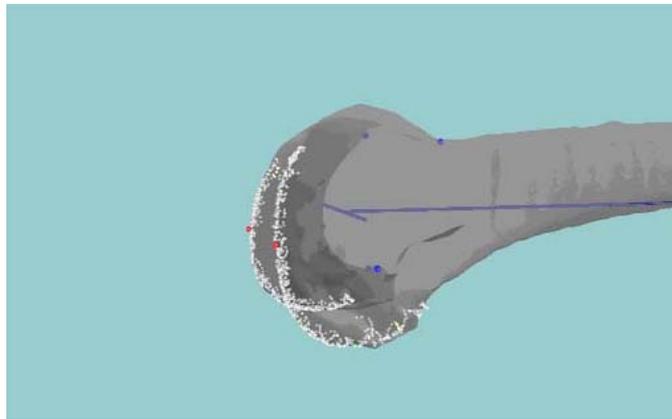


Abb. 30: Seitenansicht: Punkte=1196

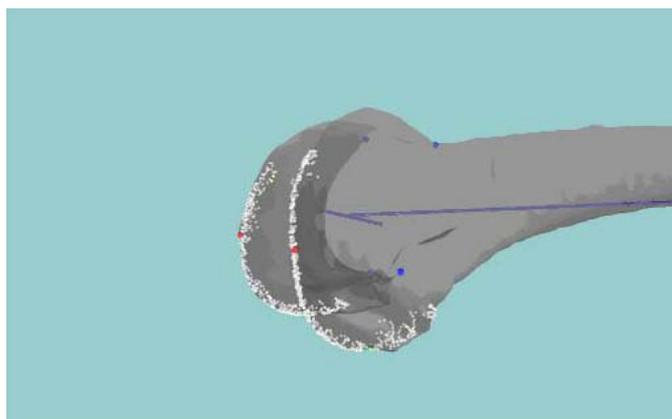


Abb. 31: Seitenansicht: Punkte=598

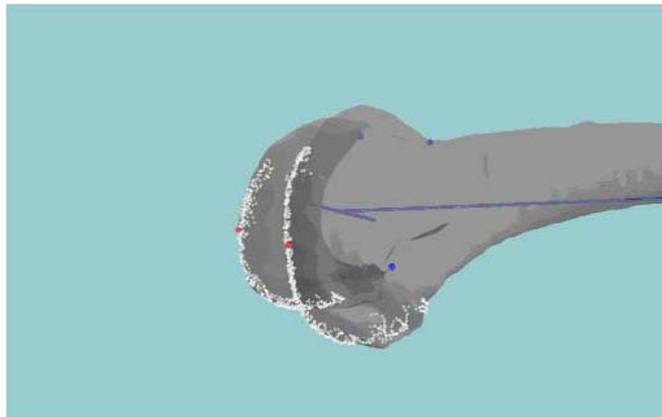


Abb. 32: Seitenansicht: Punkte=398

Bei nur noch 299 verwendeten von 1196 Punkten sieht man erste Bereiche, in denen auf Grund der fehlenden Punkte und der möglichen Entfernung durch die Korrespondenzfindung, keine Anpassung der Oberfläche mehr erfolgt (Abbildung 33). Außerdem ist bei einer Laufzeit von 32037 ms der weitere Zeitgewinn eher gering. Demnach erscheint eine Reduktion der Punktemenge von 1200 auf ca. 400-600 am erfolversprechendsten.

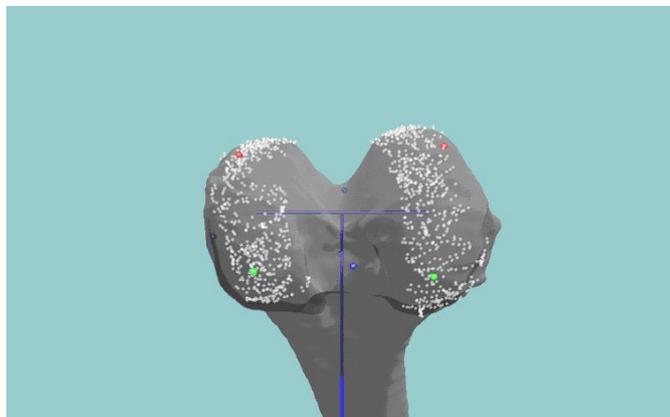


Abb. 33: Untenansicht: Punkte=299

4.2 Anwendungsfälle

Um die Anwendung der Klassen im KneeNavigator zu prüfen, wurden vier Durchläufe mit Herrn Bublat durchgeführt, die im folgenden mit den Ergebnissen aufgeführt werden. Die Freiheitsgrade des Algorithmus sind für alle Anwendungsfälle auf den optimalen Wert wie in 4.3 beschrieben eingestellt worden.

4.2.1 Optimale Messung

Mit optimaler Messung ist ein Durchlauf ohne Probleme und mit höchster Genauigkeit gemeint. Dieser Durchlauf entspricht einer normalen Anwendung des KneeNavigator. In Abbildung 34 ist das Resultat dargestellt.

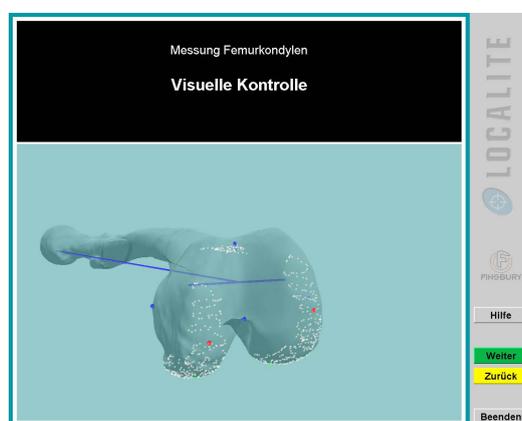


Abb. 34: KneeNavigatorsituation 1

Die Deformation führt zu einer sehr guten Möglichkeit die Messdaten zu kontrollieren. Aus dem Bild würde der Anwender schlussfolgern, dass die aufgenommenen Daten korrekt waren und die Anwendung der Software weiter geführt werden kann.

4.2.2 Anwendungsfehler: Pointer verrutschen

Der Hintergrund für dieses Thema ist ein häufig auftretender Fehler. Der Benutzer bewegt den Pointer ausserhalb der gewünschten Messbereiche. Ein Grund dafür könnte die verspätete Betätigung der Stopfunktionen während der Aufnahme sein.

Der Kochen zeigt deutliche Abweichungen zu natürlichen Knochen auf. Der Anwender würde die erneute Aufnahme der Messdaten durchführen, womit die Berechnung zu dem erwünschten Ergebnis geführt hat (Abbildung 4.2.2).

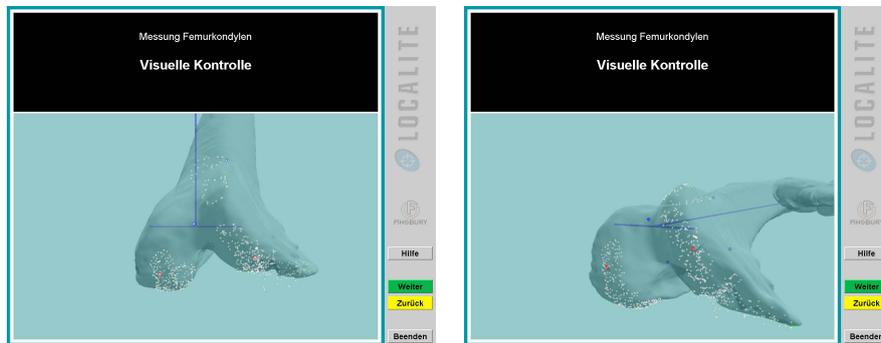


Abb. 35: Kneenavigatorsituation 2

4.2.3 Vertauschung links-rechts

Hierbei wurde geprüft, ob das Laden eines linken Femurmodells mit Abtastung eines rechten Beines, zum Beispiel durch Vertauschung, zueinander kompatibel sind. Es wäre von Vorteil, wenn dies keine Auswirkung auf das Ergebnis hätte, da für diesen Fall nur ein VRML-Objekt unabhängig von der aufzunehmenden Körperseite generiert werden müsste. In Abbildung 36 ist ersichtlich, dass der Knochen gespiegelt wird. In diesem Anwendungsfall wurden sämtliche eingelesenen Daten vertauscht, was zu einem korrekten Ergebnis geführt hat. Sollte man bei der Aufnahme einzelner Punkte die Seiten vertauschen, z.B. mediale und laterale Landmarken, so würde der Knochen deutlich sichtbare unnatürliche Verformungen aufweisen.



Abb. 36: Kneenavigatorsituation 3

4.2.4 Simulation von krankem Gewebe

Die Probeknochen für die Datenaufnahme mit dem Pointer waren im Bezug auf einen kranken Knochen, wie er in einer realen Anwendung zu erwarten ist, zu glatt an der Oberfläche. Um die Anwendung eines realistischeren Knochens dennoch prüfen zu können, wurde eine Maske aus kneteartigem Material gefertigt, den Verformungen einer kranken Knochenoberfläche angepasst und über einen Teil (siehe Abbildung 38) des Knochens gelegt. Damit wurde der Testlauf durchgeführt.

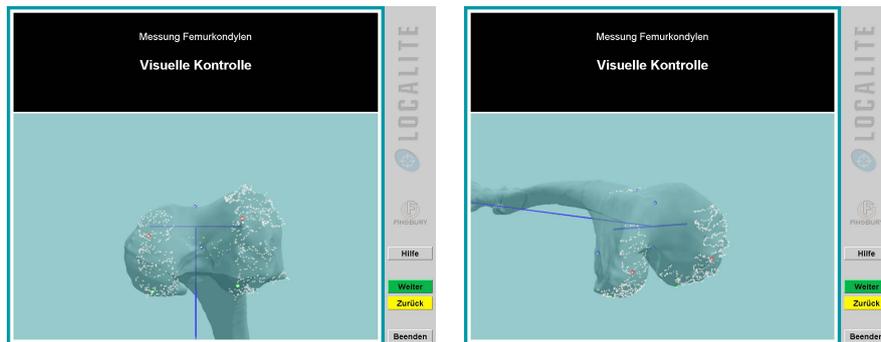


Abb. 37: KneeNavigatorsituation 4



Abb. 38: unebenes Gewebe am Knochen, simuliert mit Maske

4.3 Bewertung

Nach den Testläufen zu den Freiheitsgraden ergaben sich für die drei Parameter Approximationsgewicht, Distanz-Grenzwert und Punktmenge die folgenden Werte.

Ein Approximationsgewicht von 0,005 bietet eine sehr ansehnliches Ergebnis,

bei dem die Punkte nur gering approximiert werden und so auch gemessene Unebenheiten dargestellt werden.

Bei den Tests zur Punktemenge zeigte sich, dass bei 600 anstatt 1200 verwendeten Punkten kein Unterschied in der Qualität sichtbar ist und so ist es aus Performancegründen vorteilhaft, bei der Überschreitung einer gewissen Anzahl an Punkten diese zu reduzieren. Dieser Vorgang ist bereits in den KneeNavigator integriert.

Beim Distanz-Grenzwert erschien ein Wert von 7,5 mm als sehr geeignet, da bei dieser Entfernung die Anzahl der entfernten Punkte eher gering ausfiel und somit nicht zu Qualitätseinbußen führte. Auf Wunsch von LOCALITE wurde dieser Wert jedoch auf 15 mm gesetzt, um somit nur extreme Ausreißer aus der Punktwolke zu entfernen.

Die anschließend durchgeführten Anwendungsfälle bestätigten die Wahl der Freiheitsgrade durch hohe Geschwindigkeit und sehr guter Qualität. Die zuvor erhobenen Erwartungen an die Ergebnisse wurden zur vollsten Zufriedenheit erfüllt.

5 Zusammenfassung

Die Aufgabe war die Erstellung eines 3D-Oberflächenmodells, welches aus der Deformation eines Modells und gemessenen Stützdaten resultieren sollte. Die Anpassung des Objektes musste sowohl global als auch lokal, in einer interpolierenden sowie approximierenden Genauigkeit vollzogen werden. Bei diesen Veränderungen des Originalmodells durfte sich die Oberfläche nicht unbestimmt verändern, sondern sollte sich an definierte Bedingungen halten. Der Praxisbezug der Aufgabe wurde mit der Verbesserung des KneeNavigators geschaffen. Hier sollte eine konkrete Darstellungsszene mit Hilfe des Ergebnisses intuitiver gestaltet werden. Der Benutzer sollte eine möglichst realitätsnahe, ansprechende, intuitive und eindeutige Visualisierung ermöglicht bekommen. Es wurde gezeigt, dass die Entscheidung für die Thin Plate Splines die richtige gewesen ist. Die Implementierung und Integration in den KneeNavigator sind erarbeitet worden und die Ergebnisse getestet. Die Tests haben bewiesen, dass die Berechnungen erwartungsgemäß verlaufen und Probeverformungen exakt und identisch auf dem Bildschirm angezeigt werden können. Die Rechenzeit beträgt zu diesem Zeitpunkt noch maximale 60 Sekunden, Verbesserungsvorschläge werden aber im Kapitel 6 mitgeliefert und dürften die Laufzeit auf maximal 10-15 Sekunden senken.

6 Ausblick

Im Laufe der Programmierphase wurden einige Aspekte entdeckt, die zur Verbesserung der Performance beitragen würden. Leider konnte in der Kürze der Zeit deren Anwendung nicht eingebracht werden. Es soll aber die Gelegenheit genutzt werden den Anwendern die Ideen mit auf den Weg zu geben.

Der größte Schwachpunkt ist die Laufzeit. Das Programm erreicht inzwischen eine maximale Laufzeit von nur noch 60 Sekunden bei Punktwolke mit jeweils 300 Messpunkten und einem vorgeladenen VRML-Objekt mit 250.000 Objektpunkten. Dies bedeutet eine Gesamtmatrix von maximal 1500x1500 und 250.000x1500 Schleifendurchläufen nur innerhalb der *getTransformedCoordinates()* in der TPS-Klasse. Hier greift der erste Punkt der Verbesserung. Die Verwendung eines *TriangleIndexArrays* würde die Objektgröße auf 65.000 Punkte reduzieren, da jeder Punkt nur einmal referenziert wird, während zur Zeit ein Punkt so oft aufgerufen wird, wie er in Dreiecken vorkommt. Desweiteren nimmt die Korrespondenzsuche den zweithöchsten Zeitfaktor ein.

Hier könnte man, anstatt das gesamte Objekt nach korrespondierenden Punkten zu durchsuchen, nur das interessante Knochenende zur Suche freigeben. Dazu setzt man im einfachsten Fall einen Schwellwert an einen Punkt entlang der z-Achse und stoppt damit den Durchlauf. Außerdem suchen wir in unserem Algorithmus den nächstliegenden Vertex und nicht den dichtesten Schnittpunkt der Oberfläche. In diesem Punkt wäre eine Verbesserung der Genauigkeit möglich. Im Laufe der Ideenfindung für das Korrespondenzproblem wurde auch die Idee verfolgt, die Korrespondenzfindung durch die *PickingBehavior* von Java zu implementieren. Die Idee dahinter ist das *Picking* innerhalb einer vordefinierten Bound, wodurch der Radius der Suche bestimmt wird. Ein Durchlaufen des Objektes wäre damit entfallen. Es wurde vollständig implementiert, doch leider blieb der zeitliche Vorteil aus, der sich von der java-internen Verwendung von *Octrees* erhofft wurde bzw. drehte sich zu Kosten um. Leider war keine ausreichende Dokumentation in Java verfügbar, jedoch hat die dahinter stehende Idee Potential.

Ein weiterer Anhaltspunkt ist die Reduzierung der Objektgröße, indem die hintere liegenden Bereiche des Knochens vergrößert werden. Das feinmaschige Dreiecksnetz wird nur am Knochenende benötigt. Auch die TPS-Berechnung an sich könnte sich im hinteren Teil des Knochens (entlang der z-Achse betrachtet) mit der Verwendung von weniger Punkten beschränken. Ein Punkt zur Performance ist außerdem die Anwendung eines statistischen Knochenmodells, um Besonderheiten in dem gewählten Originalmodell von Beginn an auszuschließen. Ein weiterer Aspekt könnte je nach Benutzergruppe relevant sein.

Die Ergebnisdarstellung könnte eine farbliche Kodierung der statistischen Sicherheit des Ergebnisses enthalten. Je nach Messdatensatz und tatsächlicher Anzeige der Werte müßte eine Sicherheitsstufe entschieden werden, wobei die Landmarken kontinuierlich die höchste erhalten würden, da sie interpoliert werden. Es wurde sich bewusst dagegen entschieden, da die Darstellung schon vielsagend war und stattdessen wurden die Messdaten in allen Anzeigen dargestellt.

Eine eventuelle Abweichung der Daten zur berechneten Oberfläche ist dadurch gut zu erkennen. Das Programm ist mit dem Femur implementiert und getestet worden. Eine Erweiterung und Anpassung aller Komponenten auf VRML-Objekte im Allgemeinen ist notwendig. Wenn auf Grund dieser Aspekte alle VRML-Dateien, unabhängig von deren innerem Aufbau, in angemessen schneller Zeit deformiert werden können, sind noch viele weitere Anwendungsfälle denkbar. Interessante Themen für weitere Studien- oder Diplomarbeiten könnten das Testen der Thin Plate Splines an sich stark deformierenden Organen des menschlichen Körpers, z.B. das Gehirn oder das Implementieren eines anderen Verfahrens für den gleichen Sachverhalt als vergleichende Maßnahme sein.

Danksagung

Wir möchten uns ganz herzlich bei Professor Müller für die Betreuung und Unterstützung bedanken. Desweiteren danken wir dem Fraunhofer-Institut FIT, Forschungsbereich Life Science Informatik und der Firma LOCALITE GmbH, im Besonderen möchten wir hier unseren Betreuer Ron Schwarz erwähnen, der uns immer helfend zur Seite stand.

Literatur

- [1] Das Kniegelenk. <http://www.wissen.de>.
- [2] LOCALITE GmbH. Produktbeschreibung KneeNavigator.
- [3] Dietmar Meister. *Bilddaten getriebene Kinematik-Simulation für den Roboter gestützten Kreuzband- und Kniegelenkersatz*. PhD thesis, Fakultät für Informatik, Universität Karlsruhe, 2003.
- [4] M.M.Cerney, D.C.Adams, and J.M.Vance. Image warping of three-dimensional body scan data, 2003.
- [5] I.N. Bankman, editor. *Handbook of Medical Imaging - Processing and Analysis*. Academic Press, 2000.
- [6] Northern Digital Inc. *Polaris System Guide, Revision Number 2.1*, Februar 2003.
- [7] S. Thrun. A bayesian approach to landmark discovery and active perception in mobile robot navigation, 1996.
- [8] R. Greiner and R. Isukapalli. Learning to select useful landmarks, 1994.
- [9] S. D. Steck and H. A. Mallot. The role of global and local landmarks in virtual environment navigation. Technical report, Max-Planck-Institut für biologische Kybernetik, 1997.
- [10] J. Fischer. Aspekte medizinischer Bildverarbeitung, Registrierung von Bilddaten, 2004. Hauptseminar.
- [11] P.J. Besl and N.D. McKay. A method for registration of 3d shapes. *IEEE Trans. Patt. Anal. Machh. Intell.*, 14(2):239–256, 1992.
- [12] P. Thompson and A.W. Toga. Elastic image registration and pathology detection.
- [13] Karl Rohr, H. Siegfried Stiehl, Rainer Sprengel, Wolfgang Beil, Thorsten M. Buzug, Jürgen Weese, and M. H. Kuhn. Point-based elastic registration of medical image data using approximating thin-plate splines. In *VBC*, pages 297–306, 1996.
- [14] H. Chui and A. Rangarajan. A new algorithm for non-rigid point matching.
- [15] F. L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(6):567–585, 1989.
- [16] S. Fang, R. Raghavan, and J.T. Richtsmeier. Volume morphing methods for landmark based 3d image deformation.

- [17] G. Donato and S. Belongie. Approximation methods for thin plate spline mappings and principal warps, 2002.
- [18] H. Chui and A. Rangarajan. A new point matching algorithm for non-rigid registration, 2002.
- [19] K. Rohr, H.S. Stiehl, R. Sprengel, W. Beil, T.M. Buzug, J. Weese, and M.H. Kuhn. Nonrigid registration of medical images based on anatomical point landmarks and approximating thin-plate splines.
- [20] A. Zandifar, S. Lim, R. Duraiswami, and L.S. Davis N. Gumerov. Multi-level fast multipole method for thin plate spline evaluation.
- [21] N. Arad and D. Reisfeld. Image warping using few anchor points and radial functions, 1995.
- [22] A.M. Bazen and S. H. Gerez. Elastic minutiae matching by means of thin-plate spline models, 2002.