UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

AGAS
Arbeitsgruppe Aktives Sehen

Kamerapositionsbestimmung über Analyse durch Synthese

Diplomarbeit

zur Erlangung des Grades eines/r
Diplom-Informatikers / Diplom-Informatikerin
im Studiengang Computervisualistik

vorgelegt von

Matthias Dennhardt

Betreuer:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Erstgutachter:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Zweitgutachter:	Dipl.-Inf. Tobias Feldmann, Institut für Computervisualistik, Fachbereich Informatik

Koblenz, im März 2007

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien der Arbeitsgruppe für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ja ☐ nein ☐

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja ☐ nein ☐

Koblenz, den ..... Unterschrift

Inhaltsverzeichnis

1 Einleitung **9**

1.1 Motivation 9

1.2 Zielsetzung 11

1.3 Gliederung der Arbeit 13

2 Stand der Technik **15**

2.1 Objektverfolgung 15

2.2 Technikgestützte Objektverfolgung 17

2.2.1 Verfahren für Außenanwendung 18

2.2.2 Verfahren für Innenanwendung 20

2.2.3 Universell Einsetzbare Verfahren 22

2.3 Bildbasierte Positionsbestimmung: 2-D/3-D-Registrierung 23

2.3.1 Abstandsmaße 25

2.3.2 Demokratische Integration 33

2.3.3 Optimierungsverfahren 37

3 Eigener Ansatz **45**

3.1 Das Prinzip der Analyse durch Synthese 45

3.1.1 Anwendungsbeispiele 45
3.1.2 Prinzip 48
3.1.3 Positionsbestimmung über Analyse durch Synthese 48
3.2 GPGPU 50
3.2.1 Grafikkartenprogrammierung 51
3.2.2 OpenGL-Programmkontext 53
3.2.3 Shader-Programmkontext und Grafikprozessorpipeline 57

4 Systemaufbau **61**
4.1 Objektorientierte Analyse 62
4.1.1 Analyse im Großen 64
4.1.2 Klassendiagramme der Funktionspakete 68
4.1.3 Veränderungen und Neuentwicklungen 75

5 Experimente und Ergebnisse **83**
5.1 Versuchsaufbau 83
5.1.1 Kameraufnahme 83
5.1.2 Bildsynthese 84
5.1.3 Bildregistrierung 85
5.2 Versuchsdurchführung 87
5.3 Ergebnisse 88
5.3.1 1. Experiment: Verwendung aller Abstandsmaße 88
5.3.2 2. Experiment: Verwendung einzelner Abstandsmaßtypen 90
5.3.3 3. Experiment: Kombination verschiedener Abstandsmaßtypen . . 94
5.3.4 Vergleich aller Kombinationsmöglichkeiten 97

6 Zusammenfassung und Ausblick 99

Kapitel 1

Einleitung

1.1 Motivation

Die genaue Positionsbestimmung eines Gegenstandes ist eine Grundvoraussetzung für das nutzbringende Arbeiten in vielen technischen Bereichen. Besonders deutlich zeigt es sich in dem Gebiet der Robotik oder in Applikationen von VR/AR[1]. Im Falle der Robotik wird die genaue Bestimmung des Standortes und Orientierung eines Roboters mit einer Vielzahl von Sensoren, wie z. B. Infrarot- oder Schallsensoren und GPS[2] ermöglicht. Es zeigt sich aber, dass die Messdaten dieser Sensoreinheiten durch eine Menge von Faktoren, meist technischer Art, verfälscht werden und eine zentimetergenaue Erfassung der Position oft unmöglich wird.

Auch in dem Bereich der VR/AR wird mit einer Reihe von Positionserkennungsverfahren gearbeitet, die als Endergebnis die genaue Position und Orientierung einer Kamera liefern sollen. Diese Verfahren können in zwei Gruppen unterteilt werden. Die erste, in der das GPS zu finden ist, wird als *Grobtrackingsysteme* bezeichnet. Die zweite Gruppe beinhaltet Systeme, die das sogenannte *Feintracking* ermöglichen sollen, und denen Verfahren mit Ultraschall oder optische Positionserkennungssysteme zugeordnet werden. Beide Mengen

[1] VR/AR = Virtual Reality/Augmented Reality
[2] GPS = Global Positioning System

werden unter dem Begriff *Tracking*[3] zusammengefasst. Oft werden in einem sogenannten Trackingsystem beide Arten der Positionserkennung verwendet, indem zuerst die grobe Position eines Gegenstandes erfasst wird und anschließend mit Hilfe eines Feintrackingsystems die berechneten Standortangaben verbessert werden.

In dem Bereich des Feintrackings haben sich vor allem die optischen Verfahren mit einer hohen Genauigkeit hervorgetan. Das optische Tracking kann auf zwei verschiedene Arten durchgeführt werden. Entweder befindet sich eine Kamera in der Umgebung und nimmt Marker auf, die an dem zu erkennenden Gegenstand befestigt sind. Oder die Kamera selbst ist an dem Objekt angebracht, und die Marker sind an bestimmten Stellen des Geländes befestigt. Meist weisen im zweiten Fall die Marker bestimmte Muster auf, die es dem Positionserkennungsystem ermöglichen die genaue Standortangabe zu berechnen.

Im ersten Fall spricht man vom sogenannten „outside-in“ Tracking. Diese Bezeichnung soll andeuten, dass die Kamera nicht am zu erkennenden Gegenstand angebracht ist, sondern sich an einer beliebigen Position außerhalb befindet und die Muster, die von den Markern dargestellt werden, sucht. Im zweiten Fall ist vom „inside-out“ Tracking die Rede. Hier soll mit der Bezeichnung ausgesagt werden, dass die Kamera am zu erkennenden Objekt befestigt ist und in der Umgebung nach den Markern sucht.

Die Marker selbst können in aktive und passive unterteilt werden. Bei Aktiven handelt es sich meist um LEDs[4], deren Leuchten im Bild helle Punkte hinterlässt, die dann von einem Erkennungssystem gesucht werden können. Die passiven Marker unterscheiden sich voneinander durch die geometrische Anordnung ihrer Musterelemente.

Die Nachteile beider optischen Tracking-Systeme liegen zum einen im hohen technischen Aufwand, der durch das Beleuchten der Marker oder der am Objekt mitgeführten Kamera herrührt, und zum anderen in der Verwendung von Markern, da sie voraussetzen, dass sie vor der Positionserkennung am Objekt oder in der Umgebung angebracht werden. Darüber hinaus muss das Trackingsystem die Marker lernen, d. h. es muss eine Verknüpfung aufgebaut werden, die den Marker mit seiner Position in der Umgebung verbindet. Erst dann ist es möglich die Standortbestimmung der Kamera durchzuführen.

[3] tracking = in seiner Position und Orientierung erfassend

[4] LED = Licht emittierende Dioden

Aus den genannten Nachteilen, die mit der Benutzung von Markern einher geht, ergibt sich die Bestrebung auf diese bei der Positionserkennung zu verzichten. Eine Möglichkeit stellen die stochastischen Verfahren zur Zustandsschätzung dar, zu denen z. B. die sequentielle Monte-Carlo-Methode oder der Kalmann-Filter gehören. Diese Verfahren versuchen aus den Sensordaten, die eine ungefähre Standortangabe ermitteln, Voraussagen über die zukünftige Position eines dynamischen Objektes zu machen. Mit einem dynamischen Objekt kann z. B. eine Person gemeint sein, die sich in einer Umgebung bewegt.

Ein neuer Ansatz für die markerlose Positions- und Orientierungsbestimmung soll mit dem System dieser Arbeit getestet werden. Das Kernprinzip der Arbeit ist die Analyse durch Synthese. Mit den Daten aus einer ungefähren Standortbestimmung wird mit einem *Renderer*[5] eine Ansicht der Umgebung erstellt, wie sie von einer Kamera, die entsprechend der Grobpositionsdaten in der Umgebung positioniert ist, gesehen wird. Mit einem Vergleich zwischen synthetisch erzeugter Ansicht und dem Bild einer Kamera werden die Positionsangaben korrigiert. Mit diesem Ansatz werden die von einer Kamera aufgenommenen Daten direkt für die Positionskorrektur genutzt.

1.2 Zielsetzung

Das in dieser Diplomarbeit angestrebte System soll mit Hilfe der 2-D/3-D-Registrierung die Korrektur einer groben Positionsangabe, die z. B. durch das GPS gemacht wird, durchführen. Dabei wird ein Objekt, dessen Position in der Welt als bekannt vorausgesetzt wird, von einer Kamera betrachtet. Neben der Kenntnis über die Position liegt außerdem ein texturiertes 3-D-Modell des betrachteten Objekts vor. Eine Ansicht auf das 3-D-Modell wird mit der von einem GPS gelieferten Grobposition synthetisch, mit Hilfe eines *Renderers*, erzeugt. Durch mehrere Bildabstandsmaße werden, wird das synthetische Bild mit dem aktuellen Kamerabild verglichen. Über ein Optimierungsverfahren werden die Parameter für Rotation und Translation neu geschätzt und der Bildsyntheseeinheit für die Erzeugung einer neuen Ansicht zugeführt.

[5] Ein Programm, dass aus einer Modellbeschreibung synthetische Bilder erzeugt.

Dieser Vorgang wiederholt sich so lange, bis die Ansicht gefunden wird, die in den Bildabstandsmaßen S_i den kleinsten Abstand zum aktuellen Kamerabild aufweist. Die für diese Ansicht verwendeten Parameter für Rotation und Translation, sowie die intrinsischen Kameraparameter[6] werden abschließend für die Bestimmung der Position der Kamera in der Welt verwendet. Abbildung 1.1 veranschaulicht den beschriebenen Ablauf.

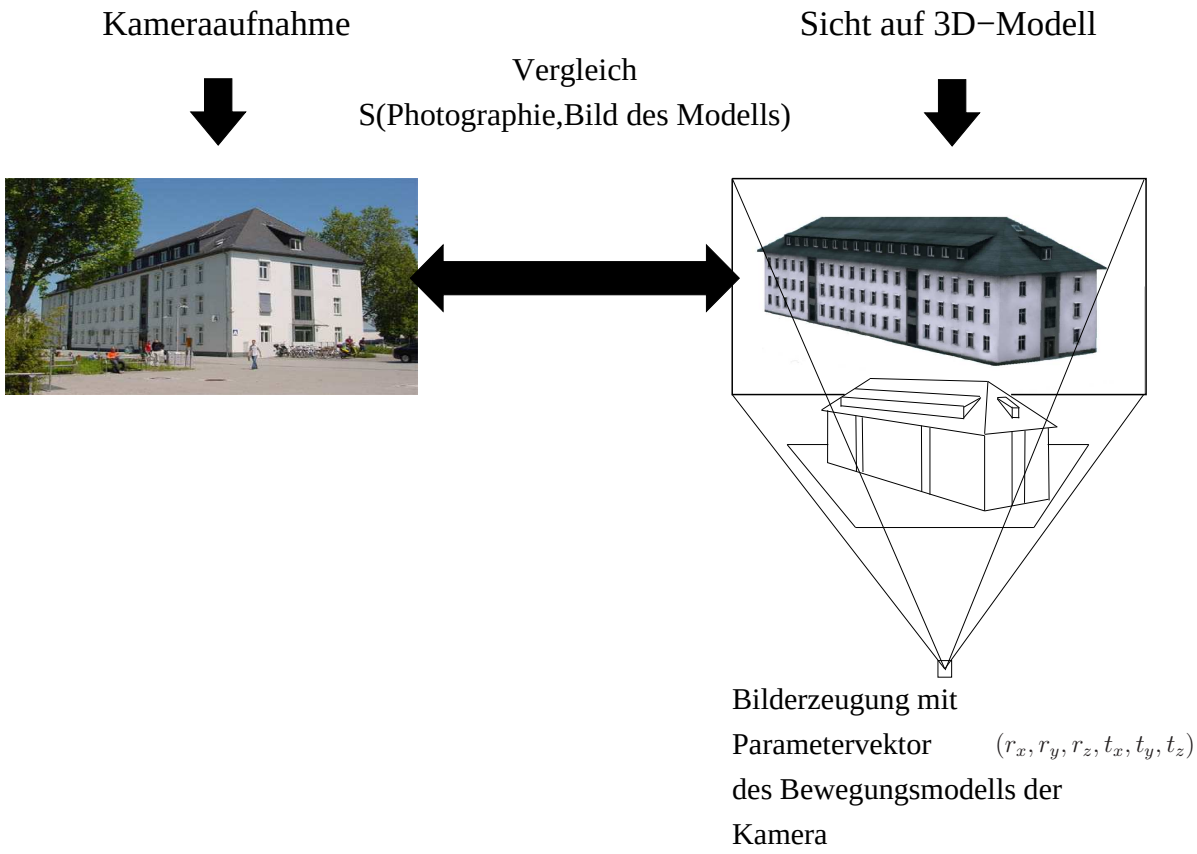


Bild 1.1: Bildvergleich zwischen Kamerabild und synthetisch erzeugtem Bild

[6]Brennweite, Größe des Sensorelementes der Kamera, Hauptpunkt, radiale Verzerrung

1.3 Gliederung der Arbeit

Die Arbeit gliedert sich in sechs Kapitel. In Kapitel 2 wird ein Überblick über die derzeit verwendeten Techniken für die Positionsbestimmung gegeben. Als Beispiel für einen bildbasierten Ansatz wird die 2-D/3-D-Registrierung vorgestellt, da dieses Verfahren auch in der vorliegenden Arbeit verwendet wird. Die Vorstellung des Verfahrens stützt sich auf die Diplomarbeiten von Tobias Feldmann [FBPD06] und Alexander Kubias [Kub06].

In Kapitel 3 wird zu Beginn das Prinzip der Analyse durch Synthese mit einem geschichtlichen Abriss und der Beschreibung einiger Anwendungbeispiele vorgestellt. Dabei wird z. B. auf die Anwendungsfelder der Sprachsignalverarbeitung und der Ornamentik eingegangen. Anschließend wird der Zusammenhang zwischen dem Prinzip der Analyse durch Synthese und der 2-D/3-D-Registrierung hergestellt, um den eigenen Ansatz dieser Arbeit zu erläutern. Abschließend wird eine Einführung in die GPGPU[7]-Programmierung gegeben, um für mögliche Nachfolgearbeiten einen schnellen Einstieg zu bieten.

Das Kapitel 4 hat den Aufbau des Programms als Inhalt und gibt einen Überblick über den generellen Programmablauf. Kapitel 5 beschreibt die mit dem System durchgeführten Experimente, den dazu verwendeten Versuchsaufbau und nimmt abschließend eine Deutung der Ergebnisse vor. Den Schluss der Arbeit bildet eine Zusammenfassung, in der auch ein Ausblick auf mögliche Nachfolgearbeiten gegeben wird.

[7] GPGPU = General-Purpose Computation on Graphics Processing Units

Kapitel 2

Stand der Technik

2.1 Objektverfolgung

Der Begriff *Objektverfolgung* (engl. Tracking) fasst eine Reihe von Aufgaben und deren Lösungen zusammen, die im Zusammenhang mit der Positionsbestimmung eines Objektes über einen festen Zeitraum hinweg auftreten. Mit diesem Begriff werden nicht nur Verfahren für die Positions- und Orientierungsbestimmung eines dynamischen Objektes bezeichnet, sondern auch Verfahren, die als Ziel die Korrektur von Messdaten eines Sensorelementes haben, um dessen Positionsbestimmung genauer werden zu lassen. Mit dynamischen Objekten ist im Rahmen der Objektverfolgung eine Vielzahl von Gegenständen gemeint. So kann z. B. die Position eines Autos oder die Augenstellung eines Menschen von Interesse sein.

Je nach Anwendungsfeld können mit der Objektverfolgung unterschiedliche Ziele verknüpft sein. In vielen Bereichen der Bildverarbeitung steht die Bestimmung und Verfolgung eines Ojektes in einer Bildfolge im Vordergrund, ohne dabei die genaue Position des Gegenstandes in Bezug auf ein festgelegtes Weltkoordinatensystem bestimmen zu wollen. Der Schwerpunkt liegt hierbei auf der Ermittlung der Position des Objektes im Bild und der Beziehung des detektierten Objektes zum Betrachter der Szene.

Im Bereich der Virtuellen Realität (engl. Virtual Reality) oder der Erweiterten Realität (engl. Augmented Reality), die beide ein praktisches Anwendungsfeld der Computergrafik darstellen, ist das Ziel der Objektverfolgung die genaue Bestimmung der Position und Orientierung eines Gegenstandes oder einer Person. Erst mit diesem Wissen können synthetisch erzeugte Bilder in die richtige Beziehung zum Betrachter gesetzt werden. Oft ist es erwünscht, dass die Positions- und Orientierungsbestimmung millimetergenau durchgeführt wird, um z. B. Informationen oder Hinweise an die richtige Stelle im Kamerabild zu setzen. Um diesen Anspruch zu erfüllen, werden häufig Algorithmen aus dem Bereich der Bildverarbeitung verwendet, um bestimmte Eigenschaften im Bild zu detektieren und zu verfolgen. Jede Veränderung der Position einer selektierten Eigenschaft ist zugleich mit einer Neubestimmung der Positions- und Orientierungsparameter verbunden. Um eindeutige Eigenschaften im Bild finden zu können, kommen in vielen aktuellen Anwendungen aktive oder passive Marker zum Einsatz.

Die optische Objektverfolgung zur Ermittlung der Positionsparameter ist derzeit nur auf kleine Umgebungen begrenzt. In größeren Bereichen kann der Fall auftreten, dass eine Zuordnung zwischen Kamerabild und der Position der Kamera nicht durchgeführt werden kann. Dieser Fall wäre z. B. mit der Aufnahme einer in ihrer Struktur homogen erscheinenden Hauswand gegeben. Aus diesem Grund wird in solchen Anwendungsfällen eine auf Sensoren gestützte Positionsbestimmung durchgeführt. Die Sensoren sind oft äußeren Einflüssen, wie Magnetfeldern, ausgesetzt oder liefern aufgrund ihres eigenen Aufbaus ungenaue Ergebnisse. Mit Hilfe von sogenannten „Tracking“- Algorithmen wird versucht, die Ungenauigkeiten in ihrer Wirkung auf das Ergebnis abzuschwächen. Dafür werden heute meist stochastische Zustandsschätzer für dynamische Systeme verwendet. Zu diesen Verfahren gehören z. B. der Kalman- (siehe [May79]) oder der Partikel-Filter (siehe [Woj04]).

Die beiden folgenden Abschnitte geben einen Überblick über die derzeitigen Techniken für die Positionsbestimmung. Der Schwerpunkt des ersten Abschnittes liegt in der Vorstellung der technischen Lösungen, wie sie z. B. mit dem GPS gegeben sind. Der zweite Abschnitt stellt als Vertreter der bildbasierten Positionsbestimmung das Verfahren der 2-D/3-D-Registrierung vor.

2.2 Technikgestützte Objektverfolgung

Im Bereich der mit Sensortechnik unterstützten Positionsbestimmung wird nach [Rot05] zwischen zwei Arten von Techniken unterschieden. Zum einen werden unter *Objektverfolgung* all jene Technologien zusammengefasst, die mit Hilfe eines Sensornetzwerkes die Position und den Fortlauf der Bewegung eines Objektes bestimmen. Dabei liegt die Positionangabe dem ermittelnden Positionierungssystem vor. Dem Objekt selbst müssen diese Angaben übermittelt werden. Zum anderen werden mit dem Begriff *Ortung* (engl. Positioning) die Technologien bezeichnet, bei denen das Objekt selbst die Position bestimmt. Die Positionsangaben liegen somit dem Objekt selbst vor.

Alle sensorgestützten Verfolgungs- und Ortungssysteme können auf fünf Basistechniken zurückgeführt werden. Die sogenannte *Zellortung* (engl. Cell of Origin (COO)) findet nach [Rot05] Anwendung, wenn das Positionierungssystem eine Zellstruktur aufweist. Dieser Aufbau ist z. B. bei Netzen von mobilen Kommunikationssystemen (engl. Global System for Mobile Communication (GSM)) zu finden. Die *Zellortung* liefert die geographischen Koordinaten der Funkzelle, in der sich ein Benutzer eingebucht hat. Die Genauigkeit dieser Technik hängt von der Größe der Zelle ab. Je kleiner sie ist, desto genauer fällt die Standortbestimmung aus.

Eine andere Grundtechnik, die als *Abstandsmessung über Zeitdifferenzen* benannt werden könnte, versucht mit Hilfe der Ankunftszeit (engl. Time of Arrival (TOA)) eines Signals beim Empfänger und der Zeitdifferenz zwischen Ankunftszeit und Absendezeit (engl. Time Difference of Arrival (TDOA)) die Entfernung zwischen Sender und Empfänger zu berechnen. Diese Technik findet vor allem bei Systemen eine Anwendung, die mit elektromagnetischen Signalen, die mit Lichtgeschwindigkeit versendet werden, arbeiten.

Mit Hilfe von Signalen kann auch die dritte Grundtechnik, die als *Positionsbestimmung über Einfallswinkel* bezeichnet werden kann, realisiert werden. Bei dieser Technik wird über den Einfallswinkel des empfangenen Signals (engl. Angle of Arrival (AOA)) auf die Position des Senders zurückgeschlossen. Der Empfänger muss mit einer Antenne mit Richtungscharakteristik ausgestattet sein, d. h. es ist möglich ohne Drehen der Antenne aus jeder Richtung den Einfallswinkel des Signals zu bestimmen.

Die Messung der Signalstärke kann ebenfalls Aufschluss darüber geben, wie groß der Abstand zwischen Sender und Empfänger ist. Aufgrund der Möglichkeit, dass die Signalstärke durch Hindernisse, wie z. B. Wänden abgeschwächt wird, kann die Anwendung dieser Grundtechnik zu ungenauen Ergebnissen führen.

Die *Auswertung von Bilddaten* kann ebenfalls dafür genutzt werden, um eine Positionsbestimmung durchzuführen. Um die Ergebnisse dieser Technik genauer werden zu lassen, kommen, wie bereits erwähnt, Marker zum Einsatz, die sich besser im Bild finden lassen.

Grundsätzlich existieren drei Anwendungsbereiche für Positionsbestimmungssysteme. Die Grafik 2.1 gibt einen Überblick über den Einsatzbereich der verschiedenen Techniken.

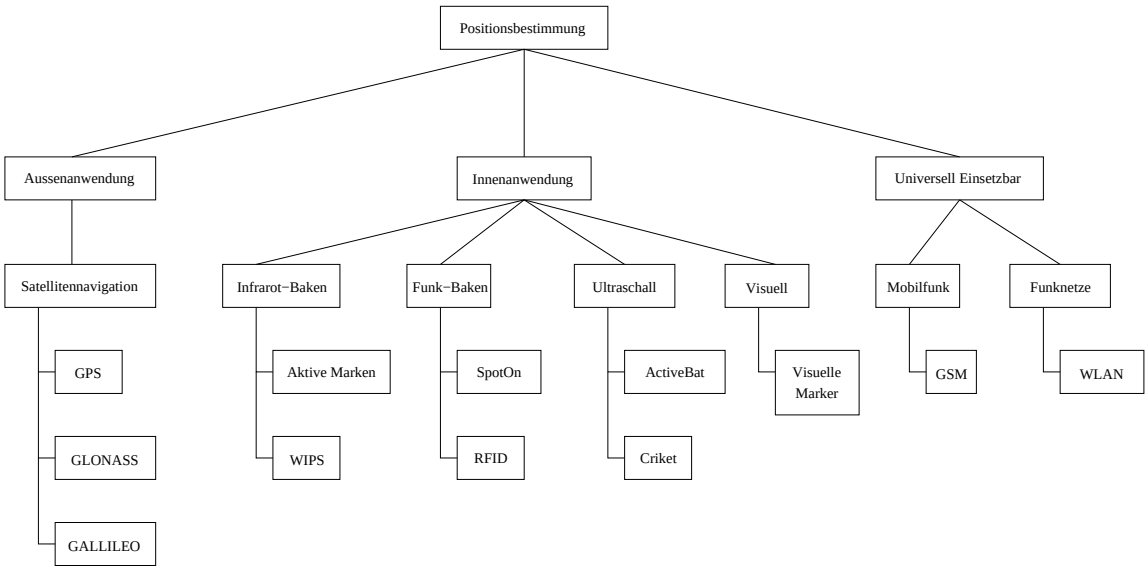


Bild 2.1: Systeme für die Positionsbestimmung und ihre Anwendungsbereiche

2.2.1 Verfahren für Außenanwendung

Die Satellitennavigation beruht auf der Bestimmung der Differenz zwischen Ankunftszeit und Absendezeit eines Signals. Um die Positionsbestimmung durchführen zu können, müssen mindestens drei Satelliten im Sendebereich eines Navigationssystems liegen. Zu

allen Satelliten wird ein Signal gesendet. Wird ein Satellit erreicht, antwortet dieser mit einem Signal, das die Absendezeit des Signals kodiert. Die Zeitdifferenz Δt zwischen Absendezeit und der Ankunftszeit beim Navigationssystem bestimmt nach Formel 2.1 die Entfernung r zwischen Satellit und Navigationssystem.

$$r = c * \Delta t, \text{ mit } c = 300000 \text{ km}/s \tag{2.1}$$

Die Abstände können, wie in Abbildung 2.2 zu sehen ist, als Radien von Kugeln, die um die Satelliten liegen, betrachtet werden. Die Schnittfläche zweier Kugeln bildet einen Kreis. Erst mit Hilfe einer dritten Kugel kann die Kreisschnittfläche auf zwei Punkte reduziert werden. Dies erklärt die Notwendigkeit, warum für eine Lagebestimmung mindestens drei Satelliten genutzt werden müssen. Von den zwei bestimmten Punkten markiert jener, der auf der Erdoberfläche liegt (Schnittpunkt 1 in der Grafik 2.2), mit seiner Lage den Standort des Navigationssystems.

Die Präzision der Standortbestimmung und ihrer Durchführung ist abhängig von einer genauen Zeitmessung. Nach [Rot05] kann ein Unterschied von 1 μsec eine Abweichung von 300 Metern verursachen. Aus diesem Grund verfügen die Satelliten über Atomuhren. Ein Problem stellt die Synchronisation zwischen der Zeitangabe des Navigationssystems und der Systemzeit dar. Durch Hinzunahme eines vierten Satelliten kann die Zeitdifferenz zwischen der Zeitangabe des Navigationssystems und der Systemzeit ausgeglichen werden.

Die Bestimmung der Positionskoordinaten auf der Erdoberfläche erfolgt mit Hilfe der Kenntnis über den Standort der Satelliten. Das Navigationssystem verfügt über ein ständig aktualisiertes Verzeichnis, das Auskünfte über alle aktiven Satelliten und derren Position gibt. Durch die berechneten Abstände und die Standortangaben der Satelliten kann das Navigationssystem auf die eigene Position zurüchschließen.

Die Vorteile dieses Systems liegen in seiner Reichweite. Überall auf der Erde kann auf diese Weise der Standort mit einer hohen Genauigkeit ermittelt werden. Es ist aber nur dann eine präzise Positionsbestimmung möglich, wenn das Signal von mehreren Satelliten empfangen wird. Ein Nachteil liegt in der Signalstärke, die nicht dazu ausreicht Gebäudewände zu durchdringen. Aus diesem Grund ist das System nur für Außenanwendungen geeignet.

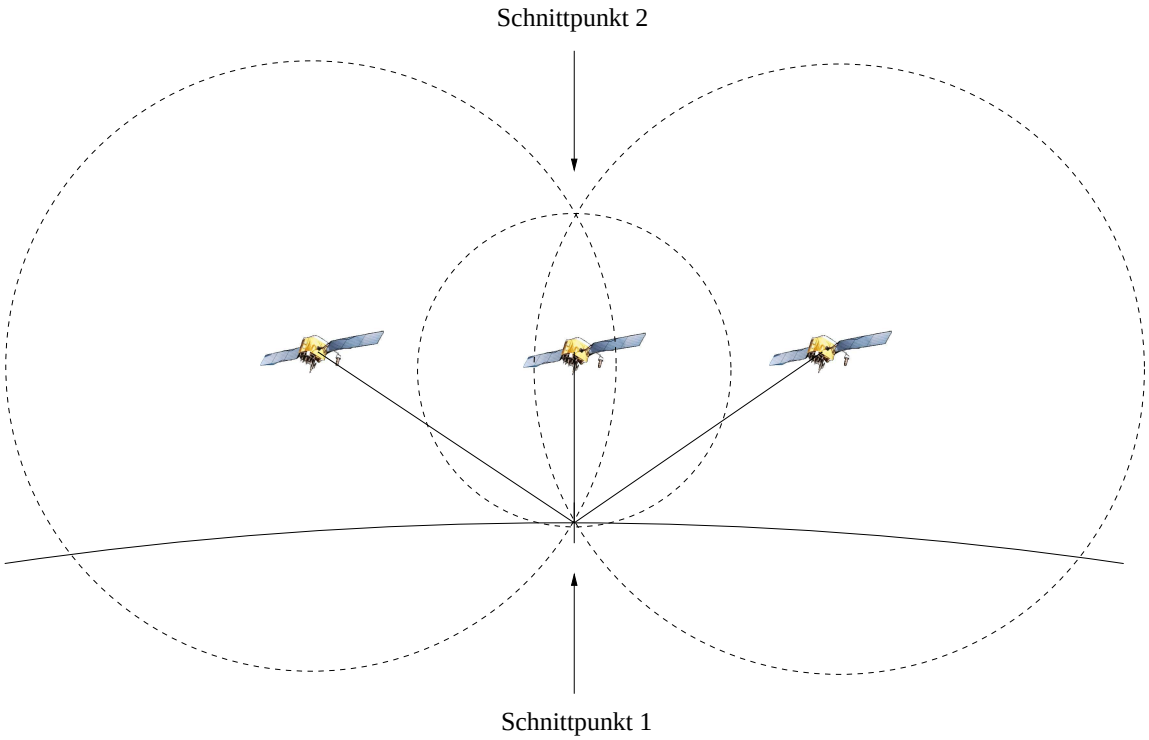


Bild 2.2: Ortsbestimmung durch Satellitennavigation

Beispiele für Satellitennavigation sind das *Global Positioning System* (GPS), *Globalnaya Navigationnaya Sputnikovaya Sistema* (GLONASS) oder das *Europäische Satellitennavigationssysteme*.

2.2.2 Verfahren für Innenanwendung

Die Positionsbestimmungsverfahren für Innenanwendungen arbeiten vorwiegend mit Schallwellen oder elektromagnetischen Wellen. Das Objekt oder eine Person wird mit Hilfe einer Bake[1] gekennzeichnet, die offen sichtbar z. B. auf der Kleidung getragen wird. Die Bake selbst kann ein Sender sein, der eine Benutzerkennung kodiert aussendet. Beim konkreten Einsatz von Infrarot-Baken geschieht die Ausstrahlung der Information alle 15 sec. Infrarotsensoren, die mit einer hohen Stückzahl in der Umgebung installiert sind, empfan-

[1] Kennzeichnung

gen die Signale und geben die Information über den Benutzer an einen Rechner für die Positionsbestimmung (engl. location server) weiter. Dieser bestimmt aus allen empfangenen Sensordaten den Standort des Benutzers.

Wird gefordert, dass die Ortsangaben auch beim Benutzer vorliegen sollen, wird das System so eingerichtet, dass die Infrarot-Baken anstatt der Infrarotsensoren fest in der Umgebung installiert sind und Signale, die ihren Standort bekanntgeben, aussenden. Der Benutzer empfängt diese Signale und reicht sie an den Rechner für die Positionsbestimmung weiter.

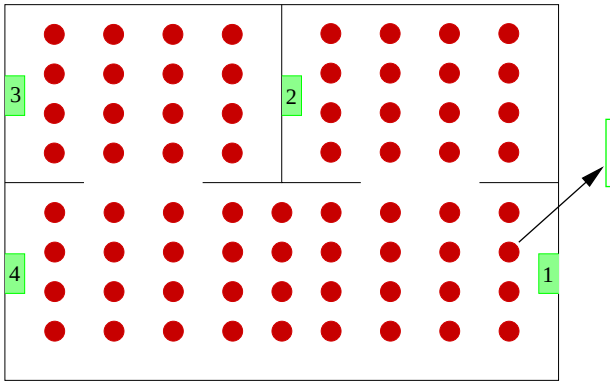
Um die Standortbestimmung eindeutig durchführen zu können, kommt häufig eine Zwei-Wege-Kommunikation zum Einsatz, um eine Authentifizierung des Benutzers durchführen zu können. Damit kann eine Positionsangabe direkt mit einem Benutzer verbunden werden. Mit dieser Erweiterung steigt auch die Anzahl der Verwaltungsrechner. Neben dem Rechner für die Lagebestimmung kommen jeweils ein Rechner für die Benutzerverwaltung, für das Senden von Nachrichten an den Benutzer und für den Zusammenschluss mehrerer Teilsysteme zu einem Gesamtsystem zum Einsatz.

Die geringe Stärke der Infrarotwelle und die daraus resultierende geringe Reichweite des Signals begründet den Einsatz von Funkwellen. Die Position eines Objektes wird durch die Messung der Signalstärke bestimmt. Die Sensoren, die das ausgesendete Funksignal empfangen, übermitteln dem Rechner für die Positionsbestimmung die Signalstärke. Dieser berechnet aus allen empfangenen Stärkeangaben die Koordinaten des Ortes.

Eine im Vegleich zu Funkwellen höhere Genauigkeit wird mit dem Einsatz von Ultraschall-Verfahren erreicht. Auf Anfrage eines Servers wird vom Benutzer ein Ultraschall-Signal in die Umgebung ausgestrahlt. Mehrere Empfänger, die in einem Abstand von 1.2 m an der Decke angebracht sind, empfangen das Signal und reichen es an den Lageserver weiter. Die daran anschließende Positionsbestimmung ähnelt der Berechnung, die auch bei der Satellitennavigation durchgeführt wird.

2.2.3 Universell Einsetzbare Verfahren

Für die Positionsbestimmung, sowohl für Innen- als auch für Außenanwendungen, werden drahtlose Netzwerke genutzt, da sie meistens in vielen Gebäuden vorinstalliert sind. Als Grundtechnik kommt dabei die Zellortung zum Einsatz. Bevor eine Standortbestimmung stattfinden kann, müssen in einer Lernphase Messungen der Signalstärke von festgelegten Wegpunkten, die in der Abbildung 2.3 durch schwarze Punkte markiert sind, zu mehreren Basisstationen durchgeführt werden. Die ermittelten Stärken werden anschließend in einer Tabelle festgehalten.



X/m	Y/m	SS1/dBm	SS2/dBm	SS3/dBm	SS4/dBm
...	...	...	...	...	...
1.0	2.0	12	3	12	4
2.5	4.0	2	14	25	7
4.0	1.5	7	8	16	25
...	...	...	...	...	...

Bild 2.3: Aufbau einer Positionstabelle für die Positionsbestimmung im WLAN

Die Positionsbestimmung wird in mehreren Schritten durchgeführt. Im ersten Schritt wird die Funkzelle, in der sich der Benutzer befindet, bestimmt. Dabei wird die aktuell gemessene Signalstärke mit den aus der Lernphase ermittelten Werten verglichen. Der kleinste Abstand zwischen den Stärkewerten bestimmt die grobe Position des Benutzers. Danach erfolgt eine genauere Eingrenzung, indem entweder Verfahren zur Laufzeitmessung oder Winkelmessungen durchgeführt werden. Die Genauigkeit der Lagebestimmung ist abhängig von der Anzahl der in der Lernphase bestimmten Wegpunkten.

2.3 Bildbasierte Positionsbestimmung: 2-D/3-D-Registrierung

Das Grundanliegen der Bildregistrierung ist die Bestimmung der Parameter eines Bewegungsmodells[2], um anschließend z. B Merkmale eines Bildes in einem anderen abbilden zu können. Wie die Bezeichnung „Bildregistrierung“ ebenfalls andeutet, geschieht die Berechnung über die Verwendung von Bildern. So soll durch Anwendung geeigneter Verfahren herausgefunden werden, wie eine Bewegung, die z. B. in einer Bildsequenz festgehalten wurde, zu beschreiben ist.

Das für die Lösung dieser Aufgabe verwendete Verfahren bestimmt die Art der Registrierung. Im Falle der 2-D/3-D-Registrierung werden die Parameter des Bewegungsmodells, das aus den affinen Transformationen Rotation und Translation besteht, über die Registrierung einer 2-D-Ansicht mit einem 3-D-Modell ermittelt. Dabei werden Position und Orientierung einer Kamera simuliert und eine 2-D-Ansicht auf das 3-D-Modell generiert. Die darauffolgende Bildregistrierung soll die Parameter der Positionsangabe bewerten und gegebenenfalls abändern, so dass eine bessere Übereinstimmung zwischen der Kameraaufnahme und dem synthetisch erzeugten Bild erzielt wird.

Die Neubestimmung der Parameter des Bewegungsmodells im Hinblick auf die Verbesserung der Bildübereinstimmung zeigt, das es sich im mathematischen Sinne um ein Optimierungsproblem handelt, das wie folgt ausgedrückt werden kann: Seien I_1 und I_2 zwei Bildfunktionen mit demselben Wertebereich. Sei $B_{\boldsymbol{x}}$ ein Bewegungsmodell mit dem Parametervektor $\boldsymbol{x}$. Suche den Parametervektor $\boldsymbol{x}$ des Bewegungsmodells B, der die Abstandsfunktion S zwischen den beiden Bildfunktionen I_1 und I_2 in Bezug auf das Bewegungsmodell $B_{\boldsymbol{x}}$ optimiert. Es gilt die in Formel 2.2 dargestellte Gleichung zu lösen.

$$\boldsymbol{x} = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, S(I_1, B_{\boldsymbol{x}}(I_2)) \tag{2.2}$$

[2]Zusammenfassung aller möglichen Transformationen, die auf ein Objekt wirken können. In einem Bewegungsmodell können z. B. alle affinen Transformation wie Rotation, Translation und Skalierung enthalten sein.

Wie bereits erwähnt besteht das Bewegungsmodell aus der Rotation und Translation im dreidimensionalen Raum. Daraus folgt, dass der Parametervektor $\boldsymbol{x}$, der die Werte für beide Bewegungen beinhaltet, aus insgesamt sechs Größen besteht.

$$\boldsymbol{x} = (r_x, r_y, r_z, t_x, t_y, t_z), \text{mit } \boldsymbol{x} \in \mathbb{R}^6 \tag{2.3}$$

Die praktische Durchführung des beschriebenen Optimierungsproblems wird im Falle der 2-D/3-D-Registrierung in drei Arbeitsschritten durchgeführt, wobei die Suche nach dem Optimum, wie in Abbildung 2.4 dargestellt, durch eine ständige Wiederholung mehrerer Teilarbeitsschritte ausgeführt wird. In der Initialisierungsphase werden das Kamerabild, gegen das registriert werden soll, und die Initialparameter des Bewegunsmodells, die z. B. durch das GPS ermittelt wurden, dem System zugeführt. Anschließend beginnt der Optimierungsvorgang, indem in einem vorher eingeschränkten Suchraum, der häufig durch die vorher ermittelten Ungenauigkeiten der Positions- und Orientierungssensoren berechnet wird, durch mehrfaches Generieren einer Ansicht, der Berechnung der Abstandsmaße zwischen den Bildern und erneuten Schätzung der Parameter für Rotation und Translation nach dem Parametervektor gesucht wird, der in den Abstandsmaßen den geringsten Wert zwischen den Vergleichsbildern bewirkt. Die Abstandsmaße stellen die zu optimierende Zielfunktion dar. Ist ein nahezu optimaler Parametervektor gefunden, ist die 2-D/3-D-Registrierung erfolgreich abgeschlossen.

Die Einschränkung des Suchraums ist für den Registrierungsvorgang unabdinglich, da sonst in einem sechsdimensionalen und kontinuierlichen Raum optimiert werden muss. Dies könnte dazu führen, dass, wenn kein Abbruchkriterium außer dem optimalen Parametervektor gewählt wird, die Suche theoretisch unendlich lange fortgesetzt werden kann.

Der eben beschriebene Ablauf der 2-D/3-D-Registrierung macht deutlich, dass der Optimierungsvorgang im Wesentlichen durch zwei Berechnungsschritte gebildet wird. Das sind zum einen die Ermittlung der Abstände zwischen den Bildfunktionen und zum anderen die Neubestimmung des Parametervektors $\boldsymbol{x}$. Im folgenden Text wird auf eine Reihe von verschiedenen Möglichkeiten der Abstandsberechnung von Bildfunktion eingegangen. Anschließend erfolgt eine Beschreibung von Techniken der Optimierung.

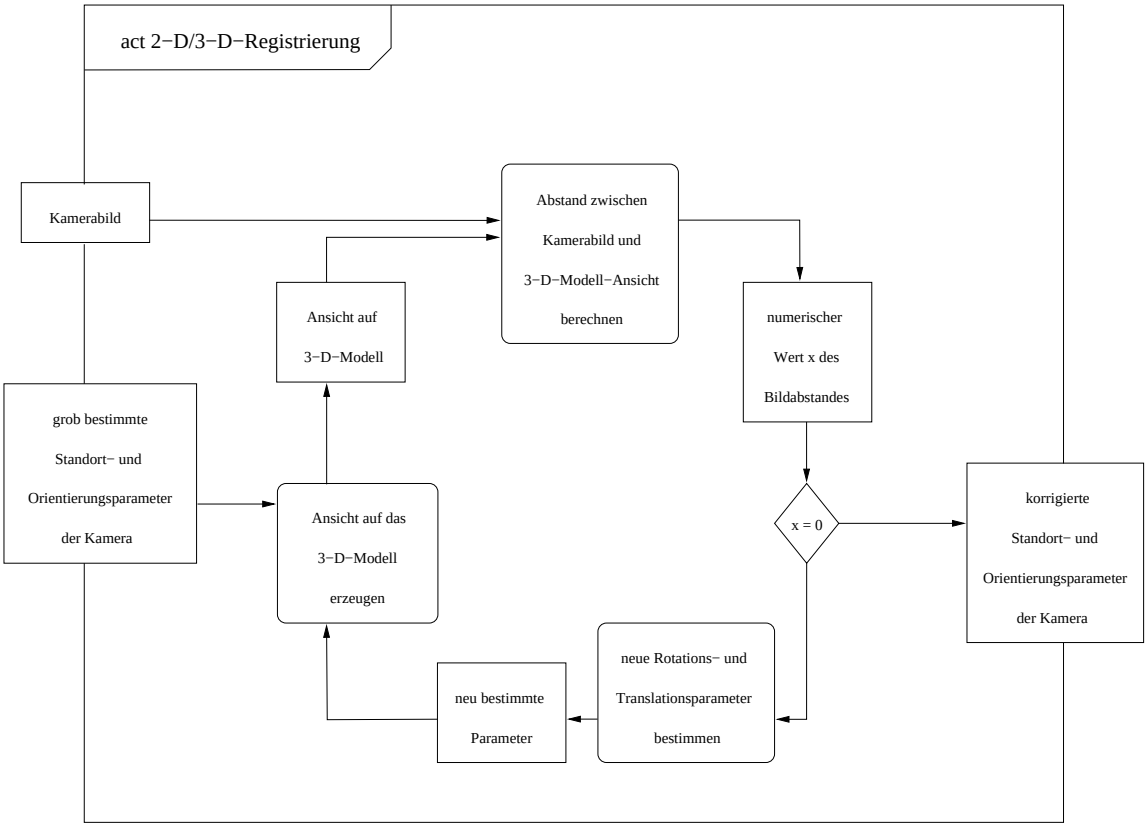


Bild 2.4: Abfolge der Arbeitsschritte bei der 2-D/3-D-Registrierung

2.3.1 Abstandsmaße

Die Bildabstandsmaße bilden den Kern der 2-D/3-D-Registrierung. Ein Abstandsmaß gibt Auskunft über die Ähnlichkeit zweier Bildfunktionen. Die Art eines Maßes wird bestimmt durch die Information, die in den Bildfunktion verglichen werden soll. Grundsätzlich wird zwischen merkmalsbasierten und intensitätsbasierten Abstandsmaßen unterschieden.

Merkmalsbasiert

Ein merkmalsbasiertes Abstandsmaß ist eine Distanzfunktion zwischen zwei aus den Vergleichsbildern extrahierten Merkmalsmengen $F(I_1)$ und $F(I_2)$. Je nach vorangegangener

Festlegung eines Merkmals kann die Menge **regionsbasiert**, **linienbasiert** oder **punktbasiert** sein. Damit entscheidet sich auch die Wahl der für die Extrahierung benötigten Vorverarbeitungsstufe. Ist die Merkmalsmenge regionsbasiert, wird mit sogenannten Segmentierungen gearbeitet. Im Falle einer punkt- oder linienbasierten Merkmalsextrahierung kommen Kanten- oder Punktdetektoren zum Einsatz.

Die daran anschließende Berechnung eines Abstandsmaßes setzt nun die aus den Bildern extrahierten Charakteristika in Beziehung, so dass eine neue Menge $M(I_1, I_2)$ von Zuordnungen entsteht. Diese Menge kann wie in Formel 2.4 beschrieben werden.

$$M(I_1, I_2) = \{(m_i, n_j) | m_i \in F(I_1) \wedge n_j \in F(I_2)\} \tag{2.4}$$

Auf der Basis der Zuordnungsmenge M muss im Falle einer Bildregistrierung das jeweilige Abstandsmaß optimiert werden. Das bedeutet es werden solange Merkmalsmengen $M(I_1, I_2)$ gebildet, bis das Abstandsmaß einen Wert erreicht, der auf eine hohe Übereinstimmung der zugeordneten Merkmale hindeutet. Die Abstandsmaße selbst sind meist von geometrischer Natur, d. h. sie setzen die geometrische Anordnung der segmentierten Merkmale in Beziehung. Es werden aber auch sogenannte Korrelationsmaße aus der Signalverarbeitung verwendet, um die Zuordnungsmenge zu bilden. Egal welches merkmalsbasierte Abstandsmaß Verwendung findet, es wird in allen Fällen ein meist sehr aufwändiger Vorverarbeitungsschritt benötigt, der darüber hinaus, je nach Qualität des verwendeten Verfahrens, auch die Genauigkeit der anschließenden Abstandsberechnung beeinflussen kann. Aus diesem Grund wird, so auch im Fall dieser Arbeit, sehr häufig auf die Verwendung dieser Art der Abstandsbestimmung verzichtet, da der Zeitfaktor in vielen Anwendungen eine große Rolle spielt und das Hauptaugenmerk häufig auf einer schnellen Positionsbestimmung liegt.

Merkmalslos/Intensitätsbasiert

Die intensitätsbasierten Abstandsmaße finden aufgrund der weniger aufwändigen Vorverarbeitungsschritte und der damit verknüpften Zeiteinsparung öfter Verwendung als die Merkmalsbasierten. Sie stellen sogar nach [Fel05] den Stand der Technik in der medizinischen Bildverarbeitung dar. Alle Abstandsmaße, die dieser Gruppe zugeordnet wer-

den, arbeiten direkt auf den Funktionswerten der Bildfunktionen, jedoch kann es, je nach verwendeten Vergleichsverfahren zu einer Neuinterpretation kommen. So wird z. B. die Bildfunktion bei den statistischen Abstandsmaßen als Wahrscheinlichkeitsfunktion angenommen. Insgesamt lassen sich drei unterschiedliche Typen von intensitätsbasierten Abstandsmaßen ausmachen. Diese wären:

- direkt
- statistisch
- räumlich

Die am einfachsten zu berechnenden Maße sind in der Gruppe der *direkt-intensitätsbasierten* Verfahren zu finden, da sie aufgrund der Verwendung der eigentlichen Bildfunktion keinen Vorverarbeitungsschritt benötigen. Die *statistisch-intensitätsbasierten* Abstandsmaße verwenden die Bildfunktion als eine Wahrscheinlichkeitsdichte und benötigen als Vorverarbeitungsschritt die Bildung eines Histogramms, in dem das Vorkommen der einzelnen Bildintensitäten erfasst wird. Bei den *räumlich-intensitätsbasierten* Vergleichsverfahren erfolgt bei einigen Abstandsmaßen im ersten Berechnungsschritt eine Berechnung der Gradientenbilder in horizontaler und vertikaler Richtung, auf denen anschließend die Abstandsberechnung durchgeführt wird.

Im nachfolgenden Text werden die Untergruppen der intensitätsbasierten Abstandsmaße im Einzelnen beschrieben. Dabei wird auch versucht, die Vor- und Nachteile der Abstandsmaßtypen, die bei der Verwendung verschiedener Bilddatensätze auftreten können, zu benennen und zu beschreiben.

Direkt-intensitätsbasierte Abstandsmaße

Summe der Intensitätsdifferenzen: Dieser Typ der intensitätsbasierten Bildabstandsmaße arbeitet direkt auf den Werten der Bildfunktionen. Die Berechnung des Abstandes erfolgt durch paarweise Differenzbildung der Intensitätswerte an derselben Position im

Bild. Diesen Berechnungsablauf zeigt z. B. das Abstandsmaß *Summe der quadratischen Abstände (SSD[3])*, das nach Formel 2.5 berechnet wird.

$$S_{SSD}(I_1, I_2) = \frac{1}{|T|} \sum_{(i,j)\in T} (I_1(i,j) - I_2(i,j))^2 \tag{2.5}$$

Die Größe $|T|$ bezeichnet die Anzahl der Pixel im Überlappungsbereich der beiden Bilder. Durch die Quadrierung der berechneten Differenzen werden negative Werte vermieden. Eine anderen Möglichkeit für die Vermeidung negativer Differenzen zeigt sich im Abstandsmaß *Summe der absoluten Differenzen (SAD[4])*. Hier werden, wie die Formel 2.6 zeigt, die Beträge der Differenzen aufsummiert und anschließend durch die Anzahl der Pixel im Überlappungsbereich dividiert. Des Weiteren wird duch die Verwendung der Absolutwerte, anstatt der quadratischen Abstände, vermieden, dass kleine Differenzen vergrößert werden.

$$S_{SAD}(I_1, I_2) = \frac{1}{|T|} \sum_{(i,j)\in T} |(I_1(i,j) - I_2(i,j))| \tag{2.6}$$

Neben der Quadrierung und der Bildung der Absolutwerte findet nach [Wei03] auch häufig die *normalisierte quadratische Differenz* $x^2/1 + x^2$ Verwendung in einem *direkt-intensitätsbasierten* Abstandsmaß . Mit dieser Form soll vermieden werden, dass das Auftreten von großen Differenzen zu starken Einfluss auf das Ergebnis hat, da alle Ergebnisse im Intervall $[0, 1]$ liegen. Als Beispiel wird in [Wei03] das in Formel 2.7 gezeigte Abstandsmaß genannt.

$$S_{NQD}(I_1, I_2) = \sum_{(i,j)\in T} \frac{(I_1(i,j) - I_2(i,j))^2}{\sigma^2 + (I_1(i,j) - I_2(i,j))^2} \tag{2.7}$$

Zusammenfassend lässt sich nach [Wei03] sagen, dass die *direkt-intensitätsbasierten* Abstandsmaße, die über die Aufsummierung der Differenzen arbeiten, sehr stabil auf Bilddatensätzen arbeiten, die dieselbe Modalität aufweisen, da in diesem Fall das Auftreten von großen Differenzen, die das Abstandsmaß stören können, sehr selten ist.

[3] SSD = sum of square differences

[4] SAD = sum of absolute differences

Korrelation: Die Korrelation wird nach [Wei03] in der Mathematik oder Informatik immer dort verwendet, wo man in einer einfachen und schnellen Art und Weise die Relation zwischen zwei Datenmengen auswerten will. In der Bildregistrierung versucht man über die Bestimmung der Korrelation Rückschlüsse auf die lineare Abhängigkeit von Pixelwerten zu ziehen, die sich im Falle eines Bildabstandsmaßes, das über die Korrelation gebildet wird, in einem großen Wert zeigt. Ein Beispiel für diese Art von Abstandsmaß ist die *Kreuzkorrelation (CC*[5]*)*. Das Abstandsmaß, das nach Formel 2.8 berechnet wird, summiert die Produkte der Differenzen zwischen einem Wert der Bildfunktion $I(i,j)$ und dem Mittelwert $\bar{I}$ der Bildfunktion auf.

$$S_{CC}(I_1, I_2) = \sum_{(i,j)\in T} (I_1(i,j) - \bar{I}_1)(I_2(i,j) - \bar{I}_2) \tag{2.8}$$

Ein weiteres Beispiel für ein mit der Korrelation gebildetes Abstandsmaß ist die *normalisierte Kreuzkorrelation (NCC*[6]*)*. Es wird ähnlich der Kreuzkorrelation berechnet jedoch wird das Produkt darüber hinaus mit der Standardabweichung normalisiert. Die Formel 2.9 zeigt die Berechnung der *normalisierten Kreuzkorrelation*.

$$S_{NCC}(I_1, I_2) = \frac{1}{\sigma_1}\frac{1}{\sigma_2} \sum_{(i,j)\in T} (I_1(i,j) - \bar{I}_1)(I_2(i,j) - \bar{I}_2) \tag{2.9}$$

$$\sigma = \sqrt{\sum_{(i,j)\in T} (I(i,j) - \bar{I})^2}$$

Mit Hilfe der Normalisierung soll das Abstandsmaß nach [Fel05] unempfindlich gegenüber Helligkeitsschwankungen gemacht werden. In [Wei03] wird darauf hingewiesen, dass je nach Modalität der Bildfunktionen Abwandlungen der *normalisierten Kreuzkorrelation* auftreten können.

Statistisch-intensitätsbasierte Abstandsmaße

Eine Größe, die bei der Verwendung von statistischen Abstandsmaßen eine wichtige Rolle

[5] CC = cross correlation

[6] NCC = normalised cross correlation

spielt, ist die Entropie. Der Wert der Entropie kann, je nach Sichtweise unterschiedlich aufgefasst werden. In [Fel05] wird die Entropie als ein Maß für die Ungenauigkeit angesehen, mit der ausgedrückt werden soll, wie gut das Auftreten eines Grauwertes v in der Bildfunktion vorhergesagt werden kann. In diesem Fall liegt der Focus auf der Betrachtung eines einzelnen Wertes. In anderen Arbeiten wie [PH03], ist vom mittleren Informationsgehalt eines Signals die Rede, wobei bei dieser Betrachtungsweise alle in einem Signal vorkommenden Werte einbezogen werden.

Was alle Interpretationsformen gemeinsam haben ist, dass ein hoher Entropiewert für das Auftreten vieler verschiedener Werte steht, und ein niedriger Wert auf den entgegengesetzten Fall hindeutet. Im praktischen Anwendungsfall der Bildverarbeitung heißt das, dass z. B. ein Grauwertbild, das aus vielen verschiedenen Grauwertstufen besteht, immer eine höhere Entropie aufweist, als ein binarisiertes Bild, das nur aus den Werten für Schwarz und Weiß besteht. Generell wird die Entropie nach Formel 2.10 berechnet.

$$H(B) = -\sum_{x_i \in S} p(x_i) \log p(x_i) \tag{2.10}$$

Der Term $-\log p(x_i)$ wird nach [PH03] in der Informationstheorie als Informationsgehalt eines Wertes x_i bezeichnet, während $p(x_i)$ die Wahrscheinlichkeit angibt, mit der ein Wert, z. B. in einem Bildsignal B, auftritt.

Um nun ein statistisches Bildabstandsmaß anwenden zu können, muss als Vorverarbeitung das Histogramm beider Vergleichsbilder ermittelt werden. Danach werden diese, je nach Vergleichsverfahren, auf unterschiedliche Art und Weise in Beziehung zueinander gesetzt. So wird z. B. in einem Fall der Bildabstandsmessung mit Hilfe der *Verbundentropie (JE[7])* die statistische Abhängigkeit zwischen zwei Bilddatensätzen untersucht und bewertet. Dabei wird die Verbundentropie aus den Histogrammen h_{I_1} und h_{I_2} der beiden Bildfunktionen I_1 und I_2 nach Formel 2.11 berechnet.

$$H(I_1, I_2) = -\sum_{(i,j) \in h_{I_1} \times h_{I_2}} p(i,j) \log p(i,j) \tag{2.11}$$

[7] JE = joint entropy

Die Größe $p(i,j)$ bezeichnet die Wahrscheinlichkeit, mit der der Wert $i \in h_{I_1}$ und der Wert $j \in h_{I_2}$ im Verbundhistogramm $h_{I_1} \times h_{I_2}$, das aus den Histogrammen der Vergleichsbilder ermittelt wird, auftreten. Für die Verbundentropie $H(I_1, I_2)$ gilt die in Formel 2.12 dargestellte Ungleichung.

$$H(I_1, I_2) \leq H(I_1) + H(I_2) \tag{2.12}$$

Für die statistische Abhängigkeit der beiden Bildfunktionen kann nun folgende Aussage getroffen werden: Sind I_1 und I_2 statistisch unabhängig, dann gilt für die Wahrscheinlichkeit der Werte $i \in h_{I_1}$ und $j \in h_{I_2}$ die Formel 2.13.

$$p(i,j) = p(i)p(j) \tag{2.13}$$

Daraus folgt für die Verbundentropie die Gleichung 2.14

$$H(I_1, I_2) = H(I_1) + H(I_2) \tag{2.14}$$

Ist die Bildfunktion I_1 statistisch unabhängig von der Bildfunktion I_2, dann ist die Verbundentropie maximal. Daraus folgt nun für den praktischen Anwendungsfall der Bildregistrierung, dass eine große Übereinstimmung der Bildfunktionen nur dadurch zu erreichen ist, dass die Verbundentropie minimiert wird.

Räumlich-intensitätsbasierte Abstandsmaße

Bei den *räumlich-intensitätsbasierten* Abstandsmaßen werden vorrangig Nachbarschaftsbeziehungen von Pixeln betrachtet. Diese Betrachtung kann entweder intensitätsbasiert oder gradientenbasiert geschehen. Ein Beispiel für den intensitätsbasierten Vergleich ist mit der sogenannten *Musterintensität (PI*[8]*)* gegeben. In ihr werden die Differenzen zu den Nachbarschaftspixeln im Differenzbild der beiden Bildfunktionen aufsummiert. Die Berechnung erfolgt in zwei Schritten. Als Ersters wird das Differenzbild gemäß Formel 2.15 berechnet.

[8] PI = pattern intensity

$$I_{diff} = I_1 - sI_2 \tag{2.15}$$

Der Skalierungsfaktor s soll einen Ausgleich der Helligkeitsunterschiede zwischen den Bildern bewirken ([Fel05]). Im zweiten Schritt werden die Differenzen zwischen einem ausgewählten Pixel und seinen Nachbarn berechnet. Die Nachbarn werden mit Hilfe einer runden Maske mit Radius r, für die die in Formel 2.16 dargestellte Bedingung gilt, ermittelt.

$$r^2 > (i-v)^2 + (j-w)^2, \text{ mit } (i,j),(v,w) \in T \tag{2.16}$$

Die Endgültige Berechnung der *Musterintensität* erfolgt gemäß der Formel 2.17.

$$S_{PI}(I_1, I_2) = \frac{1}{|T|} \sum_{(i,j)\in T} \sum_{(v,w)\in T} \frac{\sigma^2}{\sigma^2 + (I_{diff}(i,j) - I_{diff}(v,w))^2} \tag{2.17}$$

Wie bei den direkt-intensitätsbasierten Abstandsmaßen werden auch bei der *Musterintensität* negative Differenzen durch eine Quadrierung vermieden. Außerdem werden nach [Fel05] mit der Konstanten σ, für die in [PWL+98] der empirisch ermittelte Wert $\sigma = 10$ vorgeschlagen wird, versucht, Abweichungen in der Intensität zu kompensieren. Des weiteren wird in [PWL+98] für die Größe der Maske der Wert $r = 3$ nahegelegt.

Die gradientenbasierten Abstandsmaße verlangen als Vorverarbeitungsstufe die Berechnung des horizontalen und vertikalen Gradientenbildes. Mit der Bildung der Gradientenbilder werden aus den Bildfunktionen tiefe Frequenzen herausgefiltert, so dass in den Vergleichsberechnungen nur die Anteile berücksichtigt werden, die eine hohe Frequenz aufweisen. Daraus resultiert ein wesentlicher Nachteil, den man bei der Verwendung dieser Art der Abstandsbildung bedenken muss. Bei sehr homogenen Bilddaten weisen die Gradienten den Wert null auf und sind somit für die Bildung eines Abstandswertes in diesem Fall ungeeignet.

Ein weiterer Nachteil, der auch bei der intensitätsbasierten Betrachtung eine Rolle spielt, resultiert aus dem Auftreten von Bildstörungen heraus, die z. B. durch lichtreflektierende Flächen, wie Metall, entstehen können. In diesem Fall würden Gradientenwerte mit in

die Berechnung einbezogen, die in der ungestörten Bildfunktion nicht vorhanden wären. Ein Beispiel für ein gradientenbasiertes Abstandsmaß ist die *Gradientendifferenz (GDI*[9]*)*. Zuerst erfolgt die Berechnung des Gradientendifferenzbildes nach Formel 2.18. Wie bei der *Musterintensität* wird auch bei diesem Vergleichsverfahren über den Skalierungsfaktor s versucht einen Ausgleich der Helligkeitsunterschiede zu bewirken. Im zweiten Schritt wird der finale Wert des Abstandsmaßes nach Formel 2.19 bestimmt.

$$I_{diff_h}(i,j) = \frac{\partial I_1}{\partial i} - s\frac{\partial I_2}{\partial i} \tag{2.18}$$

$$I_{diff_v}(i,j) = \frac{\partial I_1}{\partial j} - s\frac{\partial I_2}{\partial j}$$

$$S_{GD}(I_1, I_2, \sigma_h, \sigma_v, s) = \sum_{(i,j)\in T} \frac{\sigma_h}{\sigma_h + I_{diff_h}(i,j)^2} + \sum_{(i,j)\in T} \frac{\sigma_v}{\sigma_v + I_{diff_v}(i,j)^2} \tag{2.19}$$

Nach [Pen99] soll mit den Gewichtungsfaktoren σ_h und σ_v das Auftreten von Rauschen und anderen kleineren Schwankungen der Bildfunktionen ausgeglichen werden.

2.3.2 Demokratische Integration

Die Wahl eines Abstandsmaßes, egal, ob es sich um ein merkmals- oder intensitätsbasiertes handelt, setzt Kenntnisse über die Modalität und verwendbare Information der zu vergleichenden Bilddatensätze voraus. So eignen sich z. B. gradientenbasierte Abstandsmaße sehr gut für die Verarbeitung von sogenannten Kantenbilder. Bei Bildern mit stark homogenen Pixelwerten ist die Benutzung des Gradienten für den Informationsvergleich ungeeignet, da aufgrund der ähnlichen Intensitätswerte auch die Steigung der Ableitung der Bildfunktion sehr gering ist. Für diesen Bildtypus wäre die Verwendung von statistischen oder direkten intensitätsbasierten Abstandsmaßen eher geeignet. Gerade im Anwendungsfeld der Positionsbestimmung im Außenbereich treten abwechselnd die beschriebenen Fälle auf.

Die vorzeitige Festlegung auf eine bestimmte Bildmodalität ist in vielen Anwendungen der Bildregistrierung unerwünscht, und man ist bestrebt, das Bildverarbeitungssystem so

[9] GD = gradient difference

einzurichten, dass es sich unabhängig zum verwendeten Bildmaterial verhält. Das setzt voraus, dass das System automatisch erkennt, welche Information in der Bildfunktion für einen Vergleich verwendet werden kann. Darüber hinaus müssen die Vergleichsverfahren, die eine ungeeignete Bildeigenschaft für den Vergleich verwenden, in ihrer Einwirkung auf das Gesamtergebnis geschwächt werden.

Des Weiteren wird immer verlangt, dass ein System ausfallsicher auf Störungen reagiert. Eine Beeinträchtigung der Bildinformation kann in Form von Rauschen auftreten und führt im konkreten Fall der Bildregistrierung bei der Berechnung einiger Abstandsmaße zu fehlerhaften Ergebnissen. Auch hierfür sollte vom System die Möglichkeit der Fehlervermeidung gegeben sein.

Eine Lösung liegt in der Verwendung mehrerer, auf unterschiedlicher Bildinformation arbeitender Vergleichsverfahren, die je nach verwendeten Bildtypus stärker in das Gesamtergebnis einbezogen werden. Eine Möglichkeit der automatischen Auswahl der geeigneten Abstandsmaße bietet das Verfahren der *Demokratischen Integration*. Die im Artikel [TvdM01] vorgestellte Methode verwendet parallel mehrere verschiedene Bildabstandsmaße. Danach werden alle Teilergebnisse der Abstandsmaße in einem Ergebnis zusammengefasst und anschließend die Differenz aller Teilergebnisse zum errechneten Gesamtresultat gebildet. Die über die Zeit ermittelten Differenzen geben Auskunft darüber, wie zuverlässig ein Abstandsmaß mit den zu vergleichenden Bilddatensätzen zurecht kommt. Der Wert der Zuverlässigkeit, der für jedes Vergleichsverfahren berechnet wird, reguliert das Einwirken des Maßes auf das Gesamtergebnis.

Darüber hinaus versucht das Verfahren der *Demokratischen Integration* die Parameter der Abstandsmaße so zu adaptieren, dass es zur größtmöglichen Übereinstimmung aller verwendeten Verfahren kommt ([Fel05]). Dies setzt voraus, dass alle Abstandsmaße den selben Wertebereich besitzen. Für das in dieser Arbeit angestrebte System wird der Wertebereich auf das Intervall $[0, 1]$ festgelegt, und es gilt für alle Maße, dass ein hoher Wert für eine große Übereinstimmung der Bilddatensätze steht.

Bestimmung der Zuverlässigkeiten der Abstandsmaße

Für die Bildregistrierung, die in dieser Arbeit angestrebt wird, werden n unterschiedliche Vergleichsverfahren S eingesetzt. Für den errechneten Abstand eines jeden Abstandsmaßes zwischen zwei Bildfunktionen $I_1(t)$ und $I_2(\boldsymbol{x})$ mit dem Parameter t, der den Zeitpunkt des Bildvergleichs angibt, und dem aktuellen Parametervektor $\boldsymbol{x}$ des Bewegungsmodells, gilt nach Formel 2.20:

$$A_i(\boldsymbol{x}, t) = S_i(I_1(t), I_2(\boldsymbol{x})) \text{ , mit } 0 \leq A_i(\boldsymbol{x}, t) \leq 1 \text{ , und } i = 1...n \tag{2.20}$$

Die Gleichung 2.20 fordert die bereits erwähnte Vereinheitlichung des Wertebereichs aller Abstandsmaße auf das abgeschlossene Intervall $[0, 1]$. Die Einzelwerte aller Abstände an allen Positionen $\boldsymbol{x}$ in $A_i(\boldsymbol{x}, t)$ werden gemäß der Formel 2.21 gewichtet mit der Zuverlässigkeit r_i aufsummiert.

$$R(\boldsymbol{x}, t) = \sum_{i=1}^{n} r_i A_i(\boldsymbol{x}, t) \tag{2.21}$$

Für die Gesamtsumme aller Teilgewichte gilt entsprechend der Formel 2.22, dass sie dem Wert eins entspricht.

$$\sum_{i=1}^{n} r_i = 1 \tag{2.22}$$

Für die in der Bildregistrierung verwendeten Optimierungsverfahren gilt mit der Verwendung des Verfahrens der Demokratischen Integration das Ziel, den Parametervektor $\boldsymbol{x}^*$ zu ermitteln, der das Gesamtergebnis $R(\boldsymbol{x}, t)$ nach Formel 2.23 maximal werden lässt.

$$\boldsymbol{x}^*(t) = \underset{\boldsymbol{x}}{\text{argmax}}\ R(\boldsymbol{x}, t) \tag{2.23}$$

Die Zuverlässigkeit r_i eines einzelnen Vergleichsverfahrens S_i zum Zeitpunkt t wird mit Hilfe der normierten Qualität q_i, die wiederum aus den relativen Qualitäten $\tilde{q}_i$ gebildet

wird, ermittelt. In [TvdM01] werden für die Berechnung der relativen Qualitäten insgesamt fünf Berechnungsformeln vorgeschlagen. Die *Uniform Quality* (UQ) ordnet jedem Vergleichsverfahren den gleichen Wert zu, indem das Ergebnis der Division von eins durch die Anzahl n_S der verwendeten Maße jedem Abstandsmaß zugeordnet wird (Formel 2.24).

$$\tilde{q}_{iUQ}(t) = \frac{1}{n_S} \tag{2.24}$$

Die sogenannte *Raw Saliency*[10] (RS) (Formel 2.25) übernimmt den jeweiligen Wert des Abstandsmaßes an der aktuell optimalen Position $\boldsymbol{x}^*(t)$, während bei der relativen Qualität *Normalized Saliency*, die in Formel 2.26 dargestellt ist, dieser Wert durch die Gesamtsumme der Werte des Abstandsmaßes an allen möglichen Positionen dividiert wird, wobei aufgrund des sechsdimensionalen und kontinuierlichen Raumes die Summe geeignet approximiert werden muss ([Fel05]).

$$\tilde{q}_{iRS} = A_i(\boldsymbol{x}^*(t), t) \tag{2.25}$$

$$\tilde{q}_{iNS} = \frac{A_i(\boldsymbol{x}^*, t)}{\sum_{\boldsymbol{x}'} A_i(\boldsymbol{x}', t)} \tag{2.26}$$

Das vierte relative Gütemaß *Distance to Average Saliency* (DAS) (Formel 2.27) bildet den Wert für die Qualität eines Abstandsmaßes mit Hilfe einer Rampenfunktion g_{ramp}. Im ersten Schritt wird die Differenz zwischen dem Wert des Abstandsmaßes an der aktuellen optimalen Position und dem Durchschnittswert $\overline{A_i(\boldsymbol{x}', t)}$ aller Positionen im Raum berechnet. Danach wird im zweiten Schritt die Differenz d entweder auf den Wert 0, wenn $d \leq 0$ ist, oder im entgegengesetzten Fall auf sich selbst abgebildet. Wie im Falle der *„Normalized Saliency“* so muss auch bei dieser relativen Qualität der Durchschnittswert, der über alle Positionen im Raum gebildet wird, geeignet approximiert werden.

$$\tilde{q}_{iDAS}(t) = g_{\text{ramp}}(A_i(\boldsymbol{x}^*(t), t) - \overline{A_i(\boldsymbol{x}', t)}\ \text{, mit } g_{\text{ramp}}(x) = \begin{cases} 0 : x \leq 0 \\ x : x > 0 \end{cases} \tag{2.27}$$

[10] engl. saliency = das Herausragen

Aus den vier relativen Qualitäten wird eine Variante für alle Abstandsmaße gewählt. Die nach Formel 2.28 zu errechnende normierte Qualität eines Abstandsmaßes ist der Quotient aus der relativen Qualität des jeweiligen Abstandsmaßes und der Summe aller verwendeten relativen Qualitäten.

$$q_i(t) = \frac{\tilde{q}_i(t)}{\sum_{j=1}^{n_S} \tilde{q}_j(t)}, \text{ mit } \sum_{j=1}^{n_S} q_i(t) = 1 \tag{2.28}$$

Die in Formel 2.21 geforderte Zuverlässigkeit r_i ergibt sich nach [Fel05] durch die Betrachtung der Veränderung der Qualität eines Abstandsmaßes über die Zeit. Die daraus resultierende Dynamik der Zuverlässigkeit kann gemäß der Formel 2.29 beschrieben werden, indem der Einfluss der Veränderung der Qualität auf die Zuverlässigkeit zum Zeitpunkt t ins Verhältnis zur Steigung der Zuverlässigkeit $r_i'(t)$ zum Zeitpunkt t gesetzt wird.

$$\tau r_i'(t) = q_i(t) - r_i(t), \text{ mit } r_i'(t) = \frac{r_i(t + \Delta t) - r_i(t)}{\Delta t} \tag{2.29}$$

Wird die Formel für die Berechnung der Steigung $r_i'(t)$ in die Gleichung zur Darstellung der Dynamik der Zuverlässigkeit eingesetzt, so kann nach geeigneter Umformung die Zuverlässigkeit nach Formel 2.30 zum Zeit $t + \Delta t$ berechnet werden. Dabei wird gefordert, dass Δt ein festes Zeitintervall beschreibt.

$$r_i(t + \Delta t) = \frac{\Delta t}{\tau} q_i(t) + (1 - \frac{\Delta t}{\tau}) r_i(t) \tag{2.30}$$

2.3.3 Optimierungsverfahren

Das Ziel der Bildregistrierung ist es, die Parameter, die dem Bewegungsmodell zugrunde liegen, so zu schätzen, dass eine maximale Übereinstimmung der Bilddatensätze, die z. B. durch ein Kamerabild und einer synthetisch erzeugten Ansicht eines 3-D-Modells gegeben sind, erreicht wird. Im Falle der 2-D/3-D-Registrierung heißt das, dass die Parameter für die Orientierung und Positionierung der Kamera in Bezug auf ein 3-D-Modell so zu ermitteln sind, dass eine 2-D-Ansicht auf das 3-D-Modell identisch mit einer Aufnahme der Kamera ist.

Da die Güte der Übereinstimmung zweier Bilder mit Hilfe von n Abstandsmaßen S_i ermittelt wird, gilt es diese in Abhängigkeit zum Vektor $\boldsymbol{x}$, der die Parameter für die Rotation und Translation der Kamera enthält, zu optimieren. Es wird nach dem Argument $\boldsymbol{x}^*$ gesucht, das die Funktion $S_i(I_1(t), I_2(\boldsymbol{x}))$ maximal werden lässt. Die affinen Transformationen des Bewegungsmodells beschreiben Bewegungen im dreidimensionalen Raum. Daraus ergibt sich für den Parametervektor, dass er ein Element des kontinuierlichen und unendlichen Raumes $\mathbb{R}^6$ ist, da die Parameter von zwei Transformationen gesucht sind. Somit wird die Suche nach den optimalen Werten weitestgehend unmöglich gemacht, wenn die Optimierung nicht von Zufällen abhängen soll, denn ein unendlicher Raum kann nicht in endlicher Zeit abgesucht werden. Aus diesem Grund gilt es, den Suchraum auf ein Intervall einzuschränken. Für die Optimierung stellt sich nun ein nichtlineares Optimierungsproblem mit Nebenbedingungen.

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmax}}\ S_i(I_1(t), I_2(\boldsymbol{x})),\ \text{mit}\ i = 1...n \tag{2.31}$$

Die Suche nach den optimalen Werten kann auf zwei verschiedene Weisen durchgeführt werden. Aus der Mathematik ist bekannt, dass Extrema einer differenzierbaren Funktion f an den Stellen zu finden sind, an denen die erste Ableitung $f' = 0$ ist. Daraus resultiert die erste Möglichkeit der Optimumssuche, indem nach Nullstellen in der ersten Ableitung der Funktion gesucht wird.

Bei nichtlinearen Optimierungsverfahren, die diese Grundstrategie anwenden, spricht man von Gradientenverfahren. Es stellt sich aber sehr häufig das Problem, dass in der Praxis die partiellen Ableitungen der Kostenfunktion nicht gebildet werden können. In solchen Fällen wird schnell aus den eigentlich effizienten und zuverlässigen Verfahren eine ineffiziente und ungenaue Lösung, da der Gradient, z. B. durch Bildung finiter Differenzen, approximiert werden muss, was von Anfang an zu Ungenauigkeiten im Ergebnis führt.

In solchen Problemfällen wird eher die zweite Variante der Optimumssuche angewendet, nämlich die direkte Verwendung der Funktionswerte. Durch mehrfache Anwendung von Vergleichsoperationen wird bei dieser Art der Suche die Stelle, an der das Optimum zu finden ist, iterativ eingeschränkt, bis der Suchraum im mathematischen Fall auf einen finalen Punkt verkleinert ist.

Darüber hinaus muss bei jedem Optimierungsproblem geklärt werden, nach welcher Art Optimum man die Suche ausrichten will. Es muss zu Beginn klar sein, ob nach einem Minimum oder Maximum gesucht wird, und ob dieses das globale oder ein lokales Extremum ist. Je nach Festlegung wird dann zwischen einer globen oder lokalen Optimierung unterschieden.

In der praktischen Anwendung der Bildregistrierung hat es sich in [Kub06] bewährt, beide Arten der Optimierung über einen **Multiresolutionsansatz** zu verbinden. Dabei wird ein globales Verfahren auf einer niedrigen Bildauflösungsstufe verwendet, um schnell eine grobe Bestimmung möglicher globalen Extrema zu bekommen. Diese Ergebnisse werden anschließend von einem lokalen Verfahren auf einer höheren Auflösungsstufe präzisiert. Die Graphik 2.5 veranschaulicht das beschriebene Verfahren.

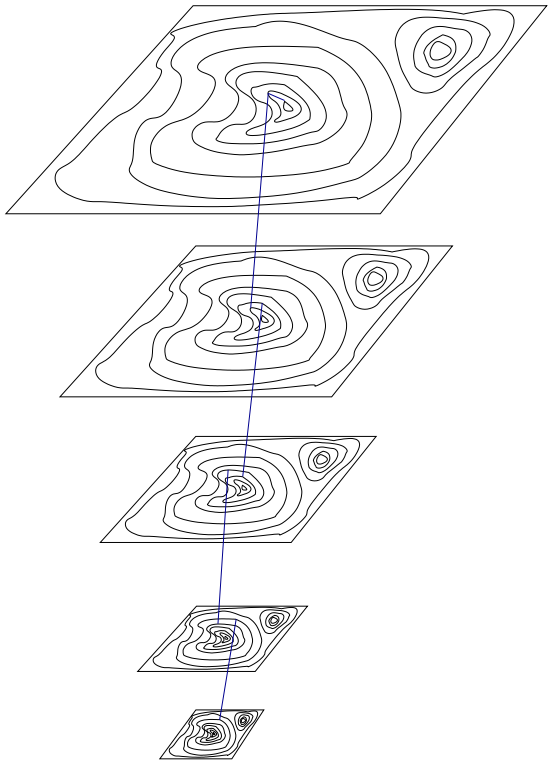


Bild 2.5: Verschiedene Auflösungsstufen der Kostenfunktion, um genauer das globale Optimum zu bestimmen

Wird im Registrierungsvorgang zusätzlich das Verfahren der *Demokratischen Integration* verwendet, müssen, wie im Abschnitt 2.3.2 bereits erwähnt, die internen Parameter, die ein Abstandsmaß steuern, so verändert werden, dass sie sich insgesamt besser an das Gesamtergebnis anpassen. In dieser Forderung steckt ebenfalls ein Optimierungsproblem, das gemäß Formel 2.32 ausgedrückt werden kann. Die Parameter eines Abstandsmaßes können, wie im Falle der Werte für die affinen Transformationen, in einem Vektor $\boldsymbol{p}_S$ zusammengefasst werden.

$$\boldsymbol{p}^*_{\boldsymbol{S_i}} = \underset{\boldsymbol{p}}{\text{argmax}}\ S_i(I_1(t), I_2(\boldsymbol{x})),\ \text{mit}\ i = 1...n \tag{2.32}$$

Um eine bessere Vorstellung der Arbeitsweise von Optimierungsverfahren zu bekommen, werden exemplarisch zwei einfache Optimierungsverfahren, die auch im praktischen Teil dieser Arbeit Anwendung finden, genauer beschrieben. Dabei handelt es sich zum einen um das gradientenfreie und globale Optimierungsverfahren *Adaptive Zufallssuche* und zum anderen um das gradientenfreie und lokale Verfahren *„Best-Neighbour“*.

Adaptive Zufallssuche:

Die *Adaptive Zufallssuche* (engl. Adaptive Random Search (ARS)) versucht sich dem globalen Optimum über zufällig ausgewählte Punkte, die normalverteilt im gesamten Suchintervall der Zielfunktion liegen, iterativ anzunähern. Dabei bilden sich die beiden Mengen $X^{(k)}$, die n Werte aus dem Definitionsbereich der Funktion beinhaltet, und $Y^{(k)}$, die die zugehörigen Funktionswerte als Inhalt hat. Mit dem Index k wird die aktuelle Generation bezeichnet. Nachdem die Mengen in der ersten Phase des Verfahrens initialisiert wurden, werden nun in Phase zwei n_{rand}[11] Parametervektoren mit den größten Funktionswerten beibehalten. Die ausgewählten Werte, deren Funktionswert einen zu kleinen Betrag aufweisen, werden aus den beiden Mengen gelöscht.

In der dritten Phase werden n_{rand} neue Punkte ausgewählt und in die Menge $X^{(k+1)}$ eingetragen. Dabei gelten nach [Kub06] die beibehaltenen Werte aus der Menge $X^{(k)}$, deren

[11] rand = random (zufällig)

Varianz mit σ_{gen}[12] gegeben ist, als Erwartungswerte der Normalverteilung. Darüber hinaus werden nur Punkte akzeptiert, deren Funktionswert nicht kleiner ist, als der kleinste der letzten Generation.

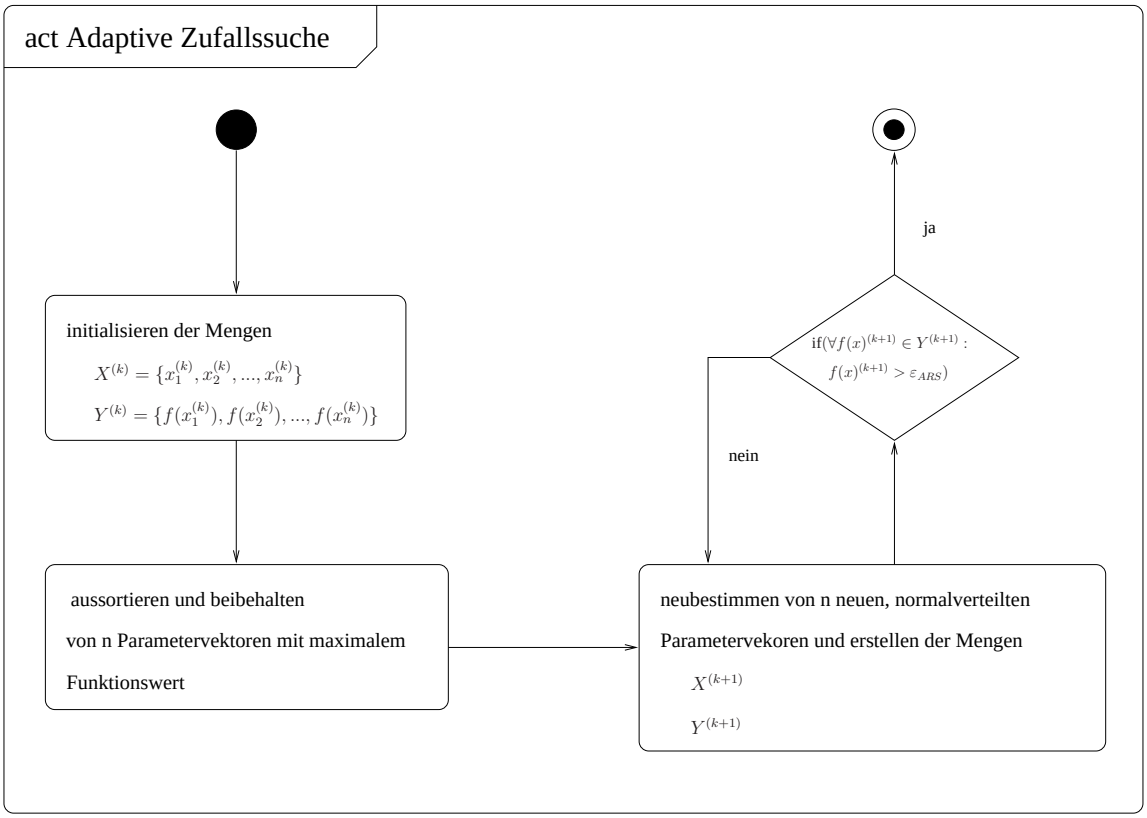


Bild 2.6: Aktivitätsdiagramm des ARS-Optimierungsverfahren

Im letzen Schritt des Verfahrens wird die Abbruchbedingung überprüft. Ist der Unterschied zwischen den Funktionswerten größer, als es der gegebene Schwellwert ε_{ARS} zulässt, dann wird k um eins erhöht und die Varianz σ_{gen} der Gaußverteilung um einen Faktor σ^2_{decr}[13] verkleinert. Danach wird die Suche nach dem globalen Optimum ab dem dritten Verfahrensschritt wiederholt. Falls mit keinem Funktionswert der Schwellwert ε_{ARS} überschritten

[12] gen = general (allgemein)
[13] decr = decrease (verringern, reduzieren)

wird terminiert das Verfahren. Das Aktivitätsdiagramm in Grafik 2.6 veranschaulicht den beschriebenen Ablauf.

„Best-Neighbour“-Optimierungsverfahren:

Mit dem lokalen Optimierungsverfahren *Best-Neighbour* wird über Vergleiche mit Nachbarpunkten, die in einem gegebenen Abstand vom aktuell betrachteten Punkt liegen, iterativ versucht, ein lokales Optimum zu finden. Dabei werden in jeder Dimension über einen festgelegten Betrag die Nachbarpunkte ermittelt. Bei einem n-dimensionalen Suchraum werden insgesamt $2n$ Punkte ermittelt, deren Funktionswerte mit dem des Startpunktes verglichen werden. Je nach Art des gesuchten Optimums entscheidet ein größerer oder kleinerer Funktionswert darüber, welcher Punkt den Vergleichswert der nächsten Iteration liefert. Befindet sich unter den Nachbarpunkten ein Funktionswert, der besser ist, so wird dieser als neuer Startpunkt ausgewählt. Wird jedoch kein besserer Funktionswert ermittelt, kommt es zu einer Verkleinerung der Schrittweite, mit der die Nachbarpunkte ermittelt werden. Das Verfahren wird beendet, wenn der Abstand der Nachbarpunkte kleiner als ein festgelegter Schwellwert ist.

In [Wei03] wird dieses Optimierungsverfahren als sehr stabil beschrieben, wenn sich die Kostenfunktion in jeder Dimension ähnlich verhält. Um zu vermeiden, dass das Verfahren in einem nicht gewollten lokalen Extremum hängen bleibt, wird in [RRS$^+$03] vorgeschlagen, die Bilder mit einem Gaußfilter zu glätten. Das in Grafik 2.7 dargestellte Aktivitätsdiagramm veranschaulicht die Abfolge der Arbeitsschritte des beschriebenen Optimierungsverfahrens.

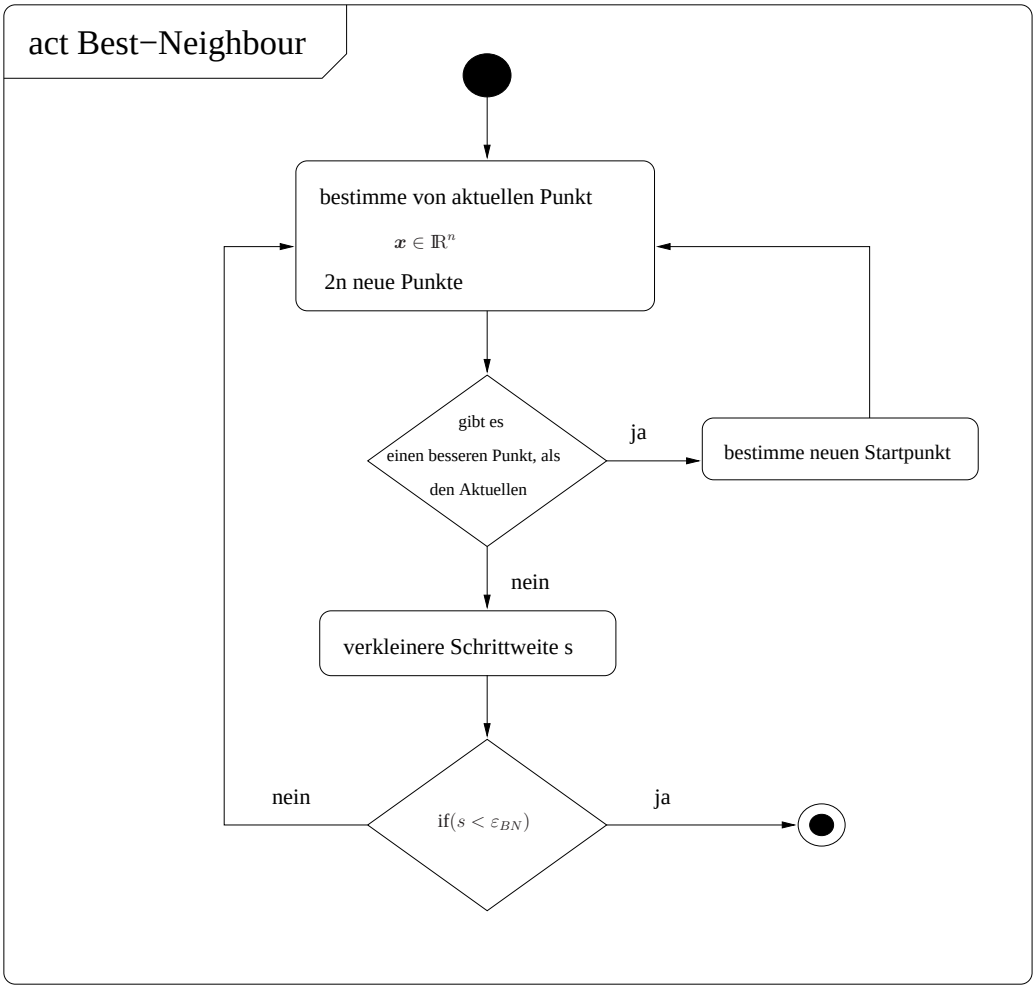


Bild 2.7: Aktivitätsdiagramm des Optimierungsverfahren Best Neighbour

Kapitel 3

Eigener Ansatz

3.1 Das Prinzip der Analyse durch Synthese

3.1.1 Anwendungsbeispiele

Das Prinzip des Modellvergleichverfahrens *Analyse durch Synthese* hat seinen Ursprung in der Sprachverarbeitung und wird erstmals mit der Veröffentlichung [Bel59] von Gordon Bell namentlich erwähnt. Die Basis dieser Arbeit bilden zum einen die weit fortgeschrittene Forschung über die Tonsignalbildung in der Phonetik und zum anderen die Möglichkeit der Durchführung rechenintensiver Analyseprozeduren auf dem Rechner. Mit dem Wissen, dass bestimmte Parameter eindeutig die Entstehung von Lauten beschreiben, kann eine sogenannte Syntheseeinheit konstruiert werden, die mit der Eingabe der Bestimmungsparameter das dazugehörige Lautspektrum generiert. Anschließend wird das synthetische Signal im Analyseschritt mit einem Eingangssignalspektrum verglichen und der Fehler zwischen beiden ermittelt. Das Verfahren wird durch die Gewinnung der Parameter, die den kleinsten Fehler zwischen beiden Signalspektren bewirken, beendet.

Den Entwicklern dieses ersten *Analyse durch Synthese*-Systems ist schon während der Arbeit an diesem aufgefallen, dass eine Möglichkeit der Reduzierung der benötigten Information für die Lautsignalgenerierung gegeben ist. Der Grund dafür liegt darin, dass das Tonsignal auf bestimmte Werte reduziert werden kann, um es eindeutig zu bestimmen.

Diese Eigenschaft findet heute direkte Anwendung in Sprachkomprimierungsverfahren moderner Kommunikationssysteme. Das Prinzip der sogenannten *Analyse durch Synthese*-Kodierung nutzt die Eindeutigkeit der Parameterbeschreibung, um schnell und effizient ein Sprachsignal $s(n)$ zwischen zwei Kommunikationspartnern zu übermitteln. Dieses Eingangssignal wird dabei in n Blöcke (engl. Frames) unterteilt. Mit Hilfe der Signalsynthese und der sich anschließenden Analyse wird der sogenannte Anregungsvektor $u(n)$ ($=$ Parametervektor) ermittelt, der den kleinsten Fehler $e'(n)$ zwischen Eingangs- und synthetisch erzeugtem Signal $s'(n)$ bewirkt. Dieser Anregungsvektor wird zusammen mit einem Skalierungsfaktor zum Empfänger übermittelt und steuert auf dessen Seite die Syntheseeinheit. Mit der Übermittlung der Anregungsvektoren der einzelnen Signalblöcke kann das Eingangssignal auf Empfängerseite reproduziert werden. Die Abbildung 3.1 stellt die einzelnen Verarbeitungsschritte der *Analyse durch Synthese*-Codierung dar.

Ende der siebziger Jahre findet das Verfahren auch Anwendung in der Bildverarbeitung. Den Anstoß hierfür bilden die beiden Arbeiten [MN78] und [BS79]. Ziel ist es nach [Moe99], der die beiden genannten Arbeiten zusammenfassend beschreibt, dass die Analyse von Bewegungen mit Hilfe synthetischer Bilder[1] effizienter gestaltet wird. Dabei wird eine Bewegung angenommen und in einem synthetischem Bild festgehalten. Dieses wird anschließend mit dem Eingabebild verglichen und der Bewegungsvektor neu bestimmt. Laut [Moe99] sind die Resultate sehr unbefriedigend ausgefallen, was auf die Verwendung von statischen Modellen zurückzuführen ist. Durch die Verbesserungen und Weiterentwicklungen in dem Bereich der Bildsynthese können heute sogenannte aktive Modelle mit einer höheren Darstellungsqualität verwendet werden.

Eine andere Anwendung der bildbasierten *Analyse durch Synthese* zeigt sich beim Ornamenteprojekt (siehe [SSP06]), das in der Arbeitgruppe Aktives Sehen der Universität Koblenz-Landau unter der Leitung von Prof. Dr. Dietrich Paulus durchgeführt wird. Die Basis wird hierbei durch die Möglichkeit der mathematischen Modellbeschreibung von Ornamenten gebildet, die ebenfalls durch festgelegte Parameter, wie Radius oder der Anzahl der Symmetrieachsen, bestimmt werden. Diese Modellbildungsvorschrift wird für die synthetisch generierte, bildliche Darstellung eines Ornaments (Abbildung 3.2) verwendet,

[1] Einfache geometrische Formen stellen den zu analysierenden Körper dar.

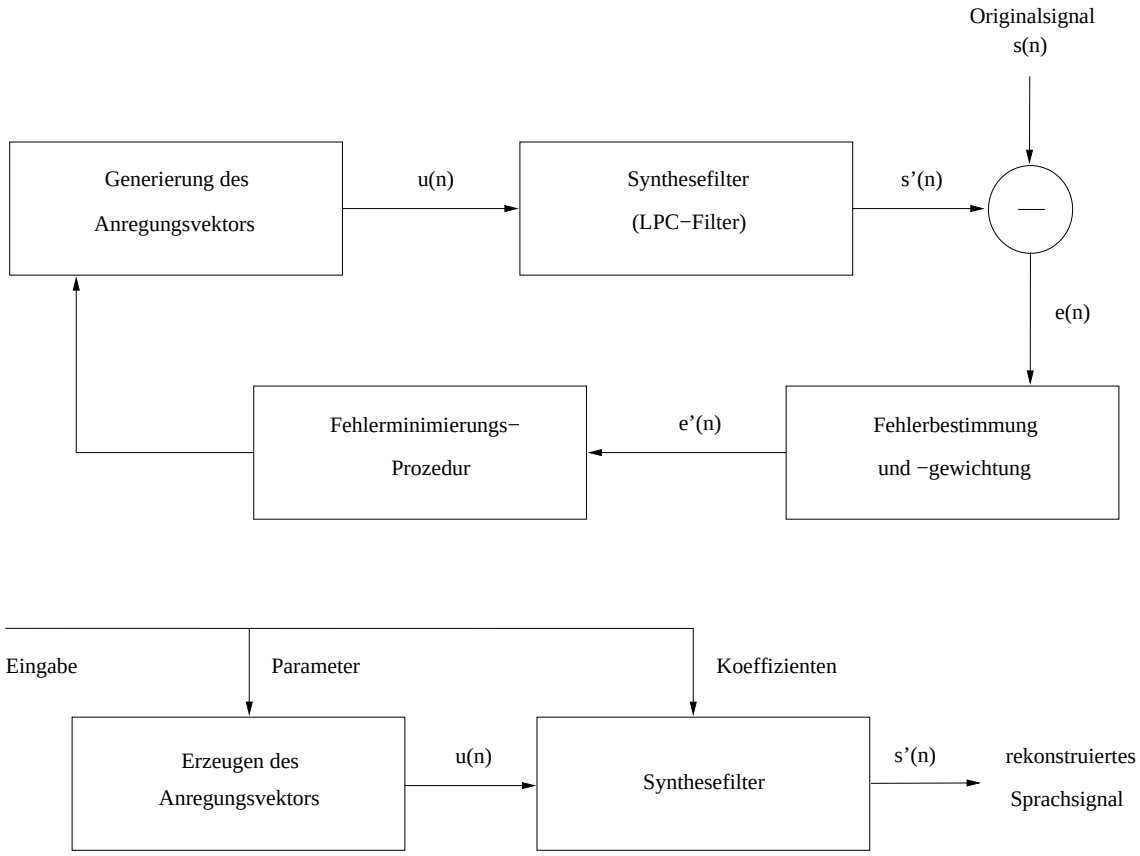


Bild 3.1: Analyse durch Synthese Kodierung und Dekodierung

die anschließend für die Analyse der zu klassifizierenden Ornamentaufnahmen weiter genutzt wird.

Aus allen beschriebenen Beispielen der Verwendung des Prinzips der *Analyse durch Synthese*, lässt sich ein generelles Vorgehen ableiten, das mit der Abbildung 3.3 in Form eines UML-Aktivitätsdiagramms näher beschrieben wird.

Der Ausgangspunkt aller *Analyse durch Synthese*-Verfahren ist zum einen das a priori Wissen über das, was analysiert werden soll, und zum anderen die Möglichkeit es mathematisch beschreiben zu können. Desweiteren gilt die Grundvoraussetzung, dass jeder Zustand des Modells eindeutig durch Parameter bestimmt werden kann.

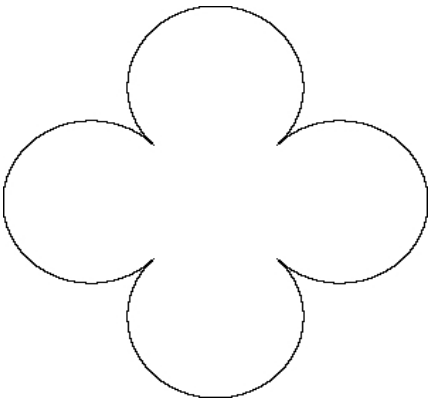


Bild 3.2: Links: ein synthetisiertes Ornamentbild, Rechts: Bild einer Ornamentanwendung

3.1.2 Prinzip

Gegeben ist ein eindeutig bestimmbares Modell $M(\boldsymbol{x})$, dessen Zustand über den Parametervektor $\boldsymbol{x}$ gesteuert wird, und ein Eingabesignal $S(\boldsymbol{y})$, dessen Bestimmungsvektor $\boldsymbol{y}$ unbekannt ist. Es gilt nach Formel 3.1 folgende Eigenschaft der Parametervektoren:

$$\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \tag{3.1}$$

Die Aufgabe eines jeden *Analyse durch Synthese*-Verfahrens ist es, den Parametervektor $\boldsymbol{x}$ so zu schätzen, so dass nach Formel 3.2 folgende Gleichungen erfüllt sind:

$$M(\boldsymbol{x}) = S(\boldsymbol{y}), \text{und damit } \boldsymbol{x} = \boldsymbol{y} \tag{3.2}$$

3.1.3 Positionsbestimmung über Analyse durch Synthese

Im eigenen Ansatz dieser Arbeit soll ein System zusammengestellt und getestet werden, dass durch Verwendung des Prinzips der Analyse durch Synthese die Kameraposition und -orientierung zu einem gegebenen Bild ermittelt. Das System wird dabei von zwei Sensoren unterstützt, die jedoch ungenaue Positions- und Orientierungsangaben liefern, bzw. an das System übermitteln.

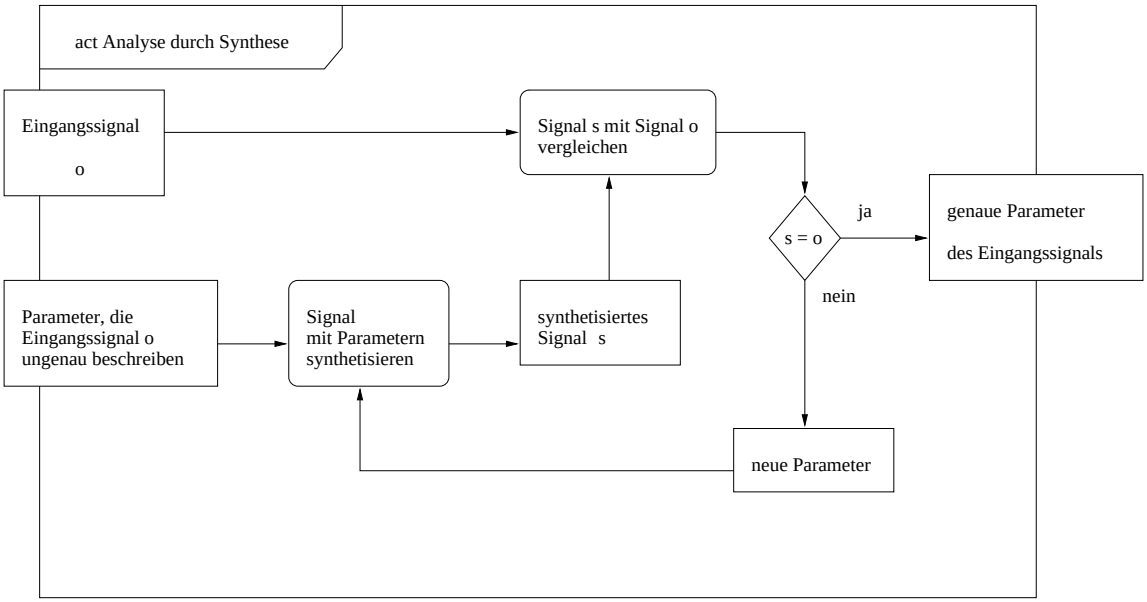


Bild 3.3: Aktiviätsdiagramm des generellen Prinzips der Analyse durch Synthese

Das Vorgehen der 2-D/3-D-Registrierung entspricht dem Prinzip der Analyse durch Synthese in der Art, dass sie für den eigenen Ansatz genutzt werden kann. Der Kreislauf der Berechnungsschritte der Bildregistrierung wird verwendet, um die Werte der Zustandsgrößen, in dem Fall Rotation und Translation einer Kamera, zu bestimmen. Ein Durchlauf beginnt mit der Synthese eines Bildes. Danach wird dieses mit einem beliebigen Eingabebild, z. B einer Kameraaufnahme, verglichen und der Abstand zwischen beiden berechnet. Der Kreislauf wird mit der Bestimmung neuer Werte der Zustandsgrößen geschlossen. Wenn im Vergleich zwischen synthetischem Bild und Eingabebild im Idealfall festgestellt wird, dass die Bilder identisch sind, wird der Kreislauf unterbrochen und die Bildregistrierung damit beendet.

Als Synthese wird im System die Berechnung eines Bildes aus Sicht einer virtuellen Kamera bezeichnet, die die Lage und Orientierung einer realen Kamera simuliert. Die Verbindung zwischen virtueller und wirklicher Kamera wird über die Parameter der Rotation und Translation gebildet. Dafür wird vorausgesetzt, dass für beide Kameras der Ursprung des Weltkoordinatensystems an derselben Position festgelegt ist. Die Ausrichtung der Koordinatenachsen muss ebenfalls gleich sein. Die Umgebung, in der sich die reale Kamera

bewegt, ist als 3-D-Szene nachgestellt. Somit können im idealen Fall Ansichten synthetisiert werden, die denselben Wertebereich aufweisen, wie das Kamerabild.

3.2 GPGPU

Der Begriff GPGPU[2] bezeichnet die Programmierung von computergrafik-fremden Berechnungen, die vom Grafikprozessor ausgeführt werden sollen. Der Grund für diese Art der Programmierung liegt in der Ausnutzung der GPU-eigenen parallelen Datenflussarchitektur, die die zeitgleiche Ausführung von Rechenoperationen ermöglicht. Darüber hinaus ist im Laufe der Grafikkartenentwicklung diese Art der Datenverrechnung sehr stark optimiert worden, um eine schnelle Darstellung von qualitativ hochwertigen Bildern zu erreichen. Dazu zählt z. B. die Verringerung von Verwaltungs- und Auslagerungsarbeiten, wie sie auf Seiten der CPU notwendig sind.

Die Möglichkeit den Grafikprozessor programmieren zu können, führte dazu, diesen auch für andere Berechnungen zu nutzen. Die programmierbaren Hardwareeinheiten des Prozessors sind der Vertex-Shader und der Fragment-Shader, die im Abschnitt 3.2.3 genauer vorgestellt werden. Seit der Einführung von DirectX 10 besteht auch die Möglichkeit einen sogenannten „Geometry-Shader“ in der Grafikpipeline des Prozessors programmieren zu können, der die Szene um weitere Eckpunkte erweitert und die Möglichkeit der geometrischen Berechungen vergrößert.

Die parallele Datenverarbeitung birgt aber auch Nachteile, die man bei der Programmierung schon frühzeitig berücksichtigen sollte. So fordert die parallele Berechnung eine Reorganisierung der zu verarbeitenden Daten, um diese vorteilhaft auf dem Grafikprozessor verrechnen zu können. Ein Beispiel hierfür ist in der Arbeit [Kub06] gegeben, in der die Neuorganisation von Daten in einer Textur dargestellt wird. Darüber hinaus ist es aufgrund der Pipelinearchitektur, die in Abbildung 3.4 dargestellt ist, auf der Grafikkarte nicht möglich, die essentiellen Lese- und Schreiboperationen, die man im Rahmen der GPU-Programmierung als „gather“ und „scatter“ bezeichnet, ohne größeren Aufwand durchzuführen. So kann nicht lesend auf eine Ausgabetextur und schreibend auf eine Ein-

[2] GPGPU = generel purpose computation on graphics processing units

gabetextur zugegriffen werden. Werden diese Operation von einem Programm zwingend verlangt, so muss auf spezielle Tricks, die in [GÖ6] beschrieben sind, zurückgegriffen werden. Meistens führen diese Kniffe auch dazu, dass die angestrebte Applikation an Komplexität zunimmt, die bei einer CPU-Implementierung hätte vermieden werden können. Darüber hinaus kann es auch dazu kommen, dass der Gewinn an Geschwindigkeit durch das mehrfache Laden von Texturen verhindert wird. Eine andere Ursache für eine Geschwindigkeitsabnahme findet sich im Datentransfer zwischen GPU und CPU. Dieser Fall, der auch bei [GÖ6] beschrieben wird, tritt meistens dann auf, wenn Berechnungen, die nicht auf der GPU durchgeführt werden können, auf die CPU ausgelagert werden. In diesem Fall muss untersucht werden, ob der Datentransfer den Geschwindigkeitsgewinn durch die GPU-Berechnung stark reduziert oder sogar die Berechnung insgesamt langsamer werden lässt.

Der nachfolgende Text beschreibt den wesentlichen Aufbau, die Bestandteile und Befehle einer typischen GPGPU-Anwendung, wie sie auch im Rahmen dieser Diplomarbeit Verwendung findet. Dabei wird versucht die Beschreibungen so hardwarenah wie nur möglich zu gestalten, um eine bessere Vorstellung von den auf der Grafikkarte stattfindenden Prozessen geben zu können. Das Hauptaugenmerk liegt hierbei auf der Grafikprozessoreinheit. Die Abkürzung GPU bezeichnet hierbei ausschließlich dieses Bauteil einer modernen Grafikkarte. Alle im nachfolgenden Text verwendeten Programmfragmente stammen aus dem Tutorial [GÖ6] von Dominik Göddeke, das eine sehr detailierte und umfassende Einführung in die GPGPU-Programmierung mit zahlreichen Programmbeispielen gibt.

3.2.1 Grafikkartenprogrammierung

Der Beginn jeder Grafikkartenprogrammierung beginnt auf CPU-Seite und wird mit Hilfe von sogenannten Grafikbibliotheken, auch Grafik-API[3] genannt, durchgeführt. Als Hauptvertreter seien hier Direct3D für Windows und OpenGL für die übrigen Plattformen einschließlich Windows genannt. Beide Bibliotheken werden als Implementierung des jeweiligen Grafikkartentreibers realisiert und stellen in Form einer Hochsprache Befehle für die Ansteuerung der Grafikprozessorpipeline zur Verfügung.

[3] API = Application Programming Interface

Spezifikation und API:

Sowohl OpenGL als auch Direct3D, das ein Bestandteil des Spezifikationspakets von DirectX ist, sind Festlegungen bestimmter Funktionen und werden von einem Konsortium bestimmt. Direct3D unterliegt den Vorgaben von Microsoft, während OpenGL von der gleichnamigen Institution bestimmt wird. Diese Institutionen legen fest, was genau mit einem Befehl der jeweiligen Bibliothek an Funktionalität gegeben sein muss. Erst mit der Programmierung des Grafikkartentreibers werden die Funktionsspezifikationen für die jeweilige Grafikkarte umgesetzt. Auf diese Weise wird erreicht, dass trotz unterschiedlicher Hersteller die Sprache für die Programmierung einer Grafikkarte einheitlich bleibt. Nur die im Hintergrund implementierte Funktionalität eines bestimmten Befehls, kann von einem Treiberhersteller nach eigenen Vorstellungen umgesetzt werden.

Die fortschreitende Entwicklung der Funktionalität der Grafikkarte führt zwangsläufig auch zu einer Erweiterung des Befehlsumfangs der Grafikbibliotheken. Jede von einem Hersteller angestrebte Funktionserweiterung führt z. B. im Falle von OpenGL über vier festgesetzte Erweiterungsstufen. Wird von einem Hersteller eine Erweiterung verlangt, so bekommt diese das Suffix der jeweiligen Firma angehängt. Im Falle von Nvidia wäre dies z. B. *GL_NV*. Wird von mehreren Herstellern beschlossen, die besagte Funktionalität in ihren Treibern mit zu realisieren, wird das Suffix des Initiators durch das Suffix *EXT* ersetzt. Unter der URL [mis07] sind alle derzeit aktuellen Erweiterungen dokumentiert. Einigt sich danach das *ARB*[4]-Konsortium die Funktionalität zu standardisieren, so wird im dritten Schritt die *EXT*-Endung durch *ARB* ersetzt. Der vierte und zugleich letzte Schritt ist die Aufnahme einer *ARB*-Erweiterung in den OpenGL-Kern. Nach diesem Schritt wird die Funktion nicht mehr zusätzlich gekennzeichnet, sondern wird wie ein normaler OpenGL-Befehl verwendet. Die zum OpenGL-Kern gehörenden Funktionen gelten für alle Treiberhersteller, die OpenGL in ihrer Treibersoftware mit realisieren möchten, als verbindlich. Sowohl der OpenGL-Kern als auch die Menge an Erweiterungen, die mit dem Treiber einer Grafikkarte realisiert sind, geben Auskunft über den Funktionsumfang der jeweiligen Grafikkarte.

[4] ARB = Architecture Review Board

3.2.2 OpenGL-Programmkontext

Jede Grafikprogrammierung, egal ob es sich um eine Anwendung in der Computergrafik oder um eine GPGPU-Anwendung handelt, muss aufgrund der Pipelinearchitektur den kompletten Datenfluss durch die Pipeline berücksichtigen. Das bedeutet, dass auch im Falle der GPGPU-Anwendung Eckpunkte (engl. vertices) und Projektion mit definiert werden müssen, obwohl sie nicht Bestandteil der Berechnungen auf der GPU sind. Der erste Schritt besteht darin einen gültigen OpenGL-Kontext anzulegen. Dies geschieht mit der Initialisierung von *GLUT*[5], die neben der Initialisierung des OpenGL-Kerns *GL* auch Fensterfunktionen, wie das Öffnen und Schließen oder die Erstellung von Menüs bereitstellt.

```
#include<GL/glut.h>
void initGLUT(int argc, char **argv){
     glutInit (&argc, argv );
     glutCreateWindow("TESTS");
}
```

Danach erfolgt die Initialisierung von *GLEW*[6] [IM06]. Damit sind neben dem OpenGL-Kern auch die Erweiterungen verfügbar, die fast immer in einer GPGPU-Anwendung benötigt werden. In diesem Zusammenhang muss bedacht werden, dass die Erweiterungen, wie bereits beschrieben, nicht Bestandteil des OpenGL-Kerns sind und damit, je nach verwendeter Grafikkarte und zugehörigem Treiber, nicht alle benötigten Erweiterungen zur Verfügung stehen. Dies sollte vor Beginn der Implementierung der Anwendung überprüft werden. Unter *Linux* kann die Information mit dem Befehl *glxinfo* abgefragt werden. Des Weiteren ist zu beachten, dass die Datei *glew.h* vor *glut.h* eingebunden wird. Mit der Initialisierung von *GLEW* wird gleichzeit auch *GLU*[7] eingebunden. Damit stehen der Applikation auch Funktionen für die Matrixmanipulation, Mipmapping[8] und Fehlerbehandlung zur Verfügung.

```
#include <GL/glew.h>
```

[5] GLUT = OpenGL Utility Toolkit
[6] OpenGL Extension Wrangler Library
[7] GLU = OpenGL Utility Library
[8] Erzeugung niedriger Auflösungsstufen einer Textur

```
#include <GL/glut.h>

void initGLEW (void) {
     int err = glewInit();
     if (GLEW_OK != err) {
         printf((char*)glewGetErrorString(err));
         exit(ERROR_GLEW);
     }
}
```

Da die berechneten Ergebnisse einer GPGPU-Anwendung nicht auf dem Bildschirm dargestellt werden sollen, müssen OpenGL und damit die Grafikhardware so konfiguriert werden, dass die Ergebnisse explizit in einem Bereich des Grafikkartenspeichers abgelegt werden, der genau nicht für die Bildschirmausgabe vorgesehen ist. Dies erreicht man mit der OpenGL-Erweiterung *Framebufferobjekt*, die das Anhängen von ein oder mehreren Ausgabetexturen ermöglicht, in die dann die Ausgabedaten geschrieben werden können. Mit den unten dargestellten Befehlen wird ein Framebufferobjekt definiert und für das Schreiben in das System gebunden.

```
GLuint fb;

void initFBO(void) {
     glGenFramebuffersEXT(1, &fb);
     glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fb);
}
```

Als nächstes müssen die Aus- und Eingabedatentypen definiert werden. Dies geschieht unter Verwendung verschiedener Texturarten und Ausgabedaten der Applikation. Die Auswahl des jeweiligen Texturtyps wird im Wesentlichen über drei Eigenschaften bestimmt. Das sind zum Einen das Textur-Ziel (engl. texture target), das Textur-Format (engl. textureformat) und das interne Format (engl. internal format). Mit dem Textur-Ziel soll dem System mitgeteilt werden, wieviele Dimensionen für die Textur verwendet werden sollen. Im zweidimensionalen Fall wird zwischen den beiden Varianten *GL_TEXTURE_2D* und *GL_RECTANGLE* unterschieden. Mit beiden Zielen ist eine Reihe von Eigenschaften gegeben, die alle in der OpenGL-Dokumentation [SWND03] nachgeschlagen wer-

den können und hier nicht näher beschrieben werden. Eine der wesentlichsten Entscheidungen ist die Wahl des Texturfomats und des internen Formats. Mit dem Texturformat wird bekannt gegeben, wieviele Farbkanäle von der Textur verwendet werden. In vielen GPGPU-Anwendungen werden die Werte in vier Kanälen abgespeichert, um optimal die Eigenschaften des Framebuffers nutzen zu können. Des Weiteren kommen auch sehr häufig Einkanaltexturen zum Einsatz. Die Nutzung von vier Kanälen wird über *GL_RGBA* bekannt gegeben, während mit *GL_LUMINANCE* das Texturformat auf einen Kanal eingestellt wird. Das interne Format gibt die Präzision an, mit der die Werte in der Textur gespeichert werden sollen. Im Rahmen einer GPGPU-Anwendung kommen des Öfteren Float-Werte mit einer Tiefe von 32 Bit zum Einsatz. Je nach Bedarf kann auch eine Tiefe von 16 Bit gewählt werden. Die einzelnen Schritte sind die Erzeugung und das Binden einer Textur, die Deaktivierung der Filter und die Allokation des benötigten Speichers. Die genaue Bedeutung der jeweiligen Funktionsaufrufe wird in [SWND03] beschrieben.

```
GLuint texID;
glGenTextures (1, &texID);
glBindTexture(texture_target,texID);
glTexParameteri(texture_target, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(texture_target, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(texture_target, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(texture_target, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
glTexImage2D(texture_target, 0, internal_format,
             texSize, texSize, 0, texture_format, GL_FLOAT, 0);
```

Der nächste Schritt ist die Bekanntgabe der zu verwendenden Projektionsart, der Größe und Position des Sichtfensters und der dazugehörigen Parameter. Diese Daten werden im ersten Abschnitt der Rendering-Pipeline verwendet, um zu entscheiden, was im Sichtfenster zu sehen ist und was gelöscht werden kann. Im Quelltextfragment wird die Rendering-Pipeline auf eine orthographische Projektion eingestellt. Damit wird erreicht, dass mit Hilfe der automatischen Interpolation der Texturkoordinaten eine 1:1-Abbildung der Pixel-Texel[9]-Geometrie durchgeführt wird.

[9] texel = texture element

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, texSize, 0.0, texSize);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glViewport(0, 0, texSize, texSize);
```

Um die Ausgabe einer Berechnung in einer Textur speichern zu können, muss die Textur an das Framebufferobjekt angehängt werden, damit dieses als Renderziel (engl. render target) benutzt wird. Um dies zu erreichen, muss das Framebufferobjekt noch in das System gebunden sein. Der zweite Parameter, des im folgenden Quelltext dargestellten Funktionsaufrufs gibt den Punkt im Framebufferobjekt an, an den die Textur angehängt werden soll. Insgesamt können vier verschiedene Texturen an ein Framebufferobjekt gebunden werden.

```
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
       GL_COLOR_ATTACHMENT0_EXT,
       texture_target, texID, 0);
```

Um Daten von einem Datenfeld, das auf CPU-Seite definiert ist, in eine Textur zu transferieren, wird als erstes die jeweilige Zieltextur gebunden und anschließend die Daten in die Textur überführt. Sollen Daten in eine im Framebufferobjekt gebundenen Textur gerendert werden, so sind die Befehle, des folgenden Quelltextbeispiel zu verwenden.

```
glBindTexture(texture_target, texID);
glTexSubImage2D(texture_target,0,0,0,texSize,texSize,
       texture_format,GL_FLOAT,data);

glDrawBuffer(GL_COLOR_ATTACHMENT0_EXT);
glRasterPos2i(0,0);
glDrawPixels(texSize,texSize,texture_format,GL_FLOAT,data);
```

Für den Datentransfer von einer Textur in ein Datenfeld, das für Berechnungen in der CPU genutzt werden kann, gibt es zwei Möglichkeiten. Entweder man bindet die gewünschte Textur in den OpenGL-Kontext und transferiert die Daten mit *glGetTexImage()* in das

Datenfeld, oder man benutzt eine an ein Framebufferobjekt gebundene Textur und liest die Daten mit dem Befehl *glReadPixels()* in das Zieldatenfeld.

Mit dieser Reihe von Befehlen kann für fast jede GPGPU-Anwendung der benötigte OpenGL-Kontext erzeugt werden. Für die eigentliche Shader-Programmierung sind aber noch eine Reihe von zusätzlichen Befehlen notwendig. Im Folgenden wird der Aufbau eines geeigneten Shader-Kontextes beschrieben. Hierbei werden zusätzlich die einzelnen Stationen der Rendering Pipeline vorgestellt und ihren Einfluss auf die Shader-Programmierung erläutert. Die Shader-Programmierung wird mit Hilfe der Shader-Sprache *Cg*[10] vorgestellt. Während der Beschreibung wird nicht direkt auf ein Shader-Programm Bezug genommen, sondern es werden die grundlegenden Befehle erläutert, die es ermöglichen, jedes Shader-Programm auf einer Nvidia-Grafikkarte, die das benötigte Shader-Model unterstützt, auszuführen. Eine umfassende Einführung in die Shader-Programmierung unter Cg bietet das Buch [FK03].

3.2.3 Shader-Programmkontext und Grafikprozessorpipeline

Die beiden programmierbaren Hardwarebestandteile der Grafikprozessorpipeline sind zum einen der Vertex-Shader und zum anderen der Fragment-Shader. Beide Prozessorbestandteile stehen an verschiedenen Stellen der Grafikpipeline und erfüllen bestimmte Berechnungsaufgaben. Um einen Cg-Shader unter OpenGL nutzen zu können, muss aus der Cg-Entwicklungsumgebung die Header-Datei *CgGL.h* und für den Linker die entsprechende Bibliothek in die Applikation mit eingebunden werden. Damit sind eine Reihe von Einrichtungsroutinen gegeben, die von OpenGL benötigt werden, um den Shader richtig in den eigenen Kontext mit einzubinden.

Als erstes muss für den Shader Platz im Grafikkartenspeicher eingerichtet werden. Dafür wird ein Cg-Kontext mit *cgCreateContext()* angelegt. Danach wird der von der Grafikkarte unterstützte Befehlssatz mit *cgGLGetLatestProfile()* für die jeweilige Shader-Art ermittelt. Für einen Vertex-Shader wird mit *CG_GL_VERTEX* und für ein Fragment-Programm mit *CG_GL_FRAGMENT* als Übergabewert der Funktion das sogenannte Funktionsprofil bestimmt. Danach muss mit dem Aufruf von *cgCreateProgram()* das Shader-Programm für

[10] Cg = C for graphics, Cg toolkit: http://developer.nvidia.com /object/cg_toolkit.html

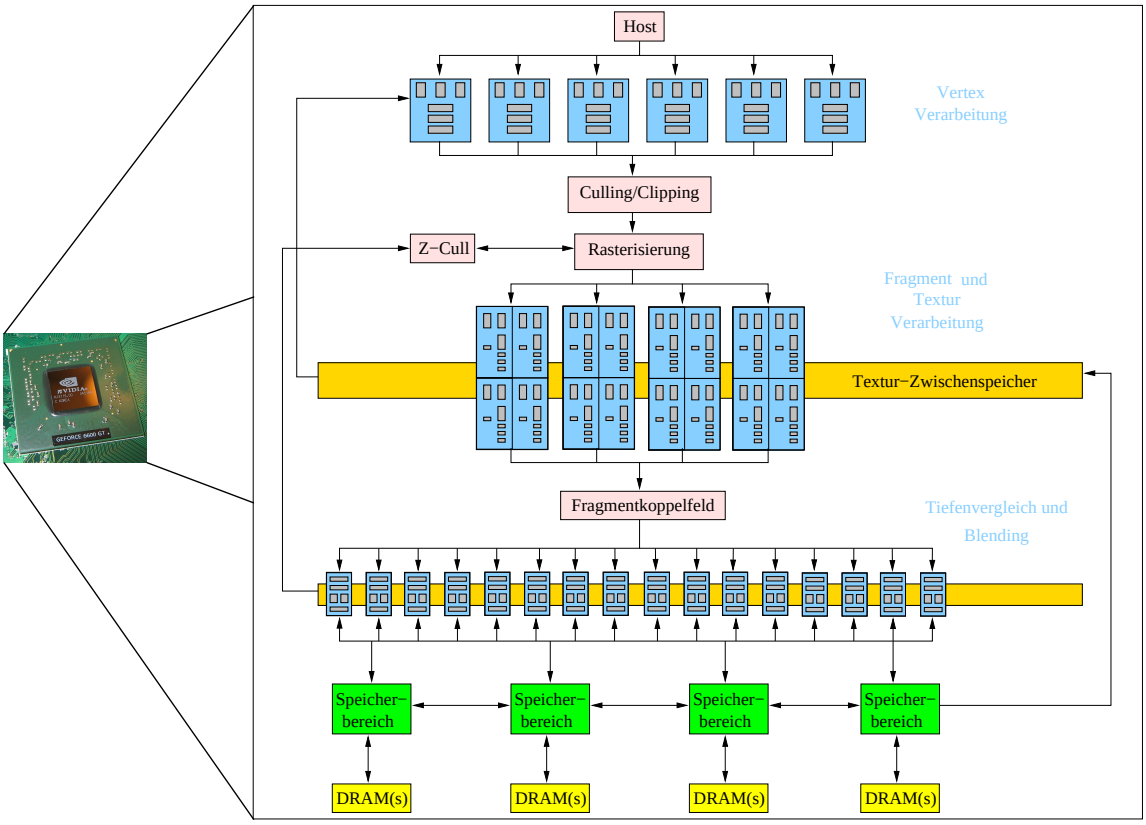


Bild 3.4: Block-Diagramm einer Geforce 6

die Ausführung auf der Grafikkarte kompiliert und anschließend mit *cgGLLoadProgram()* in den OpenGL-Kontext eingebunden werden. Bevor der Shader auf der Hardware ausgeführt werden kann, muss das ermittelte Profil mit *cgGLEnableProfile()* aktiviert und anschließend der Shader mit *cgGLBindProgram()* in den Berechnungsablauf mit eingebunden werden. Besitzt der Shader Eingabedaten in Form von Texturen, so müssen diese mit *cgGLSetTextureParameter()* und *cgGLEnableTextureParameter()* dem Shader übergeben werden. Alle beschriebenen Einrichtungsroutinen können, wie im Quelltext dargestellt, in einer eigenen Methode zusammengefasst werden.

```
void initCG(void) {
     cgContext = cgCreateContext();
     fragmentProfile = cgGLGetLatestProfile(CG_GL_FRAGMENT);
```

```
    cgGLSetOptimalOptions(fragmentProfile);
    fragmentProgram = cgCreateProgram (
    cgContext,CG_SOURCE,program_source,
    fragmentProfile,"saxpy",NULL);
    cgGLLoadProgram (fragmentProgram);
    yParam = cgGetNamedParameter (fragmentProgram,"textureY");
    xParam = cgGetNamedParameter (fragmentProgram,"textureX");
    alphaParam = cgGetNamedParameter (fragmentProgram,"alpha");
}
```

Als letzter Vorbereitungsschritt der Grafikpipeline für den Rendervorgang müssen noch die Aus- und Eingabetexturen im System aktiviert werden. Die Eingabetexturen der Shader werden, wie im Quelltext ersichtlich, mit Cg-eigenen Befehlen in das System integriert.

```
cgGLSetTextureParameter(yParam, y_oldTexID);
cgGLEnableTextureParameter(yParam);
```

Die Ausgabetexturen werden, wie bereits erwähnt, an das Framebufferobjekt mit *glFramebufferTexture2DEXT()* angehängt und mit *glDrawBuffer()* für das Schreiben aktiviert.

```
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
        GL_COLOR_ATTACHMENT0_EXT,
        texture_target, y_newTexID, 0);
glDrawBuffer (GL_COLOR_ATTACHMENT0_EXT);
```

Nachdem OpenGL eingerichtet ist, muss nun die Rendering-Pipeline mit sogenannten geometrischen Primitiven, zu denen Punkte, Linien und Dreiecke gehören, angestoßen werden. Die im Quelltext verwendeten Routinen zeichnen ein Quadrat mit der Eingabetextur, die im Shader benötigt wird. Auf diese Fläche wird dann die Eingabetextur im Maßstab $1:1$ abgebildet, d. h. dass die Pixelkoordinaten identisch sind mit den Texturkoordinaten. Mit diesen OpenGL-Routinen beginnt der Durchlauf durch die Pipeline des Grafikprozessors.

```
glPolygonMode(GL_FRONT,GL_FILL);
glBegin(GL_QUADS);
```

```
        glTexCoord2f(0.0, 0.0);
        glVertex2f(0.0, 0.0);
        glTexCoord2f(1.0, 0.0);
        glVertex2f(texSize, 0.0);
        glTexCoord2f(1.0, 1.0);
        glVertex2f(texSize, texSize);
        glTexCoord2f(0.0, 1.0);
        glVertex2f(0.0, texSize);
glEnd();
```

Wie in Abbildung 3.4 zu sehen ist, bildet der Vertex-Shader die erste Recheneinheit nach dem Vertex-Buffer, der eine Liste von Geometriedaten in Form von Vertices speichert. Der Vertex-Prozessor berechnet für jeden Vertex die Bildschirmkoordinaten, setzt die Vertices zu Dreiecken zusammen und führt Lichtberechnungen durch. Zum Beispiel für die Verwendung neuer Lichtberechnungsmodelle kommen hier neu programmierte Vertex-Shader zum Einsatz. Den nächsten Schritt bilden „Culling“ und „Clipping“ . Beim „Culling“ wird die sichtbare Seite der geometrischen Primitive ermittelt. Die verdeckten Ansichten werden aus dem Speicher gelöscht. Das „Clipping“ ermittelt die Teile einer Szene, die im definierten Sichtfeld liegen. Alle außerhalb liegenden Teile werden ebenfalls aus dem Speicher gelöscht.

In der vierten Station der Rendering-Pipeline werden bei der Rasterisierung die Primitive mit Fragmenten, die im späteren Verlauf die Pixel ausmachen, ausgefüllt. Gleichzeitig wird der Tiefenpuffer mit z-Werten gefüllt, um später am Ende des Pipeline-Durchlaufs den sogenannten Tiefentest durchführen zu können, bei dem für jedes Fragment ermittelt wird, ob es für einen Betrachter sichtbar ist oder ob es von einem anderen Fragment verdeckt wird.

Danach kommen die als sichtbar ermittelten Fragmente in die zweite programmierbare Grafikprozessoreinheit, den sogenannten Fragment- oder Pixel-Shader. In ihm werden standardmäßig die Farbwerte aller Fragmente ermittelt. Wird aber, wie in in dem vorliegenden Beispiel, statt eines Farbwertes eine Textur verwendet, so wird in diesem Fall für jedes Fragment der passende Texturwert bestimmt. Den Abschluss der Grafikpipeline bilden die optionalen Tiefen-, Alpha- und Stencil-Tests.

Kapitel 4

Systemaufbau

Der eigene Ansatz dieser Arbeit besteht im Wesentlichen aus drei Arbeitspaketen. Zunächst mussten in einem ersten Analyseschritt die wesentlichen Funktionspakete, die zur Lösung der Aufgabenstellung dieser Arbeit notwendig waren, ausgearbeitet und ihre Funktionalität im Einzelnen beschrieben werden. Als Herangehensweise wurde der in [Bal00] beschriebene und in [Bal04] im Detail vorgestellte Ansatz der *Objektorientierten Analyse* gewählt. Das Ziel war ein konkret spezifizierter Systemaufbau unter Einbeziehung bereits existierender Softwarebausteinen. Ein im Rahmen einer Diplomarbeit entwickeltes Zusatzprogramm (engl. Plug-In), wies in weiten Teilen die in dieser Arbeit benötigte Funktionalität und Aufbaustruktur auf. Während der Analyse des Programms wurde dieses in eine neue Betriebssystemumgebung importiert und zu einem eigenständigen Programm ausgebaut. Die Portierung umfasste vorallem die Abstimmung mit den Versionsvorgaben des unter Linux verwendeten gcc-Compilers 4.1.2, sowie die Ersetzung der nur unter dem Betriebssystem Windows benutzbaren Funktionalität. Der Grund für die Übertragung liegt darin, dass das vorliegende System somit auch für andere Projekte der Arbeitsgruppe Aktives Sehen genutzt werden kann, da diese ebenfalls unter Linux entwickelt werden.

Im zweiten Arbeitsschritt mussten die Teile aus der existierenden Systemlösung ersetzt werden, die für den eigenen Lösungsansatz unbrauchbar waren. Davon war die Synthese-Einheit betroffen, die durch einen in der Arbeitsgruppe „Computergrafik“ der Universität Koblenz-Landau entstandenen eigenen Bildsynthetisierer (engl. Renderer) ersetzt wurde.

Die Bildsynthese konnte zum Zeitpunkt dieser Diplomarbeit nur unter dem Betriebssystem Windows durchgeführt werden, so dass zur Durchführung der Bildsynthese unter Windows und der anschließenden Bildregistrierung unter Linux eine geeignete Schnittstelle entwickelt werden musste.

Im dritten und letzten Arbeitsschritt wurde für die Anwendung eine geeignete Benutzungsoberfläche entwickelt, die es ermöglicht, dass Konfigurationsdaten für die einzelnen Funktionspakete eingegeben werden können. Desweiteren galt es, die aus den Berechnungen resultierenden Daten für den Anwender in geeigneter Art und Weise sichtbar zu machen. Dies betraf die Ausgabe der berechneten Abstandsmaße und Parameter der Transformation sowie die Darstellung des zugehörigen Synthesebildes.

In der folgenden Beschreibung des Analyseschrittes werden die Teile vorgestellt, die wesentlich zur Zusammenstellung der Lösung beigetragen haben. Dies ist zum einen die *Analyse im Großen*, die sich aus der Bestimmung der Geschäftsprozesse sowie der Erstellung eines Paketdiagramms des Systems zusammensetzt, und zum anderen das statische Systemmodell, das sich aus mehreren Klassendiagrammen, die den einzelnen Funktionspaketen des Paketdiagramms zugeordnet werden können, zusammensetzt. Das Unterkapitel „Veränderungen und Neuentwicklung“ hat den Datentransfer zwischen Synthese- und Analyseeinheit zum Inhalt. Abschließend wird der Aufbau der Benutzungsoberfläche sowie der Datentransfer zwischen der Oberfläche und den übrigen Teilen der Applikation beschrieben.

4.1 Objektorientierte Analyse

Laut [Bal00] ist das Ziel der Objektorientierten Analyse (OOA) „die fachliche Lösung eines Softwareprodukts mit Hilfe objektorientierter Konzepte zu modellieren“([Bal00] S.376). Für diese Arbeit galt es darüber hinaus eine bereits existierende aber nur in einzelnen Teilen dokumentierte und nicht eigenständig funktionsfähige Softwarelösung mit in die Lösung einzubeziehen. Außerdem sollte durch die Analyse herausgefunden werden, inwiefern sich das vorhandene System für die eigene fachliche Lösung eignet.

Aufgrund der nur teilweise vorhandenen Dokumentation zum Zeitpunkt der Analyse und der nicht vorhandenen Artefakte des Entwicklungsprozesses, galt es diese, die normalerweise das Ergebnis der OOA[1] darstellen, nachträglich zu erstellen. Somit ergab sich ein umgekehrter Anwendungsfall der Objektorientierten Analyse.

Das Kernziel jeder OOA ist die Erstellung eines *statischen* und *dynamischen Modells* der angestrebten Problemlösung. Das *statische Modell* wird mit Hilfe eines Klassendiagramms dargestellt während im *dynamischen Modell* die Geschäftsprozesse in Form von Anwendungsfalldiagrammen (engl. Use-Case) oder Szenarien mit Hilfe von Sequenzdiagrammen beschrieben werden. Für diese Arbeit lag das Hauptaugenmerk des *statischen Modells* auf der Beschreibung der Beziehung der im System bereits deklarierten Klassen. Mit Hilfe eines Paketdiagramms, das vor der Analyse zur Festellung der Systemanforderugen erstellt wurde, konnte eine Aufteilung der Klassen in die benötigten Funktionsbereiche gemacht werden.

Für den Analyseschritt wird in [Bal00] ein methodisches Vorgehen nahegelegt, mit dem bei sorgfältiger Anwendung ein in seinen Einzelheiten komplett erfasstes Modell einer Problemlösung erarbeitet werden kann. Für dieses Vorgehen können verschiedene Orientierungspunkte gewählt werden. Folgt man den bereits bekannten Daten des angestrebten Systems, so entsteht bei konsequenter Anwendung die statische Modellbeschreibung. Richtet sich die Orientierung hingegen nach der Funktionalität resultiert hieraus die dynamische Modellbeschreibung der Problemlösung. Im konkreten Anwendungsfall dieser Arbeit waren beide Orientierungen notwendig, da vor allem im Falle der Substitution der Syntheseeinheit herausgefunden werden musste, an welchen Stellen im System ein Aufruf der Bildsynthese erfolgen muss, und wie dieser im einzelnen zu gestalten ist. Vor allem konnte mit dem dynamischen Modell schnell ermittelt werden, welche Klassen in welcher Art und Weise durch einen Austausch betroffen waren und wie die Neugestaltung des zugehörigen Aufrufs auszusehen hat, damit das System so wenig wie möglich abgeändert werden musste.

Die in [Bal04] vorgestellten Prüflisten zur Erstellung eines Artefakts (engl. Checklist) waren für diese Arbeit eine gutes Mittel, um die Analyse der insgesamt 73 Klassen strukturiert durchzuführen. Die aus der Überprüfung entstandenen Artefakte werden im Abschnitt

[1] OOA = Objektorientierte Analyse

4.1.2 vorgestellt. Alle für die OOA relevanten Prüflisten werden in der empfohlenen Reihenfolge, die in [Bal00] als *Makroprozess* bezeichnet wird, angewandt. Für die Durchführung der Untersuchung dieser Arbeit wurde der Prozess in einer abgewandelten Form angewandet. Um die Anforderungen an das System festzustellen, wurden als erstes die einzelnen Geschäftsprozesse mit den beteiligten Akteuren ermittelt und in einem Anwendungsfalldiagramm festgehalten. Aus den so ermittelten Geschäftsprozessen wurden dann die Funktionspakete ermittelt und in einem Paketdiagramm erfasst. Die Erstellung des Anwendungsfalldiagramms und des Paketdiagramms werden nach [Bal04] in der OOA als *Analyse im Großen* bezeichnet.

Auf Grundlage des Paketdiagramms wurden mit Hilfe der Prüflisten die Klassen der Siemens-Systemerweiterung den einzelnen Funktionspaketen zugeordnet. Danach konnte für jedes einzelne Systempaket die Assoziationen der darin enthaltenen Klassen ermittelt werden. Das Ergebnis dieses Analyseschrittes wird in Abschnitt 4.1.2 vorgestellt. Im letzten Schritt wurden die Beziehungen zwischen den Funktionspaketen ermittelt.

4.1.1 Analyse im Großen

Geschäftsprozesse

Das Ziel des Systems ist eine schnelle und dabei effizient durchgeführte Positions- und Orientierungskorrektur einer Kamera. Die Akteure (vgl. Diagramm 4.1), bestehend aus dem Anwender des System, der Kamera, dem Programm Coriander, das für die Kamerasteuerung verwendet wird, der Grafikkarte, dem Prozessor und dem Rendersystem, bestimmen den Ablauf der für die Positionskorrektur implementierten 2-D/3-D-Registrierung. Das System startet mit dem Geschäftsprozess *Systemstart*, der neben der Initialisierung des Systems auch die Überprüfung des Kameraanschlusses beinhaltet. Falls die angeschlossene Kamera dem System unbekannt ist, dann beinhaltet der Geschäftsprozess *Systemstart* zusätzlich den Anwendungsfall *Kamerakalibrierung*, der die Ermittlung der intrinsischen Kameraparameter[2] zum Ziel hat.

[2] Brennweite, Koordinaten des Hauptpunktes, Koeffiziente der radialen Verzerrung, Daten über das Sensorelement

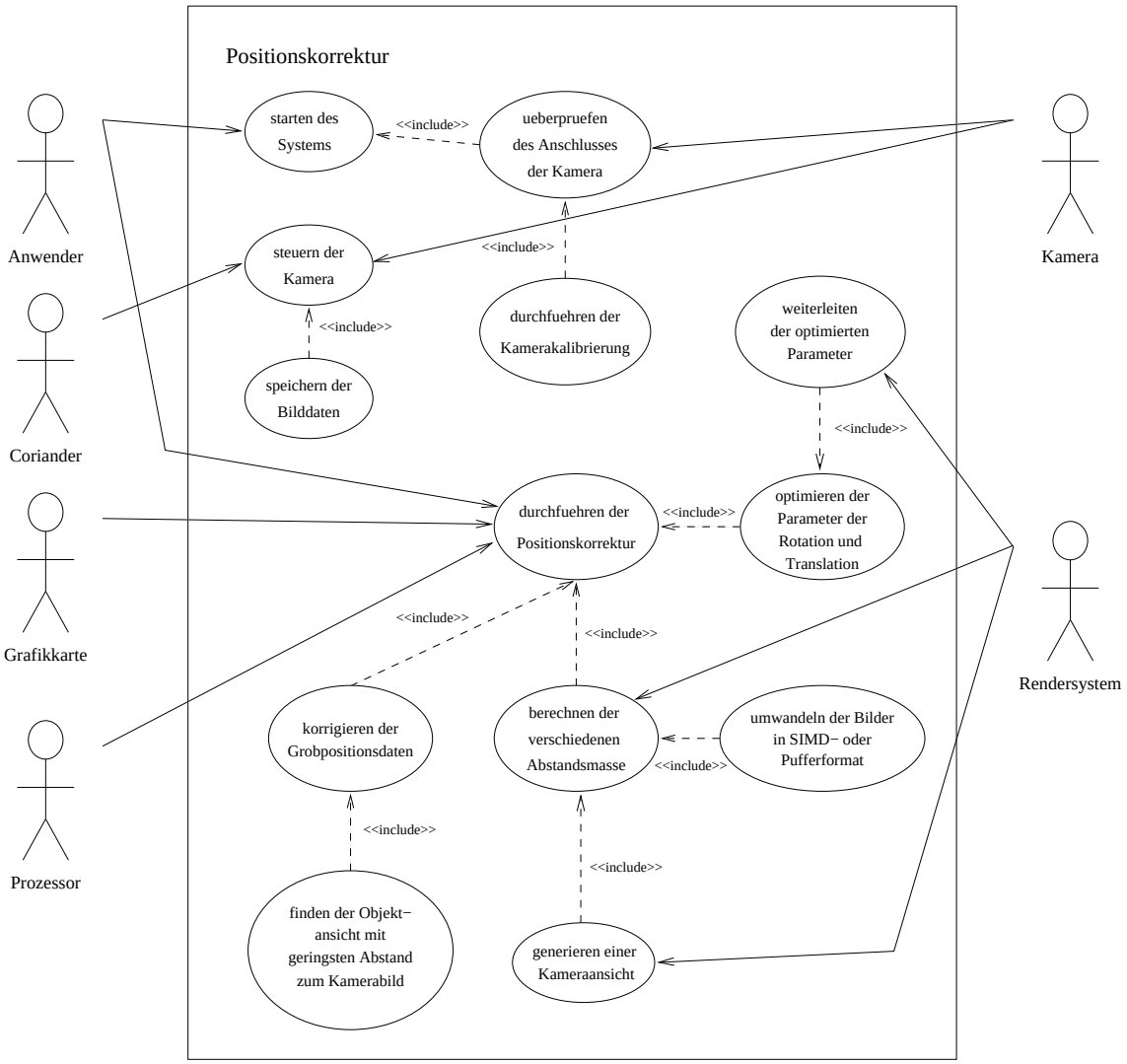


Bild 4.1: Anwendungsfälle des Systems

Der Geschäftsprozess *Start der Bildregistrierung* bilden den Übergang zur Durchführung der 2-D/3-D-Registrierung und wird vom Akteur **Anwender** gestartet. An der Registrierung selbst sind nach dem Start die Akteure **Grafikkarte**, **Prozessor** und **Rendersystem**, beteiligt. Die Positionskorrektur wird im Wesentlichen durch die Berechnung der Abstandsmaße und die Parameteroptimierung von Rotation und Translation gebildet. Die Abstandsmaßberechnung setzt voraus, dass dem System ein Kamerabild und eine synthe-

tisch erzeugte Ansicht auf das 3-D-Modell von der Syntheseeinheit zur Verfügung steht. Die Berechnung der Abstandsmaße kann entweder durch den Grafikkartenprozessor oder durch die CPU durchgeführt werden.

Funktionspakete

Aufbau: Das System teilt sich in insgesamt vier Funktionsbereiche auf. Jeweils zwei Pakete bilden die Systemsteuerung und die eingentliche Berechnungseinheit, der Positionskorrektur. Zu den Steuerelementen gehören die Kamerasteuerung und die Benutzungsoberfläche, die sowohl Einfluss auf die Kamera als auch auf einzelnen Prozesseinheiten hat.

Die Benutzungsoberfläche verbindet alle Pakete miteinander und übernimmt die Parametrisierung des Funktionspakets *Positionskorrektur*. Dazu gehört die Auswahl der Prozesseinheit, die Datenverwaltung und die Prozessverwaltung. Darüber hinaus muss die Benutzungsoberfläche eine Funktionalität für die Bildrepräsentation besitzen, um den Fortschritt der Registrierung direkt am Bild nachvollziehen zu können. Zusätzlich dazu werden auch die einer Ansicht zugeordneten Daten der Abstandsmaßberechnung und der Optimierungsergebnisse dargestellt.

Die Bildregistrierung besteht aus vier Unterpaketen. Das Funktionspaket *Bildverarbeitung* besteht zum einen aus der Bildgewinnung, die den Zugriff auf die in einer Bilddatei gespeicherten Daten bewirkt, und der Bildkonvertierung. Die Konvertierung ist notwendig, da einige der verwendeten Abstandsmaße nicht direkt auf den Bilddaten arbeiten. Aus diesem Grund müssen dem System neben den reinen Bilddaten auch die Gradientenbilder und das Histogramm zur Verfügung stehen. Darüber hinaus müssen die Bilddaten zur Benutzung auf der Grafikkarte in eine Textur überführt werden.

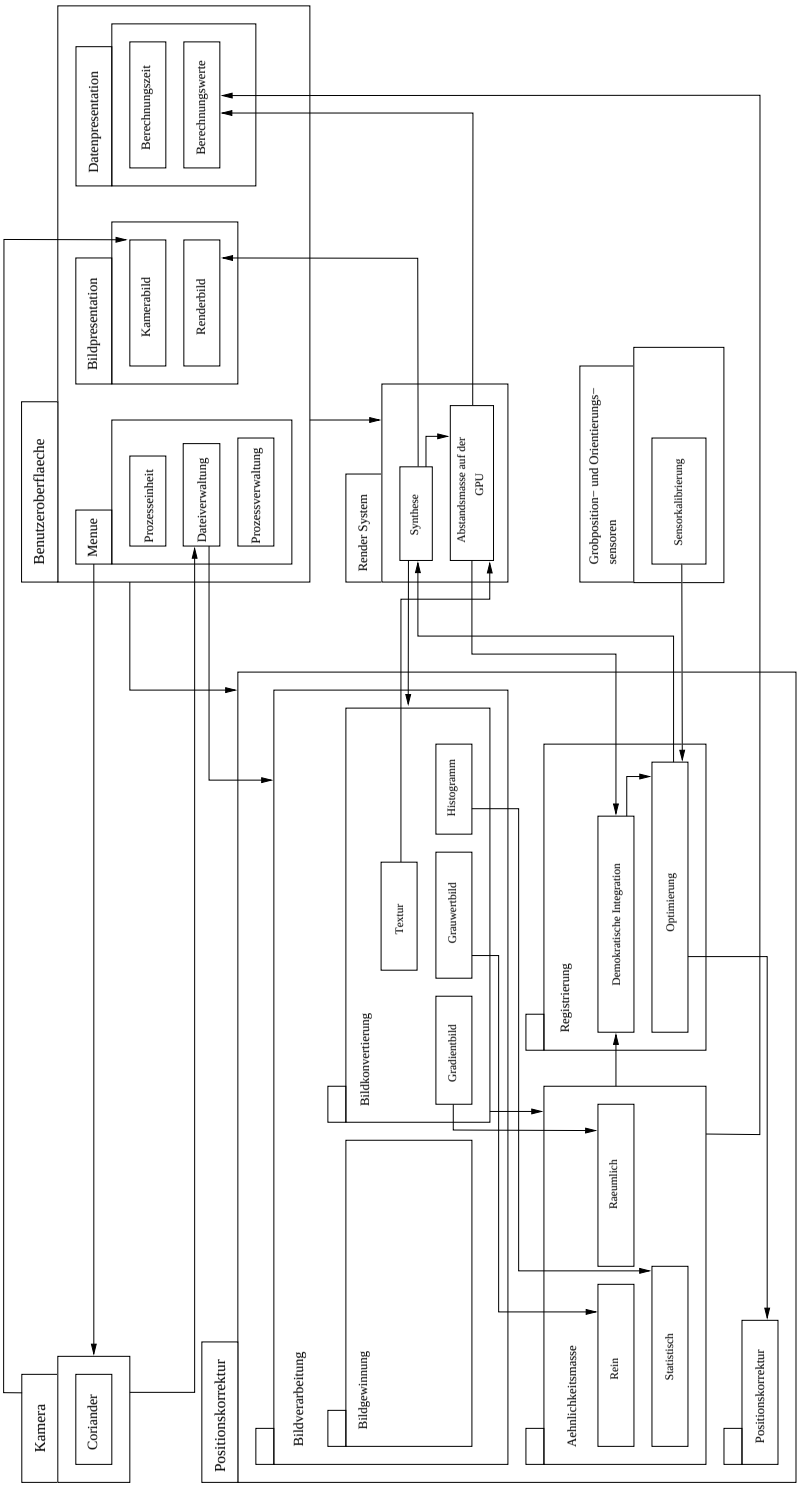


Bild 4.2: Funktionspakete des Systems

Der Kern der Registrierung wird von den drei Paketen *Abstandsmaßberechnung, Optimierung* und *Bildsynthese* gebildet. Die Abstandsmaße werden in drei Gruppen unterteilt, die bereits in Kapitel 2.3.1 ausführlich vorgestellt worden sind. Das Funktionspaket *Optimierung* wird durch zwei Bereiche gebildet: Auf Grundlage der Ergebnisse der Demokratischen Integration, die den ersten Bereich bildet, wird dann im zweiten Paket zusammengefasste *Optimierung* durchgeführt.

Verbindung zwischen den Paketen: Nach dem Start der Benutzungsoberfläche muss dem System über diese, als erster Grundbestandteil der Registrierung, eine Bildaufnahme zugeführt werden. Danach können die Einzelverfahren der Bildregistrierung für die Durchführung ihrer Berechnungen eingestellt werden.

Im Positionskorrekturpaket bekommen die Abstandsmaße die für die Berechnung notwendigen Bilddaten von der Bildkonvertierungseinheit. Nachdem alle Abstandsmaße berechnet sind, werden die Parameter für Rotation und Translation neu geschätzt. Zusätzlich werden die Abstandsmaße von der Demokratischen Integration bewertet. Nach Erhalt der neu geschätzten Parameter wird von der Bildsyntheseeinheit eine neue Ansicht berechnet und erneut den Abstandsmaßen zugeführt. Je nach Auswahl der Prozesseinheit werden die Bildabstände auf der Grafikkarte oder auf der CPU berechnet.

4.1.2 Klassendiagramme der Funktionspakete

Bildklassen:

Alle im System verwendeten Bildobjekte sind vom Typ *ImageArray*. In dieser Klasse werden die Grundoperationen, die für alle Klassen nutzbar sein müssen, deklariert. Zu diesen Grundoperationen zählen das Setzen und Ausgeben von bildbeschreibenden Daten, wie Angaben über Höhe und Breite eines Bildes oder die Anzahl der verwendeten Farbkanäle. Die von dieser Basisklasse abgeleiteten Bildklassen unterscheiden sich zum einen dadurch, dass die Pixelwerte mit unterschiedlicher Bittiefe im Bilddatenfeld kodiert werden.

Die Klasse *ImageArrayUChar* weist mit ihren Namen darauf hin, dass die Bildwerte mit einer 8 Bit Präzision abgespeichert werden. Diese Klasse wird im System für die Daten-

kapselung eines Grauwertbildes verwendet. Darüber hinaus werden Operationen definiert, die für die Berechnung des Gradientenbildes, des normalisierten Bildes und der für die Abstandsmaßberechnung benötigten Mittelwerte gebraucht werden. Diese Berechnungsoperationen dienen gleichzeitig als Zugriffsmethoden auf die Attribute dieser Klasse.

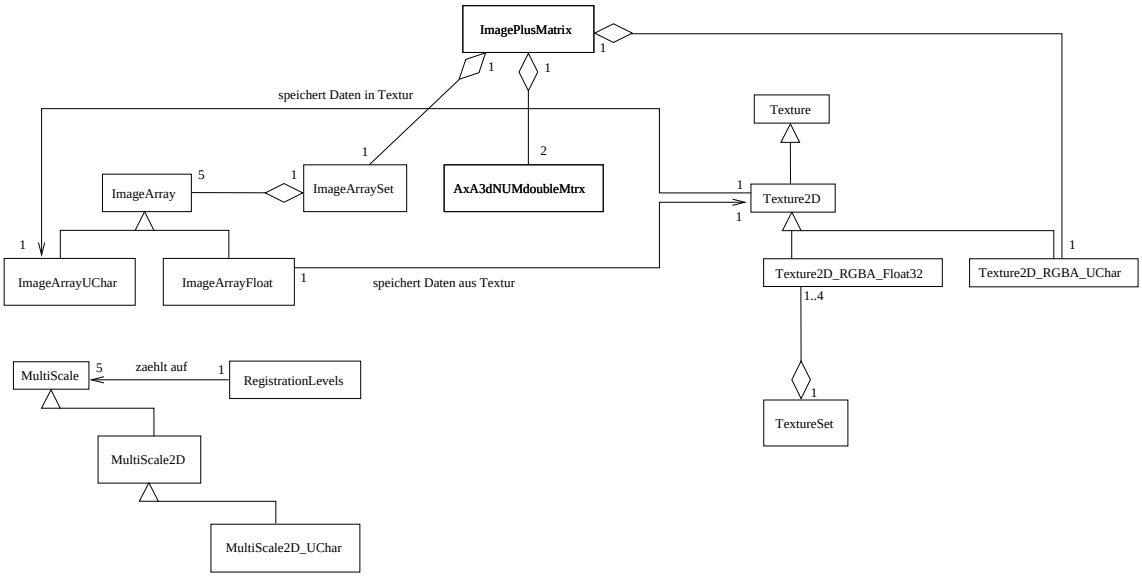


Bild 4.3: Bild- und Texturdatenklassen

Die zweite von *ImageArray* abgeleitete Klasse ist *ImageArrayFloat* und dient, wie der Name andeutet, zur Kapselung von Bilddaten, die mit einer Tiefe von 32 Bit kodiert sind. Im konkreten Anwendungsfall wird diese Klasse dazu verwendet, Texturdaten, die auf der Grafikkarte berechnet wurden, in den Arbeitsspeicher zu laden und sie gleichzeitig in einem Objekt zu kapseln.

Alle für die Registrierung benötigten Daten, einschließlich der Bilddaten, werden als Attribute in der Klasse *ImagePlusMatrix* gespeichert. Diese Klasse kann als eine Art Behältnis angesehen werden, aus dem die Berechnungsverfahren der Bildregistrierung ihre Daten beziehen. Zu diesen Daten gehören im Falle eines synthetisch erzeugten Bildes die Transformationsmatrix vom Typ *AxAdNUMdoubleMtrx*, das Bild selbst und die dem Bild erzeugten verschiedenen Auflösungsstufen. Insgesamt werden der Bildregistrierung in *RegistrationLevel* aufgezählten fünf Auflösungstufen (64×64, 128×128, 256×256, 512×512

und 1024×1024) zur Verfügung gestellt. Die Operationen für die Berechnung der verschiedenen Bildauflösungen befinden sich in den drei Klassen, *MultiScale*, *MultiScale2D* und *MultiScale2D_UChar*. Zusätzlich wird der Klasse *ImagePlusMatrix* zur Organisation der verschiedenen Auflösungsstufen die Klasse *ImageArraySet* bereit gestellt.

Zur Berechnung der Abstandsmaße auf dem Grafikprozessor wird zu jedem Bildobjekt die korrespondierende Textur erstellt und ebenfalls in der Klasse *ImagePlusMatrix* als Objekt gespeichert. Alle Texturklassen sind von der Klasse *Texture* abgeleitet. Die Klasse *Texture2D* beinhaltet als Kernoperation die Generierung der sogenannten „Mipmap-Levels“ mit deren Hilfe die Mittelwerte einer Textur berechnet werden können. Mit der Klasse *Texture2D_RGBA_UChar* werden alle Texturen zusammengefasst, die als Eingabe der Abstandsmaßberechnung auf der Grafikkarte dienen. Alle Ausgabetexturen, in denen die Endergebnisse der Abstandsmaßberechnung gespeichert werden, sind vom Typ *Texture2D_RGBA_Float32*.

Abstandsmaße:

Die Basisklasse aller Abstandsmaße ist *SimilarityMeasure*. Von ihr sind alle im System vorhandenen Abstandsmaße abgeleitet. In *SimilarityMeasure* werden grundlegende Operationen, wie z. B. die Ausgabe des Namens eines Abstandmaßes, deklariert. Außerdem werden Umrechnungsmethoden zur Verfügung gestellt, die ein errechnetes Maß für die Berechnung einer Verteilung, z. B. die Normalverteilung, umwandelt. Für die Evaluation der Abstandsmaße durch die Demokratische Integration werden ebenfalls in *SimilarityMeasure* Operationen deklariert.

In den einzelnen Abstandsmaßklassen wird die virtuelle Operation *compute()* der Klasse *SimilarityMeasure* definiert. Darüber hinaus wird bei einigen Abstandsmaßen eine Methode *assimilate()* definiert, die den errechneten Abstand so umwandelt, dass ein hoher Betrag für eine hohe Übereinstimmung der Bilder steht, und der Wert im Bereich $[0, 1]$ liegt.

Mit den Klassen *SimilarityMeasuresGPU* und *SimilarityMeasuresCPU* wird die Menge der Abstandmaße in zwei Gruppen strukturiert. In *SimilarityMeasuresGPU* werden alle Abstandsmaße zusammengefasst, die sowohl auf der CPU als auch auf der GPU berechnet

werden können. In *SimilarityMeasuresCPU* sind nur Abstandsmaße zu finden, die auf der CPU genutzt werden.

Alle Abstandsmaße werden vor der Bildregistrierung in Listen vom Typ *SimilarityMeasureList* organisiert. Die Liste wird zur Laufzeit durch ein Singelton[3]-Objekt vom Typ *SimilarityMeasureCreator* erstellt. Die Berechnung aller Bildabstände wird durch die Operation *compute()* der Klasse *SimilarityCluster* durchgeführt, indem sie die *compute()*-Operation aller in *SimilarityMeasureList* enthaltenen Abstandsmaßobjekte aufruft. Die für die Berechnung notwendigen Bilddaten werden, wie schon erwähnt, von der Klasse *ImagePlusMatrix* zur Verfügung gestellt.

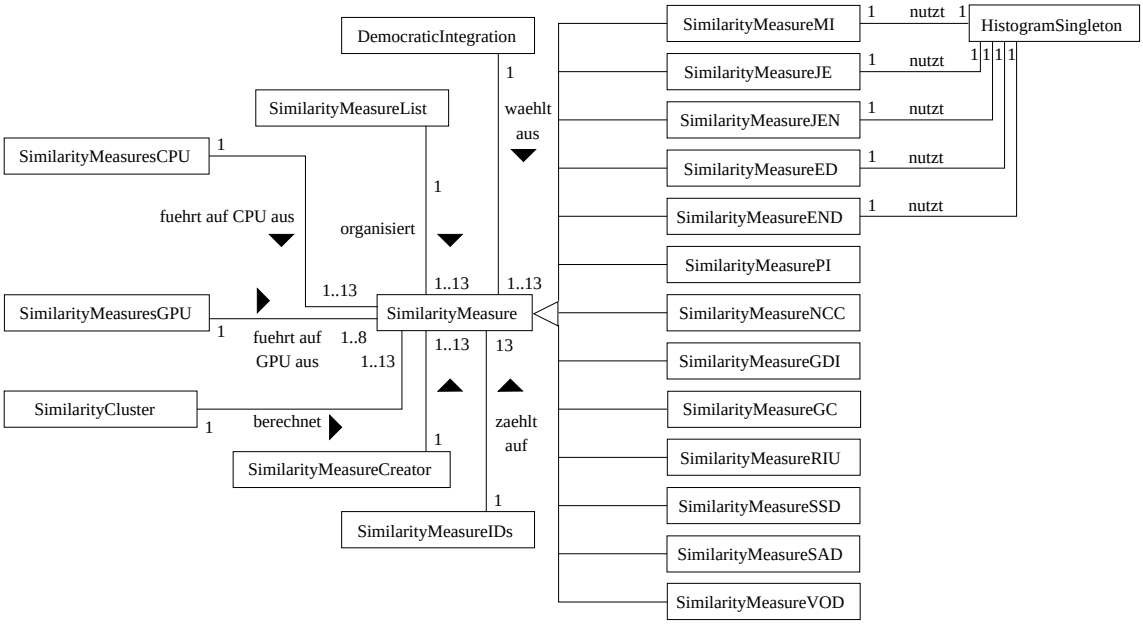


Bild 4.4: Abstandsmaßklassen und ihre Strukturklassen

Falls von der Applikation eine Benutzungsoberfläche verwendet wird, wird ein Objekt der Klasse *SimilarityMeasureList* im Singelton DataObserver gespeichert. Dieses wird für die Darstellung der errechneten Werte an die Benutzungsoberfläche weitergereicht. Eine genaue Darstellung des Datentransfers wird in Abschnitt 4.1.3 gegeben.

[3]Entwurfsmuster einer Klasse, von der zur Laufzeit nur ein Objekt existieren kann

Abstandsmaße auf dem Grafikprozessor:

Die Klassen, die zum Funktionspaket „Abstandsmaße auf der GPU“ gehören, teilen sich in zwei Gruppen. Zur ersten Gruppe gehören die OpenGL- und Shader-Konfigurationsklassen, die die in Abschnitt 3.2.1 im Detail vorgestellten Operationen deklarieren und definieren. Zu diesen Klassen gehören die Singeltons *GLManager*, für die Initialisierung eines OpenGL-Kontextes und für die Erstellung eines Cg-Programmkontextes *ProgramContext*. Zusätzlich wird in *GLManager* das Sichtfeld (engl. viewport) der im Rendersystem verwendeten Kamera definiert. Aufgrund des Singelton-Entwurfmusters und der damit verknüpften Eigenschaft, dass zur Laufzeit des Systems nur ein Objekt der Klasse existiert, gelten die in diesen Instanzen vorgenommenen Einstellungen auch für alle anderen Objekte, die auf diese Singleton Bezug nehmen. Die Klasse Framebuffer, die ebenfalls als Singleton aufgebaut ist und zu den Konfigurationsklassen gezählt werden kann, dient der Initialisierung des Framebufferobjektes, an das die Ausgabetexturen angehängt werden.

Die für die Shader-Typen (gemeint sind Vertex- und Fragmentshader) relevanten Einstellung, wie z. B. die Ermittlung des von der Grafikkarte unterstützten Profils, werden mit Operationen der Klassen *VertexProgramm* und *FragmentProgram* vorgenommenen. Beide Klassen definieren die virtuelle Methode *loadShaderProfile()* der Basisklasse *ShaderProgram*.

Mit den Klassen *PreprocessingShader* und *SimilarityShader* werden dem System die konkret zur Ausführung der Abstandsmaßberechnung benötigten Operationen gegeben. Beide Klassen sind von der Basisklasse *Shader* abgeleitet, die Operationen für das Laden, Binden und Ausführen eines Shaders bereitstellt. Der Klasse *PreprocessingShader* wird der Cg-Shader *preprocessing.cg* zugeordnet. In diesem Shader werden die für die Berechnung der räumlich-intensitätsbasierten Abstandsmaße benötigten Gradienten und die zugehörigen Mittelwerte berechnet. Die Shader *similarityMeasuresSimple_frag.cg* und *similarityMeasuresExtended_frag.cg* werden über Operationen der Klasse *SimilarityShader* gesteuert. In *similarityMeasuresSimple_frag.cg* werden die Abstandsmaße SSD, SAD, RIU, VOD und NCC berechnet. Für die in *similarityMeasuresExtended_frag.cg* definierten Abstands-

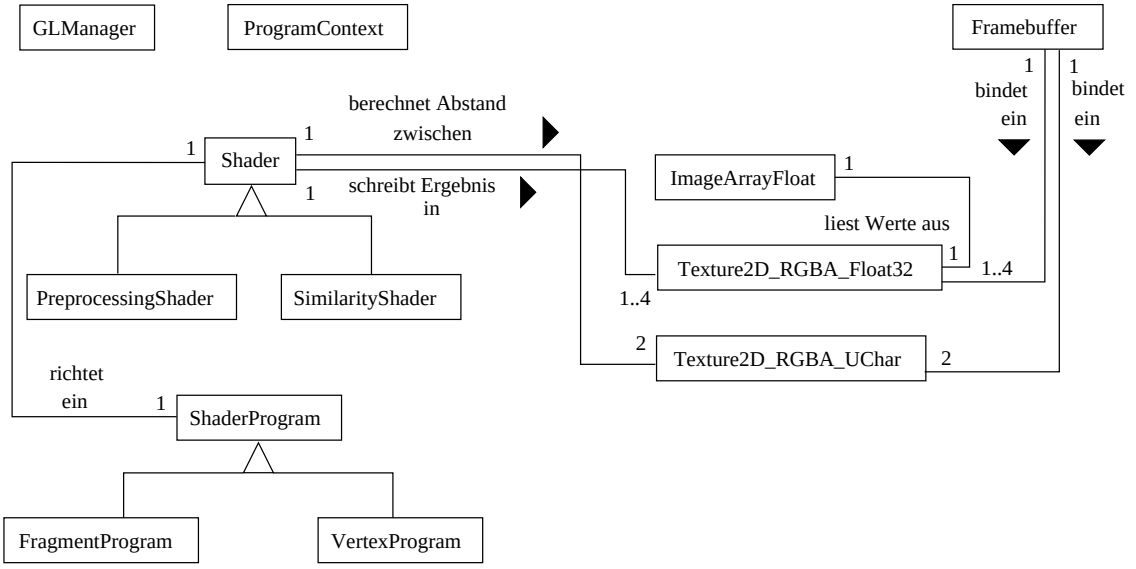


Bild 4.5: Shaderklassen und Klassen für den Aufbau des benötigten Programmkontextes

maße GC, GCI und PI werden die vorher berechneten Werte des *preprocessing*-Shaders verwendet.

Nach der Ausführung eines Shaders werden die errechneten Werte in eine angehängte Textur vom Typ *Texture2D_RGBA_Float32* geschrieben. Um die Daten aus dem Grafikkartenspeicher in den Hauptspeicher laden zu können wird die Klasse *ImageArrayFloat* verwendet. Nach dem Datentransfer werden in den einzelnen Abstandsmaß-Objekten die jeweiligen Werte gespeichert. Damit stehen sie auch für die Darstellung auf der Benutzungsoberfläche zur Verfügung.

Registrierung, Optimierung und Demokratische Integration:

Alle bisher beschriebenen Klassen werden in den Klassen, die die 2-D/3-D-Registrierung realisieren, zusammengeführt. Den Kern der Bildregistrierung bildet die Klasse *RegistrationDemocraticIntegration*. In ihren Operationen zur Parameteroptimierung werden zwei Objekte vom Typ *Optimizer* erzeugt, die die lokale und globale Optimierung der Transformationsparameter durchführen. Wie in Abschnitt 2.3.3 beschrieben, werden die Abstands-

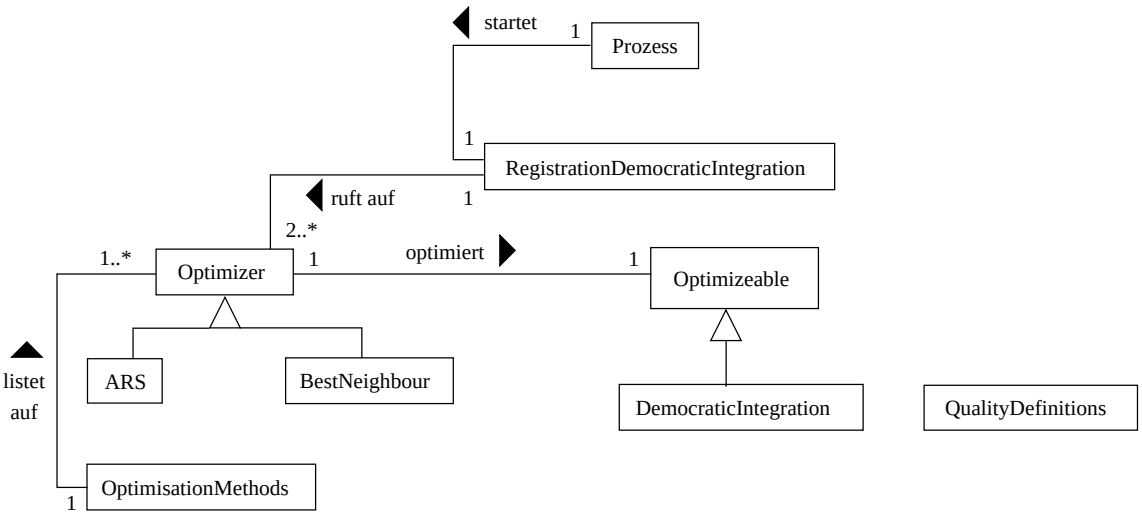


Bild 4.6: Optimierungsklassen des Systems

maße neben ihrer Optimierung zusätzlich durch die Demokratische Integration evaluiert. Dieser Zusammenhang zwischen Optimierung und Evaluierung wird im System dadurch gelöst, dass die Demokratische Integration das zu optimierende Objekt ist. Dies drückt sich im Klassendiagramm dadurch aus, dass die Demokratische Integration eine abgeleitete Klasse von *Optimizable* ist.

Alle im System verwendeten Optimierungsverfahren werden in der Klasse *Optimization-Methods* aufgezählt. Das Erzeugen und Starten der 2-D/3-D-Registrierung findet in der Klasse *Process* mit der Operation *start()* statt.

Fehlerbehandlung:

Für die Fehlerbehandlung (engl. exception handling) stehen insgesamt fünf Klassen zur Verfügung. Die vom System abgefangenen Fehler sind der Nullzeiger mit *NullPointer-Exception*, der fehlende Speicherplatz mit *OutOfMemoryException*, OpenGL-Fehler mit *OpenGLException*, Fehler im Zusammenhang mit der Verwendung der Shader mit *CgException* und Programmfehler, beispielsweise wenn eine Datei unauffindbar ist, mit *ProgramException*. Die Basisklasse aller vom System abgefangener Fehler ist *EvilException*, die ihrerseits von der Standard-C++ Ausnahmeklasse *exception* abgeleitet ist.

4.1.3 Veränderungen und Neuentwicklungen

Die Veränderungen des System, die nach der Analyse vorgenommen wurden, betrafen im Wesentlichen drei Bereiche. Zunächst musste das System, das unter dem Betriebssystem Windows entwickelt wurde, auf das Betriebssystem Linux portiert werden, was vor allem die Verwendung von Klassen betraf, die ausschließlich unter Windows verwendet werden konnten. Darüber hinaus musste die Implementierung der Klassen für eine unter Linux verwendete neuere gcc-Compiler-Version (gcc 4.1.2) abgeändert werden, um den Quelltext so aktuell wie möglich zu halten. Das betraf im verstärkten Maße die Methoden, die in ihrem Methodenrumpf eine Fehlerbehandlung benutzten. Zusätzlich wurde entsprechend der Richtlinien der Arbeitsgruppe Aktives Sehen die Klassendokumentation angepasst.

Nach Fertigstellung der Bildsynthese musste diese mit der Analyseeinheit verbunden werden. Der Aufbau der Kommunikationsschnittstellen musste so gestaltet werden, dass die Daten über Netzwerk zwischen einem Windows-Rechner und einem Linux-Rechner ausgetauscht werden konnten. Dies wurde mit der Entwicklung der Klasse *Httpget* erreicht.

Im dritten Arbeitspaket musste die DRR[4]-Erzeugung aus dem alten System herausgelöst und die betroffenen Methodenaufrufe durch äquivalente Aufrufe für die neue Syntheseeinheit ersetzen werden. Die Operationen für die Bildsynthese werden in der Klasse *Httpget* deklariert.

Einbindung der Syntheseeinheit: Die Klasse Httpget

Die Klasse *Httpget*, die beim Datentransfer zwischen der Synthese- und Analyseeinheit den „Client“ realisiert, ist nach dem Entwurfsmuster Singleton aufgebaut, um den Aufruf der in ihr deklarierten Funktionalität von jeder Stelle des Systems zu ermöglichen. Die Operationen ersetzen Methoden der Klasse *GLManager* und die komplette DRR-Erzeugung der Klassen *DRRGPU* und *DRRShader*. Die Netzwerkkommunikation wird über das Http-Protokoll in der Klasse *QHttp* der C++-Klassenbibliothek Qt implementiert ist realisiert. Um die Signal-Slot-Funktionalität von Qt ausnutzen zu können, ist die Klasse

[4]DRR = digital rekonstruiertes Röntgenbild

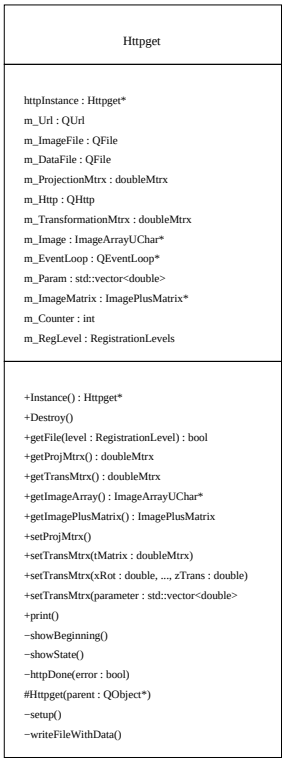


Bild 4.7: Diagramm der Klasse Httpget

von der Qt-Basisklasse *QWidget* abgeleitet. Damit können ereignisgebundene Funktionalitäten ermöglicht werden.

Die Operationen, die den Datentransfer ermöglichen, sind *getFile()* und *httpdone()*. Zusätzlich werden mit den Methoden *showBeginning()* und *showState()* Ausgaben ermöglicht, die den Stand des Datentransfers bekannt geben. Der Datentransfer wird über mehrere Schritte initialisiert und durchgeführt.

Als Erstes werden die in der Optimierung ermittelten Parameter in eine Transfomationsmatrix überführt. Die Rotation wird in Eulerwinkel angegeben, d. h. es wird jeweils eine Rotationsmatrix für jede Achse des Koordinatensystems erstellt und anschließend zu einer Matrix multipliziert. Danach wird die Matrix um eine vierte Dimension erweitert und anschließend mit der Translationsmatrix multipliziert. Diese 4×4-Matrix wird in eine Datei geschrieben, die als Eingabe der Bildsynthese dient. Mit dem Aufruf der Methode

getFile() beginnt der Datentransfer. Zuerst wird der Name der Datei zusammengestellt, in die das von der Bildsynthese geschickte und komprimierte Bild gespeichert werden soll. In diesen Namen wird als Suffix eine Zählvariable mit eingefügt, um die während eines Registrierungsdurchlaufs entstanden Bilddateien voneinander zu unterscheiden. Danach werden die komprimierte Bilddatei zum Schreiben und die Textdatei, die die Werte der Transformationsmatrix enthält, zum Lesen geöffnet. Anschließend wird der Datentransfer so konfiguriert, dass mit den Serveranfragen auch die Eingabedaten für die Bildsynthese aus der Textdatei gelesen werden. Mit der Methode *get()* wird von Serverseite die komprimierte Bilddatei gefordert. Ist die Datei im System eingetroffen, so wird ein *done()*-Signal gesendet und der Datentransfer ist damit beendet.

Die einzelnen Befehle des Http-Protokolls werden vom Qt-Laufzeitsystem in einer Ereignis-Schleife (engl. Event-Loop), die als eine Art Prozesswarteschlange verstanden werden kann, gehalten. Mit diesem Mechanismus soll verhindert werden, dass eine Benutzungsoberfläche während des Abarbeitens der Befehle „einfriert“ . Für den Anwendungsfall der vorliegenden Arbeit ist dieser Mechanismus ungeeignet, da es dadurch zu Speicherzugriffsfehlern beim Zugriff auf nicht vorhandene Bilddateien kommen kann. Aus diesem Grund muss ein Hilfsobjekt vom Typ *QEventLoop* definiert werden, das mit den Befehlen *exec()* und *exit()* die Haupt-Ereignisschleife ansteuert und damit die sofortige Ausführen aller in der Schleife gehaltenen Befehle bewirkt.

Mit der Beendigung des Datentransfers und dem Absetzen des *done()*-Signals wird die Operation *httpdone()* ausgeführt. Als Erstes wird überprüft, ob die Bilddaten vorhanden sind. Danach wird ein Prozess vom Typ *QProcess* definiert, der die komprimierte Bilddatei entpackt. Anschließend wird der Name der entpackten Datei in einer Zeichenkette vom Typ *QString* für die Anzeige auf der Benutzungsoberfläche gespeichert und im *DataObserver*-Objekt gesetzt. Zum Schluss wird die Haupt-Ereignisschleife des Qt-Laufzeitsystems mit *exit()* verlassen.

Aufbau der Benutzungsoberfläche

Die Benutzungsoberfläche repräsentiert die in der Objektorientierten Analyse ermittelten und in Abschnitt 4.1.1 beschriebenen Funktionspakete. Sie ist im Stil von Karteirei-

tern (engl. Tabs) aufgebaut. Die Hauptklasse *GeneralTab* kapselt die Unterreiter *RegistrationTab*, *SimilarityMeasureTab*, *OptimisationTab*, *ImageTab* und *DemocraticIntegrationTab*. Mit den Elementen der Klasse *RegistrationTab* können die zu verwendenden Prozesseinheiten (GPU oder CPU) ausgewählt werden. Außerdem werden in dieser Sektion Anzeigeelemente verwendet, mit deren Hilfe die Zwischenergebnisse der Registrierung angezeigt werden können. Zu den angezeigten Größen gehören die errechneten Werte der Bildabstandsmaße, die aktuellen Parameter der Rotation und Translation sowie das dazugehörige Bild. Die Registrierung selbst wird mit dem Drücken einer Schaltfläche in einem eigenen Prozessstrang (engl. thread) der mit einem Objekt vom Typ *Reg2d3dRegThread* representiert wird, gestartet. Dadurch wird gewährleistet, dass die Benutzungsoberfläche während der Zeit, in der die Registrierung durchgeführt wird, nicht „einfriert“ .

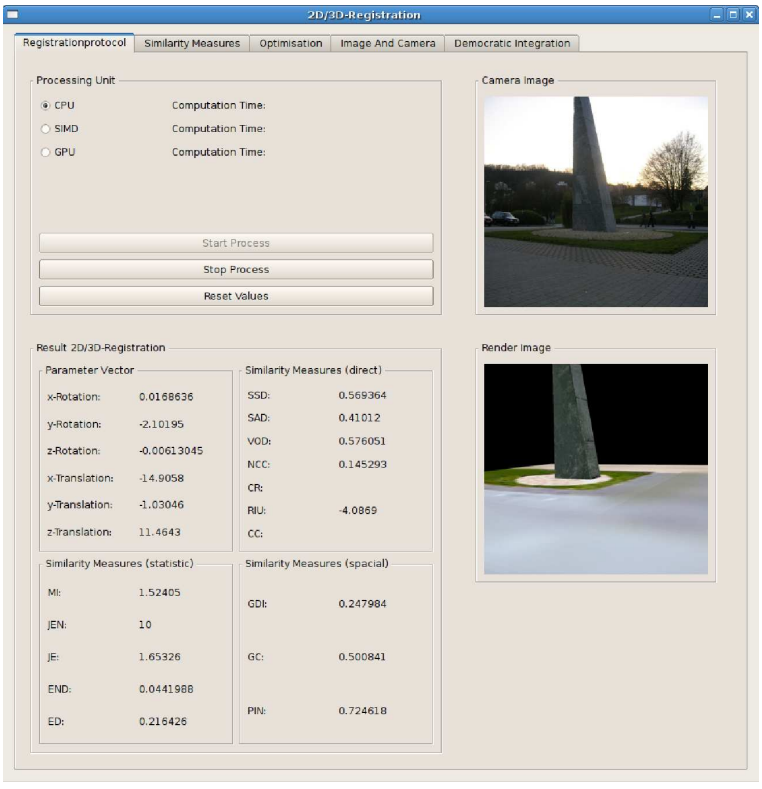


Bild 4.8: Anischt des Protokollreiters der Benutzungsoberfläche

Der zweite Karteireiter, dessen einzelne Bestandteile in der Klasse *SimilarityMeasureTab* gekapselt werden, dient zur Auswahl der Abstandsmaße. Die Auswahl ist mit der Aktivierung einer bestimmten Prozesseinheit gekoppelt. So führt die Selektion des Grafikprozessors dazu, dass keine statistischen Abstandsmaße ausgewählt werden können, da diese nicht für den Gebrauch auf der Grafikkarte implementiert sind. Zusätzlich können für die Berechnung der Abstandsmaße ein „bias“[5](zur Vermeidung von Nulldivisionen), den oberen und unteren Grenzwert der in den Bildern vorkommenden Farben sowie deren gesamten Wertebereich angegeben werden.

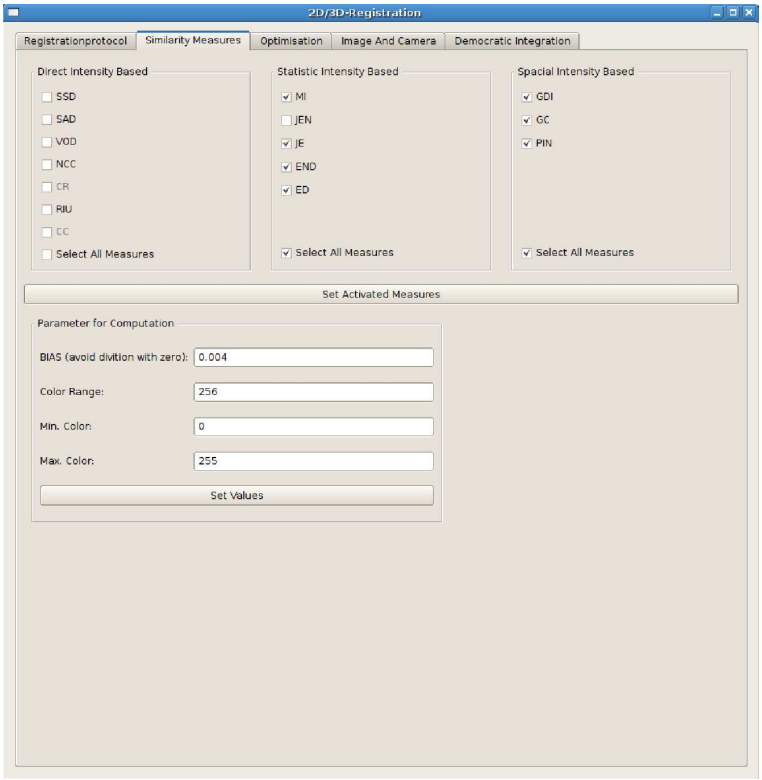


Bild 4.9: Abstandsmaßreiter für die Auswahl und Einstellung der Abstandsmaße

Der Karteireiter, in dem die Einstellungen zur Optimierung vorgenommen werden können, setzt sich aus drei Unterreitern zusammen. Im Ersten kann der für die Optimierung benötigte Bereich definiert werden, in dem die Verfahren das globale Optimum ermitteln

[5]bias = systematischer Fehler

sollen. Der zweite Unterreiter ermöglicht die Auswahl und Parametrisierung eines globalen Optimierungsverfahrens. Für die Einstellung eines lokalen Verfahrens steht der dritte Unterreiter zur Verfügung.

In dem Reiter, der durch die Klasse *ImageTab* auf der Benutzungsoberfläche entsteht, werden alle Einstellung für die Bildverarbeitung getroffen. Dazu gehören die Auswahl des Kamerabildes sowie die Möglichkeit eine Kamerakalibrierung durchzuführen. Hierfür muss ein Verzeichnis, das insgesamt fünf Bilder eines Kalibriermusters enthält, ausgewählt werden. Die fünf Aufnahmen werden dann in das System geladen. Nach Eingabe der Höhe und Breite sowie Größe eines Elementes des Kalibriermusters, kann der Prozessstrang der Kalibrierung gestartet werden. Der letzte Reiter dient zur Parametrisierung der Demokratischen Integration.

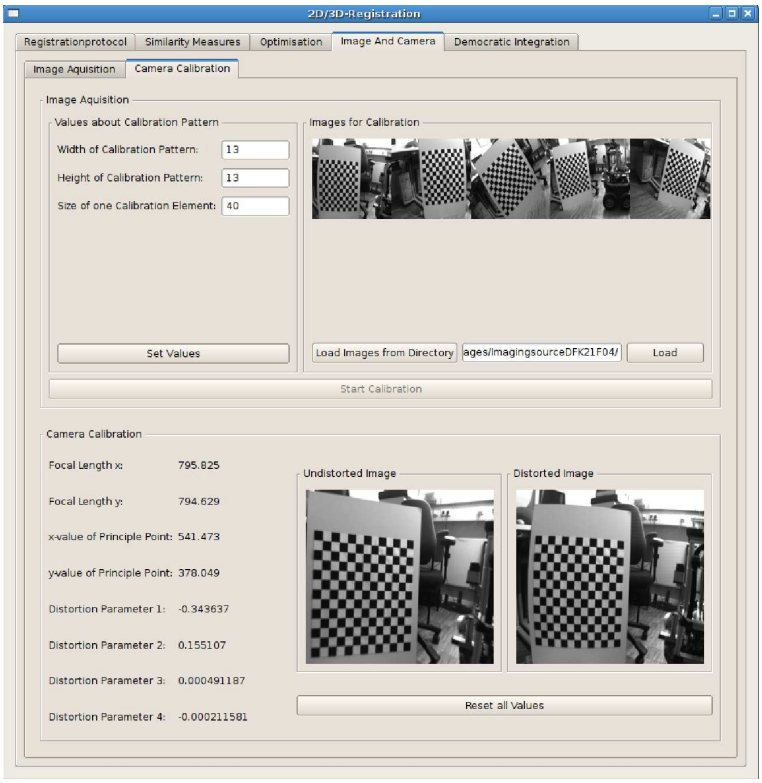


Bild 4.10: Reiter für die Kamerakalibrierung

Aufbau der Kommunikationsschnittstellen des Systems

Für den Datentransfer zwischen der Benutzungsoberfläche und Applikation stehen die Klassen *GUIDataOberserver* und *DataObserver* zur Verfügung. Beide Klassen sind in Form eines Singleton entworfen.

In der von Qt unabhängigen Klasse *DataObserver* werden alle auf der Benutzungsoberfläche eingegebenen Daten für die anschließende Verwendung in der Bildregistrierung gespeichert. Dadurch wird erreicht, dass Klassen, die die Registrierung realisieren, unabhängig von der Benutzungsoberfläche bleiben, denn die Werte können auch ohne die Benutzungsoberfläche im *DataObserver*-Objekt gesetzt werden. Die Ergebnisse, die während der Registrierung berechnet werden, werden ebenfalls im *DataObserver*-Objekt zwischengespeichert. Nach jeder Speicherung wird eine Methode *sendData()* der Klasse *GUIDataObserver* aufgerufen. Diese nutzt den Aufruf, um ein Signal abzusetzen, das der Benutzungsoberfläche signalisieren soll, dass neue Daten von der Applikation vorhanden sind. Mit dem Signal werden gleichzeitig die Werte an die Oberfläche weitergeleitet. Da die Klasse *GUIDataObserver* die Signal-Slot-Funktionalität von Qt verwendet, muss sie von *QWidget* abgeleitet werden. Somit ist diese Klasse im Gegensatz zur Klasse *DataObserver* nicht unabhängig von Qt benutzbar.

Kapitel 5

Experimente und Ergebnisse

Mit den Versuchen soll herausgefunden werden, inwieweit die einzelnen Bildabstandsmaßtypen mit den verschiedenen Bildarten zurechtkommen und inwieweit eine Positionskorrektur noch möglich ist. Mit den Tests soll darüberhinaus auch ermittelt werden, ob für eine nützliche Positionskorrektur grundsätzlich immer photorealistische Bilder notwendig sein müssen oder ob auch eine vereinfachte Darstellung ausreicht, um respektable Ergebnisse zu erzielen. Um herausfinden zu können, inwieweit eine Angleichung der Bildmodalität Auswirkung auf das Registrierungsergebnis hat, wird neben einem Kamera- und einem synthetischen Bild auch ein separiertes Kamerabild für die Experimente verwendet. In diesem Bild sind die Teile ausgeschnitten, die auch in einer synthetisch erzeugten Ansicht fehlen (z. B. Himmel).

5.1 Versuchsaufbau

5.1.1 Kameraufnahme

Für die Kameraaufnahme wurde zur genaueren Bestimmung der Position und Orientierung ein Stativ vom Typ *Hama Omega Pro II* verwendet. Dadurch war es auch möglich mit einer Genauigkeit die Orientierung der Kamera zu ermitteln. Der Standort wurde mit Hilfe eines aus Schnüren gespannten Rahmens abgesteckt. In den Eckpunkten des Rahmens wurde

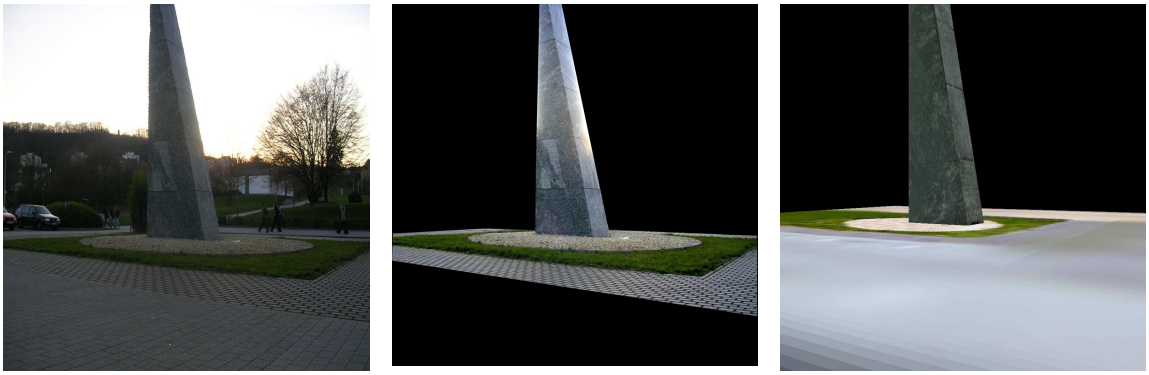


Bild 5.1: Die verschiedenen Bildmodalitäten für die Experimente (Kamerabild, separiertes Bild, synthetisches Bild)

die Kamera aufgestellt. Nach Abschluss der Kameraufnahmen wurden die zwischen den Rahmen gespannten Schnüre ausgemessen.

Durch die so gewonnenen Daten konnte für die Testauswertung eine Ansicht der optimalen Position auf das 3D-Modell synthetisiert werden. Mit der Aufnahme konnte dann eine Bildabstandsmessung mit dem aus der Registrierung stammenden Bild durchgeführt werden. Diese liefert ein Maß für die Genauigkeit dieser Sicht auf das 3D-Modell. Um die Positionskorrektur bewerten zu können, wurden die aus der Ausmessung des Rahmens stammenden Werte und die eingestellten Orientierungswinkel der Kamera genutzt.

5.1.2 Bildsynthese

Die aus der Bildsynthese stammenden Bilder zeigen, wie in Bild 5.1 zu sehen ist, eine stark vereinfachte Ansicht auf die Skulptur, die auf dem Campus der Universität Koblenz-Landau steht. In den Bildern werden keine Schatten oder Detailansichten von anderen Gebäuden oder sonstigen Dingen dargestellt. Die Skulptur steht auf einer vereinfachten Darstellung des Schotters. Darüberhinaus wird noch die umliegende Grünfläche und wenige Teile des Gehweges gezeigt. Die Bilder verfügen über keine Hintergrunddarstellungen. Die Szene wird mit einer Lichtquelle, die immer denselben Standort hat wie die Kamera, angestrahlt.

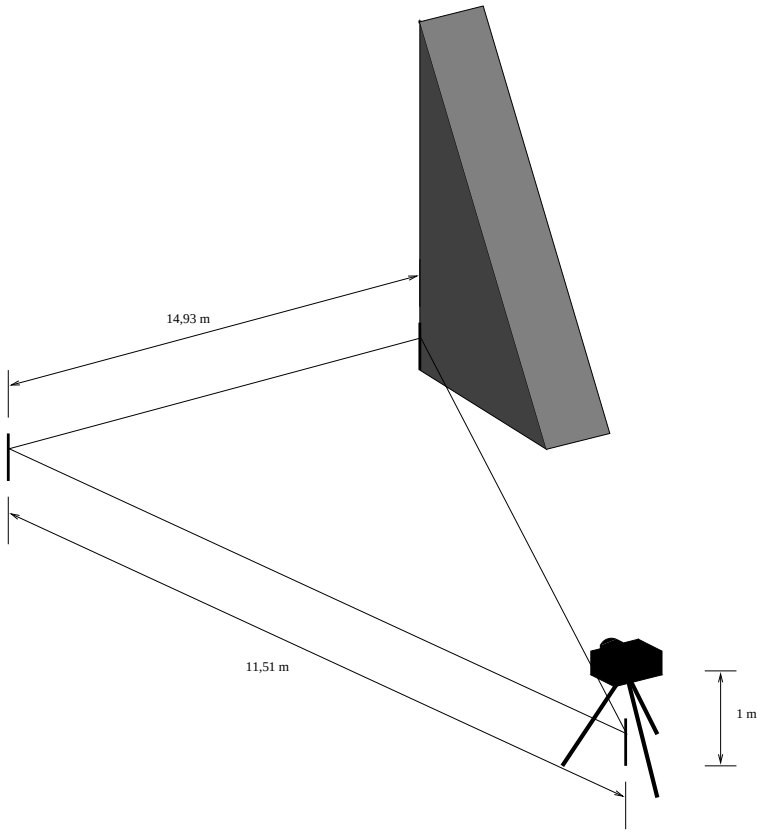


Bild 5.2: Aufbau des Rahmens für die Bestimmung der Kameraposition

Das Weltkoordinatensystem liegt, wie in Abbildung 5.3 zu sehen ist, in der linken unteren Ecke der Skulptur. Die Ausgangsposition der Kamera ist der Ursprung des Weltkoordinatensystems mit Blickrichtung entlang der positiven z-Achse. Die positive y-Achse zeigt nach unten. Die Orientierung der Achsen deutet auf ein um die z- und y-Achse gedrehtes Linkshand-System hin. Um die Transformation der Kamera richtig parametrisieren zu können, muss zuerst die Translationsmatrix mit der Rotationsmatrix multipliziert werden.

5.1.3 Bildregistrierung

Die Registrierung wird unter dem Betriebssystem Linux durchgeführt. Als globales Optimierungsverfahren wird die *Adaptive Zufallssuche (ARS)* verwendet. Die lokale Opti-

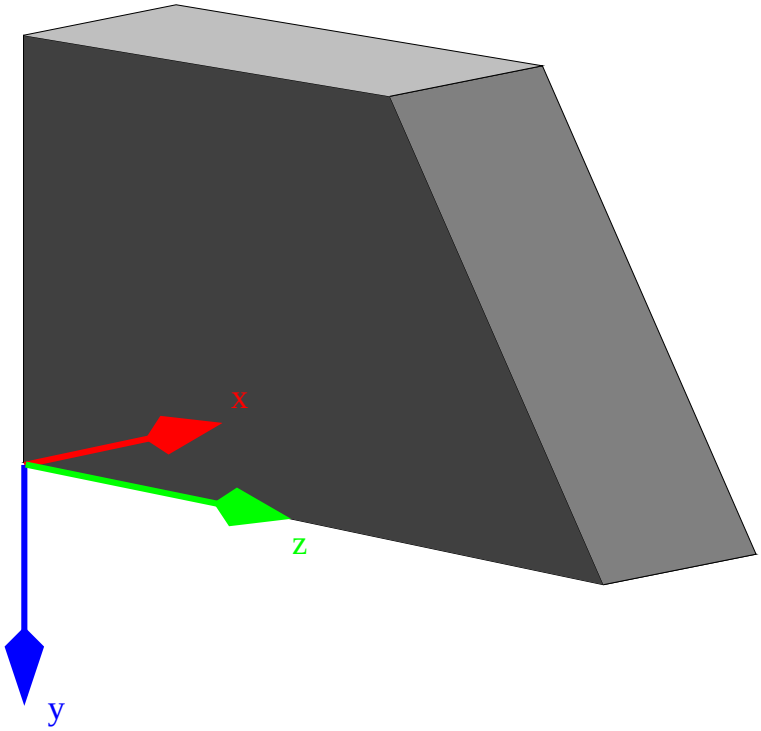


Bild 5.3: Lage des Weltkoordinatensystems im 3-D-Modell

mierung wird mit dem Verfahren *Best Neighbour (BN)* durchgeführt. Es werden alle drei Arten von Bildabstandsmaßen getestet. Je nach Versuchdurchführung werden entweder alle Abstandsmaße, Abstandsmaßtypen oder Kombinationen von Typen eingesetzt. Die Abstandsmaße, die für die Registrierung verwendet werden, sind in Tabelle 5.1 aufgelistetet.

Für die Optimierungsverfahren gelten bei allen Experimenten die gleichen Einstellungen. Der Suchraum für die Optimierungsverfahren wird um den ausgemessenen optimalen Punkt gelegt. Für die Parameter der Translation wird ein Suchraum mit einem Radius von insgesamt 70 cm aufgespannt. Für die Rotation ist eine Abweichung von insgesamt $3°$ von der optimalen Position festgelegt.

direkt-intensitätsbasiert	statistisch-intensitätsbasiert	räumlich-intensitätsbasiert
SSD	MI	GDI
SAD	JE	GC
VOD	END	PIN
NCC	ED	

Tabelle 5.1: Verwendete Abstandsmaße für die Experimente

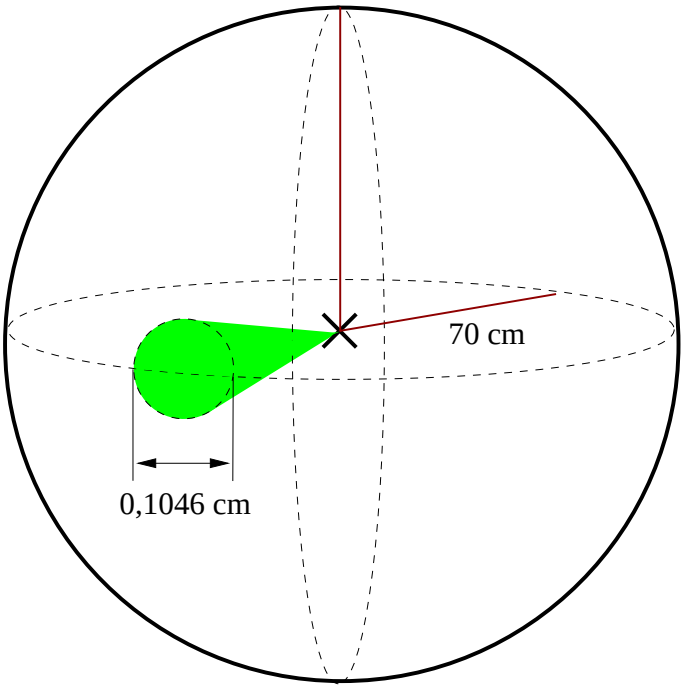


Bild 5.4: Suchraum, der um die optimale Position aufgespannt wird

5.2 Versuchsdurchführung

Alle Experimente werden automatisiert ohne Verwendung der Benutzeroberfläche durchgeführt. Für alle Experimente gelten die gleichen Einstellungen der Optimierungsverfahen und der Abstandsmaße. Insgesamt wird jeder Durchlauf 15 mal wiederholt. Im ersten Experiment werden alle Abstandsmaße eingesetzt. Im zweiten Versuch werden die ver-

schiedenen Typen von Abstandsmaßen getestet. Der letzte Test wird mit verschiedenen Kombinationen von Abstandsmaßtypen durchgeführt.

In jedem Durchlauf generiert das globale Optimierungsverfahren *Adaptive Zufallssuche* gleichverteilte Parameter im Suchraum. Die Parameter werden nach dem Zufallsprinzip ermittelt. Damit ist gewährleistet, dass die Registrierung in jedem Durchlauf von unterschiedlichen Positionen durchgeführt wird.

Die Auswertung der Experimente umfasst die Berechnung des mittleren Fehlers für jede Bildmodalität und die Berechnung der durchschnittlichen Abweichung von den optimalen Werten. Der mittlere Fehler wird über die Summe der quadratischen Differenzen gebildet.

5.3 Ergebnisse

5.3.1 1. Experiment: Verwendung aller Abstandsmaße

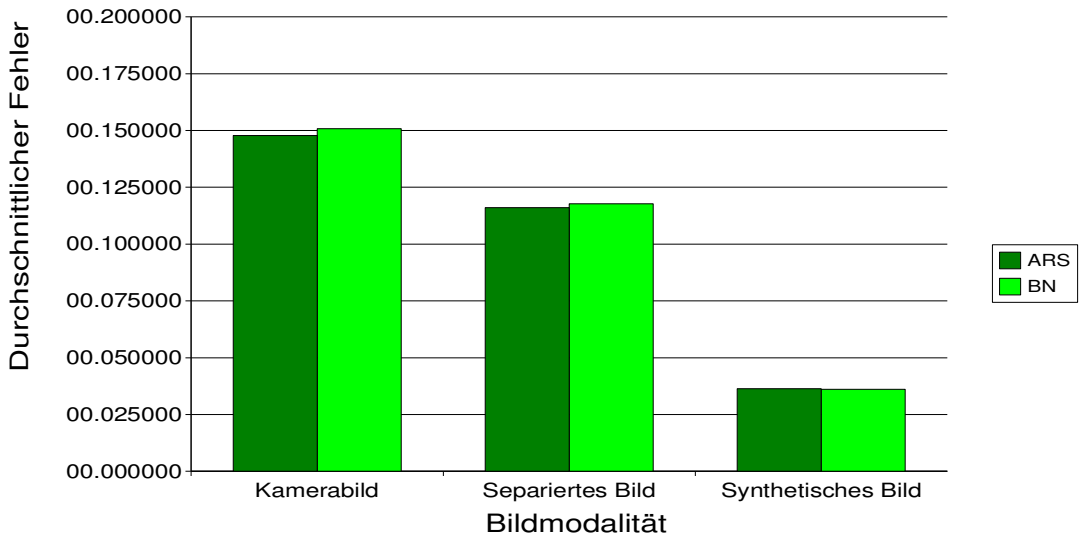


Bild 5.5: Durchschnittlicher Fehler bei der Verwendung aller Abstandsmaße

	Kamerabild	separiertes Bild	synthetisches Bild
Abweichung Translation	50 cm	42 cm	19 cm
Abweichung Rotation	2.5°	1.56°	0.6°

Tabelle 5.2: Durchschnittliche Abweichung von der optimalen Position bei Verwendung aller Abstandsmaße

Das Diagramm 5.5 zeigt, dass die Bildmodalität einen starken Einfluss auf das Ergebnis der Positionsbestimmung hat. Das lokale Optimierungsverfahren verschlechtert bei der Verwendung von Kamerabild und separiertem Bild das Ergebnis. Dieses Phänomen ist dadurch zu erklären, dass die *direkt-intensitätsbasierten* Abstandsmaße mit den im Bild vorkommenden Pixelwerten rechnen. Das synthetische Bild weist an vielen Stellen den Pixelwert null auf. Aus diesem Grund ist an diesen Bildpositionen die Pixeldifferenz sehr groß. Um das Gesamtergebnis zu verbessern wird die Kamera der Syntheseeinheit durch die Optimierungsverfahren in eine Position gesetzt aus der Bilder generiert werden können, die an möglichst vielen Stellen einen Wert ungleich Null aufweisen. Damit werden neben der Pixeldifferenz auch die Werte der *direkt-intensitätsbasierten* Abstandsmaße verringert. Diese Strategie führt dazu, dass aufgrund des Einflusses der Pixelwerte die Position der Kamera falsch gewählt wird oder die Positionsangaben durch das lokale Optimierungsverfahren verschlechtert werden

Kommt bei der Registrierung das synthetische Bild zum Einsatz, ändert sich auch die Wirkung der Pixeldifferenzen auf das Gesamtergebnis. Der Grund liegt darin, dass in dem Falle in beiden Bildern an denselben Stellen Nullwerte vorhanden sind. Somit ist auch die Pixeldifferenz an diesen Stellen gleich Null und ihr Einfluss auf das Gesamtergebnis gering.

Bei einer Angleichung der Bildmodalität wird eine Verbesserung des Ergebnisses der Positionsbestimmung erreicht. Die durchschnittliche Abweichung in den Parametern der Translation ist um fast 10 cm geringer als bei der Verwendung des unbearbeiteten Kamerabildes. Die Abweichung in den Parametern der Rotation wird um 1° verringert.

5.3.2 2. Experiment: Verwendung einzelner Abstandsmaßtypen

Direkt-Intensitätsbasiert

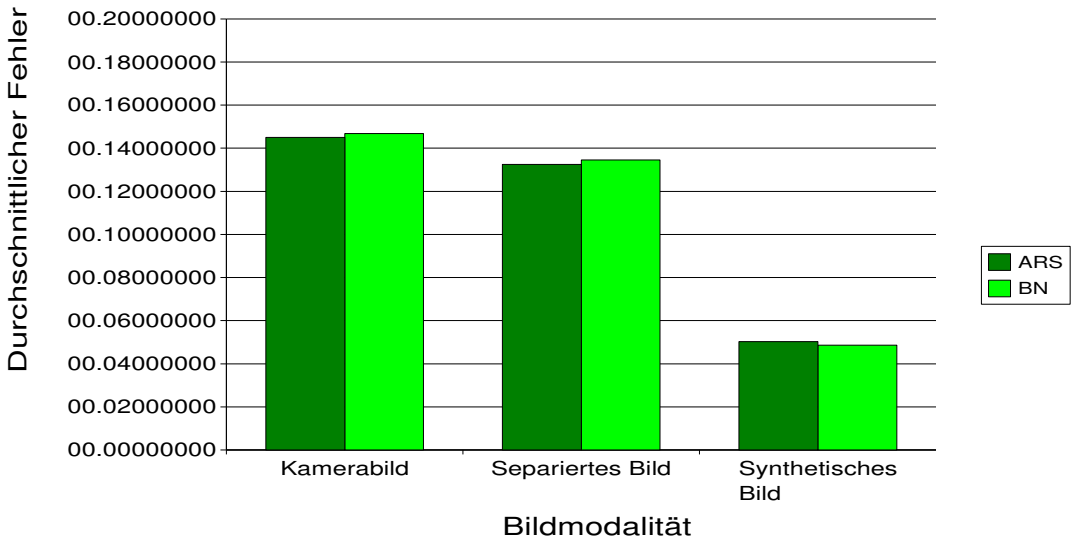


Bild 5.6: Durchschnittlicher Fehler der direkt-intensitätsbasierten Abstandsmaße

Die *direkt-intensitätsbasierten* Abstandmasse weisen einen hohen durchschnittlichen Fehler bei der Verwendung eines Kamerabildes auf. Der Grund liegt in der bereits erläuterten Wirkung der Pixeldifferenz. Kommt ein separiertes Kamerabild zum Einsatz wird eine kleine Verringerung der durchschnittlichen Abweichung erreicht. Für die Parameter der Translation beträgt die Verkleinerung 3 cm und für die Rotationsparameter ungefähr $0.7°$. Der kleinste durchschnittliche Fehler wird durch Gebrauch derselben Bildmodalität erreicht.

Statistisch-Intensitätsbasiert

Mit *statistisch-intensitätsbasierten* Abstandsmaßen wird die Positionskorrektur mit dem geringsten durchschnittlichen Fehler durchgeführt. Dieses Ergebnis wird durch die Werte

	Kamerabild	separiertes Bild	synthetisches Bild
Abweichung Translation	47 cm	44 cm	24 cm
Abweichung Rotation	2.24°	1.53°	0.5°

Tabelle 5.3: Durchschnittliche Abweichung von der optimalen Position bei Verwendung von direkt-intensitätsbasierten Abstandsmaßen

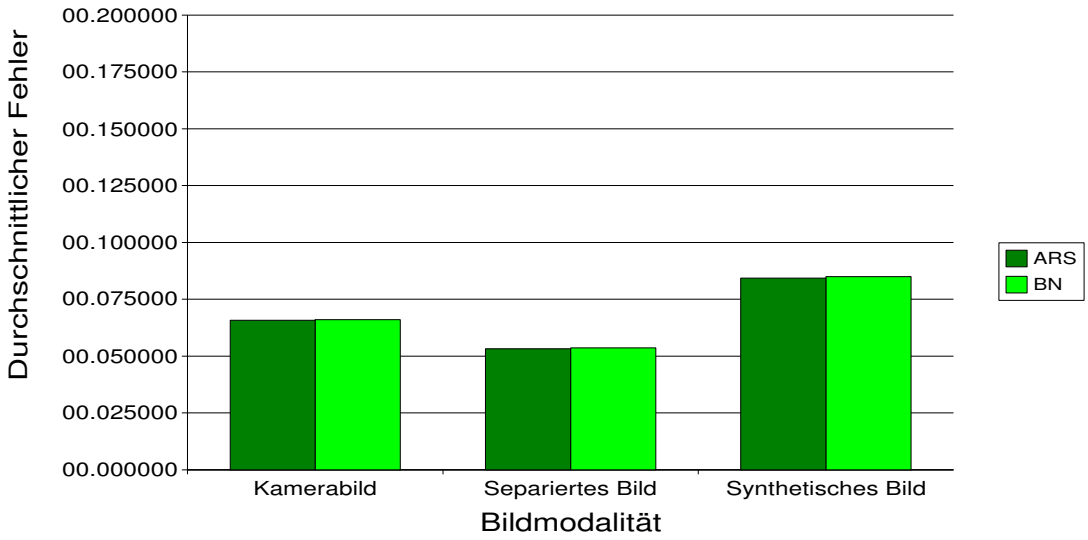


Bild 5.7: Durchschnittlicher Fehler der statistisch-intensitätsbasierten Abstandsmaße

der durchschnittlichen Abweichung von der Optimalposition bestätigt. Der größte Fehler entsteht bei der Verwendung der gleichen Bildmodalität. Dieses Ergebnis könnte mit der geringen Entropie der synthetischen Bilder zusammenhängen. Um genauere Aussagen über dieses Phänomen machen zu können, sollte dieser Abstandsmaßtyp mit einer größeren Iterationszahl getestet werden.

Eine Verbesserung der Positionsbestimmung durch Verwendung eines separierten Kamerabildes ist auch bei diesem Abstandsmaßtyp zu beobachten. Die durchschnittliche Abweichung in den Translationswerten konnte um 3 cm und in den Rotationsparametern um 0.5° verringert werden.

	Kamerabild	separiertes Bild	synthetisches Bild
Abweichung Translation	30 cm	27 cm	30 cm
Abweichung Rotation	2.10°	2.5°	2.8°

Tabelle 5.4: Durchschnittliche Abweichung von der optimalen Position bei Verwendung von statistisch-intensitätsbasierten Abstandsmaßen

Räumlich-Intensitätsbasiert

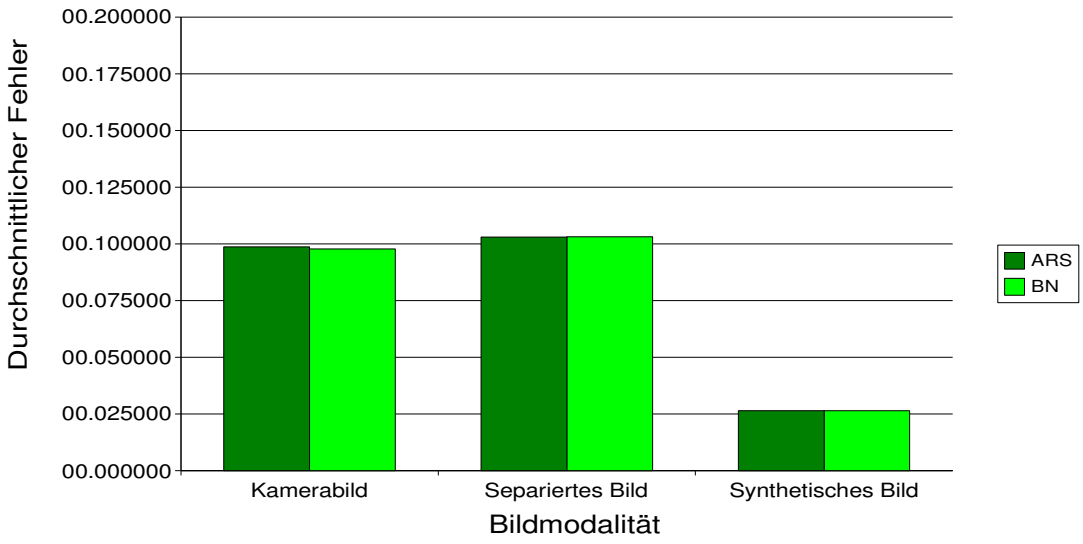


Bild 5.8: Durchschnittlicher Fehler der räumlich-intensitätsbasierten Abstandsmaße

Die *räumlich-intensitätsbasierten* Abstandsmaße weisen den höchsten durchschnittlichen Fehler beim Einsatz des separierten Kamerabildes auf. Durch das Entfernen der verschiedenen Bestandteile sind neue Kanten an Positionen im Bild entstanden, die vorher nicht existierten. Die Optimierungsverfahren versuchen, während der Registrierung, diese Kanten auch im synthetischen Bild entstehen zu lassen, indem die Parameter für die Position der Kamera entsprechend gewählt werden. Das führt, wie schon im Fall der *direkt-intensitätsbasierten* Abstandsmaße, zu einer falschen Positionierung der Kamera.

	Kamerabild	separiertes Bild	synthetisches Bild
Abweichung Translation	39 cm	37 cm	18 cm
Abweichung Rotation	1.29°	1.29°	0.79°

Tabelle 5.5: Durchschnittliche Abweichung von der optimalen Position bei Verwendung von räumlich-intensitätsbasierten Abstandsmaßen

Gegenüberstellung aller drei Abstandsmaßtypen

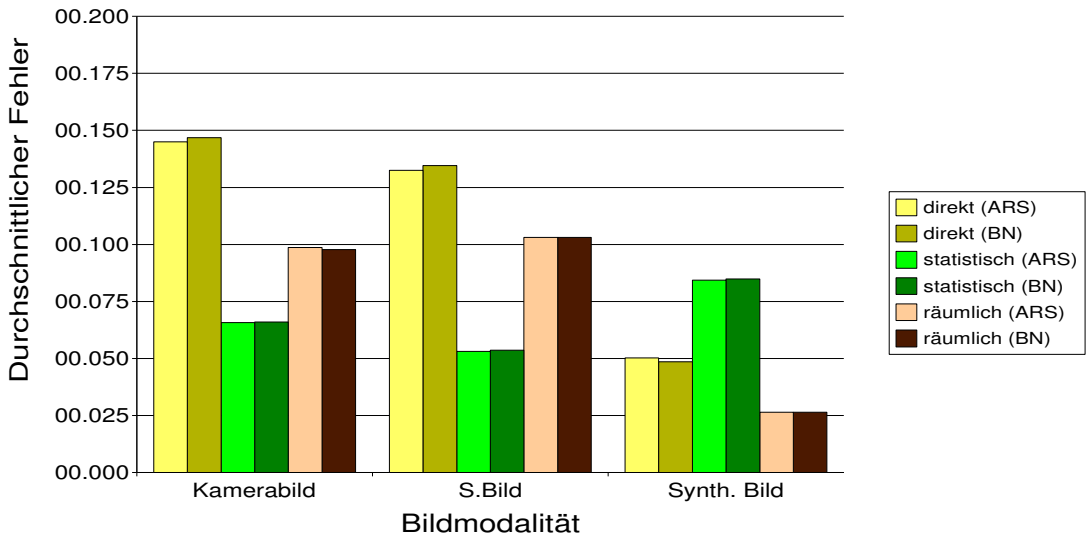


Bild 5.9: Gegenüberstellung aller Abstandsmaßtypen

Der kleinste durchschnittliche Fehler bei der Anwendung von Kamerabild und separiertem Bild wird mit den *statistisch-intensiätsbasierten* Abstandsmaßen erreicht. Werden für die Registrierung Bilder der gleichen Modalität verwendet, weisen die *räumlich-intensitätsbasierten* Abstandsmaße den kleinsten durchschnittlichen Fehler auf. Der größte durchschnittliche Fehler tritt mit der Anwendung der *direkt-intensitätsbasierten* Abstandsmaße auf. Dieses Ergebnis kann auch die Begründung dafür liefern, dass bei der Anwendung aller Abstandsmaße das Gesamtergebnis schlechter ausfällt als bei der Anwendung von rein *statistisch-* oder rein *räumlich-intensitätsbasierten* Abstandsmaßen. Während der

Registrierung werden die Ergebnisse aller Abstandsmaße zu einem Gesamtergebnis zusammengefasst. Somit wirken sich die Pixeldifferenzen auch auf das Gesamtergebnis aller Abstandsmaße aus. Die Optimierungsverfahren versuchen durch die Auswahl geeigneter Positionsparameter die Wirkung der Pixeldifferenzen zu minimieren. Dieses Verhalten führt letztlich auch bei der Anwendung aller Abstandsmaße zu einer falschen Positionierung der Kamera.

5.3.3 3. Experiment: Kombination verschiedener Abstandsmaßtypen

Direkt-Statistisch

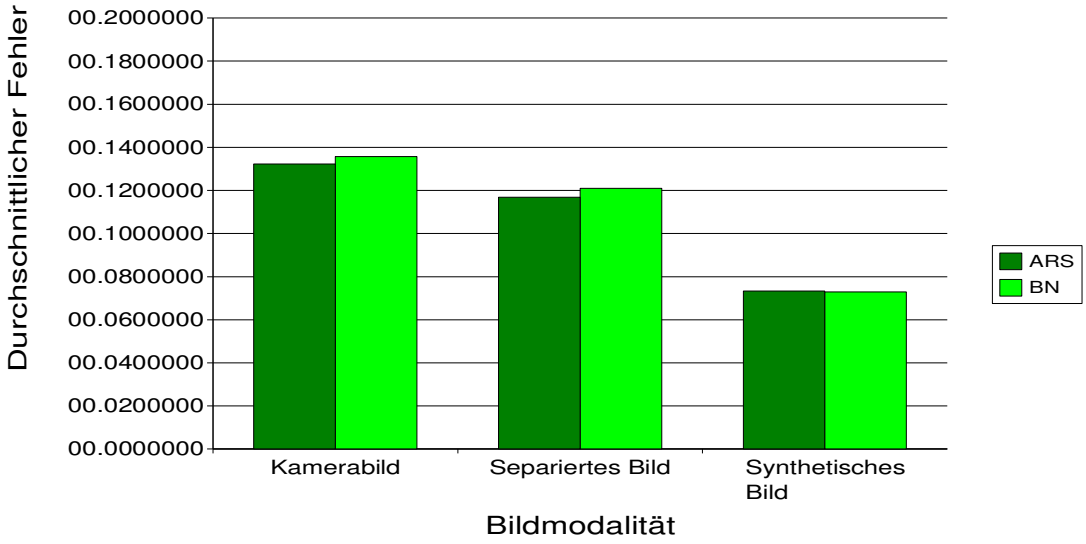


Bild 5.10: Durchschnittlicher Fehler der direkt-/statistisch-intensitätsbasierte Abstandsmaßkombination

Bei der Kombination von *direkt-intensitätsbasierten* und *statistisch-intensitätsbasierten* Bildabstandsmaßen wird das Ergebnis der Positionsbestimmung stark durch die Eigenschaften der *direkt-intensitätsbasierten* Abstandsmaßen beeinflusst. Insgesamt ist das Ergebnis erwartungskonform. Der größte durchschnittliche Fehler tritt bei der Verwendung

	Kamerabild	separiertes Bild	synthetisches Bild
Abweichung Translation	47 cm	43 cm	33 cm
Abweichung Rotation	2.82°	1.43°	1.34°

Tabelle 5.6: Durchschnittliche Abweichung von der optimalen Position bei Verwendung von direkt- und statistisch-intensitätsbasierten Abstandsmaßen

	Kamerabild	separiertes Bild	synthetisches Bild
Abweichung Translation	43 cm	41 cm	24 cm
Abweichung Rotation	1.74°	1.56°	0.97°

Tabelle 5.7: Durchschnittliche Abweichung von der optimalen Position bei Verwendung von direkt- und räumlich-intensitätsbasierten Abstandsmaßen

des Kamerabildes auf. Bei der modifizierten Variante wird der Fehler verringert. Das beste Ergebnis wird bei der Verwendung derselben Bildmodalität erreicht.

Betrachtet man das Ergebnis zusammen mit denen aus dem zweiten Experiment, so ist festzustellen, dass die Positionsbestimmung genauer durchgeführt wird als bei der Anwendung der *direkt-intensitätsbasierten* Abstandsmaße. Der Grund dafür liegt im Einfluss der *statistisch-intensitätsbasierten* Abstandsmaße. Deren hoher durchschnittlicher Fehler, der beim Gebrauch derselben Bildmodalität auftritt, wird wiederum durch die *direkt-intensitätsbasierten* Abstandsmaße vermindert.

Direkt-Räumlich

Die Ergebnisse, die bei dieser Kombination erzielt werden, weisen einen hohen Einfluss der *räumlich-intensitätsbasierten* Abstandsmaße auf. Dies zeigt sich besonders bei dem Gebrauch des separierten Kamerabildes. Der große durchschnittliche Fehler der *räumlich-intensitätsbasierten* Abstandsmaße führt dazu, dass sich der durchschnittliche Fehler im Vergleich zum Kamerabild nicht wesentlich verändert.

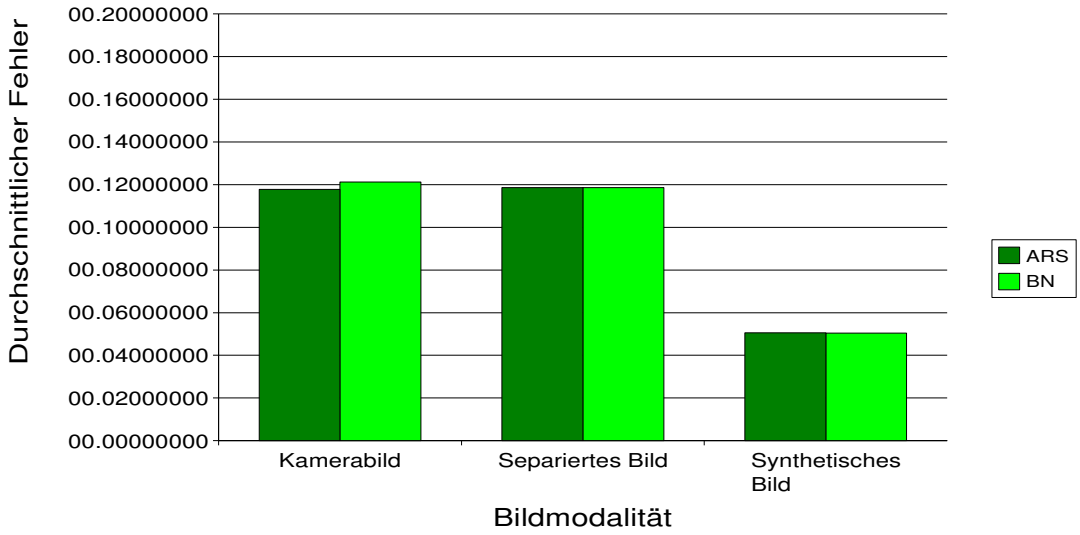


Bild 5.11: Durchschnittlicher Fehler der direkt-/räumlich-intensitätsbasierten Abstandsmaßkombination

	Kamerabild	separiertes Bild	synthetisches Bild
Abweichung Translation	36 cm	42 cm	21 cm
Abweichung Rotation	1.73°	1.63°	0.65°

Tabelle 5.8: Durchschnittliche Abweichung von der optimalen Position bei Verwendung von statistisch- und räumlich-intensitätsbasierten Abstandsmaßen

Statistisch-Räumlich

Auch bei dieser Kombination von Abstandsmaßtypen zeigt sich der Einfluss der *räumlich-intensitätsbasierten* Abstandsmaße. Der größte durchschnittliche Fehler wird mit dem Gebrauch des separierten Bildes gemacht. Im Vergleich zu der Kombination mit den *direkt-intensitätsbasierten* Abstandsmaßen wird eine wesentlich höhere Genauigkeit bei der Positionskorrektur erzielt.

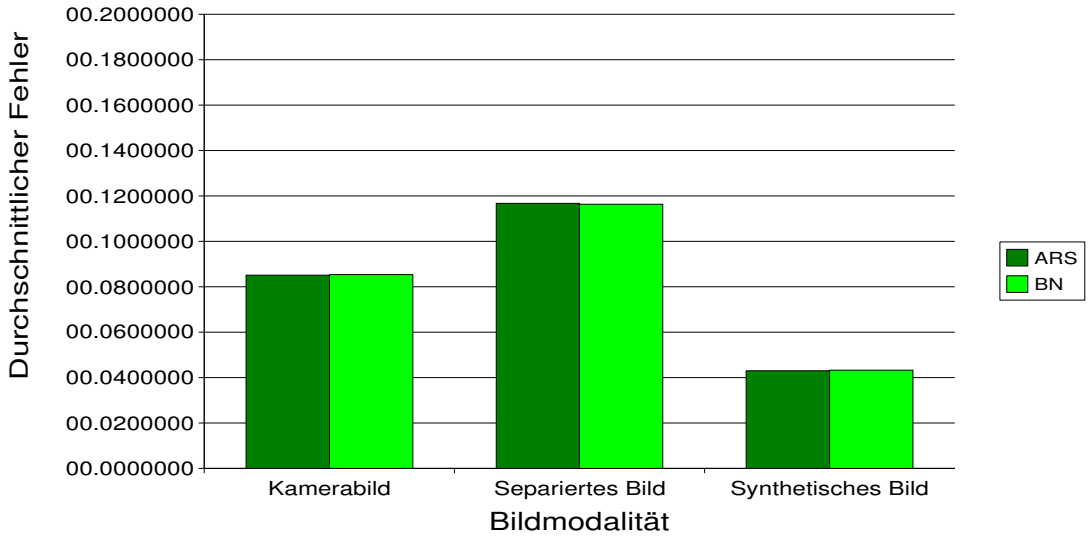


Bild 5.12: Durchschnittlicher Fehler der statistisch-/räumlich-intensitätsbasierten Abstandsmaßkombination

5.3.4 Vergleich aller Kombinationsmöglichkeiten

Der Vergleich zwischen allen Kombinationsmöglichkeiten der Bildabstandsmaßtypen zeigt, dass die besten Ergebnisse mit dem synthetischen Bild erzielt werden. Sowohl beim Kamerabild als auch beim Gebrauch des modifizierten Kamerabildes tritt die geringste Abweichung von der optimalen Position mit der Verwendung von *statistisch-intensitätsbasierten* Abstandsmaßen auf. Mit Hilfe der *räumlich-intensitätsbasierten*-Abstandsmaße wird, wie im Diagramm 5.13 zu erkennen ist, mit der Anwendung der selben Bildmodalität die genaueste Positionsbestimmung erreicht.

Die Verwendung von *direkt-intensitätsbasierten* Abstandsmaßen sowohl in Kombination als auch allein führt zu einer hohen Abweichung von der optimalen Position. Der durchschnittliche Fehler, der durch den Gebrauch dieses Abstandsmaßtyps auftritt, kann durch Kombination mit einem anderen Abstandsmaßtyp verringert werden.

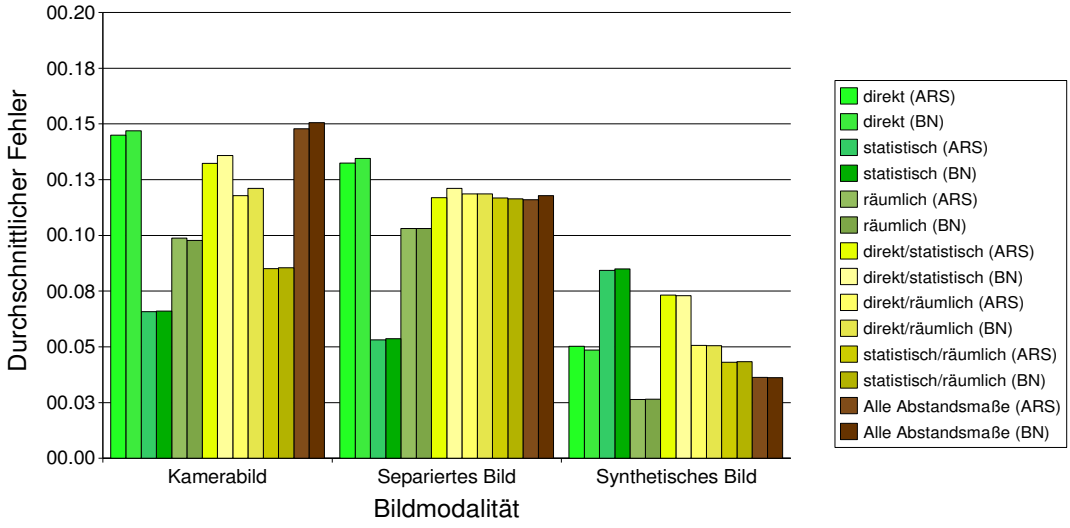


Bild 5.13: Gegenüberstellung aller Möglichkeiten

Kapitel 6

Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war die Verbesserung einer Positions- und Orientierungsangabe einer Kamera mit Hilfe von bildbasierten Registrierungsverfahren. Des Weiteren sollte herausgefunden werden, inwieweit eine Beschleunigung der Registrierung erreicht werden kann, wenn die Berechnung der Abstandsmaße auf den Grafikprozessor ausgelagert wird. Für das in dieser Arbeit angestrebte System sollte herausgefunden werden, ob und in welchem Maße eine Verbesserung der ursprünglichen Positionsangabe eingetreten ist.

Mit dieser Arbeit wurde erreicht, dass ein lauffähiges und in zahlreichen Tests evaluiertes System unter dem Betriebssystem Linux zur Verfügung steht. Die Korrektur der Positionsangabe erfolgt, wie gefordert, bildbasiert unter Verwendung aller drei Arten von intensitätsbasierten Bildabstandsmaßen. Das System verfügt über eine Benutzungsoberfläche, über die es bzw. seine Einzelbestandteile parametrisiert werden können. Eine auf Basis der Shader-Spache Cg erstellte GPU-Implementierung wurde von dem Betriebssystem Windows auf das Betriebssystem Linux portiert.

Die Benutzungsoberfläche ermöglicht die Verfolgung des Verlaufs der Bildregistrierung. Es werden für jeden Berechnungsschritt sowohl die errechneten Daten als auch das dazugehörige Bild angezeigt. Das Endergebnis jedes Registrierungsdurchlaufs wird ebenfalls auf der Benutzungsoberfläche angezeigt.

Die im Grundansatz dieser Diplomarbeit geforderte Syntheseeinheit, konnte mit Hilfe eines Renderers, der in der Arbeitsgruppe Computergrafik der Universität Koblenz-Landau

entwickelt wurde, umgesetzt werden. Zusätzlich wurde von der Arbeitsgruppe zu Testzwecken ein einfaches, texturiertes Modell eines auf dem Campus Koblenz aufgestellten Kunstwerks zur Verfügung gestellt. Zum Zeitpunkt der Durchführung dieser Arbeit konnte der Renderer nur unter dem Betriebssystem Windows benutzt werden. Die Anforderung, dass die Implementierungen für diese Arbeit unter dem Betriebssystem Linux erfolgen sollten, und der weit fortgeschrittene Stand der Arbeit, ließen eine Implementierung der Analyseeinheit auf Windows nicht mehr zu. Das Problem des Datentransfers zwischen Synthese- und Analyseeinheit konnte mit Hilfe einer eigens dafür implementierten Netzwerkschnittstelle gelöst werden. Die für die Bildsynthese benötigten Rotations- und Translationsparameter werden ebenfalls in der Netzwerkschnittstelle berechnet und zur Übermittlung an den Renderer in eine Datei geschrieben. Die Anforderung nach einer in weicher Echtzeit durchgeführten Positionskorrektur konnte aufgrund dieser Systemaufteilung und der damit verbundenen Netzwerkkomunikation nicht erreicht werden.

Zur Ermittlung der geforderten intrinsischen Kameraparameter wurde in das System eine Zhang-Kamerakalibrierung eingefügt. Diese kann mit Hilfe der Benutzungsoberfläche gesteuert werden. Die Ergebnisse werden auf der Oberfläche angezeigt. Des Weiteren wird der Erfolg der Kalibrierung durch eine Gegenüberstellung von ver- und entzerrter Kameraaufnahme auf der Benutzungsoberfläche sichtbar gemacht.

Für die Umsetzung der in dieser Arbeit geforderten Aufgaben, wurde in weiten Teilen auf eine bereits existierende Softwarelösung einer anderen Diplomarbeit zurückgegriffen. Der Entschluss, auf dieser Arbeit aufzusetzen, kam durch die in Kapitel 4.1 beschriebenen Analyse, die die Gemeinsamkeiten zwischen der vorhandenen Lösung und der gestellten Aufgabenstellung aufzeigte. Diese Entscheidung stellte sich im weiteren Verlauf als sehr nützlich heraus, da dadurch die vorliegende Arbeit trotz der erheblichen zeitlichen Verzögerung, die durch die bis November 2006 fehlende Syntheseeinheit verursacht wurde, noch in einem respektablen Umfang abgeschlossen werden konnte.

Die Shaderentwicklung unter Linux stellte sich als problematisch heraus. Alle Shader der übernommenen Arbeit, mit denen Bildabstandsmaße auf dem Grafikprozessor berechnet werden, sind unter Linux nur eingeschrängt benutzbar, da die Ergebnisse, gerade bei den räumlich-basierten Abstandsmaßen, fehlerhaft sind. Die Tests haben gezeigt, dass die unter Linux verfügbaren Grafikkartentreiber eine Ursache für die Fehler sein können. Das

zeigen z. B. die unterschiedlichen Berechnungsergebnisse der Shader trotz gleichbleibender Einstellungen des Programms. Dieses Verhalten, das nicht unter Windows wiederholt werden konnte, kann mit einem nicht unterstützten Texturformat zusammenhängen, das dazu führt, dass es zu undefinierten Zuständen der Grafikpipeline kommt.

Eine Neuimplementierung der Bildabstandsmaßshader in einer anderen Sprache hätte den zeitlichen Rahmen dieser Arbeit weit überschritten. Der Grund dafür liegt wiederum in der Entscheidung die bereits existierende Shaderimplementierung unter Linux lauffähig zu machen. Aufgrund der nicht für Linux erhältlichen Software für die Fehlersuche nahm die Portierung der Shader unter Linux erheblich mehr Zeit in Anspruch, als zunächst angenommen, da die Implementierung durch schrittweises Nachvollziehen und Nachimplementierung überprüft werden musste. Abschließend ist zum Thema GPGPU-Implementierung unter Linux zu sagen, dass aufgrund der fehlenden Implementierungsunterstützung für die Shaderentwicklung und des damit verbundenen Mehraufwands eine Entwicklung unter Windows vorteilhafter ist. Für die Programmierung der Shader kann die in dieser Ausarbeitung enthaltene Anleitung verwendet werden.

Angesichts der derzeitigen Systemarchitektur mit einer Syntheseeinheit unter Windows und der Analyseeinheit unter Linux sind objektive und aussagekräftige Geschwindigkeitsvergleiche zwischen GPU- und CPU-Implementierung nicht möglich. Die Tests wären nur unter der Voraussetzung möglich, dass sich beide Einheit auf dem selben Rechnersystem befinden. Die derzeitige Aufteilung führt jedoch dazu, dass das synthetisch erzeugte Bild zum einen über die sehr langsame Netzwerkverbindung und zum anderen über den PCI-Systembus vom Arbeitsspeicher in den Grafikkartenspeicher transferiert werden muss. Die Berechnungsgeschwindigkeit der Grafikkarte wird allerdings nur dann effektiv genutzt, wenn während des gesamten Registrierungsprozesses die Bilddaten im Grafikkartenspeicher bleiben und erst nach Abschluss aller Berechnungen wieder in den Arbeitsspeicher geladen werden. Erst unter diesen Voraussetzungen kann eine objektive Zeitmessung mit anschließendem Vergleich beider Berechnungseinheiten durchgeführt werden.

Ausblick: Um objektive Vergleichstests zwischen GPU- und CPU-Implementierung durchführen zu können, muss entweder die Syntheseeinheit auf Linux oder die Analyseeinheit auf Windows portiert werden. Aufgrund der besseren Unterstützung der Grafik-

kartentreiber unter Windows wäre diesbezüglich eine Portierung der Analyseeinheit auf Windows empfehlenswert. Darüber hinaus ist eine Portierung der Syntheseeinheit auf Linux mit einem sehr hohen Aufwand verbunden. Eine andere Möglichkeit wäre mit der Abänderung eines unter Linux verfügbaren Renderers gegeben.

In einer weiteren Testreihe sollte das Verhalten der *statistisch-intensiätsbasierten* Abstandsmaßen im Zusammenhang mit der Verwendung des synthetischen Bildes genauer getestet werden. Dabei könnte die Modellbeschreibung schrittweise um weitere Elemente erweitert werden, um das Verhalten der Abstandsmaße in Bezug auf die Zunahme der Entropie zu untersuchen. Die Erweiterung der 3-D-Modellszene würde auch eine Verringerung der Abweichung bei der Verwendung der *direkt-intensitätsbasierten* Abstandsmaße bewirken. Mit der schrittweisen Erweiterung der Modellbeschreibung könnte auch untersucht werden, ab welcher Detailstufe eine Positionsbestimmung durchgeführt werden kann, die auch für VR/AR-Anwendungen genutzt werden kann.

Für eine Neuimplementierung der Abstandsmaße auf der GPU sollte als Programmiersprache GLSL[1] eingesetzt werden. Damit wäre auch eine Möglichkeit gegeben, die angestrebte Implementierung auch auf AMD/ATI-Grafikkarten zu testen. Die Shadersprache Cg würde den Gebrauch der Abstandsmaßshader nur auf Nvidia-Grafikkarten einschränken.

[1] GLSL = OpenGL Shading Language

Verzeichnis der Bilder

1.1 Bildvergleich zwischen Kamerabild und synthetisch erzeugtem Bild . . . 12

2.1 Systeme für die Positionsbestimmung und ihre Anwendungsbereiche . . . 18
2.2 Ortsbestimmung durch Satellitennavigation 20
2.3 Aufbau einer Positionstabelle für die Positionsbestimmung im WLAN . . 22
2.4 Abfolge der Arbeitsschritte bei der 2-D/3-D-Registrierung 25
2.5 Verschiedene Auflösungsstufen der Kostenfunktion, um genauer das globale Optimum zu bestimmen 39
2.6 Aktivitätsdiagramm des ARS-Optimierungsverfahren 41
2.7 Aktivitätsdiagramm des Optimierungsverfahren Best Neighbour 43

3.1 Analyse durch Synthese Kodierung und Dekodierung 47
3.2 Links: ein synthetisiertes Ornamentbild, Rechts: Bild einer Ornamentanwendung 48
3.3 Aktiviätsdiagramm des generellen Prinzips der Analyse durch Synthese . 49
3.4 Block-Diagramm einer Geforce 6 58

4.1 Anwendungsfälle des Systems 65
4.2 Funktionspakete des Systems 67
4.3 Bild- und Texturdatenklassen 69

4.4 Abstandsmaßklassen und ihre Strukturklassen 71
4.5 Shaderklassen und Klassen für den Aufbau des benötigten Programmkontextes 73
4.6 Optimierungsklassen des Systems 74
4.7 Diagramm der Klasse Httpget 76
4.8 Anischt des Protokollreiters der Benutzungsoberfläche 78
4.9 Abstandsmaßreiter für die Auswahl und Einstellung der Abstandsmaße . 79
4.10 Reiter für die Kamerakalibrierung 80

5.1 Die verschiedenen Bildmodalitäten für die Experimente (Kamerabild, separiertes Bild, synthetisches Bild) 84
5.2 Aufbau des Rahmens für die Bestimmung der Kameraposition 85
5.3 Lage des Weltkoordinatensystems im 3-D-Modell 86
5.4 Suchraum, der um die optimale Position aufgespannt wird 87
5.5 Durchschnittlicher Fehler bei der Verwendung aller Abstandsmaße 88
5.6 Durchschnittlicher Fehler der direkt-intensitätsbasierten Abstandsmaße . 90
5.7 Durchschnittlicher Fehler der statistisch-intensitätsbasierten Abstandsmaße 91
5.8 Durchschnittlicher Fehler der räumlich-intensitätsbasierten Abstandsmaße 92
5.9 Gegenüberstellung aller Abstandsmaßtypen 93
5.10 Durchschnittlicher Fehler der direkt-/statistisch-intensitätsbasierte Abstandsmaßkombination 94
5.11 Durchschnittlicher Fehler der direkt-/räumlich-intensitätsbasierten Abstandsmaßkombination 96
5.12 Durchschnittlicher Fehler der statistisch-/räumlich-intensitätsbasierten Abstandsmaßkombination 97
5.13 Gegenüberstellung aller Möglichkeiten 98

Verzeichnis der Tabellen

5.1 Verwendete Abstandsmaße für die Experimente 87

5.2 Durchschnittliche Abweichung von der optimalen Position bei Verwendung aller Abstandsmaße 89

5.3 Durchschnittliche Abweichung von der optimalen Position bei Verwendung von direkt-intensitätsbasierten Abstandsmaßen 91

5.4 Durchschnittliche Abweichung von der optimalen Position bei Verwendung von statistisch-intensitätsbasierten Abstandsmaßen 92

5.5 Durchschnittliche Abweichung von der optimalen Position bei Verwendung von räumlich-intensitätsbasierten Abstandsmaßen 93

5.6 Durchschnittliche Abweichung von der optimalen Position bei Verwendung von direkt- und statistisch-intensitätsbasierten Abstandsmaßen . . . 95

5.7 Durchschnittliche Abweichung von der optimalen Position bei Verwendung von direkt- und räumlich-intensitätsbasierten Abstandsmaßen 95

5.8 Durchschnittliche Abweichung von der optimalen Position bei Verwendung von statistisch- und räumlich-intensitätsbasierten Abstandsmaßen . . 96

Literaturverzeichnis

[Bal00] Helmut Balzert. *Lehrbuch der Software-Technik*, volume 1. Sektrum Akademischer Verlag Heidelberg-Berlin, 2000.

[Bal04] Heide Balzert. *Lehrbuch der Objektmodellierung. Analyse und Entwurf*. Spektrum Akademischer Verlag, 2004.

[Bel59] Gordon Bell. Reduction of speechspectra by analysis-by-synthesis techniques, 1959.

[BS79] N.I. Badler and S.W. Smoliar. Digital representations of human movement. *ACM Computing Surveys*, 1979.

[FBPD06] Tobias Feldmann, Sahla Bouattour, Dietrich Paulus, and Frank Deinzer. Kombination verschiedener Ähnlichkeitsmaße für die 2D/3D-Registrierung von Röntgenbildern mittels Demokratischer Integration. In Handels et al. [HEH+06], pages 226–230.

[Fel05] Tobias Feldmann. Kombination verschiedener ähnlichkeitsmaße für die 2-D/3-D Registrierung von Röntgenbildern. Master's thesis, Universität Koblenz-Landau, 2005. Diplomarbeit teilweise extern betreut von Dr. Frank Deinzer bei Siemens Medical Solutions, Forchheim.

[FK03] Randima Fernando and Mark J. Kilgard. *The Cg Tutorial*. Addison-Wesley, 2003.

[GÖ6] Dominik Göddeke. Gpgpu::basic math tutorial, 2006. Letzter Zugriff: August 2006.

[HEH+06] Heinz Handels, Jan Ehrhardt, Alexander Horsch, Hans-Peter Meinzer, and Thomas Tolxdorff, editors. *Bildverarbeitung für die Medizin 2006*, Hamburg, 3 2006. Springer-Verlag, Berlin, Heidelberg, New York.

[IM06] Milan Ikits and Marcello Magallon. The opengl extension wrangler library, 2006. http://glew.sourceforge.net/, Letzter Zugriff: 02.08.2006.

[Kub06] Alexander Kubias. Effiziente, adaptive 2-d/3-d-registrierung von röntgenbildern und 3-d-volumen. Master's thesis, Universität Koblenz Landau, Campus Koblenz, Fachbereich 4 Informatik, Institut für Computervisualisitk, 9 2006.

[May79] Peter S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press Professional, Inc., 525 B St., San Diego, CA 92101-4412, USA, 1979.

[mis07] Opengl extension registry, 2007.

[MN78] D. Marr and H.K. Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. *Proc. Royal Society of London*, 1978.

[Moe99] T. B. Moeslund. The analysis-by-synthesis approach in human motion capture: A review, 1999.

[Pen99] Graeme P. Penney. *Registration of tomographic images to X-ray projections for use in image guided interventions*. PhD thesis, Computational Imaging Science Group, Dvision of Radiological Sciences and Medical Engineering, King's College, London, 1999.

[PH03] Dietrich Paulus and Joachim Hornegger. *Applied pattern recognition: A practical introduction to image and speech processing in C++*. Advanced Studies in Computer Science. Vieweg, Braunschweig, 4 edition, 2003.

[PWL+98] Graeme P. Penney, Jürgen Weese, John A. Little, Paul Desmedt, Derek L.G. Hill, and David J. Hawkes. A Comparison of Similarity Measures for Use in

2-D-3-D Medical Image Registration. *IEEE Trans. on Med. Img.*, 17(4):586–595, 1998.

[Rot05] Jörg Roth. *Mobile Computing*. dpunkt.verlag, 2005.

[RRS+03] Daniel B. Russakoff, Thorsten Rohlfing, Ramin Shahidi, Daniel H. Kim, John R. Adler, Jr., and Calvin R. Maurer, Jr. Intensity-based 2d-3d spine image registration incorporating one fiducial marker. *Lecture Notes in Computer Science*, 2878:287–294, 2003.

[SSP06] Clemens Schmidt, Christian Schneider, and Dietrich Paulus. Knowledge-based image analysis applied to ornaments in arts. In *Visual Media Production (CVMP 2006)*, pages 97–105, 11 2006.

[SWND03] David Schreiner, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide, Fourth Edition*. Addison-Wesley, 2003.

[TvdM01] Jochen Triesch and Christoph von der Malsburg. Democratic integration: Self-organized integration of adaptive cues. *Neural Computation*, (13):2049–2047, 2001.

[Wei03] Wolfgang Wein. Intensity based rigid 2d-3d registration algorithms for radiation therapy. Master's thesis, Technische Universität München, Fakultät für Informatik, 2003.

[Woj04] Christian Wojek. Visuelle personenverfolgung mit partikelfiltern. Studienarbeit, Institut für Logik, Komplexität und Dedutktionssysteme, Fakultät für Informatik, Universität Karlsruhe, 4 2004.